ANIL MAHESHWARI

NOTES ON ALGORITHM DESIGN

COMPUTER PROGRAMMING IS AN ART, BECAUSE IT APPLIES ACCUMULATED KNOWLEDGE TO THE WORLD, BECAUSE IT REQUIRES SKILL AND INGENUITY, AND ESPECIALLY BECAUSE IT PRODUCES OBJECTS OF BEAUTY. A PROGRAMMER WHO SUBCONSCIOUSLY VIEWS HIMSELF AS AN ARTIST WILL ENJOY WHAT HE DOES AND WILL DO IT BETTER.

— DONALD E. KNUTH

TO ME THE VERY ESSENCE OF EDUCATION IS CONCENTRATION OF MIND, NOT THE COLLECTION OF FACTS.

— SWAMI VIVEKANANDA

PUBLISHED BY USING THE STYLE FILES FROM THE TUFTE-LATEX DEVELOPERS

April 2025

Contents

Preface

These notes extensively use material from the course notes of Lars Arge, David Mount, COMP 2805/3803 Notes of myself and Michiel Smid [1], CLRS book [2], Knuth's Art of Computer Programming [3], Kleinberg and Tardos Algorithms book [4], Leskovec, Rajaram and Ullman's book on Algorithms for Massive Data Sets [5]. These notes are updated occasionally. A substantial update was done in Fall 2013. Chapters on elementary probability, locality sensitive hashing, dimensionality reduction, and several exercises have been added. Moreover, as part of the offering of COMP 5703 in the Fall of 2013, some of the students contributed significantly. Gregory Bint updated the chapter on the Minimum Spanning Trees and has added a new section on spanning-tree verification. Alexis Beingessner has provided a section on the extension of the planar separator theorem. In the Fall 2015 term, I started to work on the chapter on the Second Moment Method. The addition of this chapter was inspired by a comment from one of the referees of our paper [6] in ALGOSENSORS 2015, where s/he mentioned that the paper is an excellent introduction to the Second Moment Method at the graduate level. I have pasted the whole article, more or less verbatim, in that chapter and added a section on Cliques. Over time this chapter may evolve. In Summer 2017, I used a new style file from the Tufte-LaTeX developers to modernize these notes. In Fall 2019, I completed the first draft of the chapter on Data Streams. This is followed by a chapter on Online Algorithms and Multiplicative-Weight Update Method in Spring/Summer 2020 (thanks to COVID-19 with a forced stay abroad). In Spring 2021 I was able to complete the first draft of chapter on Dimensionality Reduction. In Summer 2023, I have added a chapter on design techniques for approximaltion algorithms. This also includes a section of FPTs, but that will likely become a chapter on its own in future. I am planning to cover some of the classical results as a series of exercises in various chapters. This will be an ongoing task. I like this way of learning and want to reflect those in the notes.

I have used parts of this material for the graduate course COMP 5703 (Algorithms), the undergraduate course COMP 3801 (Algorithms for Modern Data Sets), and the new graduate course COMP 5112 (Algorithms in Data Science) at Carleton. These notes aim to summarize the discussions in the lectures. They are not designed as stand-alone chapters or comprehensive coverage of a topic. The exercises are from numerous sources - I have tried to cite the sources. But I have likely missed the citation for many, and I will welcome the

[1] A. Maheshwari and M. Smid. *Introduction to Theory of Computation*. Free Online, 2012

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 4th edition, 2022

[3] Donald E. Knuth. *The art of computer programming, volume 1-3*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998

[4] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005

[5] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011

[6] Ahmad Biniaz, Evangelos Kranakis, Anil Maheshwari, and Michiel Smid. Plane and planarity thresholds for random geometric graphs. In *Proc. ALGOSENSORS 2015 (Patras, Greece)*, Lecture Notes in Computer Science, Berlin, Germany, 2015. Springer

right pointers.

These notes assume a basic familiarity with Data Structures (Binary Trees, Heaps), basic algorithms (searching and sorting), their analysis, a course in discrete mathematics including graph theory and combinatorics, a first-year calculus, a first-year linear algebra, and a first-year probability course. Since these chapters have been written over time and many of the TeX tools weren't available in olden times, you will see that the initial chapters don't have elegant figures or texts. Hopefully, volunteers in the future will modernize parts of these notes.

If you spot any errors or have a suggestion that can help me improve these notes, I will be glad to hear from you. If you wish to add material to these notes, including exercises, please do get in touch. Thanks in advance!

Art work is by Arti. Thanks!

Anil Maheshwari (anil@scs.carleton.ca)

1
Preliminaries

We will focus on

1. What is an Algorithm
2. Model of Computation
3. Asymptotic Notation
4. Analyzing Recurrences
5. Matrix Multiplication Algorithms

Keywords: O, Ω, Θ, Recurrences, Recursion Tree, Analyzing Recurrence Relations, Matrix Multiplication.

1.1 Introduction

These notes are about *designing* and *analyzing* algorithms for efficiently solving computational problems.

- *What is an Algorithm?*.

  Algorithms are well-defined procedures that transform an input into an output. They are not programs, but they are often specified like them. An algorithm can be implemented in several ways.

  Knuth's, Art of Computer Programming, vol.1. [1], is a good resource on the history of algorithms. He says that an *algorithm* is a finite set of rules that gives a sequence of operations for solving a specific type of problem. Algorithm has five important features:

  Finiteness: must terminate after a finite number of steps.

  Definiteness: each step is precisely described.

  Input: algorithm has zero or more inputs.

  Output: has at least one output!

  Effectiveness: Each operation should be sufficiently basic such that they can be done in a finite amount of time using pencil and

Is it Mis-spelled *logarithm?*

The first most popular algorithm is Euclid's algorithm for computing the GCD of two numbers.

A mathematician from the 9th century, Al Khwarizmi, listed algorithms for adding, multiplying, dividing, square roots, etc. of numbers in his book. He is regarded as the inventor of Algorithms.

[1] Donald E. Knuth. *The art of computer programming, volume 1-3*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998

paper.

- *Design*: The focus of these notes is on how to design good algorithms, how to prove their correctness, and how to analyze their efficiency. We will study methods/ideas/tricks for developing fast and efficient algorithms. A primary aim is to understand some key techniques that can be applied to many computational problems.
- *Analysis*: Abstract/mathematical comparison of algorithms (without implementing, prototyping and testing them).
- These notes will require proving the correctness of algorithms and their analysis. Therefore, MATH is the main tool and is required for
  - Formal specification of problems
  - Correctness proofs
  - Analysis of efficiency (time, memory usage,...)

  Please review mathematical induction, what is a proof, logarithms, the sum of series, elementary number theory, permutations, factorials, binomial coefficients, Harmonic numbers, Fibonacci numbers and generating functions (Knuth's vol 1. [2] or the book on Concrete Mathematics [3] are excellent resources).
- **Algorithms matter**: See how the algorithms are shaping the 21st century - internet, smart devices, data mining, e-commerce, online education, search engine, self-driving cars, influencing public-opinion, space exploration, drug design, ...
- When is an algorithm efficient?

  The tradition is to design algorithms for problems with the worst-case running time bounded by a polynomial in the input size. For example, algorithms with the worst-case running times of $O(n)$, $O(n^2)$, $O(n^3)$, for problems of input size n are considered efficient. Not all the problems have efficient algorithms. And there are several problems for which we don't know whether they have efficient algorithms. For example, explore what are complexity classes **P** and **NP** and the Millennium Prize Problem: Is $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$?

[2] Donald E. Knuth. *The art of computer programming, volume 1-3*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998

[3] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1994

http://www.claymath.org/millennium-problems/p-vs-np-problem

1.2 Model of Computation

A computation model typically describes how the memory, the communication, and the computation are organized in a computer. It helps us design an algorithm, irrespective of worrying about the nitty-gritty details of a specific computer (e.g. size of RAM, type of processor), and to evaluate an algorithm's computational complexity. There are several models of computation depending on the context.

For example, we study finite automata, context-free grammars, and Turing machines in a typical Theory of Computation course. In a parallel algorithms course, we may learn various parallel computing models (e.g., PRAM, Hypercubes, BSP). For these notes, we will use the RAM model of computation.

Random-access machine (RAM) model:

- Memory consists of an infinite array of cells
- Each cell can store at most one data item (bit, byte, a record, ..)
- Any memory cell can be accessed in unit time
- Instructions are executed sequentially
- All basic instructions take unit time:
  - Load/Store
  - Arithmetic's (e.g. $+, -, *, /$)
  - Logic (e.g. $>$)

The benefits of the RAM model are:

- Helps predict the resources used by the algorithm: running time and space used.
- *Running time* of an algorithm is the number of RAM instructions it executes.
- RAM model is not realistic, e.g.
  - memory is finite (even though we often imagine it to be infinite when we program)
  - not all memory accesses take the same time (cache, main memory, disk)
  - not all arithmetic operations take the same time (e.g. multiplications are expensive)
  - instruction pipelining
  - other processes
- But the RAM model if often enough to give relatively realistic results.

1.3 *Asymptotic Analysis*

We do not want to compute a detailed expression of an algorithm's run time, but instead, we would like to feel what it is like. We would like to see the trend - i.e. how does it increase when the size of the input increases- is it linear in the size of the input? Or quadratic? Or exponential? Or who knows? The asymptotics essentially captures the growth rate of the underlying functions describing the run-time. The asymptotic analysis assumes that the input size is large (since we are interested in how the running time increases when the problem size grows). It ignores the constant factors (which usually depend on the hardware, programming smartness or tricks, and compile-time optimization).

David Mount suggests the following simple definitions based on the limits of functions describing the running time of algorithms. We will describe the formal definitions later.

https://www.cs.umd.edu/class/fall2013/cmsc451/Lects/cmsc451-fall13-lects.pdf

Let $f(n)$ and $g(n)$ be two positive functions of n. What does it mean when we say that both f and g grow at roughly the same rate for large n (ignoring the constant factors), i.e.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c,$$

where c is a constant and is neither 0 nor ∞. We say that $f(n) \in \Theta(g(n))$, i.e. they are asymptotically equivalent. What about if $f(n)$ does not grow significantly faster than $g(n)$ or if it grows significantly faster? Here is the table capturing these relationships:

Asymptotic Form	Relationship	Definition
$f(n) \in \Theta(g(n))$	$f(n) \equiv g(n)$	$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$
$f(n) \in O(g(n))$	$f(n) \leq g(n)$	$0 \leq \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$
$f(n) \in \Omega(g(n))$	$f(n) \geq g(n)$	$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)}$
$f(n) \in o(g(n))$	$f(n) < g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
$f(n) \in \omega(g(n))$	$f(n) > g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Example 1.3.1 *Show that* $T(n) = \sum_{x=1}^{n} x^2 \in \Theta(n^3)$.

Proof. Note that $\sum_{x=1}^{n} x^2 = \frac{2n^3+3n^2+n}{6}$.

Thus,

$$\begin{aligned}\lim_{n\to\infty}\frac{T(n)}{n^3} &= \lim_{n\to\infty}\frac{(n^3+3n^2+2n)/6}{n^3}\\ &= \lim_{n\to\infty}\left(\frac{n^3}{6n^3}+\frac{3n^2}{6n^3}+\frac{2n}{6n^3}\right)\\ &= \lim_{n\to\infty}\left(\frac{1}{6}+\frac{1}{2n}+\frac{1}{3n^2}\right)\\ &= \frac{1}{6}\end{aligned}$$

Since $0 < 1/6 < \infty$, we have $T(n) = \sum_{x=1}^{n} x^2 \in \Theta(n^3)$. ∎

Next we formally define O, Θ, and Ω notations.

1.3.1 O-notation

$O(g(n)) = \{f(n) : \exists$ constants $c, n_0 > 0$ such that $f(n) \leq cg(n),\ \forall n \geq n_0\}$

- $O(\cdot)$ is used to asymptotically *upper bound* a function.
- $O(\cdot)$ is used to bound the *worst-case* running time (see Figure 1.1).

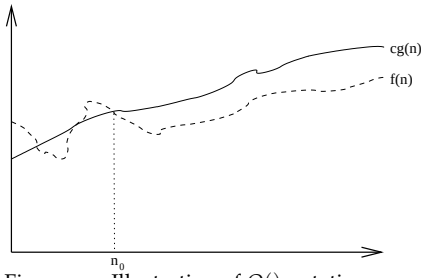


Figure 1.1: Illustration of $O()$ notation.

Example 1.3.2 *Show that* $\frac{1}{3}n^2 - 3n \in O(n^2)$.

Proof. Observe that for $c = \frac{1}{3}$ and $n > 1$, $\frac{1}{3}n^2 - 3n \leq cn^2$. Alternatively, using the limits, we have $\lim_{n\to\infty}\frac{\frac{1}{3}n^2-3n}{n^2} = \frac{1}{3}$. As $\frac{1}{3} < \infty$, $\frac{1}{3}n^2 - 3n \in O(n^2)$. ∎

Example 1.3.3 *Let* $p(n) = \sum_{i=0}^{d} a_i n^i$ *be a polynomial of degree d and assume that* $a_d > 0$. *Show that* $p(n) \in O(n^k)$, *where* $k \geq d$ *is a constant.*

Proof. Using the limits we have $\lim_{n\to\infty}\frac{\sum_{i=0}^{d} a_i n^i}{n^k} = \frac{a_d}{n^{k-d}} \leq a_d < \infty$. Thus, $p(n) \in O(n^k)$. ∎

As an exercise, find c and n_0 using the formal definition of O-notation.

Note:

- When we say "the running time is $O(n^2)$", we mean that the *worst-case* running time is $O(n^2)$ — best case may be better.
- We often abuse the notation:
  - We write $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$.

- We often use $O(n)$ in equations: e.g. $2n^2 + 3n + 1 = 2n^2 + O(n)$ (meaning that $2n^2 + 3n + 1 = 2n^2 + f(n)$ where $f(n)$ is some function in $O(n)$).
- We use $O(1)$ to denote a constant.

1.3.2 Ω-notation (big-Omega)

$\Omega(g(n)) = \{f(n) : \exists$ constants $c, n_0 > 0$ such that $cg(n) \leq f(n), \forall n \geq n_0\}$

- $\Omega(\cdot)$ is used to asymptotically *lower bound* a function (see Figure 1.2).

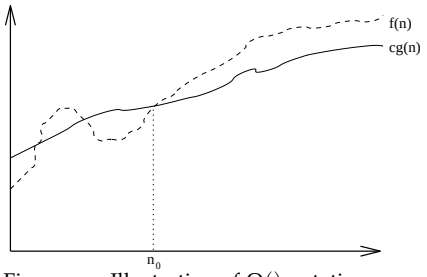


Figure 1.2: Illustration of $\Omega()$ notation.

Example 1.3.4 *Show that* $\frac{1}{3}n^2 - 3n = \Omega(n^2)$.

Proof. Observe that for $c = \frac{1}{6}$ and $n > 18$, $\frac{1}{3}n^2 - 3n \geq cn^2$. Moreover, $\lim_{n\to\infty} \frac{\frac{1}{3}n^2 - 3n}{n^2} = \frac{1}{3}$. As $0 < \frac{1}{3}$, $\frac{1}{3}n^2 - 3n \in \Omega(n^2)$. ∎

Example 1.3.5 *Let* $p(n) = \sum_{i=0}^{d} a_i n^i$ *be a polynomial of degree* d *and assume that* $a_d > 0$. *Show that* $p(n) \in \Omega(n^k)$, *where* $k \leq d$ *is a constant.*

Proof. Using the limits we have $\lim_{n\to\infty} \frac{\sum_{i=0}^{d} a_i n^i}{n^k} = a_d n^{d-k} \geq a_d > 0$. Thus, $p(n) \in \Omega(n^k)$. ∎

Note that when we say "the running time is $\Omega(n^2)$", we mean that the *best case* running time is $\Omega(n^2)$ — the worst case might be worse.

1.3.3 Θ-notation (Big-Theta)

$\Theta(g(n)) = \{f(n) : \exists\, c_1, c_2, n_0 > 0$ such that $c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall n \geq n_0\}$

- $\Theta(\cdot)$ is used to asymptotically *tight bound* a function.

$f(n) = \Theta(g(n))$ *if and only if* $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ (see Figure 1.3)

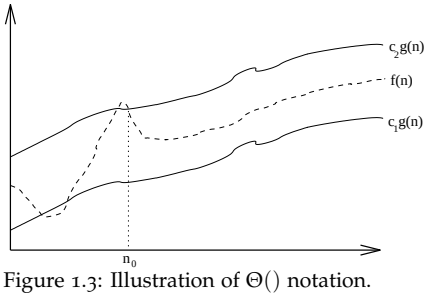


Figure 1.3: Illustration of $\Theta()$ notation.

Example 1.3.6 *Show that* $6n \log n + \sqrt{n} \log^2 n = \Theta(n \log n)$.

Proof. We need to find n_0, c_1, c_2 such that

$$c_1 n \log n \leq 6n \log n + \sqrt{n} \log^2 n \leq c_2 n \log n$$

for $n > n_0$.

Since $c_1 n \log n \leq 6n \log n + \sqrt{n} \log^2 n \Rightarrow c_1 \leq 6 + \frac{\log n}{\sqrt{n}}$.

This holds if we choose $c_1 = 6$ and $n_0 = 1$.

Now for $6n \log n + \sqrt{n} \log^2 n \leq c_2 n \log n \Rightarrow 6 + \frac{\log n}{\sqrt{n}} \leq c_2$.

This holds for $c_2 = 7$ as $\log n \leq \sqrt{n}$ for $n \geq 2$.

Therefore, $c_1 = 6$, $c_2 = 7$ and $n_0 = 2$ ensures $6n \log n + \sqrt{n} \log^2 n = \Theta(n \log n)$.

Alternatively, $\lim_{n \to \infty} \frac{6n \log n + \sqrt{n} \log^2 n}{n \log n} = 6$. Since $0 < 6 < \infty$, we have $6n \log n + \sqrt{n} \log^2 n = \Theta(n \log n)$. ∎

Example 1.3.7 *Let* $p(n) = \sum_{i=0}^{d} a_i n^i$ *be a polynomial of degree* d *and assume that* $a_d > 0$. *Then* $p(n) \in \Theta(n^d)$.

Proof. From limits we know that $\lim_{n \to \infty} \frac{p(n)}{n^d} = a_d$ and $0 < a_d < \infty$. Thus, $p(n) \in \Theta(n^d)$. ∎

1.4 How to Analyze Recurrence Relations

Many divide-and-conquer algorithms running times are expressed as a recurrence relation. For example, merge-sort and quick-sort have the recurrence relation $T(n) = 2T(\frac{n}{2}) + O(n)$, where n is the size of the problem (= number of elements to be sorted) and $T(n)$ represents the time to sort the set of n elements. To estimate the running time, we need to unfold the recurrence relation and obtain a 'closed form' expression for the running time. There are many ways of solving recurrences. We will illustrate two main methods. Algorithms books refer to them as the *recursion-tree* method and the *substitution* method.

1.4.1 Recursion Tree Method

Visualize the unfolding of the recurrence relation as a tree. The nodes of the tree represent the cost incurred at the various levels of the recursion. We illustrate this method using the following recurrence (so-called the recurrence used in the Masters' method).

Let $a \geq 1, b > 1$ and $c > 0$ be constants and let $T(n)$ be the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + cn^k,$$

defined for integer $n \geq 0$. Then

Case 1: $a > b^k$ then $T(n) = \Theta(n^{\log_b a})$.

Case 2: $a = b^k$ then $T(n) = \Theta(n^k \log_b n)$.

Case 3: $a < b^k$ then $T(n) = \Theta(n^k)$.

Before we proceed with the proof, note that in the recurrence relation $T(n) = aT\left(\frac{n}{b}\right) + cn^k$, the cn^k term accounts for the time for splitting (dividing) the problem of size n into subproblems and the time for merging the solutions of the subproblem to obtain the solution for the problem. For the case of sorting recurrence relation $T(n) = 2T(\frac{n}{2}) + O(n)$, we have $a = 2$, $b = 2$, and the time for splitting and merging a problem of size n is $O(n)$. The proof is fairly simple and uses the sum of the terms in a geometric series. We need to visualize the levels of the underlying recursion tree (see Figure 1.4).

Level 1: a subproblems are formed, each of size n/b, and the total cost is cn^k.

Level 2: a^2 subproblems are formed, each of size n/b^2, and the total cost is $a * c(n/b)^k$.

Level 3: a^3 subproblems are formed, each of size n/b^3, and the total cost is $a^2 * c(n/b^2)^k$.

...

Level $\log_b n$: $a^{\log_b n}$ subproblems are formed, each of constant size and the total cost is $\approx a^{\log_b n} c(\frac{n}{b^{\log_b n}})^k$.

Therefore the total cost is

$$T(n) = O(n^{\log_b a}) + \sum_{i=0}^{\log_b n} a^i c\left(\frac{n}{b^i}\right)^k.$$

The expression $n^{\log_b a}$ is the number of leaves of this recurrence tree. Each leaf represents a constant size problem, and all the leaves in total require $O(n^{\log_b a})$ computation time. The term $\sum_{i=0}^{\log_b n} a^i c\left(\frac{n}{b^i}\right)^k$ is the summation of the costs for the divide and conquer steps at each level of the recurrence. Note that this is a geometric series whose sum is typically governed by the first term of the series or

Geometric Series - A Summary
Consider the series $S = 1 + x + x^2 + \cdots + x^n$, where x is a constant independent of n.
If $x = 1$, $S = n + 1$.
If $x = 0$, $S = 1$.
Assume $x \neq \{0, 1\}$.
Note $Sx = x + x^2 + \cdots + x^{n+1}$.
Therefore, $S - Sx = 1 - x^{n+1}$.
Since $x \neq 1$, we have $S = \frac{1-x^{n+1}}{1-x}$
Observe that
If $0 < x < 1$, $S = \frac{1-x^{n+1}}{1-x} \leq \frac{1}{1-x} = \Theta(1)$.
If $0 < x < 1$ and $n \to \infty$, $S = \frac{1}{1-x}$.
If $x > 1$, $S_n = \frac{x^{n+1}-1}{x-1} \geq \frac{x^{n+1}-x^n}{x-1} = x^n$ and
$S_n = \frac{x^{n+1}-1}{x-1} \leq \frac{x^{n+1}}{x-1} = \frac{x}{x-1}x^n = O(x^n)$.
Therefore, for $x > 1$, $S = \Theta(x^n)$, i.e.,
S is proportional to the last term of the series.

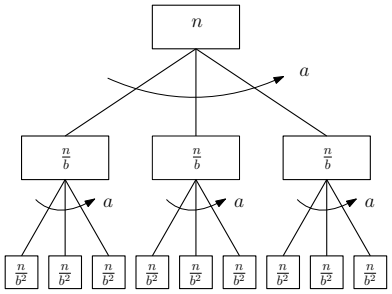


Figure 1.4: Illustration of recurrence tree.

the last term of the series or each term if they are all equal. We can derive the various cases by evaluating the geometric series.

For example, consider Case 2, where $a = b^k$. Now $\sum_{i=0}^{\log_b n} a^i c \left(\frac{n}{b^i}\right)^k = \sum_{i=0}^{\log_b n} cn^k = \Theta(n^k \log_b n)$. Thus $T(n) = O(n^{\log_b a}) + \Theta(n^k \log_b n) = \Theta(n^k \log_b n)$.

Other cases are left as exercises.

Useful log **identities**

- $2^5 = 32$
- $\log_2 32 = 5$
- $\log_2(a \cdot b) = \log_2 a + \log_2 b$
- If $b \neq 0$, $\log_2(a/b) = \log_2 a - \log_2 b$
- $\log_2(a^b) = b \log_2 a$
- $a^{\log_2 b} = b^{\log_2 a}$
- $\log_2 a = \frac{\log_d a}{\log_d 2}$
  For example $\log_2(1024) = \frac{\log_{10}(1024)}{\log_{10} 2} = \frac{3.01}{.301} = 10$
- $2^{\log_2(a)} = a$
- $\log_2(\sqrt[k]{a}) = \frac{1}{k} \log_2 a$

1.4.2 *Substitution Method*

Consider the following recurrence

$$T(n) = T(n/3) + T(2n/3) + n.$$

We can assume $T(n) = O(1)$ for small values of n. This recurrence doesn't fit the format discussed above. We may try $T(n) = T(n/3) + T(2n/3) + n \leq 2T(2n/3) + n$ and apply the above method to solve the recurrence. As $a = 2$, $b = 3/2$, and $k = 1$, the Case 1 applies and the recurrence evaluates to $T(n) \leq O\left(n^{\log_{\frac{3}{2}} 2}\right)$. Note that $\log_{\frac{3}{2}} 2 \approx 1.7$. But, using the substitution method, we will see that this recurrence evaluates to $\Theta(n \log n)$, and note that $n \log n = o\left(n^{\log_{\frac{3}{2}} 2}\right)$.

We can try the substitution method for these (or any) recurrences. Here, we guess a solution and verify that our guess is correct using induction. Typically, the complexity of an algorithm for a problem of size n will be one of $\{O(\log n), O(n), O(n \log n), O(n^2), O(n^2 \log n), O(n^3), \ldots\}$, and hence using induction, we can try to see which one of these expressions work. Next we show that $T(n) = T(n/3) + T(2n/3) + n = O(n \log n)$ by using induction on the problem size n.

We want to show that $T(n) = T(n/3) + T(2n/3) + n = O(n \log n)$ using induction on n. By assumption, it holds for small values of n. Assume that by the induction hypothesis, the statement holds for all values $k < n$, i.e. $T(k) \leq ck \log_2 k$ for some appropriately chosen constant c. We want to show that it holds for n, i.e. $T(n) \leq cn \log_2 n$.

Need to show that

$$\begin{aligned}
& cn \log_2 n \geq T(n/3) + T(2n/3) + n \\
\Leftrightarrow \quad & cn \log_2 n \geq c\frac{n}{3} \log_2 \frac{n}{3} + c\frac{2n}{3} \log_2 \frac{2n}{3} + n \text{ (Using induction hypothesis)} \\
\Leftrightarrow \quad & cn \log_2 n \geq c\frac{n}{3}(\log_2 n - \log_2 3) + c\frac{2n}{3}(\log_2 2 + \log_2 n - \log_2 3) + n \\
\Leftrightarrow \quad & 0 \geq c\frac{n}{3}(-\log_2 3) + c\frac{2n}{3} - c\frac{2n}{3} \log_2 3 + n \\
\Leftrightarrow \quad & 0 \geq -c \log_2 3 + c\frac{2}{3} + 1 \\
\Leftrightarrow \quad & c \geq \frac{1}{\log_2 3 - \frac{2}{3}} \approx 1.09
\end{aligned}$$

Therefore, the statement holds if we choose for example $c = 2$, i.e. $T(n) = T(n/3) + T(2n/3) + n = O(n \log n)$.

Consider another example, where $T(n) = 2T(\frac{n}{2}) + n \log n$, and assume that for small values of n, $T(n) = O(1)$. This recurrence relation also doesn't fit in the format to apply the recursion tree method. Using the substitution method, we will show that $T(n) = O(n \log^2 n)$ by applying induction on the problem size n.

Since $T(n) = O(1)$ for small values of n, we can assume that the base case holds. Assume that for all values of $k < n$, $T(k) \leq ck \log^2 k$ for some appropriately chosen constant c. We assume the base of log is 2.

Need to show that

$$
\begin{aligned}
& cn \log^2 n \geq 2T\left(\frac{n}{2}\right) + n \log n \\
\Leftrightarrow \quad & cn \log^2 n \geq 2c\frac{n}{2} \log^2 \frac{n}{2} + n \log n \text{ (Using induction hypothesis)} \\
\Leftrightarrow \quad & cn \log^2 n \geq cn(\log n - 1)^2 + n \log n \\
\Leftrightarrow \quad & cn \log^2 n \geq cn\left(\log^2 n + 1 - 2 \log n\right) + n \log n \\
\Leftrightarrow \quad & 0 \geq cn - 2cn \log n + n \log n \\
\Leftrightarrow \quad & c(2 \log n - 1) \geq \log n \\
\Leftrightarrow \quad & c \geq \frac{\log n}{2 \log n - 1}
\end{aligned}
$$

Therefore, the statement holds if we choose $c = 1$. Thus, $T(n) = 2T(\frac{n}{2}) + n \log n = O(n \log^2 n)$.

In summary, if we are able to 'guess' the right answer, the substitution method is a good choice to analyze the recurrence relations.

1.5 *Recurrence for Matrix Multiplication*

This is a classical example to illustrate the recurrences as well as the divide and conquer method. Consider Strassen's matrix multiplication method [4] as illustrated in the following. Let X, Y and Z be three $n \times n$ matrices, where $Z = X \cdot Y$. There are n rows, n columns and $n \times n$ entries in each of the matrices.

[4] D. Kozen. *The design and analysis of algorithms*. Springer, 1992; and V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969

- $X = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nn} \end{bmatrix}$

- $Y = \begin{bmatrix} y_{11} & \cdots & y_{1n} \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ y_{n1} & \cdots & y_{nn} \end{bmatrix}$
- We want to compute $Z = X \cdot Y$, where

$$z_{ij} = \sum_{k=1}^{n} x_{ik} \cdot y_{kj}$$

- How many operations we require?
- In all we generate n^2 entries in the matrix Z and each entry requires n multiplications and $n-1$ additions. So the total number of operations can be bounded by $O(n^3)$.
- Next we want to discuss a divide and conquer solution by Strassen that requires only $O(n^{\log_2 7}) \approx O(n^{2.81})$ operations.
- Let's first analyze the recurrence

$$T(n) = 7T(n/2) + cn^2,$$

where c is a constant, n is a positive integer, and $T(constant) = O(1)$.

- Using the recursion tree method, $a = 7$, $b = 2$, $c = c$, $k = 2$ and $a > b^k$. Hence $T(n) = O(n^{\log_2 7})$.

1.5.1 *Strassen's Algorithm*

- Divide each of the matrices into four sub-matrices, each of dimension $n/2 \times n/2$. Strassen observed the following:

$$Z = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} S_1 + S_2 - S_4 + S_6 & S_4 + S_5 \\ S_6 + S_7 & S_2 + S_3 + S_5 - S_7 \end{bmatrix}$$

where

$$\begin{aligned} S_1 &= (B - D) \cdot (G + H) \\ S_2 &= (A + D) \cdot (E + H) \\ S_3 &= (A - C) \cdot (E + F) \\ S_4 &= (A + B) \cdot H \\ S_5 &= A \cdot (F - H) \\ S_6 &= D \cdot (G - E) \\ S_7 &= (C + D) \cdot E \end{aligned}$$

- Lets test that for $S_4 + S_5$, which is supposed to be $AF + BH$.

$$\begin{aligned} S_4 + S_5 &= (A+B)\cdot H + A\cdot(F-H) \\ &= AH + BH + AF - AH \\ &= AF + BH \end{aligned}$$

- This leads to a divide-and-conquer algorithm with the recurrence relation $T(n) = 7T(n/2) + \Theta(n^2)$, since
  - We only need to perform 7 multiplications recursively. Additions/Subtractions only take $\Theta(n^2)$ time, and we need to perform 18 of them for $n/2 \times n/2$ matrices for each step of the recursion.
  - Division/Combination can still be performed in $\Theta(n^2)$ time.

Matrix multiplication is a fundamental problem, and it arises in almost all branches of Sciences, Social Sciences and Engineering. For example, high-energy physicists multiply monstrous matrices. There have been numerous improvements over Strassen's method. Note that any matrix multiplication algorithm needs to perform $\Omega(n^2)$ operations since the output matrix Z has many entries. See Exercise 1.45 to see a faster probabilistic method to verify whether the product of two matrices equals the third matrix.

1.6 Exercises

1.1 *Let $S = 1 + x + x^2 + \cdots + x^n$. Show the following*

1. *If $x = 1$, $S = n + 1$.*
2. *If $x = 0$, $S = 1$.*
3. *If $x \neq \{0, 1\}$, $S = \frac{1-x^{n+1}}{1-x}$.*
4. *If $0 < x < 1$ and $n \to \infty$, $S = \frac{1}{1-x}$.*
5. *If $0 < x < 1$, $S = \frac{1-x^{n+1}}{1-x} \leq \frac{1}{1-x} = \Theta(1)$.*
6. *If $x > 1$, $S_n = \frac{x^{n+1}-1}{x-1} \geq \frac{x^{n+1}-x^n}{x-1} = x^n$ and $S_n = \frac{x^{n+1}-1}{x-1} \leq \frac{x^{n+1}}{x-1} = \frac{x}{x-1}x^n = O(x^n)$.*
7. *For $x > 1$, $S = \Theta(x^n)$, i.e. S is proportional to the last term of the series.*

1.2 *Show that for positive constants α, β and positive number n, $\alpha^{\log_\beta n} = n^{\log_\beta \alpha}$*

1.3 *Show that*

1. $4^{\sqrt{n}} = \omega(3^{\frac{n}{3}})$
2. $2^{\frac{n}{3}} = o(3^{\sqrt{n}})$
3. $\log(n^k) = O(\log^k n)$, *for any integer constant* $k \geq 1$
4. $n^{\log_2(4)} = \Theta(2^{2\log_2 n})$
5. $n^{100} = o(2^n)$
6. $(\log n)^a = o(n^b)$, *for any constants* $a, b > 0$.
7. $n^a = O(n^b)$ *if* $1 < a \leq b$

1.4 *Are the following true or false? Justify.*

1. *Given integer constants* a, b, *where* $a < b$. *Is* $a^{b^n} = O(b^{a^n})$? *(For example, Is* $3^{4^n} = O(4^{3^n})$*?) Note that* $3^{4^2} \neq 3^{4*2}$.
2. *Given integer constants* a, b, *where* $a < b$. *Is* $n^{\log_b n} = o(n^{\log_a n})$? *(For example, Is* $n^{\log_4 n} = o(n^{\log_2 n})$*?*
3. *If* $f \in \Theta(g)$ *then* $f(n) \geq g(n)$ *for all large values of* n*?*
4. *Suppose* $f(1) = 10^6$ *and* $g(1) = 10^{-6}$ *then* $g(n) \in O(f(n))$*?*

1.5 *Let* $p(n) = a_d n^d + a_{d-1} n^{d-1} + \cdots a_1 n + a_0$, *where* $a_d > 0$, *be a* d*-degree polynomial in* n. *Also* $a_0, \cdots, a_d$ *are positive constants. Let* k *be a positive integer. Show that*

1. *If* $k \geq d$, *then* $p(n) = O(n^k)$.
2. *If* $k \leq d$, *then* $p(n) = \Omega(n^k)$.
3. *If* $k = d$, *then* $p(n) = \Theta(n^k)$.

1.6 *Let* $p(n) = \sum_{i=0}^{d} a_i n^i$ *be a polynomial of degree* d *and assume that* $a_d > 0$. *Show that* $p(n) \in \Omega(n^k)$, *where* $k \leq d$ *is a constant, using the formal definition of* Ω*-notation. What are* c *and* n_0 *for this?*

1.7 *Prove or disprove:* $g(n) = \Omega(f(n))$ *if and only if* $f(n) = O(g(n))$.

1.8 *Let* $T(n) = \sum_{x=1}^{n} x^2$. *Show that* $T(n) \in O(n^4)$ *and* $T(n) = n^3/3 + O(n^2)$.

1.9 *Evaluate the following recurrences (You can assume that* $T(1) = 1$ *in each of them).*

1. $T(n) = 2T(n/2) + O(n)$.
2. $T(n) = 4T(n/4) + O(n)$.

3. $T(n) = T(3n/4) + O(n)$.
4. $T(n) = T(7n/8) + O(n)$.
5. $T(n) = T(n-1) + O(1)$.
6. $T(n) = T(n-2) + O(1)$.
7. $T(n) = \sqrt{n}T(\sqrt{n}) + n$.

1.10 *Solve the recurrence relation*

$$T(n) = T(xn) + T((1-x)n) + cn$$

in terms of x and n where x is a constant in the range $0 < x < 1$. Is the asymptotic complexity the same when $x = 0.5, 0.1$ and 0.001? What happens to the constant hidden in the $O()$ notation.

1.11 *For any value of x, $0 < x < 1$, show that $x - (1+x)\ln(1+x) + \frac{1}{3}x^2 \leq 0$.*

1.12 *Fibonacci numbers are defined recursively as follows:*
$F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for any integer $n > 1$.
Using induction show that $F_n \geq 2^{\frac{n}{2}}$ for any $n \geq 6$.

1.13 *Suppose you need to choose between the following algorithms that solve the same problem:*

1. *Algorithm A solves the problem by dividing it into 5 subproblems of half of the size, recursively solves each of them, and combines the solution in linear time.*
2. *Algorithm B solves the problem of size n by recursively solving two subproblems of size $n-2$ and then combining the solutions in constant time.*
3. *Algorithm C solves the problem of size n by dividing it into 9 subproblems of size $n/3$ each, recursively solving each of them, and then combining the solution in $O(n^2)$ time.*

What are the running times of each of these algorithms? Which one will you choose and why?

1.14 *V. Pan has discovered a way to multiply two $70 * 70$ matrices using only 143640 multiplications. Ignoring the additions, what will the asymptotic complexity of Pan's algorithm for multiplying two $n * n$ matrices? Is it better than Strassen's? Justify your answer.*

1.15 *This is from [5] and is based on Divide-and-Conquer Multiplication. (Do not use FFTs as such for this)*

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 4th edition, 2022

1. *Show how to multiply two polynomials of degree* 1, *namely* $ax + b$ *and* $cx + d$ *using only three multiplications. (Note that* $(a + b).(c + d)$ *is considered as one multiplication.)*

2. *Give a divide-and-conquer algorithm for multiplying two polynomials of degree n that runs in* $\Theta(n^{\log_2 3})$ *time. You may think of dividing the coefficients into a high half and a low half, or in terms of whether the index is even or odd.*

3. *Show that two n-bit integers can be multiplied in* $O(n^{\log_2 3})$ *steps, where each step operates on at most a constant number of* 1*-bit values.*

1.16 *Let x be a number. Let* $n = 2^k$ *for some positive integer k. Present an algorithm running in* $O(\log n)$ *time to compute:* $z = x^n \mod 10$.

1.17 *Suppose we have n real numbers, where no two are the same. We want to report the smallest i numbers in the sorted order, where* $i < n$. *Which algorithm you think is the best option (Justify your answer)?*

A. Sort the numbers and list the first i.

B. Build a priority queue and then call Extract-Min i times.

C. Use the order statistics to find the i-th smallest number, partition the set according to this value, and then sort the i smallest numbers.

1.18 *Let A and B be two arrays, each consisting of n distinct elements in sorted order (in an increasing order). Report the median of the set* $A \cup B$ *in* $O(\log n)$ *time.*

1.19 *Given two binary strings* $a = a_0a_1 \ldots a_p$ *and* $b = b_0b_1 \ldots b_q$, *where each* a_i *and* b_j *are either o or 1. We say that* $a \leq b$ *if either of the following holds*

(1) there exists an integer j, $1 \leq j \leq \min(p, q)$, *such that* $a_i = b_i$ *for all* $i = 0, 1, ..., j - 1$ *and* $a_j < b_j$.

(2) $p < q$ *and* $a_i = b_i$ *for all* $i = 0, 1, 2, , , , p$.

Let $A \subseteq \Sigma^*$ *be a set of distinct binary strings whose lengths sums up to n. Present an algorithm that can sort the binary strings in* $O(n)$ *time. (All the strings are not of the same length!)*

1.20 *We want to sort* $n > 0$ *distinct real numbers in ascending order. Assume that these numbers are given in an array A of size n. We are also given a function* double-partition(i,j) *which takes as input two indices* $1 \leq i < j \leq n$ *of A, where* $j - i \geq 2$, *and returns two elements* $x, y \in \{A[i], A[i + 1], \ldots, A[j]\}$ *that satisfy the following:*

1. $x < y$

2. *The number of elements in* $\{A[i], A[i + 1], \ldots, A[j]\}$ *that are smaller than x are at most* $\lceil \frac{j-i}{3} \rceil$.

3. *The number of elements in* $\{A[i], A[i+1], \ldots, A[j]\}$ *that are larger than* y *are at most* $\lceil \frac{j-i}{3} \rceil$.

4. *The number of elements in* $\{A[i], A[i+1], \ldots, A[j]\}$ *that are larger than* x *but smaller than* y *are at most* $\lceil \frac{j-i}{3} \rceil$.

5. *It takes* $O(j-i)$ *time to compute* x *and* y.

Design an algorithm, running in $O(n \log n)$ *time, to sort any set of* n *distinct real numbers using the function* double-partition.

1.21 *You are given an array* A *consisting of* n *positive integers, where each element is* $\leq 10n$. *Devise an algorithm, running in* $O(n)$ *time, to sort* A *in ascending order. Justify your answer.*

1.22 *Recall that there is a lower bound for sorting. Answer the following questions*

1. *What does it mean to have a lower bound for a problem?*

2. *State clearly what is the lower bound claim for sorting a set of* n *(real)-numbers.*

3. *Why this claim does not apply to the previous problem?*

1.23 *You are given an array* A *consisting of* n *real numbers. Describe and analyze an algorithm, running in* $O(n)$ *time, that rearranges the elements of* A *so that* A *forms a binary heap. Once* A *is transformed into a Binary Heap, show how you can report the elements in* A *in sorted (ascending) order. How much time it takes to report all the elements of* A *in the sorted order? Justify your answer.*

1.24 *You are given a set of* n *real numbers which you are asked to insert incrementally in an initially empty binary search tree. Note that the time to insert an element in a binary search tree of size* x *is* $O(\log x)$. *What is the total running time of inserting all the* n *elements in the tree. Justify your answer. Assume that you have formed the binary search tree on* n *elements, show how you can report the elements in a sorted order in* $O(n)$ *time.*

1.25 *Reflecting on the answers to the previous two questions, is the construction of a binary heap in* $O(n)$ *time or reporting the elements in sorted order from a binary search tree in* $O(n)$ *time is in contradiction to the lower bound for sorting? Justify your answer.*

1.26 *Discuss a couple of scenarios where you will be using a binary heap instead of a binary search tree. Justify your answer.*

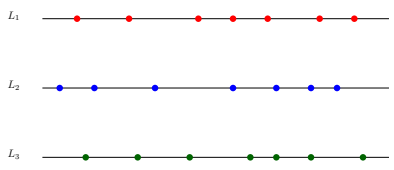
1.27 *Let* S *be a set of* n *distinct real numbers. Devise an algorithm, running in* $O(n + k \log k)$ *time, to report the* k *smallest elements of* S *in sorted order, where* $k \in \{1, \ldots, n\}$.

1.28 *Let S be a set of n-distinct real numbers and let $k \leq n$ be a positive integer (k may not be a constant). Design an algorithm, running in $O(n)$ time, that determines the k numbers in S that are closest to the median of S. For example, for the set $\{21, 70, 3, 1, 6, 7, 11, 2, 9, 8, 17, 13, 25\}$, median is 9, and the $k = 4$ closest numbers to 9 are $8, 7, 11$, and 6.*

1.29 *Let A and B be two sorted arrays, each array consists of n real numbers in ascending order. Give an algorithm, running in $O(\log n)$ time, to compute the median of the elements formed by the union of the elements in both the arrays. (You may assume that the union consists of $2n$ distinct real numbers.) Hint: Assume that the median element is from A, and assume that it is at index i. Then for $x = A[i]$ to be the median element, you can say something about how many elements in B need to be smaller than x and that can be checked in $O(1)$ time. The problem to solve here is how fast you can search for x in A.*

1.30 *Let A be an array consisting of n distinct real numbers. You need to find a real number x (x may not be an element of A) such that $\sum_{i=1}^{n} |x - A[i]|$ is minimized. Your algorithm should run in $O(n)$ time.*

1.31 *Assume that you have a set of three parallel and distinct lines in plane. Assume that Line 1 contains a set R of n red points, Line 2 contains a set B of n blue points, and Line 3 contains a set G of n green points, where n is a positive integer.*



You need to design an algorithm that determines if there is a triplet of points (r, b, g) such that they lie on a line segment, where $r \in R$, $b \in B$ and $g \in G$. (Observe that it is straightforward to design an algorithm running in $O(n^3)$ time by considering each choice for points r, b, and g, and in $O(1)$ time testing whether they all lie on a segment.) To get bonus points, design an algorithm running in $O(n^2)$ time.

1.32 *Suppose you want to have an extra operation called HEAP-DELETE (A, i), which deletes the item at node i from the heap A. Give an algorithm that can implement this in $O(\log n)$ time for an n-element max-heap. Show why it runs in the stated complexity.*

1.33 *Suppose you are given a sequence S of n integers in the range $[0, \cdots, n^3 - 1]$. Describe a simple method of sorting them in $O(n)$ time.*

1.34 *Describe an algorithm that given n integers in the range $0, \ldots, k$, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a..b]$ in $O(1)$ time, where $0 \leq a \leq b \leq k$. Your algorithm should use $\Theta(n + k)$ preprocessing time.*

1.35 *Suppose you want to merge two sorted lists A and B each of size n. Show that if two elements a_i and b_j are consecutive in the final sorted order,*

where $a_i \in A$ and $b_j \in B$, then during the algorithm there is a comparison made between them.

1.36 *The recurrence $T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + n$ shows up in the median of medians algorithm of [6]. Assume that for small values of n, $T(n) = O(1)$. Using the substitution method, show that the recurrence $T(n)$ evaluates to $O(n)$.*

[6] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973

1.37 *Consider that there is an array P containing n-photographs. It is possible that all photos are not distinct. We need to find out if there is a photograph that occurs at least $\lfloor \frac{n}{2} \rfloor + 1$ times in P (if such an element exists, then call it the* majority element*). Note that we can compare two photo's and decide in $O(1)$ time whether they are identical (i.e., the test $P[i] \stackrel{?}{=} P[j]$), but we don't have any mechanism to decide whether $P[i] < P[j]$ (i.e., the elements of P cannot be sorted).*

a) First devise an $O(n \log n)$ time algorithm for this problem using divide-and-conquer (e.g. split the array into two equal parts, and will knowing the majority element of both the halves will help in finding the majority element of the whole array?).

b) Devise an $O(n)$ algorithm for this problem. Think of forming $n/2$-pairs, each pair consisting of 2 elements, and then decide which elements to keep.

1.38 *Let G be a $n \times n$ integer grid graph, where n is a positive integer. Each vertex of G is specified by its coordinate (i, j), where $1 \leq i \leq n$ and $1 \leq j \leq n$. The neighbors of a vertex (i, j) are the vertices $(i-1, j)$, $(i+1, j)$, $(i, j-1)$, and $(i, j+1)$ (if they exist). Each vertex holds a unique number. We say a vertex (i, j) is dominant, if the number stored at that vertex is larger than the numbers stored at all of its neighbors. Show that G has a dominant vertex. Design an algorithm running in $O(n)$ time to report a dominant vertex in G. Note that G has n^2 vertices. For an illustration, see Figure 1.5.*

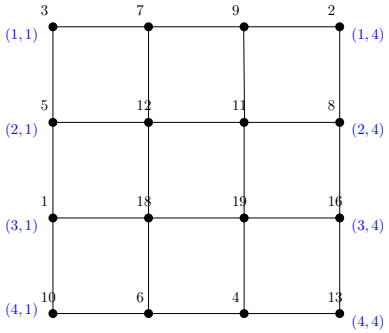


Figure 1.5: 4×4 grid graph G. Vertex coordinates are in blue color. Neighbors of vertex $(1, 1)$ are $(1, 2)$ and $(2, 1)$. Vertex at coordinate $(3, 3)$ with value 19 is a dominant vertex of G.

1.39 *Let S be a set of n points on a real line. How fast can you find a pair of points that have the smallest distance? What if the points are in 2-dimensional real plane?*

1.40 *Let A be an array of size n, where each $A[i]$ is an integer (positive or negative) and $1 \leq i \leq n$. For $1 \leq i \leq j \leq n$, define $\Delta(i, j) = \sum_{k=i}^{j} A[k]$. Find a maximum subarray of A, i.e., find a pair of indices α and β, where $1 \leq \alpha \leq \beta \leq n$, such that $\Delta(\alpha, \beta) \geq \Delta(i, j)$, for all possible choices of $1 \leq i \leq j \leq n$. Devise first a naive algorithm running in $O(n^3)$ time by considering all possible choices of i and j. Devise an $O(n \log n)$ time divide-and-conquer algorithm. Can you come up with an algorithm running in $O(n)$ time? See Exercise 4.1-5 of [7].*

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 4th edition, 2022

1.41 *Let A be an array consisting of n distinct integers. Show that we can compute the maximum and the minimum element of A by performing at most $3n/2$ comparisons.*

1.42 *Let A be an array consisting of n distinct integers. Show that we can compute the maximum and the second maximum element of A by performing at most $n + \log n$ comparisons.*

1.43 *(Problem 9.1-2 of [8]) Let A be an array consisting of n distinct integers. Show that we need to perform $\lceil \frac{3n}{2} \rceil - 2$ comparisons in the worst case to find simultaneously the maximum and the minimum element of A.*

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 4th edition, 2022

1.44 *Let A be an array consisting of n distinct real numbers. Assume that you can find the median of A in $O(n)$ time. How can use the median finding algorithm to find the k-th largest element for any given value of k, where $1 \leq k \leq n$.*

1.45 *We say two $n \times n$ matrices A and B are equal, denoted by $A = B$, if $A_{ij} = B_{ij}$ for all $1 \leq i, j \leq n$. We say a matrix $A = 0$ if $A_{ij} = 0$ for all $1 \leq i, j \leq n$. We say $A \neq 0$ if there exists some i, j for which $A_{ij} \neq 0$.*

Suppose you are given three $n \times n$ matrices X, Y, and Z. You need to determine $Z \stackrel{?}{=} XY$. Answer the following

1. *Can we determine $Z \stackrel{?}{=} XY$ in time proportional to multiplying two $n \times n$ matrices?*
2. *Let A be a matrix of dimensions $n \times n$. Let $v = (v_1, \ldots, v_n)$ be a random Boolean vector of length n, i.e., each of its coordinate v_i is set either to 0 or 1 independently and with equal probability. Consider Av, i.e. the product of the matrix A with the vector v and it results in a vector of length n. Show that for $A \neq 0$, $Pr(Av = 0) \leq \frac{1}{2}$.*
   Hint: Since $A \neq 0$, there is an entry, say $A_{ij} \neq 0$. Since $Av = 0$, we have that $\sum_{k=1}^{n} A_{ik} v_k = 0$. We can express $v_j = -\frac{\sum_{k=1}^{j-1} A_{ik} v_k + \sum_{k=j+1}^{n} A_{ik} v_k}{A_{ij}}$. Since only a specific value of v_j satisfies this equation, argue that $Pr(Av = 0) \leq \frac{1}{2}$.
3. *Assume $XY \neq Z$. Show that for any random Boolean vector v of length n, $Pr(XYv = Zv) \leq \frac{1}{2}$.*
4. *Show that the product XYv can be computed in $O(n^2)$ time.*
5. *Show that we can determine $XYv \stackrel{?}{=} Zv$ in $O(n^2)$ time.*
6. *Conclude that we can design a randomized test for checking $XY \stackrel{?}{=} Z$ that runs in $O(n^2)$ time. (Thus, it is more efficient to verify $Z \stackrel{?}{=} XY$ than multiplying matrices.)*

2
Probability for CS

We will focus on

1. Sample space and Events
2. Conditional Probability
3. Independent Events
4. Random Variables
5. Binomial Distribution
6. Cumulative Distribution
7. Expectation, Variance, and of Expectation
8. Law of Large Numbers
9. Normal Distribution, Moment generating functions, and Central Limit Theorem
10. Chernoff Bounds

Keywords: Probability, Bayes Theorem, Random Variables, Expectation, Linearity of Expectation. Variance, Indicator Random Variable, Markov Inequality, Chebyshev's Inequality, Binomial and Poisson Distributions, Central-Limit Theorem, Chernoff Bounds, Balls and Bins.

This material is adapted from the following sources:

1. Meyer's textbook [1]. (BTW, this was my textbook for the first undergraduate course in probability - way back in the Winter'83.)
2. Michiel Smid's online textbook [2]
3. Blitzstein and Hwang's textbook [3].

[1] P.L. Meyer. *Introductory probability and statistical applications*. Addison-Wesley, Boston, MA, USA, 1970

[2] Michiel Smid. Carleton University, Ottawa, Canada, 2014

[3] J.K. Blitzstein and J. Hwang. *Introduction to Probability*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press, 2014

4. Video lectures of Joe Blitzstein.

The selection of topics listed here are based on what is required to understand material presented in these notes. In no way, this is meant to be a coverage of this ever expanding field.

2.1 *Basics*

Sample Space and Events: With each probabilistic experiment, the *sample space* is the set of all possible outcomes of that experiment. For example, for rolling a dice the set of all possible outcomes are $\{1,2,3,4,5,6\}$. An *event* is also a set of possible outcomes, and is a subset of the sample space. For example, for rolling a dice and getting an even number, the events are $\{2,4,6\}$. If the sample space consists of n elements, then the total number of all possible events are 2^n. Since events are sets, we can use associated operations on sets. For example, if A and B are events for a sample space S, then we can define $A \cup B$, $\bar{A}$, $A \cap B$, with the usual meaning. Two events A and B are said to be *mutually exclusive* if they cannot occur simultaneously, i.e. $A \cap B = \emptyset$.

Probability: Let S be a sample space associated with an experiment. For each event $A \subseteq S$, associate a real number $0 \leq P(A) \leq 1$, called as the *probability* of A, and $P(A)$ satisfies following natural conditions

1. $P(S) = 1$,
2. If events A and B are mutually exclusive, $P(A \cup B) = P(A) + P(B)$ and $P(A) \cap P(B) = 0$.
3. $P(\bar{A}) = 1 - P(A)$, where $\bar{A} = S \setminus A$.
4. In general, for two events A and B, $P(A \cup B) = P(A) + P(B) - P(A \cap B)$. This is sometimes referred as the inclusion-exclusion principle.
5. If $A \subset B$, $P(A) \leq P(B)$.

Conditional Probability: Let A and B be two events associated with an experiment. We say $P(B|A)$ to be the *conditional probability* of occurrence of event B given that A has occurred. Intuitively, computation of $P(B)$ is done with respect to a reduced sample space. For example, let B be the event of getting 2 after rolling a dice. Then $P(B) = 1/6$. Let A be the event of getting an even number after rolling a dice. Then $P(A) = 1/2$. What about $P(B|A)$? It is $1/3$, since

the reduced sample space is $\{2,4,6\}$, and the probability of drawing a 2 is $1/3$ from this space. Note that $P(B|A) = \frac{P(A \cap B)}{P(A)}$ or equivalently $P(A \cap B) = P(B|A)P(A)$. Similarly we can define $P(A|B) = \frac{P(A \cap B)}{P(B)}$, or equivalently $P(A \cap B) = P(A|B)P(B)$. Note that if two events A and B are mutually-exclusive than $P(A|B) = P(B|A) = 0$. If $A \subset B$, than $P(B|A) = 1$. (For example, if B is the event of obtaining an even number and A is the event of getting a 2 on rolling a dice, than if A has occurred, than for sure B has occurred.) Now let us consider a classical theorem on conditional probabilities, called the *Bayes Theorem*.

Let $B_1, B_2, \cdots, B_k$ represent a partition of sample space S, see Figure 2.1. Let A be an event in S. Then, $P(A) = \sum_{i=1}^{k} P(A/B_i)P(B_i)$ and

$$P(B_i|A) = \frac{P(A \cap B_i)}{P(A)} = \frac{P(A|B_i)P(B_i)}{\sum_{i=1}^{k} P(A|B_i)P(B_i)}.$$

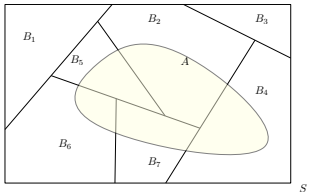


Figure 2.1: Partition of Sample Space S.

Here is a classical example from Meyer [116] showing an application of this theorem.

Example 2.1.1 *An item is produced by three factories - F1, F2, and F3. F2 and F3 produce the same number of items. F1 produces twice many items as F2 and F3. 2% of items produced by F1 and F2 are defective, and 4% of items produced by F3 are defective. All of these items are indistinguishable in terms of which factory they come from. First let us understand the partitioning by the following.*
a) What is the probability that an item is defective?

Here A = {item is defective}. B_1 = {item came from F1} and likewise define B_2 and B_3. Note that $P(B_1) = 1/2$ and $P(B_2) = P(B_3) = 1/4$. $P(A|B_1) = P(A|B_2) = 0.02$ and $P(A|B_3) = 0.04$. Then

$$P(A) = P(A|B_1)P(B_1) + P(A|B_2)P(B_2) + P(A|B_3)P(B_3) = 0.025.$$

Hence there is a 2.5% chance that an item is defective.
b) Suppose that an item is defective. What is the chance that it is made in F1?

Let us apply Bayes Theorem.

$$P(B_1|A) = \frac{P(A|B_1)P(B_1)}{\sum_{i=1}^{k} P(A|B_i)P(B_i)} = \frac{0.02 * 1/2}{0.02 * 1/2 + 0.02 * 1/4 + 0.04 * 1/4} = 0.40.$$

Hence, there is a 40% chance.

Independent Events: We say that the two events A and B are *independent* if $P(A \cap B) = P(A)P(B)$. Intuitively, this implies that the

occurrence or nonoccurence of A has no effect on the occurrence or non-occurrence of B. An example from Meyer [116] - its insightful:

Example 2.1.2 *We are rolling two fair die and have three types of events.*
A =*{The first die shows an even number}*
B =*{The second die shows an odd number}*
C =*{Both show an odd or both show an even number}*
Observe that $P(A) = P(B) = P(C) = 1/2$. *Note that* $P(A \cap B) = 1/4 = P(A)P(B)$. $P(A \cap C) = 1/4 = P(A)P(C)$. $P(B \cap C) = 1/4 = P(B)P(C)$. *But what about* $P(A \cap B \cap C)$*? Note that* $P(A \cap B \cap C) = 0 \neq P(A)P(B)P(C)$.

Hence the three events A, B and C are said to be *mutually independent* if and only if they are pairwise independent as well as $P(A \cap B \cap C) = P(A)P(B)P(C)$. In general n events are said to be mutually independent if and only if all combinations of them are mutually independent.

Random Variable: A function X assigning every element of a sample space of an experiment to a real number is called a *random variable* (r.v.), i.e. $X : S \rightarrow \Re$. For example, X may count the number of 1's in a random binary bit string of length n. A r.v. is called *discrete* if the number of possible values it can take is either finite or countably infinite. For each of those values, associate a real value between 0 and 1, i.e. its probability $P(x_i) = P(X = x_i)$. Each $P(x_i) \geq 0$ and $\sum_i P(x_i) = 1$. The function P is called the *probability mass (or density) function* and the collection of pairs $(x_i, P(x_i))$ are called the *probability distribution* of X. For example, if X is a r.v. describing the number of heads in three tosses of a coin, than the range of X is $\{0, 1, 2, 3\}$ and $P(0) = P(3) = 1/8$, $P(1) = P(2) = 3/8$. Consider the following example.

Example 2.1.3 *Suppose you want to generate several strings, each string consists of several a's followed by a single b. The random character generator spits out an a with probability* $2/3$ *and b with probability* $1/3$. *The string is generated by repeatedly invoking the 'sputter' till it outputs the first b. Let* X *be the random variable denoting the number of times the sputter is invoked. Note that* X *can take values* $1, 2, 3, ...$ *Observe that* $P(X = 1) = 1/3$, $P(X = 2) = 2/3 * 1/3$, *and* $P(X = i) = (2/3)^{i-1}1/3$. *Note that* $\sum_{i=1}^{\infty} P(X = i) = 1/3[\sum_{i=0}^{\infty}(2/3)^i] = 1$.

A r.v. X is said to be *continuous* if there exists a function f, called the probability mass function, satisfying (a) $f(x) \geq 0$, for all x. (b) $\int_{-\infty}^{+\infty} f(x)dx = 1$ and (c) For all $-\infty < a < b < +\infty$, $P(a < x < b) = \int_a^b f(x)dx$, i.e. the area of the curve under $f(x)$ between $x = a$

and $x = b$. For example, let X be a continuous r.v. with probability mass function given by $f(x) = 2x$, for $0 \leq x \leq 1$, and is 0 elsewhere. Observe that $f(x)$ satisfies the three requirements as (a) $f(x) \geq 0$ for all x, (b) $\int_{-\infty}^{+\infty} f(x)dx = \int_0^1 2xdx = 1$, and (c) $-\infty < a < b < +\infty$, $P(a < x < b) = \int_a^b f(x)dx = \int_a^b 2xdx = b^2 - a^2$. For example, if $a = 0$ and $b = 1/2$, then $P(0 \leq x \leq 1/2) = 1/4$.

Binomial Distribution: The *Binomial distribution* is defined as follows. Consider an experiment, where A is an event. Let $P(A) = p$ and $P(\bar{A}) = q = 1 - p$. Let us repeat the experiment n times and define a random variable X indicating the number of times A occurs. It is assumed that each experiment is independent of others. What is the probability that $X = k$, for some $1 \leq k \leq n$? It is easy to see that $P(X = k) = \binom{n}{k} p^k q^{n-k}$. X is called a *binomial random variable* with parameters p and n. Individual experiments (trials) are called *Bernoulli trials*. Also observe that $\sum_{k=0}^{n} P(X = k) = \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} = (p + q)^n = 1^n = 1$

CDF - Cumulative Distribution Function: In Statistics we typically perform random experiments. They can be repeated any number of times with a known set of outcomes. We associate random variables and probability distributions to such experiments. For each random variable, this association is achieved by a function called the *cumulative distribution function*.

For example, consider the following simple experiment. We flip a fair coin three times, and let the r.v. X be the number of 'Heads'. Note that the sample space

$$S = \{HHH, THH, HTH, HHT, TTH, THT, HTT, TTT\}.$$

The random variable X maps $S \to \{0, 1, 2, 3\}$. Note that $X(THH) = 2$ and $X(HTT) = 1$. We have already seen the probability mass distribution function P. Now define the cumulative distribution function F for X as $F(x) = P(X \leq x)$, i.e. the probability of X being less than or equal to x. Observe that $F(x) = 0$ if $x < 0$; $F(x) = P(0) = 1/8$ if $x \leq 0$; $F(x) = P(0) + P(1) = 1/2$ if $x \leq 1$; $F(x) = P(0) + P(1) + P(2) = 7/8$ if $x \leq 2$; and $F(x) = 1$ if $x \leq 3$.

The CDF of a continuous r.v. X with probability mass function f is given by $F(x) = P(X \leq x) = \int_{-\infty}^{x} f(s)ds$.

Expected Value: The expected value of a discrete r.v. X is defined as $E[X] = \sum_{s \in S} X(s)P(\{s\})$. Alternatively, we can express this sum by grouping all the elements of S that have the same value $X = x_i$,

and obtain $E[X] = \sum_{s \in S} X(s)P(\{s\}) = \sum_i x_i P(X = x_i)$. If this series converges, $E[X]$ is also called the *mean value* of X.

The expected value of a continuous r.v. X is defined analogously, i.e., $E[X] = \int_{-\infty}^{+\infty} xf(x)dx$. For example, consider X to be uniformly distributed over the interval $[a, b]$. We know that the probability distribution function

$$f(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

Hence $E[X] = \int_a^b \frac{x}{b-a} dx = (a+b)/2$

Next we show that for a Binomial distribution X with parameters n and p, $E[X] = np$. Note that $P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$. Thus,

$$\begin{aligned} E[X] &= \sum_{k=0}^{n} k \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k} \\ &= \sum_{k=1}^{n} \frac{n!}{(k-1)!(n-k)!} p^k (1-p)^{n-k}, \end{aligned}$$

As, for $k = 0$, $k\frac{n!}{k!(n-k)!} p^k (1-p)^{n-k} = 0$. We will perform change of variables and set $s = k - 1$. Now we have

$$\begin{aligned} E[X] &= \sum_{s=0}^{n-1} n \frac{(n-1)!}{s!(n-1-s)!} p^{s+1} (1-p)^{n-1-s} \\ &= np \sum_{s=0}^{n-1} \frac{(n-1)!}{s!(n-1-s)!} p^s (1-p)^{n-1-s} \\ &= np(p + (1-p))^{n-1} \\ &= np \end{aligned}$$

Independent Random Variables: Two random variables defined over the same sample space are said to be independent if the outcome of one doesn't depend on the outcome of the other. In case of discrete r.v., by independence we mean, $P(X = x_i, Y = y_j) = P(X = x_i) \cdot P(Y = y_j)$ for all possible values of i and j. Alternatively, we can say that $P(X = x_i | Y = y_j) = P(X = x_i)$ and $P(Y = y_i | X = x_j) = P(Y = y_i)$ for all values of i and j. As an example, let us toss a fair coin four times. Define r.v. X to be the number of heads obtained in the first two tosses and r.v. Y to be the number of heads obtained in the last two tosses. Observe that $P(X = 0, Y = 0) = P(X = 0)P(Y = 0) = 1/16$, $P(X = 0, Y = 1) = P(X = 0)P(Y = 1) = 1/8$, $P(X = 1, Y = 1) = P(X = 1)P(Y = 1) = 1/4$, $P(X = 2, Y = 1) = P(X = 2)P(Y = 1) = 1/8$, etc. In each of the possibilities we observe that $P(X = x_i, Y = y_j) = P(X = x_i)P(Y = y_j)$ and thus X and Y are independent random variables.

Linearity of Expectation: Let X and Y be two random variables mapping elements of a sample space S to real numbers. Assume that $E[X]$ and $E[Y]$ are finite. Linearity of Expectation says that $E[aX + bY] = aE[X] + bE[Y]$ for constants a and b. (Note that X and Y need not be independent.) The proof is fairly straightforward. Observe that, by definition,

$$\begin{aligned} E[aX+bY] &= \sum_{\omega \in S} (a \cdot X + b \cdot Y)[\omega] \cdot P(\omega) \\ &= \sum_{\omega \in S} (a \cdot X[\omega] + b \cdot Y[\omega]) \cdot P(\omega) \\ &= \sum_{\omega \in S} (a \cdot X[\omega] \cdot P(\omega) + b \cdot Y[\omega] \cdot P(\omega)) \\ &= a \sum_{\omega \in S} X[\omega] \cdot P(\omega) + b \sum_{\omega \in S} Y[\omega] \cdot P(\omega) \\ &= aE[X] + bE[Y] \end{aligned}$$

This easily generalizes to the linear combination of n random variables, i.e. expectation of the linear combination of n variables is same as the linear combination of the expectation of these variables. Given this, it is trivial to compute the expectation of a Binomial r.v. X. Note that $X = X_1 + X_2 + \cdots + X_n$, where each X_i is a Bernoulli indicator random variable, with success probability p. The expected value of each of the indicator variable is $E[X_i] = 1.p + 0.(1-p) = p$. Thus $E[X] = E[X_1 + \cdots X_n] = E[X_1] + \cdots + E[X_n] = np$.

Variance: The *variance* of a r.v. X is defined to be the expected value of the r.v. $(X - E[X])^2$. We will write this as $V[X] = E[X - E[X]]^2$. One of the exercises asks for verifying that $V[X] = E[X - E[X]]^2 = E[X^2] - (E[X])^2$. Let us calculate the variance of the r.v. that is uniformly distributed over the interval $[a, b]$. We know that $E[X] = (a+b)/2$. Note that $E[X^2] = \int_a^b \frac{x^2}{b-a} dx = \frac{b^3-a^3}{3(b-a)}$. Since $V[X] = E[X^2] - [E[X]]^2$, we obtain $V[X] = \frac{b^3-a^3}{3(b-a)} - (\frac{a+b}{2})^2 = \frac{(b-a)^2}{12}$.

Let X and Y be two independent r.v. defined over the same sample space. Now we can express the variance of their sum as

$$\begin{aligned} V[X+Y] &= E[(X+Y)^2] - (E[X+Y])^2 \\ &= E[X^2 + Y^2 + 2XY] - (E[X] + E[Y])^2 \\ &= E[X^2] + E[Y^2] + E[2XY] - (E[X]^2 + E[Y]^2 + 2E[X]E[Y]) \\ &= E[X^2] - E[X]^2 + E[Y^2] - E[Y]^2 \\ &= V[X] + V[Y] \end{aligned}$$

Note that due to independence $E[XY] = E[X]E[Y]$ (see Exercises). Next, we compute the variance$V[X]$ of a binomial r.v. X with parame-

ters n and p. Note that r.v. $X = X_1 + X_2 + \cdots + X_n$, where each X_i's are identical independent indicator r.v. Thus, $V[X] = nV[X_i]$. Let us now evaluate $V[X_i]$. Note that

$$\begin{aligned} V[X_i] &= E[X_i^2] - E[X_i]^2 \\ &= \{1^2.p + 0^2.(1-p)\} - p^2 \\ &= p - p^2 \end{aligned}$$

Thus, $V[X] = nV[X_i] = n(p - p^2) = np(1-p)$.

2.2 *Chebyshev's Inequality and Law of Large Numbers*

Now let us look at a famous inequality due to Chebyshev.

Theorem 2.2.1 *Let X be a random variable and let c be a real number. If $E[X-c]^2$ is finite and $\epsilon > 0$, then*

$$P(|X - c| \geq \epsilon) \leq \frac{1}{\epsilon^2} E[X-c]^2.$$

Proof. Let X be a continuous r.v. Let $R = \{x : |x - c| \geq \epsilon\}$. Note that $P(|X-c| \geq \epsilon) = \int_R f(x)dx$. Observe that $|x-c| \geq \epsilon$ implies $\frac{(x-c)^2}{\epsilon^2} \geq 1$. Thus we have

$$\begin{aligned} P(|X-c| \geq \epsilon) &= \int_R 1 \cdot f(x)dx \\ &\leq \int_R \frac{(x-c)^2}{\epsilon^2} f(x)dx \\ &\leq \int_{-\infty}^{+\infty} \frac{(x-c)^2}{\epsilon^2} f(x)dx \\ &= \frac{1}{\epsilon^2} E[X-c]^2 \end{aligned}$$

∎

For an application of this inequality consider the following. Suppose we repeat an experiment multiple times. Then the relative frequency of the occurrence of an event should converge to its actual probability. For example, a factory is producing items. Suppose we don't know what is the failure probability. One way to estimate this probability is to take a large sample and see what percentage are faulty. This percentage will be a good estimate of the failure probability. Of course this should be taken with a grain of salt. This really depends upon what kind of sample is chosen. Essentially what we are heading towards is that if the elements in the sample are chosen randomly, than the percentage of faulty items will be a true indicator of failure probability.

Consider a particular event A in an experiment. This experiment is repeated n times, and each run is independent of other runs. Let n_A be the number of times A occurs in the runs. Let $f_A = n_A/n$. Let $P(A) = p$. We show that for any positive constant $\epsilon > 0$,

$$P(|f_A - p| \geq \epsilon) \leq \frac{p(1-p)}{n\epsilon^2},$$

or equivalently,

$$P(|f_A - p| < \epsilon) \geq 1 - \frac{p(1-p)}{n\epsilon^2}.$$

Observe that n_A is a binomially distributed random variable with expected value $E[n_A] = np$ and variance $V[n_A] = np(1-p)$. Since $f_A = n_A/n$, we have

$$\begin{aligned} E[f_A] &= E[n_A/n] \\ &= \frac{1}{n}E[n_A] \\ &= np/n \\ &= p \end{aligned} \tag{2.1}$$

$$\begin{aligned} V[f_A] &= V[n_A/n] \\ &= \frac{1}{n^2}V[n_A] \\ &= \frac{np(1-p)}{n^2} \\ &= p(1-p)/n. \end{aligned} \tag{2.2}$$

Recall Chebyshev's inequality (Theorem 2.2.1 and also see Exercise 2.24) which states that

$$\begin{aligned} P(|X - E[X]| < \epsilon) &= 1 - P(|X - E[X]| \geq \epsilon) \\ &\geq 1 - \frac{1}{\epsilon^2}E[(X - E[X])^2] \\ &= 1 - \frac{V[X]}{\epsilon^2} \end{aligned} \tag{2.3}$$

Substituting $X = f_A$, $E[X] = p$, we obtain

$$P(|f_A - p| < \epsilon) \geq 1 - \frac{V[f_A]}{\epsilon^2}.$$

Set $V[f_A] = p(1-p)/n$, and we obtain

$$P(|f_A - p| < \epsilon) \geq 1 - \frac{p(1-p)}{n\epsilon^2}.$$

Note that

$$\lim_{n\to\infty} P(|f_A - p| < \epsilon) = 1.$$

This is essentially the meaning of probability of an event A, i.e. the relative frequency converges to $P(A)$ when an experiment is repeated for a large number of times. This is what is called as the *Law of Large Numbers*. To get some more intuition, think of what could have happened if this wasn't true - i.e., no matter how many times you repeat the experiment, the frequency doesn't converge to $P(A)$. What will be the state of various fields such as Physics, Nature, Evolution, ...?

Here is a different type of question. How many times should we repeat the experiment, so that the relative frequency f_A differs from $P(A)$ by at most 0.01 with probability at least 0.9?

We need to choose n so that for $\epsilon = 0.01$, $1 - \frac{p(1-p)}{n\epsilon^2} = 0.9$. This implies that $n = \frac{p(1-p)}{0.1\epsilon^2}$. For example if $p = 1/2$, than $n = 25000$. What this means is that if we toss a coin 25000 times, than we are 90% sure that the relative frequency of getting a head is within 0.01 of the theoretical probability.

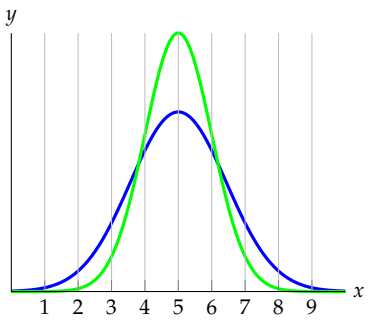


Figure 2.2: Illustration of $\mathcal{N}(5, 1.44)$ and $\mathcal{N}(5, 1)$ in blue and green, respectively

2.3 *Normal Distribution, mgf, and the Central Limit Theorem*

Next, let us discuss Normal distributions, as we will need them in later chapters. Random variable X has a *normal distribution* if its probability density function is of the form

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, -\infty < x < \infty.$$

Usually, it is denoted by $\mathcal{N}(\mu, \sigma^2)$, where $-\infty < \mu < +\infty$ is the mean (expected value) and $\sigma > 0$ is the standard deviation, i.e. positive square-root of the variance. It is also referred to as Gaussian or bell-shaped distribution. See Figure 2.2 for an illustration. It is a valid distribution, as $f(x) \geq 0$ for all values of x and $\int_{-\infty}^{+\infty} f(x)dx = 1$ (see Exercises). The function f is symmetric around $x = \mu$. If we trace the boundary of the function, it changes from being convex to concave, and these points of inflection are at $x = \mu \pm \sigma$. The distribution $\mathcal{N}(0, 1)$ is referred to as a *standardized normal distribution*.

Some quick facts about Normal distribution that are helpful includes

1. If X is $\mathcal{N}(\mu, \sigma^2)$ and $Y = aX + b$ then Y is $\mathcal{N}(a\mu + b, a^2\sigma^2)$.

2. If X has distribution $\mathcal{N}(\mu, \sigma^2)$ and if $Y = \frac{X-\mu}{\sigma}$, then Y has distribution $\mathcal{N}(0, 1)$.

3. If X has distribution $\mathcal{N}(0, 1)$, then $Pr(a \leq X \leq b) = \frac{1}{2\pi} \int_a^b e^{-x^2/2} dx$.

2.3.1 A special case of the central limit theorem

We state a theorem, which is a special case of the Central Limit Theorem (see Meyer [116]). The following theorem states that, for a large value of n, the arithmetic mean of n independent observations from the same random variable has a normal distribution.

Theorem 2.3.1 *Let $X_1, X_2, \cdots, X_n$ be n independent r.v. all of which have the same distribution. Let $\mu = E[X_i]$ and $\sigma^2 = V[X_i]$ be their common expectation and variance, respectively. Define a r.v. $S = \sum_{i=1}^{n} X_i$. Then $E[S] = n\mu$ and $V[S] = n\sigma^2$. Moreover, for large n, $T_n = \frac{S-n\mu}{\sqrt{n}\sigma}$ has essentially the distribution $\mathcal{N}(0,1)$.*

Before we discuss a proof sketch of this theorem, let us look at an example from Meyer's [116] to gain some intuition.

Example 2.3.2 *Consider a box containing three types of balls - 20 balls labelled with a zero, 30 with a one and 50 with a two. In this experiment, we draw a random ball, note its label, and then place it back. We will repeatedly pick the balls, and eventually report the average of their labels. We are interested in understanding the distribution of the averages when we repeat this experiment for a large number of rounds. The above theorem claims that the average behaves like a normal distribution. Let X_i be the label of the ball drawn in Round i of this experiment. We are interested in the random variable $M_i = \frac{1}{i}\sum_{k=1}^{i} X_k$, denoting the averages of the labels drawn up to and including the Round i. Note that each X_i takes values $0, 1,$ and 2, with probabilities $0.2, 0.3$ and 0.5, respectively. Observe that $M_1 = X_1$ and it takes values $0, 1$ or 2 with probability $0.2, 0.3$ and 0.5, respectively. Next let us look at M_2.*

$m = \frac{X_1+X_2}{2}$	0	1/2	1	3/2	2
$P(M_2 = m)$	0.04	0.12	0.29	0.30	0.25

$m = \frac{X_1+X_2+X_3}{3}$	0	1/3	2/3	1	4/3	5/3	2
$P(M_3 = m)$	0.008	0.036	0.114	0.207	0.285	0.225	0.125

Hopefully, this example convinces us that for large values of n, M_n converges to a Normal distribution.

We will sketch a proof of Theorem 2.3.1 using the *moment generating functions (mgf)* and their interesting properties. Let us first briefly explore mgf's.

2.3.2 *Moment Generating Functions*

For a random variable X, its moment generating function $M_X(t) = E[e^{tX}]$. Formally,

Definition 2.3.3 *Let X be a discrete r.v. taking values $\{x_1, x_2, x_3 \dots\}$ with probabilities $\{p_1, p_2, p_3 \dots\}$, respectively. The moment generating function (mgf) of X is*

$$M_X(t) = \sum_{i=1}^{\infty} e^{tx_i} p_i.$$

If X is a continuous r.v. with probability distribution function f, then the mgf is given by

$$M_X(t) = \int_{-\infty}^{+\infty} e^{tx} f(x) dx.$$

Consider following standard examples of mgf's.

Example 2.3.4 *Let r.v. X be uniformly distributed in the interval $[a, b]$ on real line. The mgf is given by*

$$M_X(t) = \int_a^b \frac{e^{tx}}{b-a} dx = \frac{1}{(b-a)t}(e^{bt} - e^{at}).$$

Example 2.3.5 *Let r.v. X be binomially distributed with parameters n and p. The mgf of X is*

$$M_X(t) = \sum_{k=0}^{n} e^{tk} \binom{n}{k} p^k (1-p)^{n-k} = [pe^t + (1-p)]^n.$$

Example 2.3.6 *Let r.v. X has the Normal distribution $\mathcal{N}(\mu, \sigma^2)$. Its mgf is given by*

$$M_X(t) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} e^{tx} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} dx = e^{t\mu + \sigma^2 t^2/2}.$$

By the Maclaurin series expansion, $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$. Therefore, $e^{tx} = 1 + tx + \frac{(tx)^2}{2!} + \frac{(tx)^3}{3!} + \cdots$. By definition,

$$M_X(t) = E[e^{tX}] = E[1 + tX + \frac{(tX)^2}{2!} + \frac{(tX)^3}{3!} + \cdots].$$

If we assume that the linearity of expectation will carry over for infinite sums, then

$$M_X(t) = E[e^{tX}] = 1 + tE[X] + \frac{t^2 E[X^2]}{2!} + \frac{t^3 E[X^3]}{3!} + \cdots.$$

Next we will take some higher order derivatives, and substitute the value of t as 0, to observe the following (note that this assumes that we can take derivatives of infinite series): $M'_X(0) = E[X]$, $M''_X(0) = E[X^2]$, and in general $M_X^{(n)}(0) = E[X^n]$. In fact this is the reason these functions are called the moment generating functions.

Example 2.3.7 *Recall that when X has binomial distribution with parameters n and p, its mgf is $M_X(t) = [pe^t + (1-p)]^n$. Observe that $M'_X(t) = n(pe^t + (1-p))^{n-1}pe^t$ and $M'_X(0) = np = E[X]$. Similarly, $M''_X[0] = E[X^2] = np[(n-1)p+1]$. We can also observe that $V[X] = M''_X(0) - [M'_X(0)]^2 = np(1-p)$.*

Example 2.3.8 *Let r.v. X has the Normal distribution $\mathcal{N}(\mu, \sigma^2)$. Its mgf is $M_X(t) = e^{t\mu+\sigma^2t^2/2}$. Now $M'_X(t) = (\mu + \sigma^2 t)e^{t\mu+\sigma^2t^2/2}$ and $M'_X(0) = E[X] = \mu$. Similarly, $M''_X(0) = E[X^2] = \mu^2 + \sigma^2$. Note that $Var[X] = M''(0) - M'(0)^2 = \sigma^2$.*

Observation 2.3.9 *Let r.v. X has mgf M_X. Let r.v. $Y = \alpha X + \beta$. where α and β are constants. Mgf of Y is given by*

$$M_Y(t) = E[e^{tY}] = E[e^{t(\alpha X+\beta)}] = e^{t\beta}M_X(t\alpha).$$

Observation 2.3.10 *Let X and Y be independent r.v. with mgf's M_X and M_Y, respectively. Let r.v. $Z = X + Y$. Then mgf of Z is given by*

$$M_Z(t) = E[e^{tZ}] = E[e^{t(X+Y)}]] = E[e^{tX}]E[e^{tY}] = M_X(t)M_Y(t).$$

Generalizing the above observation to n independent random variables, we obtain

Theorem 2.3.11 *Let $X_1, \ldots, X_n$ be n independent random variables with mgf's $M_{X_1}, \ldots, M_{X_n}$, respectively. Let r.v. $Z = X_1 + \cdots + X_n$. Then mgf of Z is given by $M_Z(t) = M_{X_1}(t) \cdots M_{X_n}(t)$.*

Observation 2.3.12 *Let X and Y be independent r.v. with Normal distributions $\mathcal{N}(\mu_1, \sigma_1^2)$ and $\mathcal{N}(\mu_2, \sigma_2^2)$, respectively. Let r.v. $Z = X + Y$. Then by Theorem 2.3.11 mgf of Z,*

$$M_Z(t) = M_X(t)M_Y(t) = e^{t\mu_1+\sigma_1^2t^2/2}e^{t\mu_2+\sigma_2^2t^2/2} = e^{t(\mu_1+\mu_2)+(\sigma_1^2+\sigma_2^2)t^2/2}.$$

But that corresponds to the mgf of the Normal distribution $\mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. Thus the sum of two independent Normal distributions is a Normal distribution.

From the above discussion we can conclude that

Theorem 2.3.13 *Let $X_1, \ldots, X_n$ be n independent random variables with Normal distributions $\mathcal{N}(\mu_i, \sigma_i^2)$, for $i = 1, \ldots, n$. Let r.v. $Z = \sum_{i=1}^n X_i$. Then Z has a Normal distribution $\mathcal{N}(\sum_{i=1}^n \mu_i, \sum_{i=1}^n \sigma_i^2)$. Thus the sum of independent Normal distributions is a Normal distribution.*

2.3.3 *Proof of Theorem 2.3.1*

Now we are in a position to sketch the proof of the special case of the Central Limit Theorem 2.3.1. Recall that we want to show the following:

Let $X_1, X_2, \cdots, X_n$ be n independent r.v. all of which have the same distribution (need not be Normal). Let $\mu = E[X_i]$ and $\sigma^2 = V[X_i]$ be their common expectation and variance, respectively. Let $S = \sum_{i=1}^{n} X_i$. Note that $E[S] = n\mu$ and $V[S] = n\sigma^2$. For large n, we need to show that $T_n = \frac{S-n\mu}{\sqrt{n}\sigma}$ has the distribution $\mathcal{N}(0,1)$. We will show that the mgf of T_n is same as that of the mgf of $\mathcal{N}(0,1)$.

Proof. The mgf of X_i is $M_{X_i}(t) = E[e^{tX_i}]$. Since X_i's are independent and from Observation 2.3.10, mgf of S is given by $M_S(t) = (M_{X_i}(t))^n$. Note that $T_n = \frac{S-n\mu}{\sqrt{n}\sigma} = \sum_{i=1}^{n}(\frac{X_i-\mu}{\sqrt{n}\sigma})$ is a linear function of S, and hence by Observation 2.3.9 mgf of T_n is given by

$$M_{T_n}(t) = \left(e^{-(\frac{\mu}{\sqrt{n}\sigma}t)} M_{X_i}(\tfrac{t}{\sigma\sqrt{n}})\right)^n \tag{2.4}$$

$$= e^{-(\frac{\mu\sqrt{n}}{\sigma}t)} \left[M_{X_i}(\tfrac{t}{\sigma\sqrt{n}})\right]^n \tag{2.5}$$

$$\ln M_{T_n}(t) = -\tfrac{\mu\sqrt{n}}{\sigma}t + n \ln \left[M_{X_i}(\tfrac{t}{\sigma\sqrt{n}})\right] \tag{2.6}$$

As we have seen earlier, by the Maclaurin series expansion

$$M_{X_i}(t) = E[e^{tX_i}] = 1 + tE[X_i] + \frac{t^2 E[{X_i}^2]}{2!} + \frac{t^3 E[{X_i}^3]}{3!} + \cdots$$

Since $E[X_i] = \mu$ and $E[{X_i}^2] = \mu^2 + \sigma^2$, we have

$$M_{X_i}(t) = 1 + \mu t + \frac{(\mu^2+\sigma^2)}{2} t^2 + R,$$

where R consists of all other remaining terms. Substituting this in Equation 2.6, we obtain

$$\ln M_{T_n}(t) = -\tfrac{\mu\sqrt{n}}{\sigma}t + n \ln \left[1 + \tfrac{\mu t}{\sqrt{n}\sigma} + \tfrac{(\mu^2+\sigma^2)}{2n\sigma^2} t^2 + R\right]. \tag{2.7}$$

Note that $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots$ for $|x| < 1$. For large values of n, $\frac{\mu t}{\sqrt{n}\sigma} + \frac{(\mu^2+\sigma^2)}{2n\sigma^2} t^2 + R < 1$. Hence,

$$\ln M_{T_n}(t) = -\frac{\mu\sqrt{n}}{\sigma}t + n\left[(\frac{\mu t}{\sqrt{n}\sigma} + \frac{(\mu^2+\sigma^2)}{2n\sigma^2} t^2 + R) - \frac{1}{2}(\frac{\mu t}{\sqrt{n}\sigma} + \frac{(\mu^2+\sigma^2)}{2n\sigma^2} t^2 + R)^2 + \cdots\right].$$

Omitting the algebraic manipulation, the above expression for $n \to \infty$ results in

$$\lim_{n\to\infty} \ln M_{T_n}(t) = t^2/2.$$

Equivalently,

$$\lim_{n\to\infty} M_{T_n}(t) = e^{t^2/2}.$$

Since the mgf of $\mathcal{N}(0,1)$ is $e^{t^2/2}$, hence the limiting distribution of T_n is $\mathcal{N}(0,1)$. ∎

2.4 More Distributions

We will briefly explore a few more standard discrete and continuous distributions.

Poisson Distribution: Let X be a discrete r.v. taking values $\{0, 1, 2, \dots\}$. X has Poisson distribution with parameter $\alpha > 0$ if $P(X = k) = \frac{e^{-\alpha}\alpha^k}{k!}$.

It turns out that $\sum_{k=0}^{\infty} P(X = k) = 1$, $E[X] = V[X] = \alpha$. One of the key properties of Poission distribution is that it approximates the Binomial distribution for large values of n and small values of p. Let X be a r.v. with Binomial distribution, where $P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$. As $n \to \infty$ and $np = \alpha$, we have that

$$\lim_{n\to\infty} P(X = k) = \frac{e^{-\alpha}\alpha^k}{k!}.$$

We know that $E[X] = np$ and $V[X] = np(1-p)$. For large values of n, small values of p, and $\alpha = np$, observe that $E[X] = V[X] = \alpha$.

Geometric Distribution: This distribution captures the scenario where an experiment is repeatedly executed, independent of previous executions, till a particular event occurs. Let A be the event we want and let us assume that p is the probability of its occurrence. Hence it doesn't occur with probability $1 - p$. Define the r.v. X that counts the number of times the experiment is repeated till A occurs. X takes values $\{1, 2, 3, \dots\}$. X is said to have geometric distribution where $P(X = k) = (1-p)^{k-1}p$. It turns out that $\sum_{k=0}^{\infty} P(X = k) = 1$, $E[X] = 1/p$ and $V[X] = (1-p)/p^2$. In particular, on the average, we need to execute the experiment $\lceil \frac{1}{p} \rceil$ times to see the event A.

Gamma Distribution: For any $p > 0$, the Gamma function is defined as

$$\Gamma(p) = \int_0^{\infty} x^{p-1} e^{-x} dx.$$

Let X be a continuous r.v. taking positive values. X has a Gamma distribution with parameters $r > 0$ and $\alpha > 0$, if for all $x > 0$, its probability density function is given by

$$f(x) = \frac{\alpha}{\Gamma(r)} (\alpha x)^{r-1} e^{-\alpha x}.$$

If $r = 1$, then $f(x) = \alpha e^{-\alpha x}$ and this is pdf of an *exponential distribution*. If $\alpha = 1/2$ and $r = n/2$, for some positive integer n, then we obtain the *Chi-Square Distribution*. Its pdf is given by $f(x) = \frac{1}{2^{n/2}\Gamma(n/2)} x^{n/2-1} e^{-x/2}$. Its expected value and variance is given by n and $2n$, respectively.

2.5 Chernoff Bounds

Suppose we toss a fair coin 1000 times. We expect about 500 tails. What is the probability that we will get over 750 tails? or over 900

tails? Chernoff bounds help us in determining probabilities of extreme events in a large collection of independent events. In this section we will establish these bounds. This section is derived from Hagerup and Rüb [4]. First let us establish Markov's inequality.

[4] Torben Hagerup and Christine Rüb. A guided tour of Chernoff bounds. *Inf. Process. Lett.*, 33(6):305–308, 1990

Theorem 2.5.1 *Let X be a non-negative discrete r.v. and $s > 0$ be a constant. Then $P(X \geq s) \leq E[X]/s$.*

Proof. Note that

$$\begin{aligned} E[X] &= \sum_{i=0}^{\infty} i.P(X = i) \\ &\geq \sum_{i=s}^{\infty} i.P(X = i) \\ &\geq s \sum_{i=s}^{\infty} P(X = i) \\ &= sP(X \geq s). \end{aligned} \tag{2.8}$$

Hence, $P(X \geq s) \leq E[X]/s$. ∎

Let $X_1, \cdots X_n$ be identical o-1 independent r.v's, such that $P(X_i = 1) = p_i$ and $P(X_i = 0) = 1 - p_i$. Define $X = \sum_{i=1}^{n} X_i$ and let $m = \sum_{i=1}^{n} p_i$. Note that

$$E[X] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} p_i = m.$$

Suppose we have a fair coin, which we toss n times. We define o-1 r.v's $X_1, \cdots, X_n$, where $X_i = 1$ if the i-th toss was a head otherwise $X_i = 0$. $P(X_i = 0) = P(X_i = 1) = 1/2$. Let $X = \sum_{i=1}^{n} X_i$. Now $E[X] = n/2 = m$. We are interested in estimating the probability of X deviating from $(1 \pm \epsilon)m$, for $0 < \epsilon < 1$. From Markov's inequality we obtain $P(X > (1+\epsilon)m) \leq 1/(1+\epsilon)$. We will show the following, which are collectively known as *Chernoff bounds* in the algorithms community.

$$P(X \geq (1+\epsilon)m) \leq \exp(-\epsilon^2 m/3)$$
$$P(X \leq (1-\epsilon)m) \leq \exp(-\epsilon^2 m/2).$$

Let $t > 0$. Let us first deal with proving $P(X \geq (1+\epsilon)m) \leq \exp(-\epsilon^2 m/3)$.

$$P(X \geq (1+\epsilon)m) = P(e^{tX} \geq e^{t(1+\epsilon)m}) \tag{2.9}$$
$$= e^{-t(1+\epsilon)m} e^{t(1+\epsilon)m} P(e^{tX} \geq e^{t(1+\epsilon)m}) \tag{2.10}$$
$$\leq e^{-t(1+\epsilon)m} E[e^{tX}] \tag{2.11}$$

Equation 2.11 follows from Markov inequality applied to $P(e^{tX} \geq e^{t(1+\epsilon)m}) \leq e^{-t(1+\epsilon)m} E[e^{tX}]$. Observe that $E[e^{tX}] = E[e^{t\sum_{i=1}^{n} X_i}] =$

$E[\Pi_{i=1}^n e^{tX_i}] = \Pi_{i=1}^n E[e^{tX_i}]$, as X_i's are independent. $E[e^{tX_i}] = p_i e^t + (1-p_i)e^0$. Hence $E[e^{tX}] = \Pi_{i=1}^n (p_i e^t + (1-p_i)) = \Pi_{i=1}^n (1 + p_i(e^t - 1)) \leq \Pi_{i=1}^n e^{p_i(e^t-1)} = e^{\sum_{i=1}^n p_i(e^t-1)} = e^{m(e^t-1)}$. (Note that for this we used the fact that $e^x \geq 1 + x$.) Hence,

$$P(X \geq (1+\epsilon)m) \leq e^{-t(1+\epsilon)m + m(e^t-1)}.$$

This expression holds for all values of $t \in R$. It is minimized for $t = \ln(1+\epsilon)$. To see this, one can write $e^{-t(1+\epsilon)m+m(e^t-1)} = [e^{-t(1+\epsilon)+(e^t-1)}]^m$. To minimize this, one needs to minimize $-t(1+\epsilon) + (e^t - 1)$. Differentiate this with respect to t, and we obtain that $-(1+\epsilon) + e^t = 0$ or $t = \ln(1+\epsilon)$.

Thus,

$$P(X \geq (1+\epsilon)m) \leq (1+\epsilon)^{-(1+\epsilon)m} e^{m\epsilon} = \left(\frac{e^\epsilon}{(1+\epsilon)^{1+\epsilon}}\right)^m \leq e^{-\epsilon^2 m/3}.$$

To show the last inequality, one needs to prove that $\epsilon - (1+\epsilon)\ln(1+\epsilon) \leq -\frac{1}{3}\epsilon^2$, which is left as an exercise.

Next, we show that $P(X \leq (1-\epsilon)m) \leq \exp(-\epsilon^2 m/2)$. The proof is along the same lines as the previous one.

$$\begin{aligned} P(X \leq (1-\epsilon)m) &= P(m - X \geq \epsilon m) && (2.12)\\ &= P(e^{t(m-X)} \geq e^{t\epsilon m}) && (2.13)\\ &\leq e^{-t\epsilon m} E[e^{t(m-X)}] && (2.14)\\ &= e^{tm(1-\epsilon)} E[e^{-tX}] && (2.15) \end{aligned}$$

Note that

$$\begin{aligned} E[e^{-tX}] &= \Pi_{i=1}^n E[e^{-tX_i}] && (2.16)\\ &= \Pi_{i=1}^n [p_i e^{-t\cdot 1} + (1-p_i)e^{-t\cdot 0}] && (2.17)\\ &= \Pi_{i=1}^n [p_i e^{-t} + (1-p_i)] && (2.18)\\ &= \Pi_{i=1}^n [1 - p_i(1-e^{-t})] && (2.19)\\ &\leq \Pi_{i=1}^n [e^{-p_i(1-e^{-t})}] && (2.20)\\ &= e^{-\sum_{i=1}^n p_i(1-e^{-t})} && (2.21)\\ &= e^{-m(1-e^{-t})} && (2.22) \end{aligned}$$

Therefore,

$$P(X \leq (1-\epsilon)m) \leq e^{-m(1-e^{-t})} e^{tm(1-\epsilon)}.$$

This is minimized for $t = -\ln(1-\epsilon)$. Hence,

$$\begin{aligned} P(X \leq (1-\epsilon)m) &\leq (1-\epsilon)^{-m(1-\epsilon)} e^{-m\epsilon}\\ &= \left[\left(\frac{1}{1-\epsilon}\right)^{1-\epsilon} e^{-\epsilon}\right]^m && (2.23)\\ &\leq e^{-\epsilon^2 \frac{m}{2}}. \end{aligned}$$

2.6 Bibliography

As mentioned earlier, the material for this chapter is derived from Blitzstein and Hwang [15], Meyer [116], and Smid [132]. Another excellent resource is the book by Feller [57]. Many of the exercises have been borrowed from various sources, and some of them have been asked in exams/tests in various courses. The BSP Sorting exercise is adapted from Valiant [5]. Randomized routing in a hypercube is covered in the book of Motwani and Raghavan [6]. I am not certain from which source I have picked the exercise on routing. We will also encourage the use of the free software environment *R* for statistical computing, see https://www.r-project.org/

[5] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990

[6]

2.7 Exercises

2.1 *Consider the following three events and assume that each of the dice is fair and outcome of the roll of one die is independent of the outcome of roll of any other die.*

A: Roll six dice and there is at least one 6.

B: Roll twelve dice and there are at least two 6s.

C: Roll eighteen dice and there are at least three 6s.

Which of the above events has the highest chance of occurring? What will happen if the dice are not fair?

2.2 *Let X be a binomially distributed r.v. with parameters n and p. Show that $E[X] = \sum_{k=0}^{n} k\binom{n}{k}p^k(1-p)^{n-k} = np$.*

2.3 *Let the r.v. X be such that $X = c$ where c is a constant. Show that $E[X] = c$.*

2.4 *Let X and Y be two r.v. Show that $E[X+Y] = E[X] + E[Y]$.*

2.5 *Let X and Y be two independent random variables. Show that $E[XY] = E[X]E[Y]$. Will this be true for dependent random variables?*

2.6 *Let X be a discrete r.v. and let g be a function such that $g : \mathbb{R} \to \mathbb{R}$. Show that*

1. *$E[g(X)] = \sum_x g(x)P(X = x)$.*
2. *Let X be a r.v. that represents the value of the roll of a fair die. Show that $E[X] = 3.5$. Consider the function $g : \mathbb{R} \to R$ such that $g(x) = x^2$. Show that $E[g(x)] = 15.167$.*

2.7 *Show that the variance of a r.v. X, $V[X] = E[X^2] - [E[X]]^2$.*

2.8 *If X and Y are independent r.v. than $V[X+Y] = V[X] + V[Y]$. Using a simple example, show that if X and Y are dependent then $V[X+Y] \neq V[X] + V[Y]$.*

2.9 *Let X be a r.v. and c a constant. Show that*

1. $V[X+c] = V[X]$.
2. $V[cX] = c^2 V[X]$.
3. $V[X] \geq 0$.
4. $V[X] = 0$ *if and only if the r.v. X is a constant.*

2.10 *Show that for the Normal distribution $f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2} \geq 0$ for all values of $x \in (-\infty, +\infty)$.*

2.11 *Show that for the Normal distribution $f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2}$, $\int_{-\infty}^{+\infty} f(x)dx = 1$. This is not straightforward. It is best to consider first the case where you take the product of two standard normal distributions and show that $\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-y^2/2} dy = 1$.*

2.12 *Let Z be a r.v. with standard normal distribution. In programming language C, we have the function* rand() *that returns a (pseudo-)random number uniformly distributed, say in the range from 0 to 1. Using Central Limit Theorem, can you think of a method for generating values for Z with the help of* rand() *function? Following is a possibility. Execute* rand() *k times and let the values returned be $x_1, x_2, \ldots, x_k$. Assign $z = x_1 + x_2 + \cdots + x_k - k/2$. Do the values generated by the above procedure have $\mathcal{N}(0,1)$ distribution as $k \to \infty$?*

2.13 *Show that for a Poisson r.v. X with parameter $\alpha > 0$ and $P(X = k) = \frac{e^{-\alpha}\alpha^k}{k!}$, $\sum_{k=0}^{\infty} P(X = k) = 1$ and $E[X] = V[X] = \alpha$.*

2.14 *Assume that one of the online store gets 10 Million hits where the potential customers are trying to identify what they want to buy. We can assume all the hits are independent. What is the probability that 0.001% of these hits will result in actual orders? (Hint: Think of Poisson.)*

2.15 *Show that for a geometric random variable with $P(X = k) = (1-p)^{k-1}p$, $\sum_{k=0}^{\infty} P(X = k) = 1$, $E[X] = 1/p$ and $V[X] = (1-p)/p^2$.*

2.16 *Show that for a geometric random variable with $P(X = k) = (1-p)^{k-1}p$, $P(X \geq s+t | X > s) = P(X > t)$, where s and t are positive integers. Alternatively, we can say that the geometric distribution is memoryless.*

2.17 *Show that for an integer $p > 0$, the Gamma function $\Gamma(p) = (p-1)\Gamma(p-1)$.*

2.18 *This exercise proves the Cauchy-Schwarz inequality. Let X and Y be two r.v. with finite mean and variances. Let t be any real number.*

1. *Show that* $E[(Y - tX)^2] = E[Y^2] - 2tE[XY] + t^2E[X^2] \geq 0$.

2. *Show that* $t = \frac{E[XY]}{E[X^2]}$ *minimizes* $E[Y^2] - 2tE[XY] + t^2E[X^2]$.

3. *Show that* $\sqrt{E[X^2]E[Y^2]} \geq |E[XY]|$.

2.19 *A function g is said to be* convex *if all lines that are tangent to g lie below g. Let X be a r.v. and g be a convex function. Show the following:*

1. *Using the convexity of g, show that for $\mu = E[X]$ the tangent line to g at the point $(\mu, g(\mu))$ is below g.*

2. *Let the equation of the above tangent line be $y = mx + b$. Show that for any $x \in \mathbb{R}$, $g(x) \geq mx + b$.*

3. *Show that $E[mX + b] = g(\mu) = g(E[X])$.*

4. *Show that $E[g(X)] \geq g[E[X]]$.*

2.20 *Recall Markov's inequality (see Theorem 2.5.1). Where in the proof we used the fact that X is a non-negative random variable? What goes wrong with the proof if X can also take negative values?*

2.21 *Usually the bound provided by Markov's inequality is fairly weak. Can you construct a random variable X where the bound is tight (i.e., inequality is equality). Try to design a r.v. X that takes two values, e.g. 1 and 5, with appropriate probabilities.*

2.22 *Let X be a non-negative discrete r.v. and $s > 0$. Markov's inequality establishes a bound on $P(X \geq s)$. Can we adapt Markov's inequality to estimate what will be $P(X \leq s)$?*

2.23 *This exercise provides an alternate and straightforward proof of Markov's inequality (see Theorem 2.5.1). Let X be a non-negative discrete r.v. and $s > 0$. Show the following.*

1. *Show that for an indicator random variable I, $E[I] = P[I = 1]$.*

2. *Define an Indicator r.v. I_s such that $I_s = 1$ if $X \geq s$ and 0 otherwise. Show that $sI_s \leq X$.*

3. *Show that $sE[I_s] \leq E[X]$.*

4. *Show that $P(X \geq s) \leq \frac{E[X]}{s}$.*

2.24 *This exercise provides an alternate proof of Chebyshev's inequality (see Theorem 2.2.1) using Markov's inequality. Let X be a random variable with mean μ and variance σ^2. Let $s > 0$ be a constant.*

1. *Show that $P(|X - \mu| \geq s) = P((X - \mu)^2 \geq s^2)$*

2. *Show that* $P((X-\mu)^2 \geq s^2) \leq \frac{E[(X-\mu)^2]}{s^2}$

3. *Show that* $P(|X-\mu| \geq s) \leq \frac{\sigma^2}{s^2}$

2.25 *Let X be a r.v. and let $s > 0$ and $t > 0$ be constants. Use Markov's inequality to show the following:*

1. $P(X \geq s) = P(e^{tX} \geq e^{ts})$

2. $P(X \geq s) \leq \frac{E[e^{tX}]}{e^{ts}}$

2.26 *In the last three exercises we have established various upper bounds for $P(X \geq s)$. Let the r.v. X obeys the standard normal distribution $\mathcal{N}(0,1)$. Here we are interested to know what is the cumulative density of X taking values larger than three times the standard deviation. By definition, this value is $P(X \geq 3) = \int_3^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} dx = 0.0015$. Estimate the value of $P(X \geq 3)$ using the bounds from the last three exercises and compare them against the value obtained above. (Recall that $\mathcal{N}(0,1)$ is symmetric. Evaluation of $P(X \geq 3) \leq \frac{E[e^{tX}]}{e^{3t}}$ will require some work, but this quantity is minimized when $t = 3$.)*

2.27 *For a fair coin, let Y be the indicator r.v. denoting whether the outcome of toss is a Head ($Y = 1$) or Tail ($Y = 0$). Note that $E[Y] = 1/2$ and $V[Y] = 1/4$. Suppose we toss a fair coin n times and record the total number of heads obtained in a r.v. $S \in \{0, \ldots, n\}$. By the Central Limit Theorem, for large values of n, we know that $T_n = \frac{S-n\mu}{\sqrt{n}\sigma}$ has a standard normal distribution $\mathcal{N}(0,1)$. For $n = 100$, we have that $E[S] = 50$ and $V[S] = 25$. What is the probability that (a) $45 \leq S \leq 55$, (b) $40 \leq S \leq 60$, and (c) $35 \leq S \leq 65$? (The answers should correspond to probability density of one, two, and three standard deviations around the mean for a Normal Distribution, respectively.)*

2.28 *As a promotion, the manufacturers of the* GoodForMe *cereal have placed a toy car in each of its cereal boxes. You can determine the color of the toy car, only by buying and then opening the cereal box. Each toy car is of a monochromatic color among possible $n \geq 1$ colors. Once you collect cars of all possible colors, then you win a real car. The company officials have ensured that a cereal box is equally likely to contain a car of any of the possible n-colors. Let X be the random variable that represents the number of cereal boxes purchased to obtain toy cars of each of the colors. For $j = 0, 1, \ldots, n-1$, let the random variable X_j represents the number of additional cereal boxes that are purchased after the cars of j different colors have been collected until a car of new color is obtained. Answer the following questions:*

1. *Show that* $X = \sum_{j=0}^{n-1} X_j$.

2. *Show that after the cars of j distinct colors have been obtained, the probability that the color of the car in the next cereal box that is purchased is new (i.e. different from any of the j colors) is $\frac{n-j}{n}$.*

3. *Show that X_j has a geometric distribution with parameter $\frac{n-j}{n}$.*

4. *Show that $E(X) = n \sum_{j=1}^{n} \frac{1}{j}$.*

5. *Suppose that $n = 100$. Use the approximation $\sum_{j=1}^{n} \frac{1}{j} \approx \ln n + 0.5772$ to determine the expected number of cereal boxes that need to be bought to collect cars of all the colors.*

2.29 *A hockey fan wants to collect a complete set of* 100 *hockey cards. The cards are available randomly, one per package of chewing gum, that the fan buys twice daily. On average, how long will it take the fan to collect the complete set? Justify.*

2.30 *Suppose we hash the elements of a set S having* 23 *members to a bit array of length* 100*. The array is initially all-*0*'s, and we set a bit to* 1 *whenever a member of S hashes to it. The hash function is random and uniform in its distribution. What is the expected fraction of* 0*'s in the array after hashing? You may assume that* 100 *is large enough that asymptotic limits are reached.*

2.31 *Assume that our government wants to launch a new policy but wants to take a random population sample and see whether it is even worth the effort. Everybody in the population is either for or against the proposed policy. Assume there are no biases when selecting whom to survey. I.e., each individual is equally likely to be picked for the survey (and you may assume with replacement or without, whichever scenario makes it easier for the analysis). Assume that we survey a total of n people from the population. Let p be the proportion of people in the population supporting the policy, and let p' be the proportion of the people in the survey supporting the policy. Show, using Chebyshev's inequality, that for any constant $c > 0$, $Pr(|p - p'| > c) \leq \frac{1}{4nc^2}$.*

Hint: Set up an indicator random variable X_i for the i-th person in the sample. Evaluate $E[X_i]$ and $Var[X_i]$. Note that $p(1-p) \leq \frac{1}{4}$.

For what value of n, there is at least 98% *chance that $|p - p'| < 3\sigma$, where σ is the standard deviation of the random variable $X = \sum_{i=1}^{n} X_i$.*

2.32 *A popular coffee store in Amherstburg opens at 9 A.M. Many of its customers have indicated that they would prefer that the store opens at 7:30 A.M. The store has decided to estimate the fraction p of the town's population that will like an early opening by surveying town's population.*

They need to determine the number of people n to whom to send a survey to get within 10% of the true value of p with a probability of at least 0.95. Using Chebyshev's inequality, we will get an estimate on n. Let $\mathcal{X} = \{X_1, \ldots, X_n\}$ be the $0-1$ indicator random variables corresponding to the set of n people picked, uniformly at random, from the town's population for the survey. Note that X_i is 1 if the i-th person prefers an early opening, otherwise it is 0. Let S_n be the number of people in $\mathcal{X}$ who will like an early opening. Define $A_n = \frac{1}{n}S_n$. Answer the following:

1. *Show that $E[A_n] = p$, i.e., A_n is an unbiased estimator of p.*

2. *Show that variance σ of any of X_i is $Var[X_i] = \sigma^2 = p(1-p)$.*

3. *Show that $Var(A_n) = \frac{1}{n}\sigma^2$.*

4. *Conclude that if n is large, $E[A_n]$ will be concentrated around p.*

Define the two parameters $\epsilon, \delta \in (0,1)$, where ϵ accounts for the error in the estimate ($\epsilon = 0.1$ for our problem), and δ captures the desired accuracy. We are interested to evaluate $Pr(|A_n - p| \leq \epsilon p) \geq \delta$. I.e., for our problem, we want to understand what value of n will satisfy $Pr(|A_n - p| \leq 0.1 * p) \geq 0.95$? Note that, by Chebyshev's inequality, $Pr(|A_n - p| \geq \epsilon p) \leq \frac{Var(A_n)}{(\epsilon p)^2}$. Answer the following.

1. *Show that $n \geq \frac{p(1-p)}{(1-\delta)(\epsilon p)^2}$*

2. *For $\epsilon = 0.1$ and $\delta = 0.95$, show that $n \geq \frac{2000(1-p)}{p}$.*

3. *Show that for any $0 < p' \leq p$, $\frac{2000(1-p')}{p'} \geq \frac{2000(1-p)}{p}$*

4. *If the coffee shop knows that $p' \geq 0.20$ (as gathered from the customers that come to the store everyday), show that $n \approx 8000$ people are enough to survey to get the desired estimate on p.*

5. *Conclude that if we know of a good lower bound p' of p, our estimate of n will be close to the optimal number of people to survey.*

2.33 *Let $G = (V, E)$ be a simple undirected graph, where $n = |V|$ and $m = |E|$. We partition the set of vertices V into two sets A and B such that $V = A \cup B$ and $A \cap B = \emptyset$. We decide which vertices of G will be in A or B by the following random process: For each vertex $v \in V$, independent of any other vertex, we toss a biased coin. If it shows up heads, we place v in A, otherwise we place it in B. The biased coin has 2/3rd probability of heads, and 1/3rd probability of tails. We say an edge $e = (uv)$ of G is a cut edge if either (a) $u \in A$ and $v \in B$, or (b) $u \in B$ and $v \in A$. What is the expected number of cut edges in G?*

2.34 *Suppose I have two coins in my pocket - a fair coin and a two headed coin (i.e. both sides are Heads). I pull one of the coins (randomly) from my pocket and toss it and obtain a Head. What is the probability that the coin which I tossed is the fair coin? What is the probability that it is the 2-headed coin? Provide some reasoning for your answer.*

2.35 *Assume that the Canadian population is* 40 *million. Everyone in the population, irrespective of what others are doing, has a* 1% *chance that they will attend a Zoom online meeting on any given day. Each Zoom session can handle 100 people; on average, we have 400,000 Zoom sessions running each day. The Health Agency is monitoring the entire population for three years (say 1100 days) to find if there are COVID rule breakers by analyzing the attendance in each Zoom session (as there is a chance they may be planning a physical gathering). We say a group of* $p \geq 2$ *people are potentially* spreaders *if they together attend the Zoom sessions for* d *different days out of 1100. Assume that there are* **no** *spreaders. Derive an expression for an expected number of sets of* p *people who may be (falsely) identified as spreaders. Also, show your computations for* $p = d = 3$*. If required, you can make some assumptions, e.g.,* $\binom{1100}{d} = 1100^d/d!$*, independence, . . .*

2.36 *A beer distillery has a tester (hopefully not a person) which can very quickly test the quality of each bottled beer on its assembly line and accept or reject them based on whether they pass or fail its test. If a bottle of the beer doesn't meet the standards then with 95% certainty the tester will report that the bottled beer is unacceptable. If a bottled beer is perfect, than there is still a 5% chance that the tester may say that the beer doesn't meet the standards. Suppose on the average this distillery produces 10% of the bottled beers which do not meet the standards on any given day. Let us do the following experiment. Choose a bottle of beer uniformly at random before it reaches the tester, and let us assume that the tester reports that this particular bottled beer doesn't meet the standards. What is the probability that this bottle actually doesn't meet the standards? Provide some reasoning for your answer.*

2.37 *Suppose you have a biased coin* C *(i.e.* $Pr(H) \neq Pr(T) \neq \frac{1}{2}$*). Von Neumann suggested the following method to use the biased coin* C *to simulate an unbiased coin. Strategy is to toss* C *twice and note down the outcomes. If the first toss of* C *results in Head and the second one to Tail, then we say that the outcome is* Head*. If the first toss of* C *results in Tail and second one to Head then the outcome is* Tail*. Otherwise (that is both the tosses of* C *are either Heads or Tails) we repeat the above process (i.e., we will again toss twice ...). Show that the above method simulates an unbiased coin, i.e. probability that we output* Head *is same as the probability that we output* Tail*.*

2.38 *You are given a list L of elements and want to choose a random element in this list. Each element of L should have the same probability of being chosen. Unfortunately, you do not know the number of elements in L. You are allowed to make only one pass over the list. Consider the following algorithm:*

Algorithm PickRandomElement*(L):*
u = first element of L;
$i = 1$;
while *u exists*
do *with probability* $1/i$, *set* $x = u$;
    u = successor of u in L;
    $i = i + 1$
endwhile;
return x

Prove that the output x of this algorithm is indeed a random element of L. In other words, prove the following: Let v be an arbitrary element of L. Then, the probability that $x = v$ *after* PickRandomElement*(L) has terminated is equal to* $1/n$, *where n is the number of elements in L.*

2.39 *This question extends previous algorithm. We now want to maintain* $k > 1$ *elements rather than a single element. Assume the list L consists of more than k elements. Here is the modified algorithm:*

Algorithm PickRandom-*k*-Elements*(L):*
Let $u_1, u_2, \ldots, u_k$ *be the first k-elements of L;*
Form a set $R = \{u_1, u_2, \ldots, u_k\}$;
$i = k + 1$;
Let u be the i-th element of L;
while *u exists*
do *with probability k/i, replace a random number in R with u.*
    u = successor of u in L;
    $i = i + 1$
endwhile;
return R

For this question it will be helpful to think of elements in the list L are $u_1, u_2, u_3, \ldots$, *where* u_1 *is the first element,* u_2 *is second, . . . Consider the i-th iteration of the While loop, i.e. the iteration when the i-th element* u_i *in L is considered for the first time, for any* $i > k$. *Let x be any of the elements among the first i elements of L (i.e.,* $x \in \{u_1, u_2, \ldots, u_i\}$*). Prove that the probability that* $x \in R$ *at the end of this iteration is* k/i.

2.40 *Assume that we have 100 bins, numbered 1 to 100. We also have 75 balls. In our experiment, we throw each ball uniformly at random in any of the 100 Bins. Estimate the following probabilities after we have placed all the Balls in Bins. Please provide some justification for your answers. (Note: Yor can leave your answers at the expression stage, e.g. you may say the*

probability is $\binom{100}{75}(\frac{1}{100})^{75}$ *rather than calculating the actual value. Recall that* $1 - x \leq e^{-x}$*.)*

1. *Probability that at least one bin is empty.*
2. *Probability that the first bin is empty.*
3. *Probability that at least one of the first two bins is non-empty.*
4. *Probability that no two balls occupy the same bin.*
5. *Probability that the first bin receives exactly 10 balls.*

2.41 *Suppose we have m balls and n bins. The balls are thrown independently and uniformly at random in the bins. Note that the probability that the k-th ball falls into i-th bin is* $1/n$*, where* $1 \leq k \leq m$ *and* $1 \leq i \leq n$*. A bin is* occupied *if it consists of one or more balls, otherwise it is said to be* empty*. Answer the following questions:*

1. *For a fixed index i, what is the probability that i-th bin is empty.*
2. *Give a (non-trivial) upper bound on the probability that at least one of the bins is empty.*
3. *Give a (non-trivial) lower bound on the probability that all the bins are non-empty.*
4. *Assume that* $m = 2n \ln n$*. Show that probability that all the bins are non-empty approaches* 1*, when n approaches* $+\infty$*.*

2.42 *Let us flip a fair coin a dozen time. Estimate the following probabilities.*

1. *What is the probability of getting exactly six heads?*
2. *What is the probability of getting at most six heads?*
3. *What is the probability that we get at least seven heads or seven tails in a row?*
4. *What is the probability that the sequence the heads and tails forms a palindromic sequence?*

2.43 *Assume that you have a set P of n distinct numbers. Form a sequence S of these numbers by a taking a random permutation of elements of P. We compute the minimum element,* $MIN(P)$*, of this set by an incremental algorithm as follows:*

1. $MIN(P) := S[1]$*;*
2. *For* $i := 2$ *to n do*
   if $S[i] < MIN(P)$, $MIN[P] := S[i]$

What is the expected number of times that $MIN(P)$ will be updated in Step (b) of the above algorithm?

2.44 *Let X be the total number of heads obtained in a sequence of n independent flips of a fair coin. We know that the expected value of X is $\frac{n}{2}$. Using Chernoff bounds compute the following probabilities:*

1. $Pr(|X - \frac{n}{2}| > \frac{1}{2}\sqrt{6n \ln n})$
2. $Pr(|X - \frac{n}{2}| > \frac{n}{4})$
3. *and evaluate the above expressions for different values of n and make some remarks on the values you obtain.*

2.45 *Recall that a permutation is one-to-one and onto function $\pi : [1, 2, 3 \ldots, n] \rightarrow [1, 2, 3 \ldots, n]$ such that for every integer i, $1 \leq i \leq n$, there is exactly one integer $j \in \{1, \ldots, n\}$ such that $\pi(i) = j$. For example the permutation $[12345] \rightarrow [23154]$ represents that elements are mapped as follows: $\pi[1] = 2$, $\pi[2] = 3$, $\pi[3] = 1$, $\pi[4] = 5$, and $\Pi[5] = 4$. We can visualize a permutation as a set of directed cycles as follows. Assume that we have a vertex for each number, $1 \leq i \leq n$. If $\Pi[i] = j$, we draw a directed arc from vertex i to j. (Notice that a cycle may be a self-loop if $\pi(i) = i$.) In our example permutation we have 2 directed cycles $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$ and $4 \rightarrow 5 \rightarrow 4$, and their lengths are 3 and 2, respectively. What is the expected number of cycles in a random permutation?*

(Hint: Each vertex is in some directed cycle. Show that the probability that a vertex i is in cycle of length k ($1 \leq k \leq n$) is $\frac{1}{n}$, irrespective of the length of the cycle. You may also want to define a r.v. X_i for each vertex i, where $X_i = \frac{1}{k}$, if vertex i participates in the cycle of length k. Think of the quantity $X = \sum_{i=1}^{n} X_i$.)

2.46 *Suppose you want to rent an apartment in Old Ottawa South and you have hired an agent to show all the possible apartments within your budget over the next weekend. Suppose the agent wants to show you n apartments and tells you that as soon as you make on offer to any of them, it will be accepted. As a computer scientist, you compute a random permutation of the order in which you will like to see these apartments and let the permuted order be $\pi_1, \pi_2, \pi_3, \ldots, \pi_n$. You tell the agent that on Saturday you will see the first $\frac{n}{e}$ of these, i.e. the apartments labelled $\pi_1, \pi_2, \ldots, \pi_{\frac{n}{e}}$. (We assume that the Euler number e divides n.) After viewing each of these apartments, you make some mental notes, but at the end of the day Saturday you tell the agent that you will like to see the remaining ones, in the order of the permutation $\pi_{\frac{n}{e}+1}, \ldots, \pi_n$, on Sunday. The strategy that you have decided to employ on Sunday is to make an offer to rent the very first apartment that you see which you think is better than what you have seen so far (including what you saw on Saturday) and then terminate your visit to any remaining*

unvisited apartments. On your Sunday's apartment hunting venture with the agent, you make an offer for apartment π_α. (There is a possibility that we may not find any better apartment on Sunday and hence may not make any offer - to keep the arguments simple, if you prefer, you may assume that you make an offer.) Answer the following questions:

1. *Suppose if you would have seen all the apartments, then your ranking of the apartments to rent (from highest to lowest), without loss of generality be $1, 2, 3, \ldots, n$ (i.e., 1 is best, 2 is second best, ..., n is the worst). Note that the order you visit the apartments is a random permutation of $\{1, 2, \ldots, n\}$. Suppose the smallest apartment number, i.e. the most preferred, among $\pi_1, \pi_2, \ldots, \pi_{\frac{n}{e}}$ be x. Show that $\pi_\alpha < x$.*

2. *Show that the probability of making the optimal choice in the above strategy is given by*
$$\sum_{i=\frac{n}{e}+1}^{n} Pr[\text{We see the apartment } \pi_i \text{ and } \pi_i = 1]$$
$$= \sum_{i=\frac{n}{e}+1}^{n} Pr[\pi_i = 1 \text{ and minimum of } \{\pi_1, \pi_2, \ldots, \pi_{i-1}\} \text{ is in } \{\pi_1, \pi_2, \ldots, \pi_{\frac{n}{e}}\}]$$
$$= \sum_{i=\frac{n}{e}+1}^{n} \frac{1}{n} \times \frac{\frac{n}{e}}{i-1}$$

3. *Conclude by showing that the probability that the above strategy selects the best apartment is approximately $1/e = 37\%$. (Recall that the n-th Harmonic number $\sum_{i=1}^{n} \frac{1}{i} \approx \ln n$.)*

2.47 BSP Sorting: *Let S be a set of n distinct numbers and let R be a subset of S. The sorted elements $y_1 < y_2 < \ldots < y_{|R|}$ of R partition the set $S \setminus R$ into $|R| + 1$ subsets, which we call* open intervals*:*

$$S_0 = \{x \in S : x < y_1\},$$

$$S_i = \{x \in S : y_i < x < y_{i+1}\}, i = 1, 2, \ldots, |R| - 1,$$

and

$$S_{|R|} = \{x \in S : x > y_{|R|}\}.$$

(If $R = \emptyset$, then there is one open interval S_0, which is equal to S.) If we regard R as being a sample that "represents" S, then the "ideal" sample would have the property that all open intervals have (approximately) the same number of elements. Given an integer r with $1 < r < n$, we can obtain such an "ideal" sample R of size r, in $O(n \log n)$ time: First sort the elements of S, then add each (n/r)-th element to R. (Rounding is ignored here.)

In this question, we will see a simple randomized algorithm that computes a "good" sample (to be defined below) with probability at least $1/2$, if r is not too small. For the rest of this question, we fix an integer r with $1 < r < n$.

Consider the following algorithm:

Algorithm RANDOMSAMPLE(S, r)
$p = r/n$;
$R = \emptyset$;
for each $x \in S$
do *with probability* p, *add* x *to* R
endfor;
sort the elements of R;
compute the open intervals $S_0, S_1, \ldots, S_{|R|}$;
return $R, S_0, S_1, \ldots, S_{|R|}$

We say that the sample R *is* good *if*

1. $1 \leq |R| \leq 2r$, *and*
2. *for each* i *with* $0 \leq i \leq |R|$, *the open interval* S_i *contains at most* $\frac{2n \ln r}{r}$ *elements of* S.

Otherwise, the sample R *is called* bad. *In words, a good sample* R *is (i) non-empty, (ii) at most twice as large as the sample size we are aiming for, and (iii) the elements of* $S \setminus R$ *are approximately evenly distributed over the open intervals (except for the* $\ln r$ *factor).*

Answer the following questions:

1. *Compute the expected size* $E(|R|)$ *of the set* R.
2. *Prove that*
$$\Pr(R = \emptyset) \leq e^{-r}.$$
(Hint: *Recall that* $1 - z \leq e^{-z}$ *for all real numbers* z.)
3. *Use the Chernoff bound to show that*
$$\Pr(|R| > 2r) \leq e^{-r/3}.$$
4. *Consider the sorted sequence* $x_1 < x_2 < \ldots < x_n$ *of elements of* S. *Let* k *be an integer that divides* n *(thus,* n/k *is an integer and no rounding is needed below). Partition* S *into* n/k *subsets* $B_1, B_2, \ldots, B_{n/k}$, *each containing* k *elements:* B_1 *contains* $x_1, \ldots, x_k$; B_2 *contains* $x_{k+1}, \ldots, x_{2k}$, *etc. We call each subset* B_i *a* bucket *and say that it is* empty *if* $B_i \cap R = \emptyset$.

   Argue that the following is true:

   - *If each bucket is non-empty, then each open interval contains at most* $2k$ *elements of* S.

5. *Prove the following:*
$$\Pr(\textit{ each bucket is non-empty }) \geq 1 - \frac{n}{k}(1-p)^k.$$

6. *Argue that*

$$\Pr(\text{ each open interval contains at most } 2k \text{ elements of } S\ ) \geq 1 - \frac{n}{k}(1-p)^k.$$

7. *Recall that $p = r/n$. Let $k = \frac{n \ln r}{r}$. (You may assume that k is an integer that divides n, so that no rounding is needed.) Prove that*

$$\Pr\left(\text{ at least one open interval contains more than } \tfrac{2n\ln r}{r} \text{ elements of } S\ \right) \leq \frac{1}{\ln r}.$$

(Hint: *Recall that $1 - z \leq e^{-z}$ for all real numbers z.)*

8. *Show that*

$$\Pr(\text{ the sample } R \text{ is bad }) \leq e^{-r} + e^{-r/3} + \frac{1}{\ln r}.$$

9. *Show the following: If r is chosen sufficiently large, then*

$$\Pr(\text{ the sample } R \text{ is good }) \geq \frac{1}{2}.$$

2.48 *A discrete Poisson random variable X with parameter μ is given by the following probability distribution on $j = 0, 1, 2, \ldots$*

$$Pr(X = j) = \frac{e^{-\mu}\mu^j}{j!}$$

If we throw m balls in n bins uniformly at random, then the probability that a particular bin receives r balls is given by $\binom{m}{r}(\frac{1}{n})^r(1-\frac{1}{n})^{m-r} \approx e^{-\frac{m}{n}}(m/n)^r/r!$ (for small values of r).

Suppose we throw $n^{0.97}$ balls into n bins uniformly at random. Provide the best upper bound on r so that the maximum number of balls in a bin is at most r with probability at least $1/2$. We can assume n is large.

2.49 Permutation routing in a hypercube: *A hypercube H_n of dimension $n \geq 1$ is a interconnected network with 2^n nodes, each node is labelled with a bit string of length n, and it is recursively defined as follows. A dimension 1 hypercube H_1 is a pair of interconnected nodes, one labelled as 0 and other labelled as 1. A dimension 2 hypercube H_2 is formed by taking two H_1's, and interconnecting the corresponding nodes, and adding a prefix of 0 (respectively, 1) to the label of each node in the first (respectively, second) H_1. Similarly, H_n is formed by taking two H_{n-1}'s, and interconnecting the corresponding nodes and adjusting the node labels. See Figure 2.3 for an illustration.*

First, answer the following questions regarding the topology of a n-dimension hypercube H_n.

1. *How many vertices and edges H_n has?*

2. *Show that the diameter of H_n is n.*

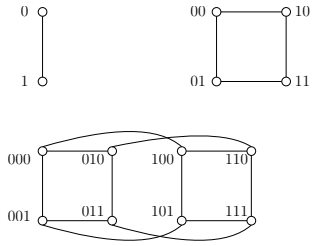


Figure 2.3: Hypercubes of dimensions 1, 2, and 3.

3. *Show that each node in H_n is connected to exactly n other nodes. (Hint: Show that a node labelled with the bit sequence $(b_n b_{n-1} \cdots_2 b_1)$ is connected to all nodes whose bit sequence differs in exactly one bit (i.e. the Hamming distance between the two bit-strings is exactly 1.))*

4. *Are all the cycles in a hypercube of even length? Is H a bipartite graph?*

Define the bit-fixing path *from a node u with label $(b_n b_{n-1} \ldots b_2 b_1)$ to a node v with label $(b'_n b'_{n-1} \ldots b'_2 b'_1)$ as follows. At the current node in the path, scan the bits in its label from left to right, and go to the node which differs in the leftmost bit with respect to the label of v. For example, bit-fixing path between $u = 11001$ and $v = 10010$ will be $u = 11001 \rightarrow 10001 \rightarrow 10011 \rightarrow 10010 = v$.*

Consider the following permutation routing problem on H_n. Initially each node consists of a packet, and each packet has the label of the destination node. Moreover no two packets have the same destination address. We will use the bit-fixing path strategy to route the packets to their respective destinations. Moreover, we do not allow more than one packet to travel on an edge of the hypercube at any given time. Thus, if more than one packet wants to use the same edge, then they need to wait for their turn. We assume that the packets can traverse an edge on the first-come-first-serve basis. Answer the following questions:

1. *Assume n is even and $N = 2^n$. Express an n-bit number i as the concatenation of two binary strings a_i and b_i of length $n/2$ each. (For example, for $n = 8$, we can express $i = (01110101)_2$ as concatenation of two 4-bit numbers $a_i = (0111)_2$ and $b_i = (0101)_2$.) Assume that we have an instance of the permutation routing problem, where each node of the n-dimensional hypercube initially contains a packet. For each node $i = (a_i b_i)$, the destination of the packet v_i stored initially at node i is $d_i = (b_i a_i)$. (For the above example, the destination of the packet v_i initially stored at node $i = (01110101)_2$ will be $d_i = (01010111)_2$). Show that the bit-fixing routing scheme, where each link can carry at most one packet in one step, requires atleast $\Omega(\sqrt{N}/n)$ steps to deliver all the packets to their final destinations.*

2. *For any node i, let the bit-fixing routing path of its packet be π_i. Let L be the maximum number of edges on any path π_i, $1 \leq i \leq N$. Let $c_e = |\{i | \text{path } \pi_i \text{ contains edge } e\}|$, i.e. the number of paths that contain e. Let $C = \max\{c_e | e \text{ is an edge of } G\}$. Show that the worst case time for accomplishing the routing in this network is $\Omega(L + C)$.*

3. *Assume that we allow any number of packets to travel on an edge in one time unit. At the start, for each packet v_i, we choose a delay x_i i.e. a random integer in $\{1, 2, \ldots, \lceil \frac{C}{\ln(NL)} \rceil\}$. Now each packet v_i first waits for the x_i units of time at the source node, then follows the path π_i to*

its destination without incurring any delays. Show that all the packets will reach their destinations in at most $L + \lceil \frac{C}{\ln(NL)} \rceil$ *time units. For a particular edge* e *in* G*, let* $m = c_e$ *be the number of packets that visit* e*. Consider one of these packets, and show that the probability that packet visits* e *at a specific time* t *is at most* $\frac{\ln(NL)}{C}$*.*

4. *Define a random variable* $X_{e,t}$ *that counts the number of packets that traverse edge* e *at time* t*. Note that* $X_{e,t}$ *can be expressed as sum of* m *identical independent indicator random variables. Show that the expected value* $E[X_{e,t}] = m\lceil \frac{\ln(NL)}{C} \rceil$*. Apply Chernoff bounds to* $X_{e,t}$ *and show that the probability that* $Pr(X_{e,t} \geq (1+\epsilon)m\lceil \frac{\ln(NL)}{C} \rceil) \leq e^{-\frac{\epsilon^2 m \ln(NL)}{3C}} = (\frac{1}{NL})^{\frac{\epsilon^2 m}{3C}}$*.*

5. *Note that* $m \leq C$ *and therefore* $E[X_{e,t}] \leq \ln(NL)$*. Furthermore, show that for* $\epsilon \geq 4$*,* $Pr(X_{e,t} \geq 5\ln(NL)) \leq (\frac{1}{NL})^5$*.*

6. *Using the fact that that there are at most* NL *edges all together in all the paths* π_i*,* $i = 1, \ldots, N$*, and there are at most* $L + \frac{C}{\ln(NL)}$ *time steps, show that the probability that there exists an edge* e *and there exists a time* t *such that more than* $\alpha \ln(NL)$ *packets travel on* e *at time* t *is at most* $\frac{1}{(NL)^2}$*, for some constant* α*.*

7. *Now consider the restriction that at most one packet can travel on any given edge in one time step. Show that with probability* $\geq 1 - \frac{1}{(NL)^2}$*, the maximum queue size is* $O(\ln(NL))$ *and all the packets reach their destinations in* $O((L + \frac{C}{\ln(NL)}) \times$ *max queue size*$) = O(C + L\ln(NL))$ *time units.*

2.50 *Let* A *be a set of* n *items. Without loss of generality assume that* $a_1, a_2, \ldots, a_k$ *are the most frequent* k *elements in* A *with frequencies* $f_1 \geq f_2 \geq \cdots \geq f_k$*, respectively. We sample each element of* A *uniformly at random (with replacement) to construct a multi-set* A'*. We are interested to know how many times we need to sample* A *so that the most frequent* k *elements have a representative in* A' *with high probability. In other words, what should be the size of* A' *so that with probability* $\geq 1 - \epsilon$*, for* $\epsilon > 0$*, so that* $a_1, \ldots, a_k \in A'$*.*

Hint: Let us assume that $s = |A'|$*. Estimate first the probability that if we choose* s *elements from* A*, each uniformly at random with replacement, what is the probability that* $a_k \notin A'$*? What is an upper bound on the probability that none of* $a_1, a_2, \ldots, a_k$ *are in* A'*? Show that by choosing* $s = O(\frac{n}{f_k} \log \frac{k}{\epsilon})$*, with probability* $\geq 1 - \epsilon$*,* $a_1, \ldots, a_k \in A'$*.*

2.51 *In this exercise, we will derive a result on two families of sets due to Bollabos, see* [7] *for a nice proof. Let* U *be a universe of* n *elements. Let* $A_1, \ldots, A_m$ *be* p *element subsets of* U *and* $B_1, \ldots, B_m$ *be* q *elements subsets of* U *with the following property*

[7] G.O.H Katona. Solution of a problem of A. Ehrenfeucht and J. Mycielski. *Journal of Combinatorial Theory, Series A*, 17(2):265–266, 1974

(P1) For each $i \in \{1, \ldots, m\}$, $A_i \cap B_i = \emptyset$.
(P2) For each $1 \leq i, j \leq m$, $i \neq j$, $A_i \cap B_j \neq \emptyset$.
With the help of the following exercises, we will show that $m \leq \binom{p+q}{p}$, i.e. m doesn't depend on the size of the universe U. Answer the following:

1. *Show that $m \leq \binom{n}{p}$ and $m \leq \binom{n}{q}$.*

2. *Let $U = 1, \ldots, p+q$. Let $A_1, \ldots, A_m$ be subsets of size p of U. Let $B_1 = U - A_1, \ldots, B_m = U - A_m$. Show the following:*
   (a) Size of each B_i is q.
   (b) $A_1, \ldots, A_m$ and $B_1, \ldots, B_m$ satisfy P1 and P2.
   (c) $m \leq \binom{p+q}{p}$.

3. *Let Π be a uniform at random permutation of elements of U, and let X_i be the event that all the elements of A_i precede all the elements of B_i in Π. Answer the following*

   (a) Show that the probability of X_i to occur is $Pr(X_i) = \frac{\binom{n}{p+q} p!q!(n-(p+q))!}{n!}$.

   (b) Show that the above expression simplifies to $Pr(X_i) = 1/\binom{p+q}{p}$.

   (c) Show that at most one of the events $X_1, \ldots, X_m$ can occur in any permutation of elements of U. I.e. these events are disjoint.

   (d) Show that $Pr\left(\bigcup_{i=1}^{m} X_i\right) \leq 1$.

   (e) Show that $Pr\left(\bigcup_{i=1}^{m} X_i\right) = \sum_{i=1}^{m} Pr(X_i)$.

   (f) Conclude that $m \leq \binom{p+q}{p}$.

2.52 *This exercise is from [8]. Let $\mathcal{P} = \{P_1, \ldots, P_n)$ be a collection of n convex polygons in plane. We say P_i and P_j intersect, if they have a point in common, otherwise they are disjoint. We need to determine if there is a collection of ϵn polygons that can be removed from $\mathcal{P}$ such that the remaining polygons are pairwise disjoint, where $0 < \epsilon < 1$. We execute the following probabilistic algorithm.*

[8] Artur Czumaj and Christian Sohler. Testing euclidean minimum spanning trees in the plane. *ACM Trans. Algorithms*, 4(3):31:1–31:23, 2008

> *Step 1: Choose $O(\sqrt{n/\epsilon})$ polygons from $\mathcal{P}$ uniformly at random. Let the set of chosen polygons be S.*
>
> *Step 2: Report TRUE if polygons within S are disjoint, otherwise report FALSE.*

Suppose that by removal of any collection S of ϵn polygons from $\mathcal{P}$ the remaining polygons aren't pairwise disjoint. Let $s = |S|$.

1. *Show that for $k = \frac{\epsilon n}{2}$ times, we can pick a pair of intersecting polygons A_i and B_i from $\mathcal{P}$ and remove them from $\mathcal{P}$.*

2. *Show that* $Pr(\exists i \in [k] : \{A_i, B_i\} \in S\} \geq \sum_{i=1}^{k} Pr(\{A_i, B_i\} \in S) - \sum_{1 \leq i < j \leq k} Pr(\{A_i, B_i, A_j, B_j\} \in S)$.

3. *Show that* $\sum_{i=1}^{k} Pr(\{A_i, B_i\} \in S) = \sum_{i=1}^{k} \frac{\binom{n-2}{s-2}}{\binom{n}{s}}$

4. *Show that* $\sum_{1 \leq i < j \leq k} Pr(\{A_i, B_i, A_j, B_j\} \in S) = \sum_{1 \leq i < j \leq k} \frac{\binom{n-4}{s-4}}{\binom{n}{s}}$

5. *For some choice of $s \in \Theta(\sqrt{n/\epsilon})$, show that $Pr(\exists i \in [k] : \{A_i, B_i\} \in S\}) \geq \frac{1}{4}$.*

Thus, if the collection $\mathcal{P}$ is almost disjoint, we need to only test if $O(\sqrt{n})$ objects in S are interesting, and with probability at least $1/4$ we get the correct result.

3
Introduction to Graphs

We will focus on

1. Undirected and directed graphs.
2. Adjacency matrix and list representation of graphs.
3. Breadth-first search.
4. Depth-first search.
5. Topological sort.
6. Equivalence Relation and Bi-connectivity.

Keywords: Vertices, Edges, undirected and directed graphs, connectivity, connected and biconnected components, strongly connected, complete graphs, complete bipartite graphs, K_5, K_{33}, graph traversal, BFS, DFS.

3.1 Introduction and Definitions

Graphs were discovered by Euler (Königsberg bridge problem)[1], Kirchoff (electrical networks)[2] and Cayley (enumeration of organic chemical isomers)[3] in different contexts. Graphs are combinatorial structures used in computer science. Lists, Trees, Directed Acyclic Graphs, Flow Charts, Control Flow Graphs, Planar Graphs, web, unit disk graphs, and communication networks are examples of graphs that are widely used in computer science. Most often, practical problems, can be cast into some sort of graph problem. Examples include the Traveling Salesperson problem (finding a route of the cheapest cost through many cities), or coloring a map so that no two neighboring countries receive the same color or finding shortest path from Carleton to National Art Gallery, or navigating hyperlinks in webpages. There are excellent books and thousands of papers discussing

[1] Leonhard Euler, 1707-1783

[2] Gustav Kirchoff, 1824-1887: At every node in an electrical circuit the sum of all currents should be equal to zero, i.e., the charge cannot accumulate at a node - or what comes in must go out!

[3] Arthur Cayley, 1821-1895

several aspects of graphs (definitions, connectivity, coloring, independent sets, matchings, Kuratowski's theorem, four color theorem, ...). Some of the classical books include [20, 21, 78, 107]. We need to get used to some of the basic definitions.

Graph A graph $G = (V, E)$ consists of a finite set of *vertices* V and a finite set of *edges* E. See Figure 3.1 for an illustration.

- *Undirected graph*: E is a set of unordered pairs of vertices $\{u, v\}$ where $u, v \in V$ (see Figure 3.1).
- *Directed graph*: E is a set of ordered pairs of vertices (u, v) where $u, v \in V$ (see Figure 3.2).

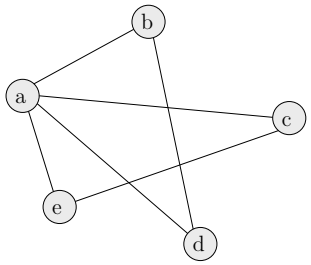


Figure 3.1: An example of a undirected graph. The edge $\{a, b\}$ is incident to the vertex labelled a and to the vertex labelled b. The degree of vertex a is 4, degree of vertex b is 2. A path from the vertex b to the vertex e consists of edges $< bd, da, ae >$. This graph is connected and has only one connected component. This graph is simple.

Incidence An edge $\{u, v\}$ is *incident* to u and v.

Degree of vertex in undirected graph is the number of edges incident to it.

In (Out) degree of a vertex in directed graph is the number of edges entering (or leaving) it.

Path A *path* from u to v is a sequence of vertices $< u = v_0, v_1, v_2, \cdots, v_k = v >$ such that $(v_i, v_{i+1}) \in E$ (or $\{v_i, v_{i+1}\} \in E$). A path is simple if all the vertices are distinct. Furthermore,

- We say that v is reachable from u
- The *length* of the path is k
- It is a *cycle* if $u = v$

Connected An undirected graph is connected if every pair of vertices are joined by a path.

Component The connected components are the equivalence classes of the vertices under the "reachability" relation.

Strongly Connected A directed graph is *strongly connected* if every pair of vertices are reachable from each other. See Figure 3.2 for an illustration.

Strongly connected components The *strongly connected components* are the equivalence classes of the vertices under the "mutual reachability" relation.

Simple connected undirected graph An undirected connected graph is called simple, if between every pair of vertices there is at most one edge, and no vertex contains a self loop (i.e. a vertex connected to itself by an edge).
Remark: In these notes, a graph specified without any qualifications is an undirected, connected, and a simple graph.

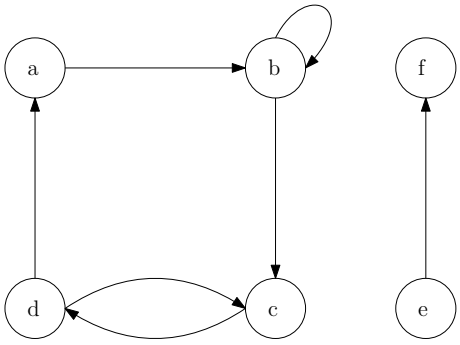


Figure 3.2: An example of a directed graph. This graph is not strongly connected since there is no way to reach from the vertex labelled a to the vertex labelled e. The strongly connected components are {{a,b,c,d},{e},{f}}.

Complete Graphs An undirected graph is called a complete graph, if every pair of (distinct) vertices are joined by an edge. Examples include K_1 (just a single vertex), K_2 (a pair of vertices joined by an edge), K_3 (a triangle), K_5 (graph on five vertices), ... K_5 is the smallest (in terms of vertices) non-planar graph (i.e. no matter how one draws it in the plane, there is a crossing). See Figure 3.3.

Bipartite Graphs A graph is called bipartite if the vertex set V can be partitioned into two subsets $S \cup T = V$, such that for any edge $\{a, b\}$, $a \in S$ and $b \in T$. A bipartite graph is complete if every vertex in S is connected to each vertex in T by an edge. For example, K_{mn} refers to a complete bipartite graph consisting of vertices $V = S \cup T$, where $|S| = m$ and $|T| = n$. Interestingly $K_{3,3}$ is the smallest graph (in terms of edges) which is non-planar.

Kuratowski's Theorem A graph is planar if and only if it has no subgraph homeomorphic to K_5 or $K_{3,3}$. [4] Two graphs are homeomorphic if both can be obtained from the same graph by a sequence of subdivisions of edges (insertion of a vertex on an edge). For example any two cycles are homeomorphic.

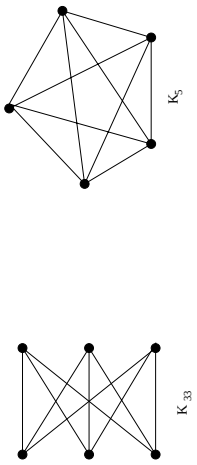


Figure 3.3: K_5 and K_{33}.

[4] This is one of the fundamental theorems in Graph Theory.

3.2 *How to represent graphs in a computer?*

There are two standard ways of representing graphs in computers: Adjacency list and Adjacency Matrix. Let $G = (V, E)$ be the graph under consideration (assume that it is undirected - for directed the same representation works as well).

In the *adjacency matrix* representation of a graph $G = (V, E)$, we form a $|V| \times |V|$ matrix A of 0s and 1s, where the $A[i, j]$-th entry is 1 if and only if there is an edge from the vertex v_i to the vertex v_j. Formally,

$$A[i, j] = \begin{cases} 1 & v_i v_j \in E \\ 0 & v_i v_j \notin E \end{cases}$$

It is easy to see that this matrix will be symmetric for undirected graphs. Also given a pair of vertices v_i and v_j, it takes constant time to check whether there is an edge joining them by inspecting the ij-th entry in the matrix A. Moreover, this representation is independent of the number of edges in G. The main drawback is that this representation requires $O(|V|^2)$ memory space whereas the graph G may have very few edges (i.e., its sparse). Just for fun and to get some insight, try to see what it means by taking products $A \times A$ or $A \times A \times A$,..., where the '.' refers to the boolean AND and '+' refers to the boolean OR? Try it for small graphs. We will come back to that later.

The other most common representation is the *adjacency list* representation. The adjacency list for a vertex v is a list of all vertices w that are adjacent to v. To represent the graph we have in all $|V|$ lists, one for each vertex. This representation requires optimal storage, i.e. $O(|V| + |E|)$. But to check whether the two vertices v and w are connected, we need to check in lists of v (or w) whether the vertex w (resp. v) is present. Searching in a list requires, in the worst case, time proportional to the size of the list. Hence, we can determine if $vw \in E$ in time $O(\min\{\text{degree}(v), \text{degree}(w)\})$.

Try to observe advantages and disadvantages of both representations.

3.3 *Graph Traversal*

Once we have a graph, represented inside the computer, what do we do with it? When we go to a new city, what is the best way to explore all the bars? What is the best way to search for a particular web page just following the hyperlinks (assume no search engine and assume that we can navigate from any web page to any other web page)? How to solve those maze problems?

- There are two standard and simple ways of traversing all vertices/edges in a graph in a systematic way
  - Breadth-first search (bfs)
  - Depth-first search (dfs)
- They are used in many fundamental algorithms as a preprocessing step.

3.3.1 *Breadth-first search (BFS)*

- Main idea of breadth-first search is (see Figure 3.4):
  - Start at a source vertex and visit:
    - All vertices at distance 1 (i.e. vertices that are neighbors of source),
    - Followed by all vertices at distance 2 (neighbors of neighbors of source),
    - Followed by all vertices at distance 3 (neighbors of neighbors of neighbors of source),
    - ⋮

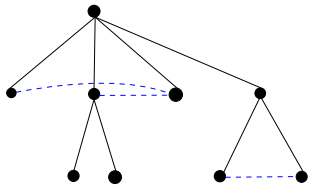


Figure 3.4: Illustration of BFS. Solid edges are tree edges and dashed edges are non-tree edges.

- BFS corresponds to computing *shortest path* distance (number of edges) from s to all other vertices in the graph.

See Algorithm 3.1 for details. It is easy to see that Algorithm 3.1 runs in $O(|V| + |E|)$ time since each vertex is inserted (enqueued)

Algorithm 3.1: Breadth-First Search
Input: Graph $G = (V, E)$ and a source vertex $v \in V$
Output: A breadth-first search tree T rooted at v. Each vertex u stores its parent $p(u)$ in T.

```
1  Initialize a Queue Q of vertices, maintained in
     First-In-First-Out order.
2  T, Q ← ∅;
3  mark[v] ← visited;
4  Q ← v;
5  while Q ≠ ∅ do
6      x ← FRONT(Q);
7      REMOVE(x, Q); // Remove x from Q.
8      foreach y ∈ V adjacent to x do
9          if mark[y] = unvisited then
10             mark[y] ← visited;
11             Insert y at the END of Q;
12             Insert the directed edge (x, y) in the tree T, where
                 p(y) ← x;
13         end
14     end
15 end
```

once in the queue Q and each edge $\{x, y\}$ is explored twice, once when the vertex x is dequeued from Q and once when the vertex y is dequeued. Insertion and Deletion of a vertex in Q can be achieved in constant time since Q is a FIFO Queue, and can be maintained as a doubly-connected list. Also the adjacency list representation will suffice and the correctness is left as an assignment problem. We can summarize this as follows:

Theorem 3.3.1 *Let $G = (V, E)$ be a simple graph. A breadth-first search traversal of G, and its corresponding tree, can be computed in $O(|V| + |E|)$ time.*

3.4 Topological sort and DFS

3.4.1 Topological Sort

Let $G = (V, E)$ be a directed acyclic graph (DAG) on the vertex set V with directed edge set E. In a DAG, there are no directed cycles. But, between a pair of nodes, there may be multiple directed paths. A *topological sort* of a DAG is a linear ordering of all its vertices such that if G contains a directed edge (u, v), then u appears before v in the ordering. One can think of this process as assigning a number $f : V \to \{1, \cdots, |V|\}$ to each vertex such that for every directed edge (u, v), $f(u) < f(v)$ (see Figure 3.5).

We will sketch two algorithms, first a slower one followed by an optimal one. You need to verify the correctness as well as the complexities of both of them. Try to decide yourself what should be a suitable graph representation for these algorithms.

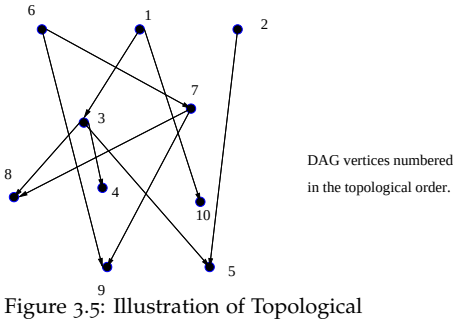


Figure 3.5: Illustration of Topological Sort

Algorithm 1: Topological sort in $O(|V|^2)$ time.

Step 1: Start from any vertex and follow edges backwards until a vertex v is found, such that v has no incoming edges.

Step 2: Make v the next vertex in the total order.

Step 3: Delete v and all of its outgoing edges.

Step 4: If the graph is non-empty, go to Step 1.

Observe that in Step 1 we will find a vertex v, having no incoming edges, as DAGs are acyclic and have a finite number of vertices. Moreover a vertex is assigned a number when it has no incoming edges. This should be sufficient to prove that the generated linear ordering of vertices satisfy the requirements of topological sort.

Algorithm 2: Topological sort in $O(|V| + |E|)$ time using adjacency list representation, where for each vertex maintain a separate list of incoming edges and outgoing edges.

Step 1: Form a queue Q of vertices which have no incoming edges.

Step 2: Pick a vertex v from Q, and make v the next vertex in the order.

Step 3: Delete v from Q and delete all of its outgoing edges. Let (v, w) be an outgoing edge. If the list of incoming edges of w becomes empty then insert w at the end of the queue Q.

Step 4: If Q is not empty then GOTO Step 2.

Which invariant(s) are maintained by Algorithm 2? Why is it correct? Why does it run in $O(|V| + |E|)$ time?

3.4.2 *Depth First Search*

Depth First Search (DFS) is another way of exploring a graph. Like BFS, DFS traversal will take linear time, will produce a DFS spanning tree and this tree will possess very interesting, useful and beautiful properties.

Assume that we have an undirected connected simple graph $G = (V, E)$. Informally DFS on G performs the following steps:

1. Select a vertex v of G which is initially unvisited.
2. Make v visited.
3. Each unvisited vertex adjacent to v is searched in-turn using DFS recursively.

DFS partitions the edges in G into two sets, the set of DFS spanning tree edges, say T, and the set of back edges, say B, where $E = T \cup B$, and $T \cap B = \emptyset$. Next we formally describe the algorithm of Aho, Hopcroft, Ullman [5] for DFS. Each vertex in G will be assigned a DFS-number, i.e., the order in which they are first visited in the DFS (see Figure 3.6).

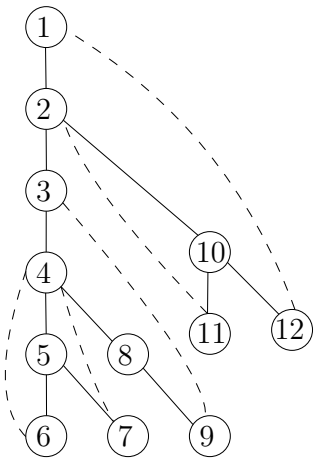


Figure 3.6: Illustration of dfs and low-numbers. Solid edges are tree edges and dashed edges are back edges. $low(6) = 4$, $low(4) = 3$, $low(10) = 1$, $low(3) = 3.$, $low(2) = 1$.

[5] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974

DFS Algorithm

Input: A (undirected simple connected) graph $G = (V, E)$, represented by adjacency list $L[v]$ for each vertex $v \in V$.

Output: Partition of $E = T \cup B$. Tree edges are given as directed edges from a child to its parent. All edges not in T are considered to be in B. DFS-number, an integer in the range $1..|V|$, assigned to each vertex.

1. $T := \emptyset$; COUNT:=1;
2. for all $v \in V$ do mark v as *unvisited*;
3. while there exists an unvisited vertex do SEARCH(v)

procedure SEARCH(v)

1. mark v as *visited*;
2. DSF-number[v]:=COUNT;
3. COUNT:=COUNT+1;
4. for each vertex w on $L[v]$ do
   if w is unvisited then
   (a) add (w, v) to T; /*edge (w, v) is a DFS tree edge */
   (b) SEARCH(w); /* Recursive call */

Complexity Analysis: Why does the above algorithm runs in $O(|V| + |E|)$ time?
We call the procedure SEARCH(v), $|V|$ times, once for each vertex. The total running time of SEARCH(v), exclusive of the recursive calls, is proportional to the degree of v. Hence the total time complexity is $O(|V| + \sum_{v \in V}(degree(v))) = O(|V| + |E|)$.

Property of Back edges: If $\{w, v\} \in B$ is a back edge, then either w is an ancestor of v or v is an ancestor of w in the DFS tree T. Why? Suppose, without loss of generality, v has a lower DFS-number than w, i.e., the vertex v is visited before the vertex w. Therefore, when SEARCH(v) is invoked, the vertex w is labeled *unvisited*. All the *unvisited* vertices visited by SEARCH(v) will become descendants of v in the DFS tree. Therefore, w will become descendant of v, since $w \in L[v]$ and each vertex in $L[v]$ is looked at while executing SEARCH(v).

3.4.3 *Computation of $low(v)$*

We introduce a quantity, called $low(v)$, for each vertex $v \in V$ with respect to the DFS tree T and the back edges B. This quantity will be used in checking whether a graph is biconnected and finding its biconnected components. We will deal with biconnectivity in the next section.

Let us first define $low(v)$. Relabel the vertices of G by their DFS-numbers. For each vertex $v \in V$, define $low(v)$ as follows:

$low(v) = MIN(\{v\} \cup \{w|$there exists a back edge $(x, w) \in B$ such that x is a descendant of v and w is an ancestor of v in the DFS tree$\})$

Intuitively, $low(v)$ is trying to capture the following. Consider the subtree T_v of the DFS-tree T, rooted at the vertex v. What is the vertex closest to the root of T that can be reached by using back edges emerging in T_v, and going to the ancestors of v? If there are no back edges going out of T_v, then $low(v) = v$; otherwise it is the minimum (i.e. closest to the root) among the set of ancestors of v, which are joined by back edges from the vertices in T_v.

To compute $low(v)$, we will compute three quantities. These quantities can be computed by simple modification to the DFS algorithm. The three quantities are

1. $w = v$; i.e. the case when there are no back edges going out of the subtree T_v.

2. $w = low(c)$ and c is a child of v; i.e. the case when $low(v)$ is the same as low value of one of its children.

3. (v, w) is a back edge in B; i.e. the back edges associated to vertex v itself.

Then, the $low(v)$ value is given by

$low(v) = MIN(\{v\} \cup \{low(c)|$ c is a child of $v\} \cup \{w|(v, w) \in B\})$.

The modified SEARCH(v) procedure that computes the low values is as follows:

```
procedure SEARCH(v)

1. mark v as visited;

2. DSF-number[v]:=COUNT;

3. COUNT:=COUNT+1;

4. low(v):=DFS-number[v]; /* low(v) is at least equal to the DFS-
   number of v */
```

5. for each vertex w on $L[v]$ do
   if w is unvisited then

   (a) add (w, v) to T; /*edge (w, v) is a DFS tree edge */

   (b) SEARCH(w); /* Recursive call */

   (c) $low(v) := \min(low(v), low(w))$ /*Compare the low value of v with its child w */

   else if w is not the parent of v then
   $low(v) := \min(low(v), \text{DFS-number}[w])$; /* (v, w) is a back edge */

Given that the DFS algorithm runs in $O(|V| + |E|)$ time, it is easy to see that this algorithm runs within the same time complexity.

3.5 Biconnectivity

3.5.1 Equivalence Relation

Before we talk about biconnectivity, we need to recall what is an equivalence relation.

Relation Let A and B be finite sets. A binary relation R from A to B is a subset of the cross product of A and B, i.e. $R \subseteq A \times B$. A relation on a set A is a relation from A to A.

Another example of an equivalence relation:

Let R be a relation on the set of integers such that $(a, b) \in R$ if and only if $a = b$ or $a = -b$. The equivalence class of integer 4 will be $[4] = \{4, -4\}$. Similarly, $[7] = \{-7, 7\}$. Observe that since $(4, -4) \in R$, then $[4] = [-4] = \{4, -4\}$. Also observe that the set of integers can be partitioned by R as follows: $\{(-1, 1), (0), (-2, 2), (-3, 3), ...\}$.

Example of an Equivalence Relation: Let $A = \{1, 2, 3, 4\}$. Let $R = \{(a, b) | a \text{ divides } b, \text{ where } a, b \in A\}$, i.e. $R = \{(1, 1), (2, 2), (3, 3), (4, 4), (1, 2), (1, 3), (1, 4), (2, 4))\}$.

Reflexive A relation R on A is reflexive if $(a, a) \in R$ for every element $a \in A$. The relation in the divide example is reflexive.

Symmetric A relation R on A is called symmetric if $(b, a) \in R$ whenever $(a, b) \in R$, where $a, b \in A$. The relation in the divide example is not symmetric.

Transitive A relation R on A is called transitive if whenever $(a, b) \in R$ and $(b, c) \in R$, then $(a, c) \in R$, for $a, b, c \in A$. The relation in the divide example is transitive.

Equivalence Relation A relation on a set A is an equivalence relation if it is reflexive, symmetric, and transitive.

Equivalence Classes Let R be an equivalence relation on a set A. The set of all elements that are related to an element $a \in A$ is called the equivalence class of a, denoted by $[a]$.

Property of Equivalence Classes Let R be an equivalence relation on A. Then if $(a, b) \in R$, then $[a] = [b]$.

Partition of A Let R be an equivalence relation on A. Then the equivalence classes of R form a partition of A.

There are many books in Discrete Mathematics discussing equivalence relations, for example, see Rosen [6].

[6] Kenneth H. Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill Higher Education, 5th edition, 2002

3.5.2 *Biconnectivity*

Most of the material in this section is from Kozen [7] and Aho, Hopcroft and Ullman [8]. Assume that the graph $G = (V, E)$ is undirected and connected. We start with definitions.

[7] D. Kozen. *The design and analysis of algorithms*. Springer, 1992

[8] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974

Articulation vertex A vertex $v \in V$ is called an articulation vertex, if its removal disconnects the graph. Equivalently, vertex v is an articulation vertex if there exists vertices a and b, so that every path between a and b goes through v and a, b, and v are all distinct.

Biconnected A connected graph is biconnected if any pair of distinct vertices lie on a simple cycle (one with no repeated vertices). Equivalently, for every distinct triple of vertices v, a, and b, there exists a path between a and b not containing v. Observe that G is biconnected if and only if it has no articulation vertices. Note that a graph just consisting of a single edge (two vertices joined by an edge) is biconnected!

Relation on edges For edges $e, e' \in E$, define that the two edges are equivalent, $e \equiv e'$, if e and e' lie on a simple cycle.

Lemma 3.5.1 *The relation $\equiv$ defined above is an equivalence relation.*

Proof. To prove that it is an equivalence relation, we need to show that it is reflexive, symmetric and transitive.

Reflexive: Obviously $e \equiv e$, $\forall e \in E$, since an edge by itself lies on a cycle.

Symmetric: If $e \equiv e'$, then $e' \equiv e$, since they are on the same cycle.

Transitivity: Suppose that $e \equiv e'$ and $e' \equiv e''$. We want to show that $e \equiv e''$. Say e and e' are on a simple cycle X and e' and e'' are on a simple cycle Y. If e'' is also on the cycle X then we are done because e is on that cycle too. Otherwise, follow the edges of Y from one end of e'' until it hits X. Call that intersection vertex p. Do the same for the other end of e'', and call that intersection vertex q. Now p and q are

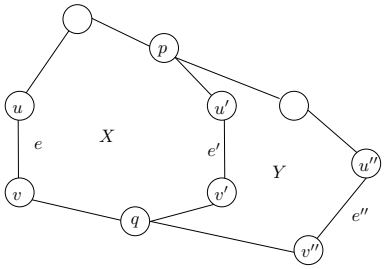


Figure 3.7: Illustration of Transitivity. The cycle c'' contains e and e''.

distinct and there are two disjoint paths between them using edges of X. One of these paths contains edge e. Combing this path with the ones of Y we used to get from e'' to X gives us a simple cycle containing both e and e''. [9]

[9] Thanks to Danny Sleator for providing this proof.

■

Now we have an equivalence relation. What are the equivalence classes of this relation with respect to the edge set of G.

Biconnected Components The equivalence classes of the relation $\equiv$ are the biconnected components of G.

Now we discuss critical lemmas that relate articulation vertices, biconnected components, DFS number and low values. These lemmas are '*if and only if*' type - or '*necessary and sufficient*' type. Such types of lemmas are extremely useful, especially in computer science, since they provide a complete characterization of the object/structure under consideration, and often lead to an algorithm.

Lemma 3.5.2 *A vertex v is an articulation vertex if and only if it is contained in at least two distinct biconnected components.*

Proof. Suppose v is an articulation vertex. Then its removal disconnects G. That means that there are two vertices a and b neighboring v so that each path between a and b goes through v. See Figure 3.8. Then the edges (a, v) and (v, b) cannot lie on a simple cycle, and hence they belong to two distinct biconnected components. This implies that v is contained in at least two distinct biconnected components.

Now suppose that v is contained in two distinct biconnected components, and is adjacent to vertices a and b in these components, respectively. Then $(va) \not\equiv (vb)$. Then all paths between a and b goes through v, and hence removing v disconnects G. So v is an articulation vertex. ■

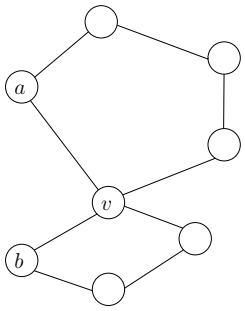


Figure 3.8: An articulation vertex in two distinct biconnected components.

Lemma 3.5.3 *Let (uv) and (vw) be two adjacent tree edges in a DFS tree T of G. Then $(uv) \equiv (vw)$ if and only if there exists a back edge from some descendant of w to some ancestor of u.*

Proof. Recall that the descendants of w are w and all the vertices in the subtree rooted at w. Ancestors of u include u and all vertices on the path from u to the root of the DFS tree.

If there exists a backedge from some descendant w' of w to some ancestor u' of u, then $(uv) \equiv (vw)$, since there is a simple cycle consisting of the tree path between u' and w' and the backedge $w'u'$. See Figure 3.9.

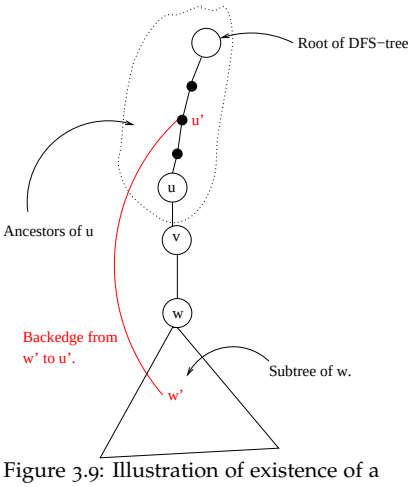


Figure 3.9: Illustration of existence of a back edge

Suppose $(uv) \equiv (vw)$. By definition, there is a simple cycle that contains both of them. The edges (uv) and (vw) must appear in this order in the cycle, as the vertex v appears exactly once on the cycle. This implies that there is an edge (actually a backedge) from some vertex w' in the subtree rooted at w to some ancestor u' of u. (This ancestor could be the vertex u or a vertex on the path from u to the root of the DFS tree.) ∎

Lemma 3.5.4 *Vertex v is an articulation vertex if and only if either*
(a) v is the root of the DFS tree and has more than one child.
(b) v is not the root, and for some some child w of v there is no backedge between any descendant of w (including w) and a proper ancestor of v.

Part (a) is easy to prove and part (b) follows from the previous lemma. The modifications to the DFS procedure to compute the biconnected components are as follows:

See Figure 3.10 for an illustration for the biconnected components computed by the following search procedure.

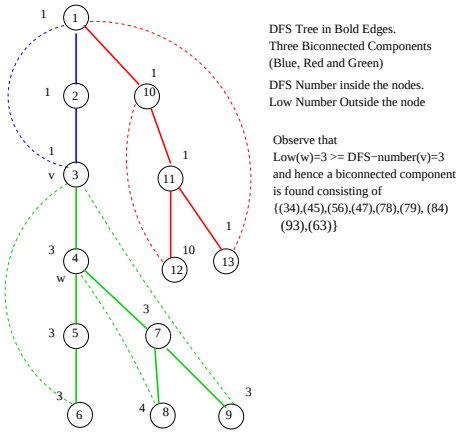


Figure 3.10: Illustration of biconnected components

procedure SEARCH(v)

1. mark v as *visited*;
2. DSF-number[v]:= COUNT;
3. COUNT:= COUNT+1;
4. $low(v)$:= DFS-number[v]; /* $low(v)$ is at least equal to the DFS-number of v */
5. for each vertex w on $L[v]$ do
   if w is unvisited then
   (a) add (w, v) to T; /*edge (w, v) is a DFS tree edge */
   (b) SEARCH(w); /* Recursive call */
   (c) If $low(w) \geq$ DFS-number$[v]$ then a biconnected component has been found;
   (d) $low(v) := \min(low(v), low(w))$ /*Compare the low value of v with its child w */

   else if w is not the parent of v then
   $low(v) := \min(low(v)$,DFS-number$[w])$; /* (v, w) is a back edge */

end

Why does the above algorithm compute biconnected components?

Actually we will need a STACK to figure out the edges in a biconnected component! How do we do that? When a vertex w is encountered in the SEARCH procedure places the edge (v, w) on the STACK if it is not there. After discovering a pair (v, w), such that w is a child of v and $low(w) \geq$ DFS-number$[v]$, POP all the edges from the STACK up to and including (v, w). These edges form a biconnected component. This extra step can be accomplished in linear time as well. To prove that the above SEARCH procedure indeed computes the biconnected components, we need to argue by induction on the number of biconnected components.

3.6 Exercises

3.1 *This problem is related to the representation of graphs. Assume that the number of edges in the graph $G = (V, E)$ is small, i.e., it is a sparse graph. In the adjacency matrix representation of G, the normal tendency is to first initialize the matrix, requiring $O(|V|^2)$ time. Is there any way we can initialize the adjacency matrix in time proportional to $O(|E|)$ and still have $O(1)$ adjacency test?*

3.2 *Provide a clear, concise, and complete proof for the correctness of the BFS algorithm.*

3.3 *Let G=(V,E) be a directed acyclic graph with two designated vertices, the start and the destination vertex. Write an algorithm to find a set of paths from the start vertex to the destination vertex such that (a) no vertex other than the start or the destination vertex is common to two paths. (b) no additional path can be added to the set and still satisfy the condition (a). Note that there may be many sets of paths satisfying the above conditions. You are not required to find the set with the most paths but any set satisfying the above conditions. Your algorithm should run in $O(|V| + |E|)$ time. State the algorithm, its correctness and analyze the complexity.*

3.4 *Let $G = (V, E)$ be a directed acyclic graph. We say that G is* semi-connected *if for every pair of distinct vertices $u, v \in V$, we have that there is a directed path from u to v or there is a directed path from v to u in G. Given G in the adjacency list representation, design an algorithm running in $O(|V| + |E|)$ time to determine whether G is semi-connected. (Hint: First, construct examples of directed-acyclic graphs on four vertices that are semi-connected and that aren't. Determine what property (with respect to their linear order) distinguishes them.)*

3.5 *Clearly describe the modifications you need to make in the SEARCH procedure to compute and output the biconnected components. Prove that your algorithm is correct, i.e. it computes all the biconnected components.*

3.6 *How can we find in $O(|V| + |E|)$, whether a graph $G = (V, E)$ is a bipartite graph (Hint: Use BFS).*

3.7 *An Euler circuit for an undirected graph is a path that starts and ends at the same vertex and uses each edge exactly once (vertices might be repeated). A connected, undirected graph G has an Euler circuit if and only if every vertex is of even degree. Give an $O(|E|)$ algorithm to find an Euler circuit in G, if it exists.*

3.8 *Assume that you are given n positive integers, $d_1 \geq d_2 \geq \cdots \geq d_n$, each greater than 0. You need to design an algorithm to test whether these integers form the degrees of an n vertex simple undirected graph $G = (V, E)$ (Think of a greedy algorithm.)*

3.9 *Show that in a depth-first search, if we output a left parenthesis '(' when a node is accessed for the first time and output a right parenthesis ')' when a node is accessed for the last time, then resulting parenthesization (or bracketing sequence) is proper. Each left '(' is properly matched with a right ')'.*

3.10 *Let $G = (V, E)$ be a simple undirected graph. Provide an algorithm running in $O(|V| + |E|)$ time, which outputs whether G contains a cycle or not. If it contains a cycle - then it needs to output at least one cycle. What graph representation you have used for your algorithm. Justify why you used that and remember to link this justification with your complexity analysis.*

3.11 *Typically departments in universities (like Carleton) offer many courses, but to register in a course, one needs to have completed all the required prerequisite courses. We can easily model this relationship as a directed graph, where each course is a vertex, and a directed edge from course u to v if and only if u is a prerequisite course for taking w. It should be clear that this graph should not contain any directed cycles (otherwise we won't graduate!). (For example, if COMP 1405 and COMP 1805 are required for taking COMP 2402, and COMP 2402 is required for taking COMP 3804, we will have directed edges from vertices corresponding to COMP 1805 and COMP 1405 to COMP 2402, and a directed edge from COMP 2402 to COMP 3804.) Given a directed graph $G = (V, E)$ in adjacency list representation, representing the courses and their prerequisites, your task is to compute minimum number of terms one needs to spend in the department to complete the degree, where you can assume that you can do any number of courses in any term, provided that the prerequisite conditions are met.*

3.12 *Let s and t be two specific vertices of an undirected connected simple graph $G = (V, E)$ on $n = |V|$ vertices, where any path between s and t in G consists of at least $n/2 + 2$ vertices. Show that there is a vertex $v \in V$,*

$v \neq s$ and $v \neq t$, such that any path from s to t passes through v. Also, provide an algorithm, running in $O(|V| + |E|)$ time, for identifying such a vertex v for a given pair of vertices $s, t \in V$. (Note that by removing v from G, we disconnect s and t.)

3.13 *Assume that $G = (V, E)$ is biconnected. Our task is to identify those edges $E' \subseteq E$, so that if we remove any edge $e \in E'$ from G, then the resulting graph is not biconnected. Intuitively, edges in E' are essential in maintaining the biconnectivity of G. It is fairly straightforward to test whether an edge $e \in E$ is critical, by just removing e from G and running the biconnectivity algorithm to test whether the resulting graph is biconnected. A question worth trying, but is likely to be nontrivial, is to compute the set $E' \subseteq E$ in $o(|E|(|V| + |E|))$ time.*

3.14 *Consider the following modified pseudo-code.*

Modified DFS Algorithm
Input: A graph $G = (V, E)$, represented by adjacency list $L[v]$ for each vertex $v \in V$.
Output: A pair of integers $(pre[v], post[v])$ assigned to each vertex v.

1. *$Clock := 1$;*
2. *for all $v \in V$ do mark v as* unvisited*;*
3. *While there exists an unvisited vertex v do SEARCH(v)*

procedure SEARCH(v)

1. mark v as *visited*;
2. $pre[v] := Clock$;
3. $Clock := Clock + 1$;
4. for each vertex w on $L[v]$ do
   if w is unvisited then SEARCH(w);
5. $post[v] := Clock$;
6. $Clock := Clock + 1$;

Answer the following questions:

1. *Suppose $G = (V, E)$ is undirected graph. Show that for any pair of nodes u and v in G, the two intervals $[pre[u], post[u]]$ and $[pre[v], post[v]]$ are either disjoint or one interval contains the other.*

2. *Execute the modified dfs algorithm on the directed graph in Figure 3.2.*

3. *Suppose $G = (V, E)$ is directed graph. Show that for any pair of nodes u and v in G, the two intervals $[pre[u], post[u]]$ and $[pre[v], post[v]]$ are either disjoint or one interval contains the other. Moreover, show that if for a directed edge $(u, v) \in E$, $pre[v] < pre[u] < post[u] < post[v]$, then there is a directed cycle in G.*

4. *Call an edge $e = (u, v)$ of a directed graph a* back edge *if $pre[v] < pre[u] < post[u] < post[v]$. Show that a directed graph has a directed cycle if and only if the modified dfs algorithm reveals a back edge.*

5. *Design an algorithm that determines whether a directed graph $G = (V, E)$ is an acyclic graph (i.e., it doesn't contain a directed cycle). Your algorithm must run in $O(|V| + |E|)$ time.*

6. *Let $G = (V, E)$ be a directed acyclic graph. Show that for any directed edge $e = (u, v) \in E$, $post[u] > post[v]$.*

7. *All vertices with no incoming edges in a directed acyclic graphs are called the* source *vertices, and all the vertices that have no outgoing edges are called the* sink *vertices. In any directed acyclic graph, can you say what property the vertex with the largest post number satisfies? the vertex with the smallest post number? Does the ordering of the vertices with respect to decreasing post number results in a linear order?*

3.15 *A directed graph $G = (V, E)$ is said to be* strongly connected *if every pair of vertices is joined by a directed path. That is, for any pair of vertices $u, v \in V$, there is a directed path from u to v and there is a directed path from v to u. If G is not strongly connected, then its vertices can be partitioned into strongly connected components. Answer the following:*

1. *What are the strongly connected components of a directed acyclic graph?*

2. *Identify strongly connected components of the graph in Figure 3.11.*

3. *Show that if A and B are two strongly connected components in G and there is an edge from some vertex in A to some vertex in B, then the highest post number in A is bigger than the highest post number in B.*

4. *Is it possible to linearise the strongly connected components of a directed graph $G = (V, E)$ with respect to the decreasing order of the highest post number of its components?*

5. *Is it possible to identify a vertex in* source *strongly connected component? Is it possible to identify a vertex in* sink *strongly connected component?*

6. *Can the sink strongly connected component of a directed graph $G = (V, E)$ can be identified in $O(|V| + |E|)$ time?*

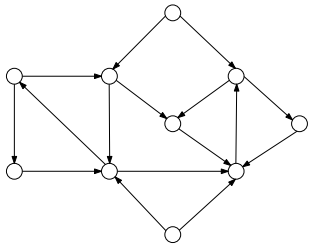


Figure 3.11: A directed graph.

3.16 *Given a directed graph $G = (V, E)$, where each vertex has a distinct integer label. For each vertex v, define $R(v)$ to be the set of all vertices $w \in V$ for which there is a directed path from v to w in G. Furthermore, for each vertex $v \in V$, define* $\mathsf{MinLabel}(v)$ *to be the vertex with the minimum label in the set $R(v)$. Provide an algorithm, running in $O(|V| + |E|)$ time, that computes* $\mathsf{MinLabel}(v)$ *for all vertices $v \in V$.*

3.17 *Let $G = (V, E)$ be a directed acyclic graph. Is it possible to find a (directed) Hamiltonian path in G, i.e. a directed path that touches each vertex exactly once, in $O(|V| + |E|)$ time.*

3.18 *In the summer vacation, you decided to travel to various communities in Northern Canada by your favourite ATV (All-Terrain Vehicle). Each of the communities you want to visit is represented as a vertex in your travel graph (a total of $|V|$ communities). Moreover, you are provided with distances between all pairs of communities. Think of your input graph as a complete graph (i.e. every pair of vertices are joined by an edge), and the weight of an edge, say $e = (uv)$ is the distance between the community u and v. Since this is in the far North, and the routes between communities are not used that often, the gas stations are only located in communities (there are no gas stations outside a community). Furthermore, we can assume that each community has at least one gas station. Once you fill-up the tank of your ATV, it has an upper limit, say of Δ kilometres, which it can travel, and to travel any further, it needs to fill up (which means at that point it needs to be in a community!). Answer the following questions:*

1. *First design a method, running in $O(|V| + |E|)$ time, which can answer whether there is some path which your ATV can take so that you can travel between two particular communities, say s and t. If the distance between s and t is at most Δ, you can travel directly without refuelling. Otherwise, you can travel between s and t, provided there are communities where we can refuel and proceed.*

2. *Design an algorithm running in $O(|E| \log |V|)$ time to determine the smallest value of Δ, which will enable you to travel from s to t. (Please present Pseudocode, correctness, analysis) and use the algorithms discussed in the class/book as black boxes).*

3.19 *Consider a connected graph $G = (V, E)$ where each edge has a non-zero positive weight. Furthermore, assume that all edge weights are distinct. Show that for each vertex $v \in V$, the edge incident to v with minimum weight belongs to a Minimum Spanning Tree (MST). Can you use this to devise an algorithm for MST? Note that the above step identifies at least $|V|/2$ edges in MST. Now we can collapse these edges by* identifying *the vertices and then recursively applying the same technique - the graph in the next step has at most half of the vertices that you started with - and so*

on. The recursion terminates when we are left with a single vertex. At that point, we would have collected $|V| - 1$ edges that are in an MST.

Note that for an edge $e = uv$ in the graph $G = (V, E)$, identifying *vertex u with v or* collapsing *e is the following operation: Replace the vertices u and v by a new vertex, say u'. Remove the edge between u and v. If there was an edge from u (respectively, v) to any vertex w ($w \neq u$ and $w \neq v$), then we add an edge (with the same weight as of edge uw (respectively, vw)), between the vertices u' and w. This transforms graph G to a new graph $G' = (V', E')$, where $|V'| < |V|$ and $|E'| < |E|$. Note that G' may be a multigraph (i.e., between a pair of vertices, there may be more than one edge). For example, if uv, uw, and vw are edges in G, then G' will have two edges between u' and w when we identify u with v. We can transform G' to a simple graph by keeping the edge with the lower weight among uw and vw as the representative for $u'w$ for the computation of MST.*

3.20 *Let $G = (V, E)$ be an edge-weighted directed connected graph. Assume that each edge weight is positive and distinct. Let $s, t \in V$ be two specific vertices in G and let $\pi(s, t)$ be a shortest path in G between s and t. Suppose we modify the weight of each edge of G to be the square root of its original weight. Will $\pi(s, t)$ continue to be a shortest path between s and t in the modified graph? What if we increase the weight of each edge to square of its value? Will $\pi(s, t)$ continue to be a shortest path between s and t in the modified graph?*

3.21 *Let $G = (V, E)$ be a directed graph given in the adjacent matrix representation. Define the square of G to be the graph $G' = (V', E')$ where $V' = V$ and $(u, v) \in E'$ if and only if there is a directed path consisting at most two edges between u and v in G. Given G show how you can compute G' efficiently.*

3.22 1. *Show that every simple undirected graph on $n \geq 2$ vertices has at least two vertices of equal degree.*

2. *Show that a tree with $n \geq 2$ vertices have at least two vertices of degree 1.*

3.23 *Let $G = (V, E)$ be a simple, connected, and undirected graph on $n \geq 3$ vertices. Answer the following:*

1. *Suppose we are also given that G is not a complete graph. Show that three vertices always exist u, v, and $w \in V$ such that $uv, vw \in E$ and $uw \notin E$.*

2. *Show that any two longest paths in G have a vertex in common.*

3.24 *Let $G = (V, E)$ be a simple undirected bipartite graph. Let A be its node-edge incidence matrix of dimension $|V| \times |E|$, defined as*

$$A_{ve} = \begin{cases} 1, & \text{if } v \text{ is an endpoint of an edge } e \\ 0, & \text{otherwise} \end{cases}$$

Show that every square submatrix of A has determinant 0 or ± 1.

Note that the rows (respectively, columns) of A corresponds to the vertices (resp., edges) of G. Consider the column corresponding to the edge $e = (uv) \in E$. That column consists of exactly two 1s, one corresponding to the row of u and the other to the row of v, and all the remaining entries are 0.

4
Matrices with Applications to CS

We will focus on

1. Solutions to the system of linear equations $Ax = b$.
2. Row, Column, and Null Spaces.
3. Expressing a square matrix A that has n linearly independent eigenvectors as $A = X\Lambda X^{-1}$. Λ is a diagonal matrix of its eigenvalues and X consists of its eigenvectors as columns.
4. Any real symmetric matrix S can be expressed as $S = Q\Lambda Q^T$, where Q is a collection of orthonormal eigenvectors.
5. A symmetric matrix S is positive definite if all its eigenvalues are > 0.
6. Singular Value Decomposition for any matrix.
   $A = U\Sigma V^T$ and $Av_i = \sigma_i u_i$.
7. Approximating A by a sum of tensor products.

Keywords: Rank, Vector Spaces, Eigenvalues & Eigenvectors, Markov Matrix & Page Rank, Symmetric and Positive Definite Matrices, SVD, principal component analysis, Least square approximations.

4.1 Basics

Let A be a matrix consisting of 3 rows and 3 columns on real numbers. We view each row or column as a vector in $\mathbb{R}^3$. For example, let

$$A = \begin{bmatrix} 2 & 2 & 0 \\ 2 & 4 & 8 \\ 10 & 16 & 24 \end{bmatrix}$$

The three rows are $r_1 = (2,2,0)$, $r_2 = (2,4,8)$, and $r_3 = (10,16,24)$. The three columns are $c_1 = (2,2,10)$, $c_2 = (2,4,16)$, and $c_3 = (0,8,24)$. All of them are vectors in $\mathbb{R}^3$. We further observe that $r_3 = 2r_1 + 3r_2$. (Two typical operations on vectors include scalar multiplication and vector addition. In a scalar multiplication of a vector $v = (2,3,7)$ by a scalar $c = 3$, we obtain $cv = (6,9,21)$. In vector addition of two vectors $u = (1,3,-5)$ and $v = (-6,4,1)$, we obtain $u + v = (-5,7,-4)$). Let us find the row reduced echelon form (RREF) of A by finding its pivots. We will denote the entry in the i-th row and the j-th column of A by a_{ij}. Since $a_{11} \neq 0$, it is the first pivot. Now we take the suitable multiples of r_1 and subtract them from r_2 and r_3 so that the only non-zero entry that remains in the first column is in the first row. This can be achieved by replacing r_2 by $r_2 - r_1$, and r_3 by $r_3 - 5r_1$, and we obtain the following matrix:

$$\begin{bmatrix} 2 & 2 & 0 \\ 0 & 2 & 8 \\ 0 & 6 & 24 \end{bmatrix}$$

Next we find the second pivot. The entry in 2nd row and 2nd column is non-zero, hence that is the pivot. We replace r_3 by $r_3 - 3r_2$ and obtain

$$\begin{bmatrix} 2 & 2 & 0 \\ 0 & 2 & 8 \\ 0 & 0 & 0 \end{bmatrix}$$

The last row only contains zero's. Therefore, A doesn't have a non-zero third pivot. To obtain the RREF, we will like the pivots to be 1, and moreover the sub-matrix consisting of the pivot rows and columns to be the identity matrix. To obtain the RREF form, we divide the first row by 2, the second row by 2, and obtain

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 4 \\ 0 & 0 & 0 \end{bmatrix}$$

We are almost there, except that the sub-matrix formed by the first two rows and first two columns is not an identity matrix. To obtain the RREF, we replace r_1 by $r_1 - r_2$ and obtain

$$R = \begin{bmatrix} 1 & 0 & -4 \\ 0 & 1 & 4 \\ 0 & 0 & 0 \end{bmatrix}$$

Next, we will make several observations on A and its RREF R. (For a deeper understanding on various properties of A and R, refer to the textbook of Gilbert Strang [1].) Since the number of non-zero pivots

[1] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, Wellesley, MA, fifth edition, 2016

is 2, the rank r of A is $r = 2$. Moreover, the dimension of its row space is $r = 2$ (row space is the vector space consisting of all linear combinations of row vectors). The basis vectors of the row space are the rows corresponding to the non-zero pivots in R, i.e., $v_1 = \begin{bmatrix} 1 \\ 0 \\ -4 \end{bmatrix}$ and $v_2 = \begin{bmatrix} 0 \\ 1 \\ 4 \end{bmatrix}$. Similarly, the column space, i.e., the vector space formed by linear combinations of the columns of A has dimension $r = 2$. Its basis vectors are the columns of A corresponding to the non-zero pivots. In our example, it will be the first and the second column of A, i.e., $u_1 = \begin{bmatrix} 2 \\ 2 \\ 10 \end{bmatrix}$ and $u_2 = \begin{bmatrix} 2 \\ 4 \\ 16 \end{bmatrix}$, respectively. Interestingly, now A can be expressed as the sum of its rank 1 components as follows:

$$A = u_1 v_1^T + u_2 v_2^T = \begin{bmatrix} 2 \\ 2 \\ 10 \end{bmatrix} \begin{bmatrix} 1 & 0 & -4 \end{bmatrix} + \begin{bmatrix} 2 \\ 4 \\ 16 \end{bmatrix} \begin{bmatrix} 0 & 1 & 4 \end{bmatrix}$$

Next we discuss briefly the null spaces of A. There is a column null space (usually referred to as the null space) and there is a left null space. The null space of A represents all vectors x such that $Ax = 0$. In other words, what linear combinations of the vectors corresponding to the columns will result in a 0 vector. For our example, the vector $[0, 0, 0]$ is in the null space as $A \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$. Are there any non-zero vectors x in the null space of A? In other words, is there a vector $x = (x_1, x_2, x_3) \in \mathbb{R}^3$, such that $x_1 \begin{bmatrix} 2 \\ 2 \\ 10 \end{bmatrix} + x_2 \begin{bmatrix} 2 \\ 4 \\ 16 \end{bmatrix} + x_3 \begin{bmatrix} 0 \\ 8 \\ 24 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$. We can see that $x_1 = 1$, $x_2 = -1$, and $x_3 = 1/4$, satisfies this condition. Hence, the vector $x = (1, -1, 1/4)$, or any of its scalar multiples, is in the null-space of A. In fact the dimension of the null-space of A is the number of its columns minus its rank. In our case, the dimension of the null-space will be $3 - 2 = 1$. Now for the left null space, we are looking for vectors $y \in \mathbb{R}^3$ such that $A^T y = 0$ (equivalently, $y^T A = 0$). This represents what linear combinations of row vectors can result in a 0 vector. In our example, we have $\begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} 2 & 2 & 0 \\ 2 & 4 & 8 \\ 10 & 16 & 24 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$. Note that the vectors $y = [0, 0, 0]$, and $[2, 3, -1]$ or any of their linear combinations satisfies the left nullity conditions. The dimension of the left null space is the number of rows minus the rank of A, i.e., $3 - 2 = 1$ in our example.

In general, for an $m \times n$ matrix A, we have the following. Assume that its RREF is R and it consists of $r \leq \min\{m, n\}$ non-zero pivots. Then, rank of A is r. The column space is a subspace of $\mathbb{R}^m$ of dimension r, and its basis vectors are the columns of A corresponding to the non-zero pivots in R. The row space is a subspace of R^n of dimension r, and its basis vectors are the rows of R corresponding to the non-zero pivots.

The null-space of A consists of all the vectors $x \in \mathbb{R}^n$ satisfying $Ax = 0$. They form a subspace of dimension $n - r$. Lastly, the left

null space of A, i.e., all the vectors $y \in \mathbb{R}^m$ such that $A^T y = 0$ (or equivalently $yA^T = 0$, and thus the name). For the null space, we looked at which linear combinations of columns yield a zero vector. For the left null-space, we are interested to know which linear combinations of rows yield a zero vector. Its dimension is $m - r$.

The fundamental theorem of linear algebra states that for an $m \times n$ matrix A with its row-reduced echelon form R with rank $r \leq \min\{m, n\}$, we have the following:

1. A and R have the same same row space. Its dimension is r and the basis is the same (e.g., the row vectors of R corresponding to the r non-zero pivots). Row operations to obtain R from A changes rows, but it doesn't change the row space.

2. The column space of A has dimension r. Note that the basis vectors of the column space of A and R may not be the same, but for all $x \in \mathbb{R}^n$, $Ax = 0$ exactly when $Rx = 0$.

3. The number of independent columns of A is the same as the number of independent rows of A.

4. The null space of A has dimension $n - r$. A and R have the same basis. Note that the solutions space of $Ax = 0$ and $Rx = 0$ is the same, as the row operations don't alter the solution.

5. Dimension of the column space plus the dimension of the null space of A equals n (= the dimension of $\mathbb{R}^n$).

6. The left null-space of A has dimension $m - r$.

7. Dimension of the row space plus the dimension of the left null space of A equals m (= the dimension of $\mathbb{R}^m$).

8. Furthermore, we can choose a basis for these vector spaces in such a way that the r vectors forming the basis for the column space are orthonormal, and the $n - r$ vectors forming the basis for the null space are orthonormal and they are also orthogonal to the column basis vectors. Together they form an orthonormal basis for $\mathbb{R}^n$. Analogously, we can choose r orthonormal vectors forming the basis for the row space, and $m - r$ orthonormal vectors forming the basis of the left null-space and together they form an orthonormal basis for $\mathbb{R}^m$.

4.2 *Introduction to Eigenvalues*

Consider a square matrix A of dimension $n \times n$ on real numbers. Let x be a vector of dimension n. Let $A = (a_{ij})$, where $i = 1, \ldots, n$,

$j = 1, \ldots, n$, $a_{ij} \in \mathbb{R}$, and let $x = (x_1, x_2, \ldots, x_n)$, $x_i \in \mathbb{R}$. Consider the product of A and x, and we know that it results in another vector $y = (y_1, y_2, \ldots, y_n)$ of dimension n such that $y_i = \sum_{j=1}^{n} a_{ij}x_j$, for $i = 1, 2, \ldots, n$. We can view A as a function that transforms a vector x to another vector y. We are in particular interested in those vectors x, such that the product Ax results in a vector that is parallel to x. Clearly, if $x = 0$, it is true that $Ax = x = 0$. We are interested to know if there are non-zero vectors x such that $Ax = \lambda x$, for constant λ. Such vectors are called *eigenvectors* and the corresponding constant value λ is called the *eigenvalue*. As it will turn out that there are several applications of eigenvalue-eigenvectors in many fields of Sciences and Engineering. Let us first see a couple of examples.

Example 4.2.1

$$A = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix}$$

Observe that

$$\begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} = 5 \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

and

$$\begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = 1 \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Thus, $\lambda_1 = 5$ and $\lambda_2 = 1$ are the eigenvalues of A and the corresponding eigenvectors are $v_1 = [1, 3]$ and $v_2 = [1, -1]$, respectively, as $Av_1 = \lambda_1 v_1$ and $Av_2 = \lambda_2 v_2$. Note that v_1 is stretched five times when multiplied by A, whereas v_2 is left unchanged.

Example 4.2.2 *Let us consider the same example as above, but now the rows are permuted. Let*

$$B = \begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix}$$

Observe that

$$\begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = 5 \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

and

$$\begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = -1 \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Thus, $\lambda_1 = 5$ and $\lambda_2 = -1$ are the eigenvalues of B and the corresponding eigenvectors are $v_1 = [2, 1]$ and $v_2 = [1, -1]$, respectively, as $Bv_1 = \lambda_1 v_1$ and $Bv_2 = \lambda_2 v_2$. Here v_2 flips its direction when multiplied by B, but its magnitude remains the same.

Let us consider the eigenvalues and eigenvectors of A^2. Since, $Av_i = \lambda_i v_i$. Note that by multiplying by A on both the sides on the left, we have

$$A^2 v_i = A(Av_i) = A(\lambda_i v_i) = \lambda_i (Av_i) = \lambda_i (\lambda_i v_i) = \lambda_i^2 v_i.$$

Thus for A^2, the eigenvectors are the same as that of A, but eigenvalues are squared. In fact for an integer $k > 0$, A^k has the same eigenvectors as A, but the eigenvalues are λ^k.

Let us see how to compute the eigenvalues and eigenvectors of an $n \times n$ matrix A. We are interested to find vectors x (especially the non-zero vectors) such that $Ax = \lambda x$, or equivalently,

$$(A - \lambda I)x = 0, \tag{4.1}$$

where I is an $n \times n$ identity matrix. Note that all the eigenvectors x that satisfy $(A - \lambda I)x = 0$ constitutes the null space of $A - \lambda I$. (Recall from the last section that the null space of a matrix B is the set of all the vectors v such that $Bv = 0$. Clearly $v = 0$ is in this set and this set is closed under vector addition and scalar multiplication. Hence the null space forms a vector space.) If $(A - \lambda I)x = 0$ has a non-zero solution, then the matrix $(A - \lambda I)$ is singular, i.e., it is not invertible. This implies that the determinant of $(A - \lambda I)$, $det(A - \lambda I) = 0$. The equation $det(A - \lambda I) = 0$ results in a polynomial of degree n, and this equation has n (real or complex) roots. The polynomial $det(A - \lambda I)$ is referred to as the *characteristic polynomial*. Note that the roots of the characteristic polynomial may not be distinct (e.g. consider the eigenvalues of identity matrix).

Example 4.2.3 *Consider the matrix A given above. Its characteristic polynomial is given by*

$$det(A - \lambda I) = det \begin{bmatrix} 2-\lambda & 1 \\ 3 & 4-\lambda \end{bmatrix} = (2-\lambda)(4-\lambda) - 3 = \lambda^2 - 6\lambda + 5$$

Roots of $\lambda^2 - 6\lambda + 5 = (\lambda - 5)(\lambda - 1) = 0$ *are* $\lambda_1 = 5$ *and* $\lambda_2 = 1$. *These are the eigenvalues of A and they are distinct.*

Next, let us see how to find the eigenvector v corresponding to an eigenvalue λ. Since v is in the null space of $A - \lambda I$, we solve the system of equations for $(A - \lambda I)v = 0$ to find all the components of v. For the above example, let us find the eigenvector $v_1 = \left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$ corresponding to the eigenvalue $\lambda_1 = 5$.

$$(A - \lambda_1 I)v_1 = \begin{bmatrix} 2-5 & 1 \\ 3 & 4-5 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} -3 & 1 \\ 3 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Observe that the rows in the matrix $A - \lambda_1 I$ are dependent. Now we obtain the equation $-3a + b = 0$ or equivalently $3a = b$, and thus the vector $v_1 = [1, 3]$ (or any of its scalar multiple) is an eigenvector corresponding to $\lambda_1 = 5$. Similarly we can compute that $v_2 = \left[\begin{smallmatrix} 1 \\ -1 \end{smallmatrix}\right]$ is an eigenvector corresponding to $\lambda_2 = 1$ by solving the following:

$$(A - \lambda_2 I)v_2 = \begin{bmatrix} 2-1 & 1 \\ 3 & 4-1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 3 & 3 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

We conclude this section with the following example of rotation matrix.

Example 4.2.4

$$Q = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Note that this matrix rotates any vector by 90° in anticlockwise direction. Since each vector v is rotated, there cannot be any non-zero real vector v such that Qv is parallel to v. The characteristic polynomial of Q is $det(Q - \lambda I) = \lambda^2 + 1$. Note that this does not have real roots and thus the eigenvalues of Q are imaginary numbers $\lambda_1 = i$ and $\lambda_2 = -i$. What about its eigenvectors? For that we solve

$$(Q - \lambda_1 I)v_1 = \begin{bmatrix} -i & -1 \\ 1 & -i \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and obtain that $a - bi = 0$ or $a = bi$ and thus $v_1 = [i, 1]$. It is a complex eigenvector and satisfies $Qv_1 = \lambda_1 v_1$. Similarly, we have $v_2 = [1, i]$ corresponding to $\lambda_2 = -i$. Therefore, even if all the entries in a matrix are real, its eigenvalues and eigenvectors may have complex numbers.

4.3 Diagonalizing Square Matrices

Let A be an $n \times n$ real matrix with n distinct eigenvalues. For such matrices, their corresponding eigenvectors are linearly independent (see exercises). Let $\lambda_1, \ldots, \lambda_n$ be the distinct eigenvalues and let $x_1, \ldots, x_n$ be the corresponding eigenvectors, respectively. Let each $x_i = [x_{i1}, x_{i2}, \ldots, x_{in}]$. Define an eigenvector matrix X, where the ith column of X is the eigenvector x_i, $1 \leq i \leq n$. Formally,

$$X = \begin{bmatrix} x_{11} & x_{21} & \ldots & x_{n1} \\ \vdots & \vdots & \vdots & \vdots \\ x_{1n} & x_{2n} & \ldots & x_{nn} \end{bmatrix}$$

Since eigenvectors are linearly independent, we know that X^{-1} exists. Define a diagonal $n \times n$ matrix Λ whose entries are as follows.

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 \\ 0 & 0 & \lambda_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & \lambda_n \end{bmatrix}$$

Consider the matrix product AX,

$$AX = A \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix} = \begin{bmatrix} \lambda_1 x_1 & \dots & \lambda_n x_n \end{bmatrix} = X\Lambda$$

Since X^{-1} exists, we multiply by X^{-1} on both the sides from left and obtain

$$X^{-1}AX = X^{-1}X\Lambda = \Lambda \tag{4.2}$$

and when we multiply on the right we obtain

$$AXX^{-1} = A = X\Lambda X^{-1} \tag{4.3}$$

For our example from the last section, where

$$A = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix}$$

with eigenvalues $\lambda_1 = 5$ and $\lambda_2 = 1$ and the eigenvectors $[1, 3]$ and $[1, -1]$ respectively, we have

$$AX = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 3 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 1 \\ 15 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 3 & -1 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} = X\Lambda$$

and

$$X^{-1}AX = \begin{bmatrix} 1/4 & 1/4 \\ 3/4 & -1/4 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 3 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} = \Lambda$$

Similarly,

$$A = X\Lambda X^{-1} = \begin{bmatrix} 1 & 1 \\ 3 & -1 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1/4 & 1/4 \\ 3/4 & -1/4 \end{bmatrix}$$

An alternate way to think about diagonalization, eigenvalues, and eigenvectors is as follows. We say an $n \times n$ matrix A is diagonalizable if there exists an invertible $n \times n$ matrix X such that $X^{-1}AX$ is a $n \times n$ diagonal matrix Λ. Therefore, $X^{-1}AX = \Lambda$, or equivalently $AX = X\Lambda$. This can be expressed as $A \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} \Lambda$. This can also be expressed as for $i = 1, \dots, n$, $Ax_i = \lambda_i x_i$. Thus, A

is diagonalizable if there exists n scalars $\lambda_1, \ldots, \lambda_n$ and n linearly independent vectors $x_1, \ldots, x_n$, such that $Ax_i = \lambda_i x_i$.

Consider the diagonalization given by equation $A = X\Lambda X^{-1}$. Consider $A^2 = (X\Lambda X^{-1})(X\Lambda X^{-1}) = X\Lambda(X^{-1}X)\Lambda X^{-1} = X\Lambda^2 X^{-1}$. Thus, A^2 has the same set of eigenvectors as A, but its eigenvalues are squared. In general, for an integer $k > 0$, $A^k = X\Lambda^k X^{-1}$, and its eigenvectors are same as that of A and its eigenvalues are raised to the power of k.

Let $u_0, u_1, u_2, \cdots \in \mathbb{R}^n$ are vectors and $u_{k+1} = Au_k$ for $k \geq 0$. Any vector in $\mathbb{R}^n$ can be expressed as a linear combination of the eigenvectors $x_1, x_2, \ldots x_n$. Thus, for constants $c_1, c_2, \ldots, c_n$,

$$\begin{aligned}
u_0 &= c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
u_1 &= Au_0 = A(c_1x_1 + c_2x_2 + \cdots + c_nx_n) \\
u_1 &= c_1Ax_1 + c_2Ax_2 + \cdots + c_nAx_n \\
u_1 &= c_1\lambda_1x_1 + c_2\lambda_2x_2 + \cdots + c_n\lambda_nx_n \\
u_2 &= Au_1 = c_1\lambda_1Ax_1 + c_2\lambda_2Ax_2 + \cdots + c_n\lambda_nAx_n \\
u_2 &= c_1\lambda_1^2x_1 + c_2\lambda_2^2x_2 + \cdots + c_n\lambda_n^2x_n \\
&\vdots \\
u_{k+1} &= Au_k = c_1\lambda_1^kx_1 + c_2\lambda_2^kx_2 + \cdots + c_n\lambda_n^kx_n
\end{aligned}$$

Note that for large values of k, if for any λ_i, $|\lambda_i| < 1$, $|\lambda_i|^k \to 0$.

Here is an interesting exercise from the text book of Gilbert Strang to illustrate the above concept. Define $G(n)$ for integers $n \geq 0$ as follows:

$$G(n) = \begin{cases} 0, & \text{for } n = 0 \\ 1, & \text{for } n = 1 \\ \frac{G(n-1)+G(n-2)}{2}, & \text{otherwise.} \end{cases}$$

Now, define

$$g_k = \begin{bmatrix} G_{k+1} \\ G_k \end{bmatrix}$$

We obtain

$$g_{k+1} = \begin{bmatrix} G_{k+2} \\ G_{k+1} \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ 1 & 0 \end{bmatrix} g_k$$

Define,

$$A = \begin{bmatrix} 1/2 & 1/2 \\ 1 & 0 \end{bmatrix}$$

The eigenvalues and eigenvectors of A are $\lambda_1 = 1$ and $\lambda_2 = 1/2$ and $x_1 = (1, 1)$ and $x_2 = (1, -2)$, respectively. Thus,

$$g_k = c_1\lambda_1^k x_1 + c_2\lambda_2^k x_2 = c_1 x_1 + c_2(-1/2)^k x_2$$

For large values of k,

$$\lim_{k\to\infty} g_k \approx c_1 x_1$$

To find the value of c_1, we use that $g_k = c_1\lambda_1^k x_1 + c_2\lambda_2^k x_2$ for $k = 0$, and obtain

$$g_0 = \begin{bmatrix}1\\0\end{bmatrix} = c_1 \begin{bmatrix}1\\1\end{bmatrix} + c_2 \begin{bmatrix}1\\-2\end{bmatrix}$$

This gives $c_1 = 2/3$ and $c_2 = 1/3$. Thus, for large values of k,

$$g_k = 2/3 \begin{bmatrix}1\\1\end{bmatrix}$$

and G_k approaches $2/3$ for large values of k.

4.4 *Symmetric and Positive Definite Matrices*

Let S be an $n \times n$ real symmetric matrix where its ij-th entry is identical to the ji-th entry for all $1 \le i \le n$ and $1 \le j \le n$, i.e., $S = S^T$. We will show that S has n real eigenvalues and it consists of n orthonormal eigenvectors $Q = q_1, q_2, \ldots, q_n$. Moreover, the diagonalization of S is given by $S = Q\Lambda Q^T$, where Λ is the diagonal matrix consisting of real eigenvalues on its principal diagonal and Q consists of orthonormal eigenvectors as columns. First we present an example.

Example 4.4.1 *Consider the symmetric matrix* $S = \left[\begin{smallmatrix}3 & 1\\ 1 & 3\end{smallmatrix}\right]$. *Its characteristic equation is* $\lambda^2 - 6\lambda + 8 = 0$ *and the eigenvalues are* $\lambda_1 = 4$ *and* $\lambda_2 = 2$ *and the corresponding eigenvectors are* $q_1 = (1/\sqrt{2}, 1/\sqrt{2})$ *and* $q_2 = (1/\sqrt{2}, -1/\sqrt{2})$, *respectively. Note that eigenvalues are real and the eigenvectors are orthonormal. Furthermore,*

$$S = \begin{bmatrix}3 & 1\\ 1 & 3\end{bmatrix} = \begin{bmatrix}1/\sqrt{2} & 1/\sqrt{2}\\ 1/\sqrt{2} & -1/\sqrt{2}\end{bmatrix} \begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix} \begin{bmatrix}1/\sqrt{2} & 1/\sqrt{2}\\ 1/\sqrt{2} & -1/\sqrt{2}\end{bmatrix}$$

Lemma 4.4.2 *All the eigenvalues of a real symmetric matrix S are real.*

Proof. By definition of eigenvalues,

$$Sq = \lambda q \tag{4.4}$$

By taking the complex conjugate, we have that $\bar{S}\bar{q} = \bar{\lambda}\bar{q}$. Since we are given that S is real, $S = \bar{S}$. (Note that for a complex number $a + bi$, its

complex conjugate is $a - bi$.) Using the fact that $S = S^T$, the transpose of $S\bar{q} = \bar{\lambda}\bar{q}$ is given by

$$\bar{q}^T S = \bar{q}^T \bar{\lambda} \tag{4.5}$$

We multiply by $\bar{q}^T$ on the left in Equation 4.4 and obtain $\bar{q}^T Sq = \bar{q}^T \lambda q$. Similarly, we multiply by q on the right in Equation 4.5 and obtain $\bar{q}^T Sq = \bar{q}^T \bar{\lambda} q$. Thus, $\bar{q}^T \lambda q = \bar{q}^T \bar{\lambda} q$. This implies that $\lambda = \bar{\lambda}$ and this can only happen if λ's are real. ∎

Lemma 4.4.3 *All components of the eigenvectors of a real symmetric matrix S are real.*

Proof. Each eigenvector q is a solution of the equation $(S - \lambda I)q = \lambda q$, where all elements of S are real and all λ's are real. Thus all entries in q are real. ∎

Lemma 4.4.4 *Any pair of eigenvectors of a real symmetric matrix S corresponding to two different eigenvalues are orthogonal.*

Proof. Let q_1 and q_2 be two eigenvectors corresponding to $\lambda_1 \neq \lambda_2$, respectively. Thus, $Sq_1 = \lambda_1 q_1$ and $Sq_2 = \lambda_2 q_2$. Since S is symmetric, $q_1^T S = \lambda_1 q_1^T$. Multiply by q_2 on the right and we obtain $\lambda_1 q_1^T q_2 = q_1^T Sq_2 = q_1^T \lambda_2 q_2$. Since $\lambda_1 \neq \lambda_2$ and $\lambda_1 q_1^T q_2 = q_1^T \lambda_2 q_2$, this implies that $q_1^T q_2 = 0$ and thus the eigenvectors q_1 and q_2 are orthogonal. ∎

First we consider the special types of symmetric matrices S having n distinct eigenvalues. Then its corresponding eigenvectors are orthogonal by Lemma 4.4.4. In fact we can assume that they are orthonormal (as we can always normalize them). Then using the diagonalization discussed in the previous section, the matrix S can be expressed as $S = Q\Lambda Q^{-1}$, where Q is the matrix of orthonormal eigenvectors (see Equation 4.3). Equivalently,

$$S = Q\Lambda Q^{-1} = \lambda_1 q_1 q_1^T + \lambda_2 q_2 q_2^T + \cdots + \lambda_n q_n q_n^T.$$

For our example we will have

$$S = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} = 4 \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} + 2 \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$$

Now we handle the general case of the symmetric matrices where all of its eigenvalues may or may not be distinct. Let us first start with an example.

Example 4.4.5 *Consider the symmetric matrix* $S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

Its characteristic polynomial is given by the determinant of

$$\begin{bmatrix} 1-\lambda & 0 & 0 & 0 \\ 0 & 1-\lambda & 1 & 0 \\ 0 & 1 & 1-\lambda & 0 \\ 0 & 0 & 0 & 1-\lambda \end{bmatrix}$$

and it equals $\lambda^4 - 4\lambda^3 + 5\lambda^2 - 2\lambda$. *The roots of this polynomial are the eigenvalues. They are* $\lambda_1 = 2$, $\lambda_2 = 1$, $\lambda_3 = 1$, $\lambda_4 = 0$. *The corresponding eigenvectors are* $q_1 = (0, 1/\sqrt{2}, 1/\sqrt{2}, 0)$, $q_2 = (0, 0, 0, 1)$, $q_3 = (1, 0, 0, 0)$, $q_4 = (0, -1/\sqrt{2}, 1/\sqrt{2}, 0)$, *respectively. It is easy to verify that* $Sq_i = \lambda_i q_i$.

To see the eigenvectors corresponding to $\lambda = 1$, *consider the nullspace of the matrix* $S - \lambda I$. *Equivalently, what forms the basis for the nullspace of*

$$\begin{bmatrix} 1-1 & 0 & 0 & 0 \\ 0 & 1-1 & 1 & 0 \\ 0 & 1 & 1-1 & 0 \\ 0 & 0 & 0 & 1-1 \end{bmatrix} x = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Clearly the basis consists of two unit vectors $q_2 = (0, 0, 0, 1)$ *and* $q_3 = (1, 0, 0, 0)$. *Similarly, the eigenvector corresponding to* $\lambda = 2$ *is given by*

$$\begin{bmatrix} 1-2 & 0 & 0 & 0 \\ 0 & 1-2 & 1 & 0 \\ 0 & 1 & 1-2 & 0 \\ 0 & 0 & 0 & 1-2 \end{bmatrix} x = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The basis for the nullspace consists of the unit vector $q_1 = (0, 1/\sqrt{2}, 1/\sqrt{2}, 0)$. *Finally, the eigenvector corresponding to* $\lambda = 0$ *is given by*

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The basis for the nullspace consists of $q_4 = (0, -1/\sqrt{2}, 1/\sqrt{2}, 0)$. *By our choice, it is not hard to see that all the eigenvectors are orthonormal. Thus,*

$$Q = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1/\sqrt{2} & 0 & 0 & -1/\sqrt{2} \\ 1/\sqrt{2} & 0 & 0 & 1/\sqrt{2} \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

and

$$\Lambda = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Moreover, it is easy to see that

$$S = Q\Lambda Q^T = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1/\sqrt{2} & 0 & 0 & -1/\sqrt{2} \\ 1/\sqrt{2} & 0 & 0 & 1/\sqrt{2} \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1/\sqrt{2} & 0 & 0 & -1/\sqrt{2} \\ 1/\sqrt{2} & 0 & 0 & 1/\sqrt{2} \\ 0 & 1 & 0 & 0 \end{bmatrix}^T$$

Moreover, S can be expressed as the summation $S = \sum_{i=1}^{4} \lambda_i q_i q_i^T$.

Theorem 4.4.6 *All eigenvalues of a real symmetric $n \times n$ matrix S are real. Moreover, S can be expressed as $S = Q\Lambda Q^T$, where Q consists of orthonormal basis of $\mathbb{R}^n$ formed by n eigenvectors of S, and Λ is a diagonal matrix consisting of n eigenvalues of S.*

Proof. By Lemma 4.4.2, we know that all eigenvalues of S are real. Now we show the existence of n orthonormal eigenvectors of S that forms a basis of $\mathbb{R}^n$. We will provide a proof for 3×3 real symmetric matrices S. Induction can be used to derive the proof for the general case by extending the ideas. We will assume that given any three basis vectors v_1, v_2, and v_3, spanning $\mathbb{R}^3$, we can find a set of corresponding orthogonal basis vectors v_1', v_2', and v_3', by executing the Gram-Schmidth orthogonalization process. Furthermore, we can obtain an orthonormal basis by dividing each of the vectors by their norm.

Gram-Schmidth orthogonalization applied to v_1, v_2, and v_3 will result in
$v_1' = v_1$
$v_2' = v_2 - \frac{\langle v_2, v_1 \rangle}{\langle v_1, v_1 \rangle} v_1$
$v_3' = v_3 - \frac{\langle v_3, v_1 \rangle}{\langle v_1, v_1 \rangle} v_1 - \frac{\langle v_3, v_2 \rangle}{\langle v_2, v_2 \rangle} v_2$
and a corresponding orthonormal basis will be
$(\frac{v_1'}{||v_1'||}, \frac{v_2'}{||v_2'||}, \frac{v_3'}{||v_3'||})$

Let λ_1 be an eigenvalue of S and let $q_1 \in \mathbb{R}^3$ be a unit eigenvector corresponding to the null space of $S - \lambda_1 I$, where I is a 3×3 identity matrix. Let $\mathcal{B}_1 = (q_1, v_2, v_3)$ forms an orthonormal basis of $\mathbb{R}^3$ (i.e., take any three vectors $u_1 = q_1$, u_2, and u_3 that forms a basis of $\mathbb{R}^3$ and then apply Gram-Schmidth orthogonalization resulting in an orthonormal basis $\mathcal{B}_1 = (q_1, v_2, v_3)$). Consider

$$S\mathcal{B}_1 = (Sq_1 \; Sv_2 \; Sv_3) = (\lambda_1 q_1 \; Sv_2 \; Sv_3)$$

Observe that

$$\mathcal{B}_1^T S \mathcal{B}_1 = \begin{pmatrix} q_1^T \\ v_2^T \\ v_3^T \end{pmatrix} (\lambda_1 q_1 \; Sv_2 \; Sv_3) = \begin{pmatrix} \lambda_1 q_1^T q_1 & ? & ? \\ \lambda_1 v_2^T q_1 & ? & ? \\ \lambda_1 v_3^T q_1 & ? & ? \end{pmatrix}$$

where ? indicates that we don't know what these entries are (but soon we will get to know them). Using orthogonality of unit vectors in $\mathcal{B}_1$ and the fact that $\mathcal{B}_1^T S \mathcal{B}_1$ is symmetric since S is, we obtain

$$\mathcal{B}_1^T S \mathcal{B}_1 = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & ? & ? \\ 0 & ? & ? \end{pmatrix}$$

Notice that the 2×2 unknown matrix in $\mathcal{B}_1^T S \mathcal{B}_1$ is symmetric. Call it S'. We can apply the same construction on S', and it will result in a

another basis $\mathcal{B}_2'$ of $\mathbb{R}^2$ such that

$$\mathcal{B}_2'^T S' \mathcal{B}_2' = \begin{pmatrix} \lambda_2 & 0 \\ 0 & \lambda_3 \end{pmatrix}$$

Let $\mathcal{B}_2' = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. Construct $\mathcal{B}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & a & b \\ 0 & c & d \end{bmatrix}$. Observe that $\mathcal{B}_2$ is an orthonormal basis of $\mathbb{R}^3$. Also, observe that $\mathcal{B}_1\mathcal{B}_2$ is orthogonal. Now consider,

Note that $\mathcal{B}_1^T\mathcal{B}_1 = I$ and $\mathcal{B}_2^T\mathcal{B}_2 = I$. Thus, $(\mathcal{B}_1\mathcal{B}_2)^T\mathcal{B}_1\mathcal{B}_2 = \mathcal{B}_2^T\mathcal{B}_1^T\mathcal{B}_1\mathcal{B}_2 = I$.

$$\begin{aligned} (\mathcal{B}_1\mathcal{B}_2)^T S \mathcal{B}_1\mathcal{B}_2 &= \mathcal{B}_2^T\mathcal{B}_1^T S \mathcal{B}_1\mathcal{B}_2 \\ &= \mathcal{B}_2^T \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & ? & ? \\ 0 & ? & ? \end{bmatrix} \mathcal{B}_2 \\ &= \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & & \lambda_3 \end{bmatrix} \end{aligned}$$

Define $Q = \mathcal{B}_1\mathcal{B}_2$ and $\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & & \lambda_3 \end{bmatrix}$. Since Q is orthogonal $Q^{-1} = Q^T = \mathcal{B}_2^T\mathcal{B}_1^T$, we have

$$(\mathcal{B}_1\mathcal{B}_2)^T S \mathcal{B}_1\mathcal{B}_2 = Q^T S Q = \Lambda$$

Thus, $S = Q\Lambda Q^T$. ∎

The above theorem is a special case of Schur's lemma that states that given any $n \times n$ complex matrix A, there exists an $n \times n$ unary matrix U (consisting of n orthonormal vectors forming the columns of U) such that $U^H A U$ is an upper triangular matrix. The matrix U^H is the (conjugate) transpose matrix of U, where each entry u_{ij} in U becomes $\overline{u_{ji}}$ in U^H. Note that if U is a real matrix, then U^H is simply the transpose of U. A matrix M is said to upper triangular if all the entries $m_{ij} = 0$ for $i > j$. If A is Hermitian (i.e., $\forall i, j : \ a_{ij} = \overline{a_{ji}}$), then $U^H A U$ will be a diagonal matrix. The above statement, when translated for real symmetric matrices S will imply that $Q^T A Q$ is a diagonal matrix Λ.

From Theorem 4.4.6 we know that a real symmetric $n \times n$ matrix $S = Q^T \Lambda Q = \sum_{i=1}^{n} q_i^T \lambda_i q_i$. The above representation enables us to express any symmetric matrix as the sum of rank one matrices $\lambda_i q_i q_i^T$. In the Section 4.5, we will have a similar way of expressing any rectangular matrix.

A symmetric matrix S is said to be *positive definite* if all its eigenvalues > 0. It is called *positive semi-definite* if all the eigenvalues are ≥ 0. An alternative way to define a positive definite matrix is as follows. A symmetric matrix $S \in \mathbb{R}^n \times \mathbb{R}^n$ is positive definite (respectively, positive semi-definite) if for for all non-zero vectors $x \in \mathbb{R}^n$, $x^T S x > 0$ (respectively, $x^T S x \geq 0$).

Lemma 4.4.7 *Let $S \in \mathbb{R}^n \times \mathbb{R}^n$ be a symmetric matrix. For all non-zero vectors $x \in \mathbb{R}^n$, if $x^T S x > 0$ holds, then all the eigenvalues of S are > 0.*

Proof. Let λ_i be an eigenvalue of S and its corresponding unit eigenvector is q_i. Note that $q_i^T q_i = 1$. Since S is symmetric, we know that λ_i is real. Now we have, $\lambda_i = \lambda_i q_i^T q_i = q_i^T \lambda_i q_i = q_i^T S q_i$. But $q_i^T S q_i > 0$, hence $\lambda_i > 0$. ∎

4.5 *Singular Value Decomposition*

In previous sections we learnt about diagonalization of square matrices. We have seen that a square matrix A can be expressed as $A = X\Lambda X^{-1}$, where X consists of n linearly independent eigenvectors as columns. There we needed A to be square, (usually) all its eigenvalues to be distinct so that the eigenvectors are independent. Here we will not make any such assumptions on A - not even that it is a square matrix - but we will be able to diagonalize it! The singular value decomposition (SVD) enables us to diagonalize any type of matrices.

Let A be a $m \times n$ matrix of rank r. Recall that the rank is the number of linearly independent rows (or columns) of A. We will show that we can find orthonormal vectors $v_1, \ldots, v_r \in \mathbb{R}^n$, orthonormal vectors $u_1, \ldots, u_r \in \mathbb{R}^m$, and $\sigma_1, \ldots, \sigma_r \in \mathbb{R}$, such that $Av_i = \sigma_i u_i$, for $i = 1, \ldots, r$. Note that since A may not be a square matrix, the vectors as a result of the product Av_i are not of the same dimension as v_i. But the beauty of SVD is that we can still find orthonormal vectors in $\mathbb{R}^n$ such that their product with A results in a scaled copy of orthonormal vectors in $\mathbb{R}^m$. So this mimics the eigenvalue/vector properties of $Av = \lambda v$ when A is a square matrix. The key is to work with the square symmetric matrices AA^T and A^TA, instead of the matrix A. Of course, the resulting diagonalization is slightly more complex, and will use the four orthonormal bases associated with column and row subspaces of A to understand the process.[2] These are:

1. The vectors $u_1, \ldots, u_r$ will form orthonormal bases of the columns of A.
2. The vectors $u_{r+1}, \ldots, u_m$ will form an orthonormal basis for the nullspace of A^T.
3. The vectors $v_1, \ldots, v_r$ will form an orthonormal bases of the rows of A.
4. The vectors $v_{r+1}, \ldots, v_n$ will form an orthonormal basis for the nullspace of A.

Moreover, these vectors will satisfy the following $Av_1 = \sigma_1 u_1, \ldots, Av_r = \sigma_1 u_r$, where $\sigma_i > 0$ for $i = 1, \ldots, r$. This can equivalently be expressed

[2] To get some insight in these subspaces, consider $Ax = b$, where A is an $m \times n$ matrix and assume all elements in A, b, x are real. This system has solution when b is equal to some linear combination of the columns of A (specified by x). In that case we say that b is in the column space of A. The column space of A consists of all linear combinations of the columns of A and $Ax = b$ is solvable if and only if b is in the column space of A. Note that A has n columns, but each column only consists of m entries. Each column is in $\mathbb{R}^m$ and the column space is a subspace of $\mathbb{R}^m$. The rank of a matrix is the number of independent columns (or rows) of A. Equivalently, when we do a row reduction (e.g. in Gaussian elimination) the rank is the number of non-zero rows in A. Furthermore, we define the null space of A to consist of all solutions x to the equation $Ax = 0$. Note that these are the vectors in $\mathbb{R}^n$.

in matrix notation as

$$A\begin{bmatrix} v_1 & v_2 & \dots & v_r \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & \dots & u_r \end{bmatrix} \begin{bmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & . & & \\ & & & . & \\ & & & & \sigma_r \end{bmatrix}$$

We further expand this to include the whole set of orthonormal vectors $U = [u_1, \dots, u_r, u_{r+1}, \dots, u_m]$ and $V = [v_1, \dots, v_r, v_{r+1}, \dots, v_n]$ as follows

$$A\begin{bmatrix} v_1 \dots v_r v_{r+1} \dots v_n \end{bmatrix} = \begin{bmatrix} u_1 \dots u_r u_{r+1} \dots u_m \end{bmatrix} \begin{bmatrix} \sigma_1 & & & & & \\ & . & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & 0 & \\ & & & & & 0 \end{bmatrix}$$

and more compactly we can write this as $AV = U\Sigma$ or equivalently $A = U\Sigma V^{-1} = U\Sigma V^T$, where Σ is the diagonal matrix consisting $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r, 0, 0, \dots$ and $V^T = V^{-1}$ (as V is a matrix of orthonormal vectors). Let us try to see what are these σ_i's are.

First we assume that $m \geq n$ and $rank(A) = n$, and later on we will consider the case that $rank(A) < n$. Consider the matrix product A^TA. This is an $n \times n$ matrix. Moreover it is symmetric as $(A^TA)^T = A^T(A^T)^T = A^TA$.

Lemma 4.5.1 *The matrix A^TA is positive definite.*

Proof. Take any non-zero vector $x \in \mathbb{R}^n$. Consider $x^TA^TAx = (Ax)^T(Ax) = ||Ax||^2 \geq 0$. We have assumed that $rank(A) = n$. This implies that the rank of the null-space of A is $n - rank(A) = 0$. Hence, the only vector x for which $Ax = 0$ is $x = 0$. This implies that $x^TA^TAx = ||Ax||^2 > 0$ and hence A^TA is positive definite. ∎

From the positive definiteness of A^TA, we know that its eigenvalues are positive and the corresponding eigenvectors are orthonormal. Let its eigenvalues be $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n > 0$ and let the corresponding orthonormal eigenvectors be $v_1, v_2, \dots, v_n$, respectively. We know that $A^TAv_i = \lambda_i v_i$, or equivalently $v_i^TA^TAv_i = \lambda_i$. [3] Define $\sigma_i = ||Av_i||$. Then $\sigma_i^2 = ||Av_i||^2 = v_i^TA^TAv_i = \lambda_i$. Thus, $\sigma_i = ||Av_i|| = \sqrt{\lambda_i}$. Note that $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n > 0$. We define the vectors $u_1, \dots, u_n \in \mathbb{R}^m$ as follows. Each $u_i = Av_i/\sigma_i$. Note that u_i's are well defined as $\sigma_i \neq 0$.

[3] $v_i^Tv_i = 1$ as v_i is orthonormal.

Lemma 4.5.2 *The set of vectors $u_i = Av_i/\sigma_i$, for $i = 1, \dots, n$, are orthonormal.*

Proof. First, let us consider

$$||u_i|| = ||Av_i||/\sigma_i = \sigma_i/\sigma_i = 1.$$

Now we show that the dot product of any two vectors u_i and u_j for $1 \leq i \neq j \leq n$ is zero.

$$u_i^T u_j = (Av_i/\sigma_i)^T(Av_j/\sigma_j) = \frac{1}{\sigma_i\sigma_j} v_i^T A^T A v_j = \frac{1}{\sigma_i\sigma_j} v_i^T \lambda_j v_j = \frac{\lambda_j}{\sigma_i\sigma_j} v_i^T v_j = 0.$$

[4] ∎

[4] As $A^T A v_j = \lambda_j v_j$ and v_i and v_j are orthonormal.

Now we have that $v_1, \ldots, v_n$ are orthonormal and $u_1, \ldots, u_n$ are orthonormal and for $i = 1, \ldots, n$, $Av_i = \sigma_i u_i$. In the matrix notation we can write this as $AV' = U'\Sigma'$, where A is the given ($rank(A) = n$) $m \times n$ matrix, V' is $n \times n$ matrix consisting of orthonormal vectors $v_1, \ldots, v_n$, U is $m \times n$ matrix consisting of orthonormal vectors $u_1, \ldots, u_n$, and Σ' is $n \times n$ diagonal matrix where each $(i,i) - th$ entry is σ_i for $i = 1, \ldots, n$ and $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_n > 0$. Since V' is square and orthonormal, $V'^{-1} = V'^T$. Multiply by V'^T on both the sides on the right of $AV' = U'\Sigma'$ and we obtain $A = U'\Sigma' V'^T$.

The matrix U' is not a square matrix. It consists of orthonormal vectors $u_1, \ldots, u_n$ but it has more rows than columns (as $m \geq n$). Hence the null space of U' is non-empty. Let $u_{n+1}, \ldots, u_m$ be orthonormal vectors to $u_1, \ldots, u_n$ (i.e., in the null space of A^T). Therefore, together with $u_1, \ldots, u_n$ we form the matrix U of dimension $m \times m$ consisting of m orthonormal vectors $u_1, \ldots, u_m$. Thus $U^T U = I$ and $U^{-1} = U^T$.

Lastly, we transform Σ' to Σ. It is $m \times n$ matrix, where each of its entry is 0 except the (i,i)-th entry equals σ_i for $i = 1, \ldots, n$.

Observe that under the assumption that $rank(A) = n$ and $m \geq n$, now we have that $A = U\Sigma V'^T$, where $U^T U = I$ and $V^T V = I$. Note that we can also express $A = \Sigma_{i=1}^n \sigma_i u_i v_i^T$. This will be the rank one decomposition of A. Since $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$, we can see that the sum of $\sigma_i u_i v_i^T$ corresponding to the large values of σ_i essentially approximate A.

Example 4.5.3 *Let* $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix}$. *$A$ is 3×2 matrix, where $m = 3$ and $n = 2$. Rank of A is $n = 2$.* $A^T A = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix}$. *Eigenvalues of $A^T A$ are given by det* $\begin{bmatrix} 5-\lambda & 2 \\ 2 & 2-\lambda \end{bmatrix} = 0$. *This results in* $\lambda^2 - 7\lambda + 6 = (\lambda - 6)(\lambda - 1) = 0$, *or* $\lambda_1 = 6$ *and* $\lambda_2 = 1$. *Note that* $\lambda_1 \geq \lambda_2 > 0$. *The corresponding eigenvectors are*

$A^T A v_1 = \lambda_1 v_1$ *is* $v_1 = \begin{bmatrix} 2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix}$, *and*

$A^T A v_2 = \lambda_2 v_2$ *is* $v_2 = \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}$.

Note that v_1 and v_2 are orthonormal vectors as $v_1^T v_2 = 0$ and $||v_1|| = 1$ and $||v_2|| = 1$. Next we compute the vectors u_1 and u_2. First $\sigma_1 = \sqrt{\lambda_1} = \sqrt{6}$ and $\sigma_2 = \sqrt{\lambda_2} = \sqrt{1} = 1$. Then,

$u_1 = \frac{1}{\sigma_1} A v_1 = \frac{1}{\sqrt{6}} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix} = \begin{bmatrix} 2/\sqrt{30} \\ 1/\sqrt{30} \\ 5/\sqrt{30} \end{bmatrix}$.

Similarly,

$u_2 = \frac{1}{\sigma_2} A v_2 = \frac{1}{\sqrt{1}} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix} = \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \\ 0 \end{bmatrix}$.

Note that u_1 and u_2 are orthonormal vectors.

Set $U' = \begin{bmatrix} 2/\sqrt{30} & -1/\sqrt{5} \\ 1/\sqrt{30} & 2/\sqrt{5} \\ 5/\sqrt{30} & 0 \end{bmatrix}$, $\Sigma' = \begin{bmatrix} \sqrt{6} & 0 \\ 0 & 1 \end{bmatrix}$, and $V' = \begin{bmatrix} 2/\sqrt{5} & -1/\sqrt{5} \\ 1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}$.

Observe that $A = U'\Sigma' V'^T$, i.e.

$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 2/\sqrt{30} & -1/\sqrt{5} \\ 1/\sqrt{30} & 2/\sqrt{5} \\ 5/\sqrt{30} & 0 \end{bmatrix} \begin{bmatrix} \sqrt{6} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2/\sqrt{5} & 1/\sqrt{5} \\ -1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}$.

Next, we extend U' to U by taking a unit vector in the null space of U' that is orthonormal to u_1 and u_2. Let $u_3 = (a, b, c)$. Then we set $u_1^T u_3 = 0$ and $u_2^T u_3 = 0$ and $||u_3|| = \sqrt{a^2 + b^2 + c^2} = 1$ and obtain that $u_3 = (2/\sqrt{6}, 1/\sqrt{6}, -1/\sqrt{6})$. We derive Σ from Σ' by setting $\Sigma = \begin{bmatrix} \sqrt{6} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$.

Now we have that $A = U\Sigma V'^T$, i.e.

$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 2/\sqrt{30} & -1/\sqrt{5} & 2/\sqrt{6} \\ 1/\sqrt{30} & 2/\sqrt{5} & 1/\sqrt{6} \\ 5/\sqrt{30} & 0 & -1/\sqrt{6} \end{bmatrix} \begin{bmatrix} \sqrt{6} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 2/\sqrt{5} & 1/\sqrt{5} \\ -1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}$.

Lastly, in terms of rank one decomposition of A, observe that

$A = \sqrt{6} \begin{bmatrix} 2/\sqrt{30} \\ 1/\sqrt{30} \\ 5/\sqrt{30} \end{bmatrix} \begin{bmatrix} 2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix}^T + \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \\ 0 \end{bmatrix} \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}^T$.

Now we handle the case when $m \geq n$ and $rank(A) < n$. Let $rank(A) = r < n$. Thus the rank of the null space of A is $n - r$. We claim the following.

Lemma 4.5.4 *The $n - r$ eigenvalues of $A^T A$ equals 0.*

Proof. Consider a basis of the null space of A. Let $x_1, \ldots, x_{n-r}$ be a basis of the null space of A. This implies that $Ax_j = 0$ for $j = 1, \ldots, n - r$. Now, $A^T A x_j = 0 = 0 x_j$. Thus, 0 is an eigenvalue of $A^T A$ corresponding to each x_i's. Thus $n - r$ eigenvalues of $A^T A$ equals 0. ∎

As in the case of $rank(A) = n$, we first compute the eigenvalues and eigenvectors of $A^T A$. Let $\lambda_1 \geq \lambda_2 \geq \lambda_n \geq 0$ be the eigenvalues of $A^T A$. Since $n - r$ of them are 0, we know that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0$ and $\lambda_{r+1} = \cdots = \lambda_n = 0$. Let $v_1, \ldots, v_n$ be the corresponding orthonormal vectors, where $v_{r+1}, \ldots, v_n$ are in the null space of $A^T A$, i.e. $A^T A v = 0$ for each $v \in \{v_{r+1}, \ldots, v_n\}$. The vectors $v_1, \ldots, v_n$ define the orthonormal matrix V. Next we define $\sigma_i = ||Av_i|| = \sqrt{\lambda_i}$. Note that $\sigma_1 \geq \ldots \sigma_r > 0$ and $\sigma_{r+1} = \cdots = \sigma_n = 0$. We have to

take some care in defining the vectors $u_1, \ldots, u_n$ as some of the σ_i's are 0. For $i = 1, \ldots, r$, we define $u_i = \frac{1}{\sigma_i} A v_i$. As before $u_1, \ldots, u_r$ are orthonormal and $Av_i = \sigma_i u_i$. For $i = r+1, \ldots, n$ we construct u_i's that are orthonormal to all the vectors $u_1, \ldots, u_{i-1}$. Note that each of these u_i's satisfy $\sigma_i u_i = Av_i = 0$ as $\sigma_i = 0$, for $i = r+1, \ldots, n$. This will result in a matrix of orthonormal vectors $u_1, \ldots, u_n$ of dimension $m \times n$. We will further extend it to form a matrix U of orthonormal vectors $u_1, \ldots, u_m$. We also construct the matrix Σ of dimension $m \times n$, where all its entries are 0 except the (i,i)-th entry equals σ_i, for $i = 1, \ldots, r$. Given that $Av_i = \sigma_i u_i$ for $i = 1, \ldots, n$, we have that $A = U\Sigma V^T$. Furthermore, the rank one decomposition of $A = \Sigma_{i=1}^{r} \sigma_i u_i v_i^T$.

If $m < n$, then we can work with A^T in place of A, and the details are similar and hence we skip them. Given that $A = U\Sigma V^T$, consider $A^T A = (U\Sigma V^T)^T (U\Sigma V^T) = (V\Sigma U^T)(U\Sigma V^T) = VU^T\Sigma^2 UV^T = V\Sigma^2 UU^T V^T = V\Sigma^2 V^T$. Note that $A^T A$ is a symmetric matrix, and since it is expressed in the diagonalized form $A^T A = V\Sigma^2 V^T$, σ_i^2's are its eigenvalues and V is its eigenvectors matrix. Similarly, consider AA^T and we obtain that $AA^T = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma V^T V\Sigma U^T = U\Sigma^2 U^T$. Thus U is the eigenvector matrix for the symmetric matrix AA^T with the same eigenvalues as $A^T A$. Overall, the singular value decomposition of a $m \times n$ matrix A can be captured in the following theorem.

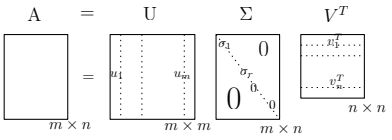


Figure 4.1: Illustration of SVD of a $m \times n$ matrix A expressed as $U\Sigma V^T$.

Theorem 4.5.5 *Let A be a $m \times n$ matrix of real numbers of rank $r \leq \min(m,n)$. Then $A = U\Sigma V^T$, where*

U is a orthonormal $m \times m$ matrix and $U^T U = I$,

V is a orthonormal $n \times n$ matrix and $V^T V = I$, and

Σ is an $m \times n$ matrix where each of its entries is 0, except the (i,i)-th entry is σ_i for $i = 1, \ldots, r$. Note that $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_r > 0$ and $\sigma_i = \sqrt{\lambda_i}$ where λ_i are the eigenvalues of $A^T A$. The set of orthonormal vectors $v_1, \ldots, v_r$ and $u_1, \ldots, u_r$ are eigenvectors of $A^T A$ and AA^T, respectively. Furthermore, the orthonormal basis $v_{r+1}, \ldots, v_n$ for the null space of A together with $v_1, \ldots, v_r$ forms the set V. Analogously, the orthonormal basis $u_{r+1}, \ldots, u_m$ for the null space of A^T together with $u_1, \ldots, u_r$ forms the set U. The vectors v's and u's satisfy the equation $Av_i = \sigma_i u_i$, for $i = 1, \ldots, r$. Alternatively, we can express $A = \Sigma_{i=1}^{r} \sigma_i u_i v_i^T$ in terms of its rank one components.

Note: Note that if rank of A is r, we can restrict U and V to the first r columns, and Σ to $r \times r$ diagonal matrix, and still $A = U_{m \times r} \Sigma_{r \times r} V_{r \times n}^T$. This is because of the nature of the Σ matrix, as it consists of a diagonal square $r \times r$ submatrix and all of its remaining entries are 0.

Note: Suppose A represents a very large matrix where the rows cor-

respond to individual shoppers in an online store and the columns correspond to items sold by the store. This matrix possibly contains millions of rows and possibly thousands of columns, and the total space required will easily be in range of billions of entries. Assume, $m = 10^8$ and $n = 10^5$. The matrix A has 10^{13} cells. Let the SVD of $A = U\Sigma V^T$ by Theorem 4.5.5. Let there be r $\sigma_i's$ that are > 0, and let $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_r > 0$. We know that $A = \Sigma_{i=1}^{r} \sigma_i u_i v_i^T$.

Define $\mathcal{E} = \sum_{i=1}^{r} \sigma_i^2$. This is termed as the *energy* of A. Define $\mathcal{E}' = 0.99\mathcal{E}$, and let $j \leq r$ be the maximum index such that $\sum_{1=1}^{j} \sigma_i^2 \leq \mathcal{E}'$. We approximate A by $\Sigma_{i=1}^{j} \sigma_i u_i v_i^T$. Let us see how many cells we need to store in this representation. We need to store the first j columns of U, j diagonal entries of Σ, and j rows of V^T. In all that amounts to $jm + j^2 + jn$ cells. For our example, if $j = 20$, then we need to store $20 \times 10^8 + 20^2 + 20 \times 10^5 = 5,005,000$ cells. This number is only .02% of 10^{13}. A huge saving in the space (and thus processing)!

4.6 Low Rank Approximation Using SVDs

In the previous section, using the singular value decomposition, we have seen that a rank r real matrix A of dimension $m \times n$ can be expressed as $A = U\Sigma V^T$, where U is a orthonormal $m \times r$ matrix, V is a orthonormal $n \times r$ matrix, and Σ is an $r \times r$ matrix where each of its entries is 0, except the (i,i)-th entry is σ_i for $i = 1, \ldots, r$. Note that $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_r > 0$ and $\sigma_i = \sqrt{\lambda_i}$ where λ_i are the eigenvalues of $A^T A$. An alternate way to express A is $A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$, where $u_1, \ldots, u_r$ and $v_1, \ldots, v_r$ are orthonormal eigenvectors of AA^T and $A^T A$ forming the matrices U and V, respectively.

Suppose we want to approximate A by a rank k matrix, where $k \leq r$. First, we need to understand what does an approximation of a matrix mean? We define three norms for a matrix A:

Frobenius Norm: $||A||_F = \sqrt{\sum_i \sum_j A_{ij}^2}$.

Spectral Norm: $||A||_2 = \max_{x \in \Re^n} \frac{||Ax||}{||x||} = \sigma_1$

Nuclear Norm: $||A||_N = \sigma_1 + \ldots + \sigma_r$

Example 4.6.1 *For an $n \times n$ identity matrix I, $||I||_F = \sqrt{n}$, $||I||_2 = 1$, and $||I||_N = n$.*

For $k = 1, \ldots, r$, define $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$. The theorem of Eckart-Young states that among all possible rank k matrices B of the same dimensions as A, the matrix A_k minimizes the above three norms, i.e. $||A - A_k||_{\{F,2,N\}} \leq ||A - B||_{\{F,2,N\}}$.

First let us prove the theorem for the spectral norm.

Theorem 4.6.2 *Let A be a $m \times n$ real matrix of rank r with SVD $A = U\Sigma V^T$. Let $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$. Let B be a $m \times n$ matrix of rank $k \leq r$. Then, $||A - A_k||_2 \leq ||A - B||_2$.*

Proof. From Exercise 4.49, we know that $||A - A_k||_2 = || \sum_{i=k+1}^{n} \sigma_i u_i v_i^T||_2 = \sigma_{k+1}$.

If rank of A is k, then $||A - A_k||_2 = 0 \leq ||A - B||_2$, and hence the statement holds. Now assume that $rank(A) > k$. We want to show that $||A - A_k||_2 = \sigma_{k+1} \leq ||A - B||_2$. Consider the $k+1$ vectors $v_1, \ldots, v_{k+1}$ corresponding to the first $k+1$ columns of V. Take a non-zero unit vector $w \in \Re^n$ such that it can be expressed as a linear combination $w = c_1 v_1 + \ldots + c_{k+1} v_{k+1}$ and $Bw = 0$. This is always possible as the dimension of the null space of B is $n - k > 0$. Now

$$
\begin{aligned}
||A - B||_2^2 &\geq ||(A - B)w||_2^2 \\
&= ||Aw - Bw||_2^2 \\
&= ||Aw||_2^2 && \text{as } Bw = 0 \\
&= w^T A^T A w \\
&= (c_1 v_1 + \ldots + c_{k+1} v_{k+1})^T A^T A (c_1 v_1 + \ldots + c_{k+1} v_{k+1}) \\
&= (c_1 v_1 + \ldots + c_{k+1} v_{k+1})^T (c_1 A^T A v_1 + \ldots + c_{k+1} A^T A v_{k+1}) \\
&= (c_1 v_1 + \ldots + c_{k+1} v_{k+1})^T (c_1 \sigma_1^2 v_1 + \ldots + c_{k+1} \sigma_{k+1}^2 v_{k+1}) && \text{as } A^T A v_i = \sigma_i^2 v_i \\
&= c_1^2 \sigma_1^2 + \ldots + c_{k+1}^2 \sigma_{k+1}^2 && v_i \perp v_j \text{ and } ||v_i|| = 1 \\
&\geq (c_1^2 + \ldots + c_{k+1}^2) \sigma_{k+1}^2 && \text{since } \sigma_1^2 \geq \ldots \geq \sigma_{k+1}^2 \\
&= \sigma_{k+1}^2 && \text{since } w \text{ is a unit vector}
\end{aligned}
$$

Thus, $||A - B||_2 \geq \sigma_{k+1} = ||A - A_k||_2$. ∎

Next we turn our attention to Frobenius norm.

Claim 4.6.3 *Let A be $m \times n$ real matrix of rank r. Let $A = U\Sigma V^T$ be its singular-value decomposition. The following statements hold:*

1. $||A||_F^2 = \sum_i \sum_j A_{ij}^2$.
2. $||A||_F^2 = Trace(A^T A) = \sum_{i=1}^{n} (A^T A)_{ii}$.
3. $||A||_F^2 = \sigma_1^2 + \ldots + \sigma_r^2$.

Proof. The first part follows from the definition of Frobenius Norm $||A||_F = \sqrt{\sum_i \sum_j A_{ij}^2}$.

For the second part, consider the term $(A^T A)_{11}$. It is the dot product of the first column of A with itself, i.e., $(A^T A)_{11} = a_{11}^2 + a_{21}^2 +$

$a_{31}^2 + \ldots + a_{m1}^2$. Similarly, $(A^T A)_{ii}$ is the dot product of the i-th column of A with itself. Thus, $||A||_F^2 = \sum_{i=1}^{n} (A^T A)_{ii}$.

For the last part use the fact that the SVD of $A = U\Sigma V^T$, $A^T A = V\Sigma^T U^T U\Sigma V^t = V\Sigma^T\Sigma V^T = V\Sigma^2 V^T$. The trace of $A^T A$ is same as the trace of $V\Sigma^2 V^T$ and it equals $\sigma_1^2 + \ldots + \sigma_r^2$. Note that U and V are orthonormal matrices. ∎

Claim 4.6.4 *Let A be a $m \times n$ real matrix of rank r with SVD $A = U\Sigma V^T$. Let $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$. Then, $||A - A_k||_F^2 = \sigma_{k+1}^2 + \ldots + \sigma_r^2$.*

Proof. Follows from the definition of A_k, and the previous claim, as SVD of $A - A_k = U(\Sigma - \Sigma_k)V^T$, and only non-zero diagonal entries of $\Sigma - \Sigma_k$ are $\sigma_{k+1}, \ldots, \sigma_r$. ∎

Now we prove the Eckart-Young Theorem for the Frobenius Norm. We want to show that $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$ is the best rank k approximation of A under Frobenius norm. For the proof, we assume that rank of A is n, though we can work with any rank r, where $n \geq r \geq k$.

Theorem 4.6.5 *If $rank(B) = k$, then $||A - A_k||_F \leq ||A - B||_F$.*

Proof. Let the singular-value decomposition of $A = U\Sigma V^T$. We know that $||A - A_k||_F^2 = \sum_{i=k+1}^{n} \sigma_i^2$ by Claim 4.6.4.

We need to show that $||A - B||_F^2 \geq ||A - A_k||_F^2 = \sum_{i=k+1}^{n} \sigma_i^2$.

Let us denote the i-th singular value of A (respectively, $A - B$) by $\sigma_i(A)$ (respectively, $\sigma_i(A - B)$).

Note that $||A - B||_F^2 = \sum_{i=1}^{n} \sigma_i^2(A - B)$.

We will show that $\sigma_i(A - B) \geq \sigma_{k+i}(A)$ for all $i \geq 1$. This will imply that $||A - B||_F^2 = \sum_{i=1}^{n} \sigma_i^2(A - B) \geq \sum_{i=k+1}^{n} \sigma_i^2(A) = ||A - A_k||_F^2$.

Let us try to show that $\sigma_i(A - B) \geq \sigma_{k+i}(A)$ for any $i \geq 1$.

Consider the $(k+i)$-th singular value $\sigma_{k+i}(A)$ of A. Using the spectral norm $||\cdot||_2$, we can express $\sigma_{k+i}(A) = ||A - A_{k+i-1}||_2$ as $\sigma_{k+i}(A)$ is the largest eigenvalue of the matrix $A - A_{k+i-1}$ (see Exercise 4.49). Moreover, A_{k+i-1} is the best rank $k + i - 1$ approximation of A with respect to the spectral norm.

Let us define $Y = A - B$, i.e., $A = B + Y$. Now $\sigma_{k+i}(A) = ||A - A_{k+i-1}||_2 = ||B + Y - A_{k+i-1}||_2$.

By similar reasoning, $\sigma_i(Y) = ||Y - Y_{i-1}||_2$, where Y_{i-1} is the best rank $i - 1$ approximation of Y with respect to the spectral norm.

We know that B is of rank k, hence $\sigma_{k+1}(B) = 0$. Furthermore, we can express $\sigma_{k+1}(B) = ||B - B_k||_2 = 0$, where B_k is the best rank k approximation of B.

Consider the sum Y_{i-1} and B_k and let this be the matrix Z, i.e. $Z = B_k + Y_{i-1}$. Now $rank(Z) \leq rank(B_k) + rank(Y_{i-1}) = k + i - 1$.

Now we are almost ready to show that $\sigma_i(A - B) = \sigma_i(Y) \geq \sigma_{k+i}(A)$.

$$\begin{aligned}
\sigma_i(Y) &= ||Y - Y_{i-1}||_2 + ||B - B_k||_2 \\
&\geq ||Y - Y_{i-1} + B - B_k||_2 \qquad \text{using triangle inequality for norms} \\
&= ||A - Y_{i-1} - B_k||_2 \\
&= ||A - Z||_2 \\
&\geq ||A - A_{i+k-1}||_2 \\
&= \sigma_{i+k}(A)
\end{aligned}$$

Note that for any rank $\leq k + i - 1$ matrix (such as Z), $||A - A_{i+k-1}||_2 \leq ||A - Z||_2$ as A_{k+i-1} is the best rank $k + i - 1$ approximation of A with respect to the spectral norm. ∎

4.7 Markov Matrices

Let $X_0, X_1, \ldots$ be a sequence of random variables that are evolving over time. At time 0, we have the r.v. X_0, followed by r.v. X_1 at time 1 and so on. Assume that each X_i takes value from the set $\{1, \ldots, n\}$ that represents the set of states. This sequence is a *Markov chain* if the probability that X_{m+1} equals a particular state $\alpha_{m+1} \in \{1, \ldots, n\}$ only depends on what is the state of X_m and is completely independent of the states of $X_0, \ldots, X_{m-1}$, i.e.,$P[X_{m+1} = \alpha_{m+1} | X_m = \alpha_m, X_{m-1} = \alpha_{m-1}, \ldots, X_0 = \alpha_0] = P[X_{m+1} = \alpha_{m+1} | X_m = \alpha_m]$, where $\alpha_0, \ldots, \alpha_{m+1}, \cdots \in \{1, \ldots, n\}$. This is called the *memoryless property* of Markov chains as the most recent r.v. X_m determines the value of X_{m+1}, and it doesn't matter how X_m acquired its state value. To understand the transition of Markov chain from one state to another, we define *state transition probabilities*, i.e., what is the probability to go from state i to state j for all $1 \leq i, j, \leq n$. See Figure 4.2 for an example of a Markov chain with three states. This is usually represented in a transition matrix defined as follows.

A square matrix A of dimension $n \times n$ is a *Markov matrix* if all its entries are non-negative and the entries within each column sums to 1. This matrix represents a system with n states where the (ij)-th entry is the transition probability from state j to state i. Suppose that $A[i, j] = 0.3$. This means that if $X_m = j$, then there is a 30% chance

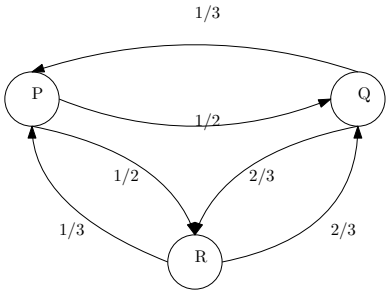


Figure 4.2: Markov chain with three states and transition probabilities.

$$A = \begin{array}{c} \\ P \\ Q \\ R \end{array} \begin{array}{c} \begin{array}{ccc} P & Q & R \end{array} \\ \begin{bmatrix} 0 & 1/3 & 1/3 \\ 1/2 & 0 & 2/3 \\ 1/2 & 2/3 & 0 \end{bmatrix} \end{array}$$

Figure 4.3: Markov matrix A corresponding to Figure 4.2.

that $X_{m+1} = i$. If $A[k, j] = 0$ then a direct transition from the state j to the state i is impossible. If $A[j, j] = 0.4$ then there is a 40% chance that the next state transition doesn't change the state.

It is common to represent a Markov chain as a directed graph $G = (V, E)$. Its node set V consists of all the states $\{1, \ldots, n\}$. There is a directed edge from a node j to node i, if there is a non-zero probability to make a transition from state j to state i in one step, i.e., $A[i, j] > 0$. We typically start with an initial state in Markov chain given by the value of r.v. X_0 (representing a node in G), and in each successive step we make a state transition from the current state given by $X_m = j$ to the next state given by $X_{m+1} = i$ (i.e., follow a directed edge from node j to node i) that respects the probability of the state transition from the node j to i in G. One of the questions that is commonly asked is what is the probability of reaching the state i after taking n steps starting from the state j? In general, given an initial probability vector representing the probabilities of starting in various states, we are interested to know what is the steady state, i.e., after traversing the chain for a large number of steps, what is the probability of landing in various states. This concept will become clear in the next section after we establish some more (graph-theoretic) properties of Markov chains.

We say that a state i is *recurrent* if starting from the state i, with probability 1, we can return to the state i after making finitely many transitions. Otherwise, it is transient, i.e., there is a non-zero probability of not returning to the state i. See Figure 4.4 for an example.

We say a Markov chain is *irreducible* if it is possible to go between any pair of states in a finite number of steps. Otherwise it is called reducible. Note that the chain in Figure 4.4 is reducible as state 4 cannot be reached from state 1. In terms of graph theory, if G is strongly connected then it is irreducible. Exercise 4.26 asks you to show that if a Markov chain is irreducible, then all its states are recurrent. Observe that if all states of a Markov chain are recurrent, then it doesn't mean that it is irreducible. For example, consider a Markov chain with two states, and the only transition that is allowed is to stay in the same state with probability 1. Clearly both the states are recurrent, but there is no way to reach one state from the other. *Period* of a state i is the greatest common divisor (GCD) of all possible number of steps it takes the chain to return to the state i starting from i. Formally, the period of the state i equals $GCD\{m > 0 : P[X_m = i | X_0 = i] > 0\}$. If there is no way to return to i starting from i, then its period is undefined. We say a Markov chain is *aperiodic* if the periods of each of its states is 1.

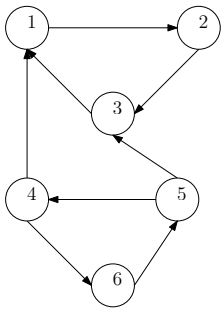


Figure 4.4: Recurrent States={1,2,3}. Transient States={4,5,6}

4.7.1 *Eigenvalues of a Markov matrix*

Recall that a Markov matrix (or a stochastic matrix or a transition matrix or a probability matrix) A is a square matrix that captures the probability of transition from one state to another in a Markov process. Each $A[i,j]$, $1 \leq i \leq n$ and $1 \leq j \leq n$, is non-negative and its value is equal to the probability of transition from the state j to state i. Observe that the sum of the values within any column is 1 (as that equals the probability of leaving from a state to any of the possible states). Assume that A is a $n \times n$ Markov matrix. Its eigenvalues are the roots of the equation given by the determinant of the matrix $A - \lambda I$, i.e., $det(A - \lambda I) = 0$, where I denotes the identity matrix. We are interested in establishing some properties of the eigenvalues of A. First an example.

The Markov matrix A corresponding to the Markov chain from Figure 4.5 is given in Figure 4.6.

Let us compute the eigenvalues of A by determining the determinant of the following matrix and equating it to zero:

$$A - \lambda I = \begin{bmatrix} 0-\lambda & 1/3 & 1/3 \\ 1/2 & 0-\lambda & 2/3 \\ 1/2 & 2/3 & 0-\lambda \end{bmatrix}$$

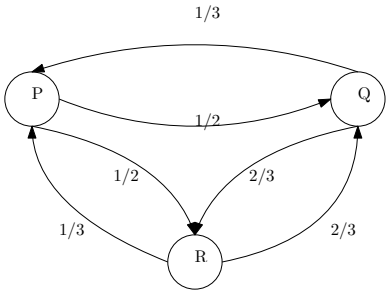


Figure 4.5: Markov chain with three states and transition probabilities.

$$A = \begin{array}{c} \\ P \\ Q \\ R \end{array} \begin{array}{c} \begin{array}{ccc} P & Q & R \end{array} \\ \begin{bmatrix} 0 & 1/3 & 1/3 \\ 1/2 & 0 & 2/3 \\ 1/2 & 2/3 & 0 \end{bmatrix} \end{array}$$

Figure 4.6: Markov matrix A corresponding to Figure 4.5.

Now $\det(A - \lambda I) = 0$ gives us the equation $9\lambda^3 - 7\lambda - 2 = 0$. The roots of this equation are $\lambda_1 = 1, \lambda_2 = -2/3$, and $\lambda_3 = -1/3$ and the corresponding eigenvectors are $v_1 = (2/3, 1, 1)$, $v_2 = (0, -1, 1)$, and $v_3 = (-2, 1, 1)$, respectively. We observe that the largest (principal) eigenvalue is 1 and the corresponding (principal) eigenvector is $(2/3, 1, 1)$. Note that $Av_i = \lambda_i v_i$, for $i = 1, \ldots, 3$. If required, we can convert any eigenvector to a unit vector $(\frac{v_i}{||v_i||})$ or take its scalar multiple, without altering its corresponding eigenvalue. Moreover, it is straightforward to see that A will have an eigenvalue of 1, as in A^T all the elements in each row add to 1 and the determinant (and the eigenvalues) of a matrix is same as that of its transpose (see Exercise 4.13). The next theorem states that the largest eigenvalue of a Markov matrix is 1.

Theorem 4.7.1 *The largest eigenvalue of any Markov matrix is* 1.

Proof. Let A be a Markov matrix of dimension n and let $B = A^T$. Let $\mathbf{1} = (1, 1, \ldots, 1)$ be a vector of dimension n. All the elements within each row of B sums to 1, thus $\sum_{j=1}^{n}(b_{ij} \cdot 1) = 1$ and $B\mathbf{1} = 1 \cdot \mathbf{1}$. Hence 1 is an eigenvalue of B (and therefore for A as $A = B^T$). Now we show that the largest eigenvalue of B is 1. We prove this by contradiction. Suppose there exists an eigenvalue $\lambda > 1$ and a

non-zero vector $\mathbf{x}$ such that $B\mathbf{x} = \lambda\mathbf{x}$. Let x_i be among the largest coordinate(s) of $\mathbf{x}$. We can assume that $x_i > 0$, otherwise we multiply by a scalar and that also satisfies this equation. Entries in B are non-negative (they are probabilities), and the sum of all the elements in any row of B is 1. We can interpret each entry in $\lambda\mathbf{x}$ as a convex combination of the elements of $\mathbf{x}$. For example, the i-th row yields $b_{i1}x_1 + b_{i2}x_2 + \cdots + b_{in}x_n = \lambda x_i$. But $b_{i1}x_1 + b_{i2}x_2 + \cdots + b_{in}x_n \leq b_{i1}x_i + b_{i2}x_i + \cdots + b_{in}x_i = (b_{i1} + b_{i2} + \cdots + b_{in})x_i = x_i$. Because of the convex combination, no entry in $\lambda\mathbf{x}$ can be larger than x_i. But since $\lambda > 1$, and in particular $\lambda x_i > x_i$, the ith row sum $\sum_{j=1}^{n}(b_{ij}x_j) = \lambda x_i > x_i$ leads to a contradiction. Therefore, the largest eigenvalue of B, and hence of A, is 1. ∎

Consider the Markov matrix A corresponding to Figure 4.5.

$$A = \begin{bmatrix} 0 & 1/3 & 1/3 \\ 1/2 & 0 & 2/3 \\ 1/2 & 2/3 & 0 \end{bmatrix}$$

Note that all the entries in A^2 are > 0 and all the entries within a column still adds to 1.

$$A^2 = \begin{bmatrix} 1/3 & 2/9 & 2/9 \\ 1/3 & 11/17 & 1/6 \\ 1/3 & 1/6 & 11/17 \end{bmatrix}$$

In fact, we can observe that if the entries within each column of A adds to 1, then entries within each column of A^k, for any integer $k > 0$, will add to 1.

Next consider multiplying A by a vector u_0, where each coordinate of $u_0 \geq 0$ and the sum of the coordinates of u_0 is 1. Think of u_0 as a initial probability distribution of a random surfer, i.e., the probability of the surfer in any given state. Consider $u_1 = Au_0$. Observe that each of the coordinates of $u_1 \geq 0$ and they also add to 1. The vector u_1 is the probability of various states in which the surfer is after making a state transition conditioned on that the starting probability distribution is u_0. For example, for $u_0 = (1/3, 1/3, 1/3)$,

$$u_1 = Au_0 = \begin{bmatrix} 0 & 1/3 & 1/3 \\ 1/2 & 0 & 2/3 \\ 1/2 & 2/3 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = \begin{bmatrix} 4/18 \\ 7/18 \\ 7/18 \end{bmatrix}$$

Similarly, define $u_2 = Au_1 = A^2u_0$, and for our example it is

$$u_2 = Au_1 = \begin{bmatrix} 0 & 1/3 & 1/3 \\ 1/2 & 0 & 2/3 \\ 1/2 & 2/3 & 0 \end{bmatrix} \begin{bmatrix} 4/18 \\ 7/18 \\ 7/18 \end{bmatrix} = \begin{bmatrix} 7/27 \\ 10/27 \\ 10/27 \end{bmatrix}$$

Likewise, we compute $u_3 = Au_2 = [20/81, 61/162, 61/162]$, $u_4 = Au_3 = [61/243, 91/243, 91/243]$, $u_5 = Au_4 = [182/729, 547/1458, 547/1458]$, $\ldots$, $u_\infty = [0.25, 0.375, 0.375] = [2/8, 3/8, 3/8]$. Note that by taking the repeated powers, $\lim_{k\to\infty} A^k u_0 = [2/8, 3/8, 3/8]$ and that corresponds to the steady state. In fact the steady state corresponds to the principal eigenvector. We formalize this notion next.

We can express the vector u_0 as a linear combination of eigenvectors. For our example, let $u_0 = c_1v_1 + c_2v_2 + c_3v_3$, where c_1, c_2, c_3 are constants. In particular,

$$\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = c_1 \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix} + c_2 \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} + c_3 \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}$$

Now u_1 can be expressed as

$$\begin{aligned} u_1 &= Au_0 \\ &= A(c_1v_1 + c_2v_2 + c_3v_3) \\ &= c_1Av_1 + c_2Av_2 + c_3Av_3 \\ &= c_1\lambda_1v_1 + c_2\lambda_2v_2 + c_3\lambda_3v_3 \qquad\qquad as Av_i = \lambda_i v_i. \end{aligned}$$

Thus,

$$u_1 = A \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = c_1\lambda_1 \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix} + c_2\lambda_2 \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} + c_3\lambda_3 \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}$$

In general, for any integer $k > 0$, $u_k = A^k u_0 = c_1\lambda_1^k v_1 + c_2\lambda_2^k v_2 + c_3\lambda_3^k v_3$, i.e.,

$$u_k = A^k \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = c_1\lambda_1^k \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix} + c_2\lambda_2^k \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} + c_3\lambda_3^k \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}$$

and that equals

$$u_k = c_1 1^k \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix} + c_2(-\frac{2}{3})^k \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} + c_3(-\frac{1}{3})^k \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}$$

For large values of k, $(\frac{2}{3})^k \to 0$ and $(\frac{1}{3})^k \to 0$. The above expression reduces to

$$u_k \approx c_1 \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix} = \frac{3}{8} \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2/8 \\ 3/8 \\ 3/8 \end{bmatrix}$$

Note that the value of c_1 is derived by solving the equation for $u_0 = c_1 v_1 + c_2 v_2 + c_3 v_3$ for $u_0 = [1/3, 1/3, 1/3]$.

Let us see what happens if the initial vector $u_0 = [1/4, 1/4, 1/2]$. Then $u_1 = Au_0 = [1/4, 11/24, 7/24]$, $u_2 = Au_1 = [1/4, 23/72, 31/72]$, $u_3 = Au_2 = [1/4, 89/216, 73/216]$, $\ldots$, $u_\infty = [2/8, 3/8, 3/8]$. Hence a different initial value $u_0 = [1/4, 1/4, 1/2]$ still leads to the same steady state corresponding to the principal eigenvector. The reasoning is same as before. Express $u_0 = d_1 v_1 + d_2 v_2 + d_3 v_3$ for constants d_1, d_2 and d_3. Now, for any $k > 0$, $u_k = A^k u_0 = d_1 \lambda_1^k v_1 + d_2 \lambda_2^k v_2 + d_3 \lambda_3^k v_3 = d_1 v_1 + d_2 (-\frac{2}{3})^k v_2 + d_3 (-\frac{1}{3})^k v_3$. Now take $\lim_{k \to \infty} u_k = d_1 v_1$. We solve for d_1 using the initial condition

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/2 \end{bmatrix} = d_1 \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix}$$

and obtain that $d_1 = 3/8$. Thus $u_\infty = [2/8, 3/8, 3/8]$.

Consider the following example that illustrates the number of fans of the Senators and the Leafs NHL Teams at the end of the season. Assume that there are 3,000,000 hockey fans all over Canada for these two teams and at the end of the season, depending on the performance of the two teams, certain fraction of the fans switch loyalties. Assume that the following transition matrix captures the change in loyalties:

$$A = \begin{array}{c} \\ \\ \end{array} \begin{array}{c} \begin{array}{cc} Sens & Leafs \end{array} \\ \begin{bmatrix} 0.9 & 0.3 \\ 0.1 & 0.7 \end{bmatrix} \end{array} \begin{array}{c} \\ Sens \\ Leafs \end{array}$$

Note that only 10% of Sens fans switch loyalties as opposed to 30% of the Leafs fans. We assume that this trend stays forever. Let us assume that there are $50,000$ Sens fans and $2,500,000$ leafs fans at the start (say 20 years ago) and we want to know what will be the steady state of the fans distribution. Let us first find the eigenvalues and eigenvectors of A. It is easy to see that $\lambda_1 = 1$ and $\lambda_2 = 0.6$ and the corresponding eigenvectors are $v_1 = (3, 1)$ and $v_2 = (1, -1)$, respectively. We know that $u_0 = (50000, 2500000)$ and we want to find u_k for large values of k, where $u_k = Au_{k-1}$. From the theory developed above we know that for constants c_1 and

c_2, $u_k = c_1(\lambda_1)^k v_1 + c_2(\lambda_2)^k v_2 = c_1 1^k v_1 + c_2(0.6)^k v_2$. The initial condition $u_0 = [500000, 2500000] = c_1[3,1] + c_2[1,-1]$ results in $c_1 = 750,000$ and $c_2 = -1,750,000$. As $k \to \infty$, $0.6^k \to 0$. Thus, $\lim_{k\to\infty} u_k = 750,000[3,1]$ or there are $2,250,000$ Sens fans and $750,000$ Leafs fans in the steady state.

The above examples leads to the following abstraction. Assume that all the entries of a Markov matrix A, or of some finite power of A, i.e., A^k for some fixed integer $k > 0$, are strictly > 0. These conditions imply that A corresponds to an irreducible aperiodic Markov chain M. (Recall that in an irreducible chain M, for any pair of states i and j, it is always possible to go from state i to state j in finite number of steps with positive probability. Informally, period of a state i is the greatest common divisor of all possible number of steps it takes the chain to return to the state i starting from i. M is aperiodic if the GCD is 1 for period of each of the states in M.) As a consequence of the Perron-Frobenius theorem from linear algebra it will turn out that almost always (a) the largest eigenvalue 1 of A will be unique, (b) all other eigenvalues of A have magnitude strictly smaller than 1, (c) all the coordinates of the eigenvector v_1 corresponding to the eigenvalue 1 are > 0, and (d) the steady state corresponds to the eigenvector v_1.

4.7.2 *PageRank*

Now a days we cannot imagine a life without the www. Google, Safari, Yahoo, ... are among the various search engines that search the internet to answer our web queries on a wide range of topics. In this section, we sketch how the ranking of the web-pages is done by the page rank algorithm from the founders of Google [5] (see also [6]). Ranking assigns a real number to each web-page. The higher the number, the more important the page is. Since the web is extremely large, the ranking cannot be done manually. First we will provide a very simple model of the web and see how the Markov matrices can help us to rank the web pages.

[5] Sergey Brin and Lawrence Page. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 56(18):3825–3833, 2012

[6] Sergey Brin, Rajeev Motwani, Lawrence Page, and Terry Winograd. What can you do with a web in your pocket? *IEEE Data Eng. Bull.*, 21(2):37–47, 1998

Consider the web as a directed graph $G = (V, E)$ defined as follows. Each web-page is a vertex of G. If a web-page u points (links) to the web-page v, there is a directed edge from u to v. The weight of an edge uv is $\frac{1}{\text{out-degree}(u)}$. Assume $V = \{v_1, \ldots, v_n\}$. Define an $n \times n$ adjacency matrix M as follows.

For $1 \le i, j \le n$, $M(i,j) = \begin{cases} \frac{1}{\text{out-degree}(v_j)}, & \text{if } v_j v_i \in E \\ 0 & \textit{otherwise.} \end{cases}$

For example consider the following simple web-graph (Figure 4.7) and its associated matrix (Figure 4.8).

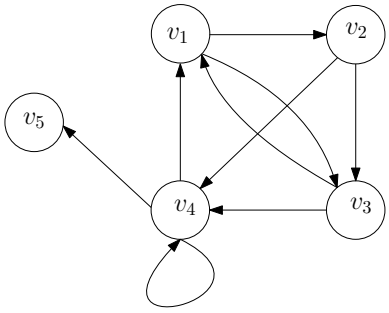


Figure 4.7: Web graph with 5 nodes.

$$M = \begin{array}{c} \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 0 & 1/2 & 1/3 & 0 \\ 1/2 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 1/3 & 0 \end{bmatrix} \end{array}$$

In the above example the node v_4 has out-degree 3 and hence

the weight on each of its outgoing edges $\{v_4v_1, v_4v_4, v_4v_5\}$ is $1/3$. Assume that a web-surfer starts the surfing at the web-page v_1. It has two outgoing edges and with equal probability ($= 1/2$) the surfer decides to follow one of them, say v_3. Now at v_3, he/she decides between two possible outgoing edges and picks one of them with equal probability, say v_4. At v_4 there are three possibilities and with equal probability (to return to v_1, to stay at v_4, or to advance to v_5) the surfer chooses to go to v_5. Node v_5 has no outgoing edges (its column is zero) and hence the web surfer is stuck. To overcome this, we say that the web-surfer jumps to a random page and restarts the whole process. This can be reflected by modifying the above matrix to the following:

$$Q = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1/2 & 1/3 & 1/5 \\ 1/2 & 0 & 0 & 0 & 1/5 \\ 1/2 & 1/2 & 0 & 0 & 1/5 \\ 0 & 1/2 & 1/2 & 1/3 & 1/5 \\ 0 & 0 & 0 & 1/3 & 1/5 \end{bmatrix} \end{array}$$

Note that M corresponding to Figure 4.8 is updated to reflect that from node v_5, with equal probability, the surfer will go to any of the five web-pages.

The creators of Google, in addition to the above modifications, suggested the following. During the surfing, the web-surfer at each of the node (i.e., a web-page) flips a coin. If the outcome is Heads, it follows the outlined approach using the matrix Q. But if the outcome is tails, it 'teleports' to a page, chosen uniformly at random among all the webpages, and continues the surfing from there. Let the probability of heads be α, then we can express the transition matrix as $K = \alpha Q + \frac{1-\alpha}{n} E$, where E is an $n \times n$ matrix and all of its entries are 1. We make a few remarks about K. Each of its entries > 0, and the entries within each column sums to 1.

Hence K is Markov matrix corresponding to a aperiodic irreducible Markov chain. Its largest eignenvalue is 1 and its corresponding eigenvector has positive entries, they add to 1, and corresponds to the steady state of K. Thus, the values in this eigenvector corresponds to the page rank of the web-pages.

Note that for the purpose of the computation of the page ranks, since K is an extremely large matrix, it is not advisable to compute its eigenvector corresponding to its principal eigenvalue directly. This requires executing Gaussian elimination and it has relatively large computational complexity. The computational issues are addressed by exploiting the fact that Q is extremely sparse and E is a special

Theorem 4.7.2 (**Perron-Frobenius theorem** [136]) *Let A be a square real matrix such that all of its entries are > 0. Then, all the coordinates of the eigenvector corresponding to the largest eigenvalue are strictly positive.*

Observe that the underlying directed graph is biconnected, and GCD of the periods is 1 for each node as there is a self-loop at each node.

matrix. For a vector $v = (1/n, \ldots, 1/n)$ corresponding to the uniform probability distribution of a random web surfer initially, the computation of $Kv = \alpha Qv + \frac{1-\alpha}{n} Ev$ can exploit the sparsity of Q and the properties of E. Similarly $K^2 v = K(Kv) = Kv'$ has a similar computational flavour as we are again multiplying K by a vector v'. Thus, we can repeatedly compute this product and stop when we think the successive vectors are very close to each other and the computation has converged. Hopefully, the resulting vector represents the steady state and we can deduce the page rank of each of the web page.

4.8 Bibliography

Acknowledgements to wonderful textbooks and/or video lectures of Gilbert Strang [7], Alan Tucker [8], R. Vittal Rao [9], and Jim Carrel [10]. Exercises on approximating the product of two matrices are based on Gilbert Strang [11] and Drineas and Mahoney [12].

[7] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, Wellesley, MA, fifth edition, 2016

[8] A. Tucker. *Linear algebra: an introduction to the theory and use of vectors and matrices*. Macmillan Publishing Company, 1993

[9] Vittal Rao. Advanced Matrix Theory and Linear Algebra for Engineers. https://nptel.ac.in/syllabus/111108066/, 2019. [Online; accessed 2-May-2019]

[10] James Carrel. Fundamentals of Linear Algebra. https://www.math.ubc.ca/~carrell/NB.pdf, 2005. [Online; accessed 2-May-2019]

[11] G. Strang. *Linear Algebra and Learning from Data*. Wellesley-Cambridge Press, 2019

[12] Petros Drineas and Michael W. Mahoney. Lectures on randomized numerical linear algebra. *CoRR*, abs/1712.08880, 2017

4.9 Exercises

4.1 *Let A be a $n \times n$ real symmetric matrix of rank $r < n$. What can we say about the four fundamental subspaces of A.*

4.2 *Find the basis and dimensions for all the four fundamental spaces associated with the following matrices: $\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \end{bmatrix}$ and $\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 6 \end{bmatrix}$.*

4.3 *Show that every rank one $m \times n$ matrix A can be expressed as a product two vectors $x = (x_1, x_2, \ldots, x_m)$ and $y = (y_1, y_2, \ldots, y_n)$, i.e., $A = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_m \end{bmatrix} \begin{bmatrix} y_1 & y_2 & \ldots & y_n \end{bmatrix}$*

4.4 *Let $q_1, \ldots, q_n$ be a set of n independent orthonormal vectors in $\mathbb{R}^m$. Let Q be a $m \times n$ matrix, where its i-th column is the vector q_i, for $i = 1, \ldots, n$. Show that $Q^T Q = I$, where I is $n \times n$ identity matrix. Show that if $m = n$, then $Q^{-1} = Q^T$. That is, the inverse of a square matrix of orthonormal vectors is its transpose.*

4.5 *Let A be a $m \times n$ matrix where all its columns are linearly independent. Show that $A^T A$ is invertible. (Hint: One way to show that a matrix B is invertible is to show that $Bx = 0$ only for $x = 0$.)*

4.6 *Let A be a square matrix, where each of its diagonal entry is strictly larger than the sum of all other entries within that row. Show that A is invertible.*

4.7 *Show that $n \times n$ matrix A is invertible if and only if $Ax = 0$ has a unique solution.*

4.8 *Show that $n \times n$ matrix A is invertible if and only if 0 is not an eigenvalue of A.*

4.9 *Let A be an invertible matrix and λ be an eigenvalue of A and let v be the corresponding eigenvector. Show that $1/\lambda$ is an eigenvalue for A^{-1} and its corresponding eigenvector is v.*

4.10 *Show that an $n \times n$ real matrix S is symmetric if and only if $u^T Av = v^T Au$ for all $u, v \in \mathbb{R}^n$.*

4.11 *Consider the matrix* $A = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix}$.

Answer the following:

1. *Find the Eigenvalues and Eigenvectors of A. Show your work.*
2. *What are (approximately) the Eigenvalues and Eigenvectors of A^2 and A^{100}?*

4.12 *Let A be a square matrix of dimension $n \times n$, where each entry of A is a real number. Show that the products AA^T and $A^T A$ are symmetric matrices? Is $AA^T = A^T A$ for all (real) square matrices? Justify your answer.*

4.13 *For a square matrix A show that*

1. *The product of its eigenvalues equals the determinant of A. (Hint: Consider the characteristic polynomial and set $\lambda = 0$.)*
2. *The sum of its eigenvalues equals the sum of the diagonal entries of A (called the* trace *of A).*
3. *The eigenvalues of A are same as that of A^T. Do they have the same eigenvectors?*

4.14 *Answer the following:*

1. *Compute Eigenvalues of matrix* $B = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 2 & 3 \\ 1 & 3 & 1 \end{bmatrix}$.

   Notice that B is symmetric, i.e., $B = B^T$ (B is same as its transpose B^T). Do you notice something about the Eigenvalues of B? Try a couple more symmetric matrices to test your hypothesis.

2. *Compute Eigenvalues of matrix* $C = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 2 & 3 \\ 3 & 4 & 4 \end{bmatrix}$.

   Notice that this is not a full rank matrix, and one of its Eigenvalues is zero. Is it true in general if the matrix C is not of full rank then some of its Eigenvalues are zeros? How many of them will be zeros, if any?

4.15 *What are eigenvalues and eigenvectors of an identity matrix.*

4.16 *Show that for a real symmetric matrix* $S = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$*, its eigenvectors are* $x_1 = \left[\begin{smallmatrix} b \\ \lambda_1 - a \end{smallmatrix}\right]$ *and* $x_2 = \left[\begin{smallmatrix} \lambda_2 - d \\ c \end{smallmatrix}\right]$ *where* λ_1 *and* λ_2 *are the eigenvalues, respectively.*

4.17 *For a square matrix* A *, show that if all its eigenvalues are distinct, then its corresponding eigenvectors are linearly independent. First show that for a pair of distinct eigenvalues* $\lambda_1 \neq \lambda_2$*, their corresponding eigenvectors* v_1 *and* v_2 *are linearly independent. (Note that* v_1 *and* v_2 *are linearly independent if* $c_1 v_1 + c_2 v_2 = 0$ *then* $c_1 = c_2 = 0$*.)*

4.18 *Assume that a square matrix* A *can be expressed as* $A = X \Lambda X^{-1}$*, where* Λ *is a diagonal matrix consisting of eigenvalues of* A*, and the columns of* X *are the corresponding (linearly independent) eigenvectors. What are are eigenvalues and eigenvectors of* A^{-1}*? Can any of the eigenvalues of* A *equals to* 0*?*

4.19 *Let* A *be a square matrix and let* $p(\lambda) = det(A - \lambda I) = 0$ *be its characteristic polynomial. We are interested to find out whether all the roots (eigenvalues) of* $p(\lambda)$ *are distinct or not without actually computing its roots. Show that this can be achieved by finding the GCD of* $p(x)$ *and its derivative* $p'(x)$*.*

4.20 *Let* $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$*. What are the conditions on* a, b, c*, and* d*, so that* A *is positive definite. What about positive semi definite?*

4.21 *Recall that a symmetric matrix* A *is positive definite if for all non-zero vectors* x*,* $x^T A x > 0$*. For a symmetric positive definite matrix* A *show that*

1. *All its eigenvalues* > 0*.*
2. *All its pivots, in the row reduced echelon process, are* > 0 *in the*
3. *Determinants of all the upper left submatrices are positive.*
4. *All diagonal elements of* A *are* > 0*.*

4.22 *Let* A *be a* $m \times n$ *real matrix. Show that* $A^T A$ *is a symmetric positive semi definite matrix.*

4.23 *Assume all the matrices involved are* $n \times n$ *matrices. Answer the following:*

1. *If* A *is symmetric positive definite. Is* A^{-1} *symmetric positive definite?*
2. *If* A *and* B *are symmetric positive definite matrices. What about* $A + B$*?*

4.24 *Let A be a real symmetric $n \times n$ matrix. Show that the following statements are squivalent.*

1. *A is positive semi-definite.*
2. *For any vector $x \in \Re^n$, $x^T Ax \geq 0$.*
3. *For some $n \times n$ matrix U, $A = U^T U$.*
   (Hint: Consider SVD of A.)

4.25 *Show that for a Markov matrix A, and an identity matrix I, $A - I$ is singular, i.e., the determinant of $A - I$ is 0. Show that the rows of $A - I$ are not independent and hence $det(A - I) = 0$. This implies that 1 is an eigenvalue of a Markov matrix.*

4.26 *Show that in an irreducible Markov chain all the states are recurrent.*

4.27 *In a Markov chain we say that a pair of states (i, j)* communicates *with each other if it is possible to reach from the state i to the state j with non-zero probability after a finite number of steps, and similarly it is possible to reach from the state j to the state i in a finite number of steps with non-zero probability. Show that 'communicates' is an equivalence relation. Show that an irreducible Markov chain has a unique equivalence class.*

4.28 *Suppose that the underlying graph of a Markov chain is a bipartite graph. Can this chain be aperiodic?*

4.29 *Let A be a square Markov matrix of dimension $n \times n$. Recall that in A, all entries are non-negative and the entries within a column sum to 1. Show that $\lambda = 1$ is an Eigenvalue of A. (Hint: Show that for a Markov matrix A, and an identity matrix I, $A - I$ is singular, i.e., the determinant of $A - I$ is 0. Show that the rows of $A - I$ are not independent and hence $det(A - I) = 0$. This implies that 1 is an eigenvalue of a Markov matrix. Note that there are many more ways to answer this question.)*

4.30 *Assume that the total population of the Province of Ontario is 14 million. Each resident of Ontario is either a fan of the Ottawa Senators or Maple Leafs Hockey Club. Each year, 5% fans of Senators switch their aligns to Leafs, and the 10% of Leafs fans switch their aligns to Senators. Suppose that this trend continues year after year, and the population of Ontario doesn't change, what will be the steady state of the number of fans of each of the Hockey Clubs? Is that affected by the initial number of fans each team has?*

4.31 *Let the Singular-Value Decomposition of a rank r matrix $[A]_{m \times n}$ be $A = U\Sigma V^T$. Show that the vectors $u_1, \ldots, u_r$, corresponding to the first r columns of U, form an orthonormal basis for the column space of A. (For any vector x of dimension n, consider $Ax = U\Sigma V^T x$. Observe that $\Sigma V^T x$ is a vector whose only non-zero components are among its first r components.)*

The following exercises will lead us to the *principal component analysis* (PCA). We will use the following notation. Let $p_1, p_2, \ldots, p_n \in \mathbb{R}^n$ denote a set of linearly independent orthonormal vectors that form a basis of $\mathbb{R}^n$. We arrange these vectors in a $n \times n$ matrix P, whose i-th column is the vector p_i. Let X be a $m \times n$ real matrix, where rows represents $n-$ dimensional vectors $x_1, x_2, \ldots, x_m$. Furthermore, we are given that each column of X is centred with mean 0, i.e., the sum of the entries in a column is 0.

In simple terms PCA is used for the following: Think of rows of X as objects and the columns as different (numeric) attributes of these objects. It is possible that there may be lot of redundant information in X. For example, a column of X may be a scalar multiple of some other column (e.g., one column represents prices in US$ and other in Canadian $ of the same item), or all the columns of X can be expressed as a linear combination of a very few columns, or it may be possible to project row vectors on to a different basis so that the properties of the objects are easy to deduce. In other words, PCA will help us to identify an orthonormal basis (dimensions) in $\mathbb{R}^n$ such that after projecting X onto the new basis, the covariance between the new columns is 0. Within each new dimension, the variance is maximized. Answer the following:

4.32 *We know that any vector $x \in \mathbb{R}^n$ can be expressed as a linear combination of orthonormal basis vectors $p_1, p_2, \ldots, p_n \in \mathbb{R}^n$, i.e., there are unique $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{R}$, such that $x = \sum_{i=1}^{n} \alpha_i p_i$. Show that for all $i = 1, \ldots, n$, $\alpha_i = \langle x \cdot p_i \rangle$, i.e., α_i is given by the dot product of vectors x and p_i.*

4.33 *Show that the matrix product XP results in mapping each row vector x_i to the coordinate system defined by the orthonormal basis $p_1, p_2, \ldots, p_n$.*

4.34 *Show that the matrix product $\hat{X} = XP$ has the property that sum of all the elements in each column is 0.*

4.35 *Covariance of two random variables A and B is defined as $\sigma_{AB} = E[(A - E[A])(B - E[B])]$. Show that $\sigma_{AB} = E[AB] - E[A]E[B]$.*

4.36 *Consider the product $\mathcal{C} = \frac{1}{m-1} X^T X$. Show the following:*

Dividing by $m - 1$ instead of m has to do with degrees of freedom. Don't ask why!

1. *$\mathcal{C}$ is a $n \times n$ symmetric matrix.*
2. *For $i = 1, \ldots, n$, the diagonal entry $\mathcal{C}_{ii}$ of $\mathcal{C}$ is the variance of the elements in the i-th column of X, i.e., the variance of $\{x_{1i}, x_{2i}, x_{3,}, \ldots, x_{mi}\}$.*
3. *ij-th entry $\mathcal{C}_{ij}$ is the covariance of the i-th and the j-th columns of X.*

4.37 *Consider the covariance matrix of $\hat{X}(= XP)$, i.e., the matrix $\frac{1}{m-1} \hat{X}^T \hat{X}$.*

1. *Show that $\frac{1}{m-1} \hat{X}^T \hat{X}$ is a square symmetric matrix.*

2. *Suppose we construct P by choosing the orthonormal eigenvectors $p_1, p_2, \ldots, p_n$ of $\mathcal{C}$ as its columns. Show that $\frac{1}{m-1}\hat{X}^T\hat{X} = P^T\mathcal{C}P$.*

3. *From Equation 4.2, conclude that $P^T\mathcal{C}P$ is a diagonal matrix.*

4. *Consider the projection of the vectors of X on to P, i.e., XP. Show that the covariance of any two distinct columns of $XP = 0$.*

The eigenvectors of $\mathcal{C} = \frac{1}{m-1}X^TX$ determine the principal directions/axes. For example, the largest variance among all possible directions after projecting all row vectors in X is given by the direction of the eigenvector corresponding to the largest eigenvalue of $\mathcal{C}$. Among all the directions that are orthogonal to the first, the direction given by the eigenvector corresponding to the second largest eigenvalue has the largest variance, and so on.

4.38 *Solve the following exercises.*

1. *Find the principal directions/axes of the row vectors of the matrix*

$$X = \begin{bmatrix} -2 & -6 \\ -1 & -3 \\ 0 & 0 \\ 1 & 3 \\ 2 & 6 \end{bmatrix}$$

Determine the projections of the row vectors onto the principal axes. Let p_1 be the unit vector corresponding to the largest eigenvalue of $\mathcal{C} = \frac{1}{m-1}X^TX$ and let p_2 be the unit vector corresponding to second largest eigenvalue. Any row vector of X can be expressed as linear combinations of p_1 and p_2. For example, the second row vector can be expressed as

$$\begin{bmatrix} -1 \\ -3 \end{bmatrix} = \alpha_1 p_1 + \alpha_2 p_2$$

Find α_1 and α_2. Observe that $\alpha_2 = 0$. Thus, by knowing α_1 and p_1, we can create the row vector. Do this for all the rows of X. What is the saving in the space if we store α's and p_1 instead of X? Also, let p_1 and λ_1 be the principal eigenvalue and eigenvector of $\mathcal{C}$. What can you say about $\lambda_1 p_1 p_1^T$? How well it approximates $\mathcal{C}$?

2. *Repeat the exercise for*

$$X = \begin{bmatrix} 1/\sqrt{2} & 0 \\ 2/\sqrt{2} & 3/\sqrt{2} \\ 3/\sqrt{2} & 2/\sqrt{2} \\ 4/\sqrt{2} & 5/\sqrt{2} \\ 5/\sqrt{2} & 4/\sqrt{2} \end{bmatrix}$$

What can you say about $\lambda_1 p_1 p_1^T$ in relation to X^TX?

3. *Repeat the exercise for*

$$X = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 0 & 1 \\ 0 & 2 & -1 \\ 0 & -2 & -1 \end{bmatrix}$$

What is the most dominant direction?

Next we consider relationship between PCA and SVD's for a $m \times n$ matrix X. We assume that X is centred, i.e., the sum of elements within any column is 0. We will establish that PCA is a special case of SVD. Recall that the covariance matrix of X is given by $\mathcal{C} = \frac{1}{m-1} X^T X$.

4.39 *Answer the following.*

1. *Given that $\mathcal{C}$ is a square symmetric matrix, using Theorem 4.4.6 show that $\mathcal{C} = PDP^T$, where D is a diagonal matrix made of eigenvalues of $\mathcal{C}$ and P consists of eigenvectors of $\mathcal{C}$.*

2. *Let the SVD decomposition of $X = U\Sigma V^T$ (see Theorem 4.5.5). Show that $\mathcal{C} = \frac{1}{m-1} X^T X = V \frac{\Sigma^2}{m-1} V^T$.*

3. *Show that given SVD, we can obtain the PCA of X by observing that the eigenvectors (principal directions) are given by $P = V$ and the eigenvalues of the covariance matrix $\mathcal{C}$ are given by $\frac{1}{m-1}\Sigma_{ii}^2$. Moreover, show that $XV = U\Sigma$ results in the projection of X on to the orthonormal basis given by the vectors in V. Therefore, given SVD we can easily obtain the PCA of X.*

4. *Compute SVD's of matrices in the previous question and verify the claims.*

The following exercises will help us understand *least squares approximations*. The problem can be described as follows. Given a set of m points in plane, find a line that passes through these points. In case there is no such line, find the line that is'close' to these points. This is a typical scenario when the system of equations $Ax = b$ has no solution and we are trying to find a 'best' possible solution $\hat{x}$ such that $A\hat{x} \approx b$. This happens when, for example, we have many more equations than variables (i.e., $m > n$ for a $m \times n$ matrix A defining the system $Ax = b$). We will formalize these notions through a series of exercises.

4.40 *Let $p = (p_x, p_y)$ and $q = (q_x, q_y)$ be two distinct points in plane. Show that there is a unique line passing through them. Let the equation of*

the line by $y = C + Dx$, where C and D are constants. Note that we can express this configuration in a system of equation $Ax = b$ as follows.

$$\begin{bmatrix} 1 & p_x \\ 1 & q_x \end{bmatrix} \begin{bmatrix} C \\ D \end{bmatrix} = \begin{bmatrix} p_y \\ q_y \end{bmatrix}$$

Using the matrix notation, find the line passing through the points $(2,5)$ and $(3,6)$.

4.41 *In each of the cases determine whether there is a line passing through the points.*

1. $(2,5)$, $(3,6)$, $(4,7)$.
2. $(2,5)$, $(3,6)$, $(4,8)$.

Write the corresponding system of equations in the matrix format $Ax = b$. Notice that A is 3×2 matrix. Answer the following:

1. *Compute column spaces for A, i.e., space determined by the span of column vectors of A.*
2. *Determine if b belongs to the column space of A.*
3. *Can the membership of b in the column space of A help in determining if $Ax = b$ has a solution?*

4.42 *Consider a system of equations $Ax = b$ that has no solution. Then b is not in the column space of A. Answer the following. (It is easy to visualize this in three dimensions ($m = 3$) and then think of higher dimensions.)*

1. *Show that the closest point to b in the column space of A is the point p obtained by projecting b onto the column space of A.*
2. *Since p is in the column space of A, show that there is a vector $\hat{x}$ such that $A\hat{x} = p$, i.e., p can be expressed as linear combinations of columns of A.*
3. *Define $e = b - p$. This is the error vector. Show that the least square error is given by $e^T e$. Show that if b is in the column space of A then $e^T e = 0$, that is we can find a solution to the system $Ax = b$.*
4. *Let $a_1, \ldots, a_n$ be the columns of A. Show that e is perpendicular to each of the column vectors, i.e., $e \cdot a_i = 0$.*
5. *Show that $A^T e = 0$.*
6. *Show that $A^T A\hat{x} = A^T b$.*
7. *Show that if the columns of A are independent (i.e., $a_1, \ldots, a_n$ are linearly independent) then $A^T A$ is invertible.*

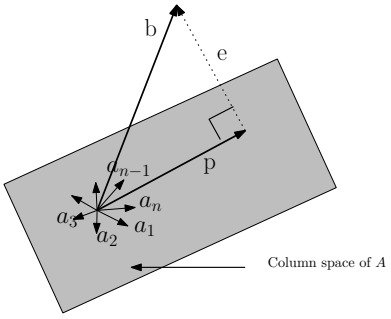


Figure 4.9: Projection of b onto the column space of A.

8. *Assume that A^TA is invertible. Define the* projection matrix $\mathcal{P} = A(A^TA)^{-1}A^T$. *Show that $p = \mathcal{P}b$, i.e., the projection of b onto the column space of A is given by $\mathcal{P}b$.*

9. *Show that $\mathcal{P}^2 = \mathcal{P}$. (The interpretation is that once b is projected in the column space (as p), the next projection doesn't do anything.)*

10. *Suppose A is a square invertible matrix ($m = n$ and $AA^{-1} = A^{-1}A = I$) then show that $\mathcal{P}$ is an identity matrix. Note that in this case the columns of A spans whole of $\mathbb{R}^n$ and hence any vector $b \in R^n$ is a linear combination of the column vectors.*

4.43 *Find the best line that passes through the following sets of points. Following the notation of the previous problem, find p, $\hat{x}$, $\mathcal{P}$, e, and e^Te.*

1. $(2,5)$, $(3,6)$, $(4,8)$.

2. $(1,2), (-1,-2), (3,0), (2,4)$.

Following set of exercises will help in approximating the product of two matrices. Let A be a $m \times n$ matrix and B be a $n \times p$ matrix. We will denote columns of A by A_j, $j = 1, \ldots, n$ and rows of B by B_j^T, for $j = 1, \ldots, n$. The Frobenius norm of a matrix A is defined as $||A||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} A_{ij}^2}$.

4.44 *Show that $AB = \sum_{j=1}^{n} A_jB_j^T$. Apply this to the product of $A = \begin{bmatrix} 1 & 2 \\ 2 & 0 \\ 3 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 2 & 1 \\ 3 & 2 \end{bmatrix}$.*

4.45 *Let $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$. Show that $||xy^T||_F = ||x||_2||y||_2$. In words, for any two vectors x and y, show that the Frobenius norm of the matrix given by the product of xy^T is the same as the product of their Euclidean norms.*

Note that the probabilities of selecting the columns of A (or the rows of B) are based on their norms - higher the norm better are the chances of selecting them. The division by $\sqrt{cp_{k(t)}}$ biases the entries in X and Y appropriately so that XY is close to AB. We will explore these in the following exercises.

4.46 *Answer the following.*

1. *Show that after executing the randomized algorithm, $XY = \sum_{t=1}^{c} \frac{1}{cp_{k(t)}} A_{k(t)}B_{k(t)}^T$.*

Algorithm 4.1: Randomized Algorithm for approximating AB.
Input: Matrices $[A]_{m\times n}$ and $[B]_{n\times p}$
Output: Approximation to AB as the product of two matrices $[X]_{m\times c}$ and $[Y]_{c\times p}$

1 $\mathcal{C} = \sum_{i=1}^{n} ||A_i||_2||B_i^T||_2$
2 For $k = 1, \ldots, n$, compute $p_k = \frac{||A_k||_2||B_k^T||_2}{\mathcal{C}}$
3 **foreach** $t \in \{1, c\}$ **do**
4 Pick a number $k(t) \in \{1, \ldots, n\}$ independently with probability $p_{k(t)}$
5 Set the t-th column of X as $X_t = \frac{1}{\sqrt{cp_{k(t)}}} A_{k(t)}$
6 Similarly, set the t-th row of Y as $Y_t^T = \frac{1}{\sqrt{cp_{k(t)}}} B_{k(t)}^T$
7 **end**
8 **return** XY

2. *Show that the expected value $E[(XY)_{ij}] = (AB)_{ij}$. For this, you can define random variables $X_1, X_2, \ldots, X_c$, where X_t refers to ij-th entry of the matrix $\frac{1}{cp_{k(t)}} A_{k(t)} B_{k(t)}^T$, i.e., $X_t = (\frac{1}{cp_{k(t)}} A_{k(t)} B_{k(t)}^T)_{ij}$. First show that $E[X_t] = \frac{1}{c}(AB)_{ij}$ and then use the independence of X_t's to show that $E[(XY)_{ij}] = E[X_1 + \cdots + X_c] = (AB)_{ij}$.*

3. *Show that the variance $V[(XY)_{ij}] \leq \frac{1}{c}\sum_{k=1}^{n} \frac{A_{ik}^2 B_{kj}^2}{p_k}$. Using the notation of previous exercise, first show that $V[(XY)_{ij}] = V[X_1 + \cdots + X_c]$, $V[X_t] \leq E[X_t^2]$, and $E[X_t^2] = \sum_{k=1}^{n} \frac{A_{ik}^2 B_{kj}^2}{c^2 p_k}$.*

4. *Show the following:*

   (a) $V[(XY)_{ij}] = E[(XY - AB)_{ij}^2]$

   (b) $E[||AB - XY||_F^2] = \sum_{i=1}^{m}\sum_{j=1}^{p} V[(XY)_{ij}]$

   (c) $E[||AB - XY||_F^2] \leq \frac{1}{c}\sum_{k=1}^{n} \frac{1}{p_k}||A_k||_2^2||B_k^T||_2^2$

5. *Show that the function $f(p_1, \ldots, p_n) = \sum_{k=1}^{n} \frac{1}{p_k}||A_k||_2^2||B_k^T||_2^2$ is minimized if we choose $p_k = \frac{||A_k||_2||B_k^T||_2}{\mathcal{C}}$. (We can use the Lagrange multiplier's as we are trying to minimize $f(p_1, \ldots, p_n)$ subject to the constraint that $\sum_{k=1}^{n} p_k = 1$. Define the function $g(p_1, \ldots, p_n, \lambda) = \sum_{k=1}^{n} \frac{1}{p_k}||A_k||_2^2||B_k^T||_2^2 - \lambda(\sum_{k=1}^{n} p_k - 1)$ and take partial derivatives of g with respect to p_k's to obtain the values for p_k's that minimize f.)*

6. *Show that if we choose* $p_k = \frac{||A_k||_2||B_k^T||_2}{\mathcal{C}}$ *as in the algorithm then we obtain* $E[||AB - XY||_F^2] \leq \frac{1}{c}\mathcal{C}^2$.

4.47 *Let* $A = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$ *and* $B = \begin{bmatrix} 2 & 1 \\ 0 & 3 \\ 1 & 2 \end{bmatrix}$. *Execute the above algorithm for different values of* $c = 1, 2, 3$ *and evaluate* $||AB - XY||_F$.

The following exercise helps us to understand Rayleigh quotients. They relate eigenvalues to an optimization problem and are instrumental in partitioning graphs, a topic that will be addressed in Chapter 7.

4.48 *Let* S *be a* $n \times n$ *real symmetric matrix and let* $v_1, \ldots, v_n$ *be its* n *orthonormal eigenvectors corresponding to its* n *eigenvalues* $\lambda_1 \geq \ldots \geq \lambda_n$, *respectively. Let* $x \in \Re^n$ *be any vector, and we can express it as linear combination of vectors* $v_1, \ldots, v_n$, *i.e.,* $x = \alpha_1 v_1 + \ldots + \alpha_n v_n$, *for some constants* $\alpha_1, \ldots, \alpha_n$. *Answer the following:*

1. *Show that* $x^T x = \alpha_1^2 + \ldots + \alpha_n^2$.
2. *Show that* $x^T S x = \lambda_1 \alpha_1^2 + \ldots + \lambda_n \alpha_n^2$.
3. *For a vector* $x \in \Re^n$, *consider the expression* $f(x) = \frac{x^T S x}{x^T x}$. *Show that for any eigenvector* v_i *of* S, $f(v_i) = \lambda_i$.
4. *Show that* $\max_{x \in \Re^n} \frac{x^T S x}{x^T x} = \lambda_1$.
5. *Show that* $\max_{x \in \Re^n, x \perp v_1} \frac{x^T S x}{x^T x} = \lambda_2$. *(Maximum is over the set of vectors* $x \in \Re^n$ *that are orthogonal to* v_1.*)*
6. *In general, show that if we restrict ourselves to vectors* $x \in \Re^n$, *such that* x *is orthogonal to* $v_1, \ldots, v_{k-1}$, *then* $\max_{x \in \Re^n, x \perp v_1, \ldots, x \perp v_{k-1}} \frac{x^T S x}{x^T x} = \lambda_k$.

4.49 *Let* A *be* $m \times n$ *real matrix. Consider* $X = A^T A$. *Note that* $A^T A$ *is* $n \times n$ *symmetric matrix. Let* $U = \{u_1, \ldots, u_n\}$ *be* n *orthonormal eigenvectors of* $A^T A$ *corresponding to eigenvalues* $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_n \geq 0$. *Answer the following.*

1. *Show that* $A^T A u_i = \lambda_i u_i$.
2. *Show that* $u_i^T A^T A u_i = \lambda_i$.
3. *Recall that for a vector* x, *its norm* $||x||_2 = \sqrt{x^T x}$. *Show that* $||Au_i||_2 = \sqrt{\lambda_i}$.
4. *Using Exercise 4.48, show that* $\max_{x \in \Re^n, ||x||_2 = 1} ||Ax||_2 = \sqrt{\lambda_1}$.

5. *Conclude that the spectral norm of a matrix A, defined by $\max_{x \in \Re^n, ||x||_2=1} ||Ax||_2$, is its largest singular value $\sigma_1 = \sqrt{\lambda_1}$. (Recall that the singular values are the diagonal entries in the matrix Σ in the singular-value decomposition $A = U\Sigma V^T$.)*

4.50 *Consider the adjacency matrix of a complete graph on n-vertices. Show that its eigenvalues are $\lambda_1 = n - 1$ and $\lambda_2 = \ldots = \lambda_n = -1$.*

4.51 *Consider the adjacency matrix of complete bipartite graph K_{mn}. Show that it has exactly two non-zero eigenvalues $\sqrt{mn}$ and $-\sqrt{mn}$.*

4.52 *Let Q be an orthonormal $n \times n$ real matrix. Compute its norms: $||Q||_F, ||Q||_2$, and $||Q||_N$.*

4.53 *Let A be $n \times n$ real matrix and let Q be an orthonormal $n \times n$ real matrix. Show that $||QA||_F = ||A||_F, ||QA||_2 = ||A||_2$, and $||QA||_N = ||A||_N$.*

4.54 *Let A be $n \times n$ real matrix and let Q be an orthonormal $n \times n$ real matrix. Show that $||QAQ^T||_F = ||A||_F, ||QAQ^T||_2 = ||A||_2$, and $||QAQ^T||_N = ||A||_N$.*

4.55 *Consider a utility matrix M that is typically used in recommender systems. The rows represents users, the columns represents items, and the entry M_{ij} in the matrix represents the ranking the user i gives to the item j. Assume that these ranks are non-negative integers. Consider the SVD of $M = U\Sigma V^T$. Assume that rank of matrix M is r, it has m rows (users) and n columns (items). From SVD, we know that U is an $m \times r$ and V is $n \times r$ matrices consisting of orthonormal columns, and Σ is a $r \times r$ diagonal matrix. The interpretation associated to r in the context of recommender system is that it represents the latent concepts that connects rows with columns. For example, if the utility matrix represents rankings of movies for users, the concepts may be: type of movie - action, sci-fi, classic, autobiographical; director and cast; ... Answer the following:*

1. *Consider the ranking matrix $M = \begin{bmatrix} 4 & 4 & 0 & 0 \\ 5 & 4 & 0 & 1 \\ 1 & 0 & 3 & 4 \\ 1 & 0 & 4 & 5 \\ 0 & 0 & 5 & 5 \end{bmatrix}$ that represents 5 book readers as rows, and 4 books as columns, and the entry in the matrix represents the ranking of books by readers. The rankings are integers in $[0, 5]$. Compute the SVD of $M = U\Sigma V^T$.* Use a software package.
2. *Deduce that rank of M is 3. Implicitly there are three concepts, and the strength of these concepts are 10.97, 8.38, and 1.07, respectively. These are the entries in the matrix Σ.*

3. *Observe that matrix* $U = \begin{bmatrix} .14 & -.64 & -.47 \\ .23 & -.71 & .32 \\ .45 & .07 & .42 \\ .58 & .12 & .30 \\ .61 & .23 & -.62 \end{bmatrix}$ *relates readers to concepts. From the magnitude of the values in the matrix* $U\Sigma$*, give some interpretation of the strength of the three concepts for each reader.*

4. *Similar, matrix* $V = \begin{bmatrix} .25 & -.70 & .43 \\ .13 & -.64 & -.55 \\ .61 & .22 & -.56 \\ .73 & .16 & .42 \end{bmatrix}$*, relates concepts with books. Give a similar interpretation of strength of concepts for each of the books.*

5. *For the third readers row* $R_3 = (1, 0, 3, 4)$*, show that the entries in* R_3V *maps this reader to the concept space.*

6. *Suppose a new reader enters the system who has only read one book, say 2nd book, and has given it a ranking of* 4*. The row corresponding to this user in the matrix will be* $q = (0, 4, 0, 0)$*. Our task is to recommend this reader other possible books that they may like. Note that* qV *maps this reader to the concept space. What will be the interpretation of* $(qV)V^T$*? Based on the entries in* $(qV)V^T$*, what books we will like to recommend this reader?*

7. *What will be the best rank* 2 *approximation of* M *with respect to Frobenius norm? Call the resulting matrix* M'*. How much energy will be lost? Calculate the affect of working with* M' *instead of* M *in making recommendation with respect to the reader* q*. Does the recommendation change?*

4.56 *Suppose you have two boxes - one colored red and the other colored blue. The red box contains two red balls, and similarly, the blue box contains two blue balls. In each step, with equal probability, you select a box. If the box is not empty, you randomly choose one of the balls from that box and place that in the other box. You repeat this process till the blue box contains both the red balls and the red box has both the blue balls. Represent this process using a Markov chain. What are the states and the transition probabilities in this Markov chain? Which states are recurrent and which are transient?*

4.57 *Assume that we have a star graph* $S = (V, E)$ *on* $n + 1$ *nodes. It consists of one central node (a resort), say* r*, and* n *satellite nodes (say attractions) only connected to the central node. Assume that the satellite nodes are labelled* $1, 2, \ldots, n$*. Thus* $V = \{r, 1, 2, \ldots, n\}$ *and* $E = \{(r, 1), (r, 2), \ldots, (r, n)\}$*. Each morning, we wake up at the central node, choose a satellite node uniformly at random, hike to that node during the day, and return to the central node by evening. We repeat this process*

till we visit all the n attractions. Note that the choice of which attraction to visit on the i-th day is independent of which attractions we have visited on days $1, \ldots, i-1$. *Each morning we select the attraction uniformly at random among the n attractions and may land up visiting the same attraction we have seen before. Let* X_n *represent the number of attractions that have been visited by the end of the day n. (Clearly,* $X_1 = 1$, *and* $X_2 = 1$ *or* 2 *depending on whether we visit the same attraction on Day 2 as that on Day 1 or a new one.) Answer the following questions:*

1. *Design a Markov chain where you may consider* X_n *to be the states for* $n = 1, 2, \ldots$. *Or, you may consider* Y_i *to be the state that indicates that i different attractions have been visited so far. Then we will only have a finite number of sites* $Y_0, Y_1, \ldots, Y_n$.

2. *What are the transition probabilities, i.e., write an expression for* $Pr(X_n = j | X_{n-1} = i)$, *where* $i, j \in \{1, \ldots, n\}$. *You have think something similar with respect to* Y_i*'s.*

3. *Which states are recurrent? What are the periods of various states?*

4. *What is the expected number of days required to visit all the attractions?*

5
Minimum Spanning Trees

We will focus on

1. Cut, light and heavy edges.
2. Kruskal's MST algorithm
3. Prim's MST algorithm
4. Borůvka's algorithm
5. Randomized algorithm
6. MST verification algorithms

5.1 Minimum Spanning Trees

Let $G = (V, E)$ be an undirected connected graph with a cost function w mapping edges to positive real numbers. A spanning tree is an undirected tree connecting all vertices of G. The *cost* of a spanning tree is equal to the sum of the costs of the edges in the tree. A *minimum* spanning tree (MST) is a spanning tree whose cost is minimum over all possible spanning trees of G. It is easy to see that a graph may have many MSTs with the same cost (e.g., consider a cycle on 4 vertices where each edge has a cost of 1; deleting any edge results in a MST, each with a cost of 3).

As in the CLRS book[1], we will describe the two main algorithms for building MSTs, Kruskal's and Prim's. Both of these algorithms are greedy algorithms and are based on the following generic algorithm (Algorithm 9.1). The algorithm maintains a subset of edges A, which is a subset of some MST of G.

Intuitively this algorithm is straight-forward except for two pressing questions: What is a safe edge, and how do we find one? To answer these questions, we first need a few definitions.

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009

Algorithm 5.1: Generic-MST
Input: Graph G, cost function w
Output: A minimum spanning tree of G

```
1 A ← 0
2 while A ≠ MST do
3 |   find a safe edge {u, v} for A
4 |   A ← A ∪ {u, v}
5 end
6 return A
```

Cut A cut $(S, V \setminus S)$ of $G = (V, E)$ is a partition of vertices of V.

Edge crossing a cut An edge $(u, v) \in E$ crosses the $cut(S, V \setminus S)$ if one of its end point is in the set S and the other one in the set $(V \setminus S)$.

Cut respecting A A cut $(S, V \setminus S)$ respects the set A if none of the edges of A crosses the cut.

Light edge An edge which crosses the cut and which has the minimum cost of all such edges.

Theorem 5.1.1 *Let A be a subset of the edges of E which is included in some MST, and let $(S, V \setminus S)$ be a cut which respects A. Let (u, v) be a light edge crossing the cut $(S, V \setminus S)$, then (u, v) is safe for A.*

Proof. Assume that T is a MST that includes A (similarly, you may think of A as being a subset of the edges of T, or being "the makings of" a MST). If T includes the edge (u, v) then (u, v) is safe for A. If T does not include (u, v), then we will show that there is another MST, T', that includes $A \cup \{(u, v)\}$, and this will prove that $\{(u, v)\}$ is safe for A. Since T is a spanning tree, there is a path, say $P_T(u, v)$, from the vertex u to vertex v in T. By inserting the edge (u, v) in T we create a cycle. Since u and v are on different sides of the cut, there is at least one edge $(x, y) \in P_T(u, v)$ that crosses the $cut(S, T \setminus S)$. Moreover $(x, y) \notin A$, since the cut respects A. But the cost of the edge (x, y) is at least the cost of the edge (u, v), since edge (u, v) is a light edge crossing the cut. Construct a new tree T' from T by deleting the edge (x, y) in T and inserting the edge (u, v). Observe that the cost of the tree T' is at most the cost of the tree T since the cost of (x, y) is at least the cost of (u, v). Moreover $A \cup \{(u, v)\} \subset T'$ and $(x, y) \notin A$, hence edge (u, v) is safe for A. ∎

The above theorem leads to the following corollary, where we fix a particular cut (i.e. the $cut(C, V \setminus C)$).

Corollary 5.1.2 *Let $A \subset E$ be included in some MST. Consider the forest consisting of $G_A = (V, A)$, i.e., the graph with the same vertex set as G but restricted to the edges in A. Let $C = (V_C, E_C)$ be a connected component of G_A. Let (u, v) be a light edge connecting C to another connected component in G_A, then (u, v) is safe for A (See Figure 5.1).*

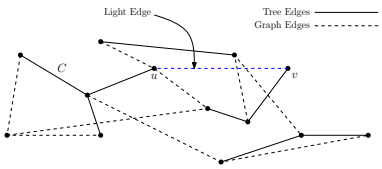


Figure 5.1: An example of Corollary 5.1.2. The edge (u, v) connects C to some other component of G_A and is a light edge; it is therefore safe to add to the MST.

5.2 *Kruskal's Algorithm for MST*

Proposed by Kruskal in 1956, this algorithm follows directly from Corollary 5.1.2. Here are the main steps. To begin with the set A consists of only isolated vertices, and no edges (so, $|V|$ "connected" components in all).

1. Sort the edges of E in non-decreasing order with respect to their cost.
2. Examine the edges in order; if the edge joins two components then add that edge (a safe edge) to A.

To implement Step 2, we do the following. Let e_i be the edge under consideration, implying that all edges with a lesser cost than $e_i = (a, b)$ have already been considered. We need to check whether the endpoints a and b are within the same component or whether they join two different components. If the endpoints are within the same component, then we discard the edge e_i. Otherwise, since it is the next lightest edge overall, it must be the lightest edge between some pair of connected components, and so we know from Corollary 5.1.2 that it is safe to add to A. We will need to merge these two components to form a bigger component.

To accomplish all of this, we will need some data structure which supports the following operations:

- MAKE-SET(v) - create a new set containing only the vertex v.
- FIND(v) - Find the set which presently contains the vertex v.
- MERGE(V_x, V_y) - Merge the two sets V_x and V_y together such that FIND will work correctly for all vertices in merged set.

We can implement this data structure as follows. For each vertex we keep track of which component it lies in using a label associated with the vertex. Initially each vertex belongs to its own component, which is done with MAKE-SET. During the algorithm the components will be merged, and the labels of the vertices will be updated.

Assume that we need to merge the two components V_a and V_b corresponding to the end points a and b of the edge $e_i = (a, b)$. We use FIND(a) and FIND(b) to get the sets V_a and V_b respectively. We then call MERGE which will relabel all of the vertices in one of the components to have the same labels as the vertices of the other. The component which we relabel will be the one which is smaller in size. Given such a data structure, we can implement Kruskal's algorithm as in Algorithm 5.2.

Algorithm 5.2: Kruskal-MST
Input: Graph $G = (V, E)$, cost function w
Output: A minimum spanning tree of G

```
1  A ← ∅
2  foreach v ∈ V do
3  |  MakeSet(v)
4  end
5  sort the edges of E in non-decreasing order w.r.t. w
6  foreach e = {a, b} ∈ E, where e is taken in sorted order do
7  |  V_a ← Find(a)
8  |  V_b ← Find(b)
9  |  if V_a ≠ V_b then
10 |  |  A ← A ∪ {e}
11 |  |  Merge(V_a, V_b)
12 |  end
13 end
14 return A
```

Let us analyze the complexity of Kruskal's algorithm. Sorting the edges takes $O(|E| \log |E|)$ time. The test for an edge, whether it joins two connected components or not, can be done in constant time. (In all $O(E)$ time for all edges.) What remains is to analyze the complexity of merging the components which can be bounded by the total complexity of relabeling the vertices. Consider a particular vertex v, and let us estimate the maximum number of times this will be relabeled. Notice that the vertex gets relabeled only if it is in a smaller component and its component is merged with a larger one. Hence after merging, the size of the component containing v becomes at least double. Since the maximum size of a component is $|V|$, this implies that v can be relabeled at most $\log_2 |V|$ times. Therefore, the total complexity of the Step 2 of the algorithm is $O(|E| + |V| \log |V|)$ time. These results are summarized in the following theorem.

Theorem 5.2.1 (Kruskal) *A minimum (cost) spanning tree of an undirected connected graph $G = (V, E)$ can be computed in $O(|V| \log |V| +$*

$|E| \log |E|)$ *time.*

5.3 *Prim's MST algorithm*

Prim's algorithm is very similar to Dijkstra's single source shortest path algorithm[2], and, in fact, their complexity analysis will be the same. Here the set A at any stage of the algorithm forms a tree, rather than a forest of connected components as in Kruskal's. Initially the set A consists of just one vertex. In each stage, a light edge is added to the tree connecting A to a vertex in $V \setminus A$.

[2] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959

The key to Prim's algorithm is in selecting that next light edge efficiently at each iteration. For each $v \in V \setminus A$, we keep track of the least cost edge which connects v to A, and the cost of this edge is used as the "key" value of v. These key values are then used to build a priority queue Q. See Figure 5.2 for an example of these sort of light edges.

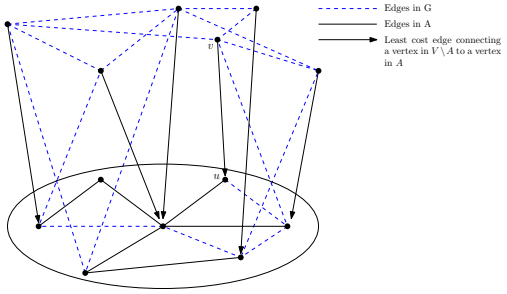


Figure 5.2: Example of Prim's Algorithm showing the set A (encircled) and the least cost edges associated with each vertex in $V \setminus A$.

In each step of the algorithm, the vertex v with the least priority is extracted out of Q. Suppose that corresponds to the edge $e = \{u, v\}$, where $u \in A$, then observe that e is a safe edge since it is the light edge for $cut(A, V \setminus A)$. We update $A := A \cup \{e\}$. Finally, after extracting v out of Q, we need to update Q. The details are explained in Algorithm 5.3.

The vertices that are in the set A at any stage of the algorithm are the vertices in $V \setminus Q$, i.e., the ones that are not in Q. $kev(v)$ is the weight of the light edge $\{v, \pi(v)\}$ connecting v to some vertex in the MST A. Notice that the *key* value for any vertex starts at infinity, when it is not adjacent to A via any edge, and then keeps decreasing.

Let us analyze the complexity of the algorithm. The main steps are the priority queue operations, namely *decrease-key* and *extract-min*. We perform $|V|$ extract-min operations in all, one for each vertex. We also perform $O(|E|)$ decrease-key operations, one for each edge. The following table shows the complexity of these operations depending on the type of priority queue you choose. These complexities are per operation, although the complexities of Fibonacci Heaps are *amortized*

Algorithm 5.3: PRIM-MST
Input: Graph $G = (V, E)$, cost function w, root vertex r
Output: A minimum spanning tree of G

```
1  foreach v ∈ V do
2  |  key(v) ← ∞
3  |  π(v) ← nil /* π keeps track of the parent of a vertex
   |     in the tree.                                    */
4  end
5  key(r) ← 0
6  Q ← V /* Priority queue consists of vertices with
      their key values                                    */
7  while Q ≠ ∅ do
8  |  u ← Extract-Min(Q)
9  |  foreach v adjacent to u do
10 |  |  if v ∈ Q and w(u, v) < key(v) then
11 |  |  |  π(v) ← u
12 |  |  |  key(v) ← w(u, v)
13 |  |  end
14 |  end
15 end
```

(kind of an average over the worst possible scenario! - more on that later).

	Binary Heaps	Fibonacci Heaps
Extract-min	$O(\log n)$	$O(\log n)$
Decrease-key	$O(\log n)$	$O(1)$

5.4 *Randomized Algorithms for Minimum Spanning Trees*

Here we discuss some results related to randomized algorithms for computing minimum spanning trees. These results are based on Section 10.3 of Raghavan and Motwani's book on Randomized Algorithms [3] and T. Chan's simplified analysis from [4]. Assume that all edge weights are distinct and hence there is a unique MST in the given graph $G = (V, E)$. The randomized algorithm uses a few concepts which are discussed in the following subsections followed by the actual algorithm itself in Section 5.4.3. The first concept is an algorithm due to Boruvka from 1926[5] which helps us to reduce the number of vertices in the graph. The second is about heavy and light edges with respect to a spanning tree.

[3] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995

[4] Timothy M. Chan. Backwards analysis of the Karger-Klein-Tarjan algorithm for minimum spanning. *Inf. Process. Lett.*, 67(6):303–304, 1998

[5] Otakar Borůvka. O jistém problému minimálním. *Práce mor. přírodověd. spol. v Brně III*, 3:37–58, 1926; and Otakar Borůvka. Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí. *Elektrotechnickỳ obzor*, 15:153–154, 1926

5.4.1 *Boruvka's Algorithm*

Observe that for any vertex $v \in V$, the edge, say $\{v, w\}$, with the minimum weight incident to that vertex will be included in the MST, as per Corollary 5.1.2. This leads to a simple way to compute the MST as given in Algorithm 5.4.

Algorithm 5.4: Boruvka-MST
Input: Graph G, cost function w (all costs distinct)
Output: A minimum spanning tree T of G

```
1 T ← ∅
  // Each iteration of this loop is called a Phase
2 while G contains more than a single vertex do
3 |   Mark the edges with the minimum weight incident to each
  |    vertex. Add these edges to T
4 |   Identify the connected components of the marked edges
5 |   Replace each connected component by a vertex
6 |   Eliminate all self loops. Eliminate all multiple edges
  |    between a pair of vertices, except the edge with the
  |    minimum weight
7 end
8 return T
```

A few observations about this algorithm:

- Let G' be the graph obtained from G after contracting the edges in a single phase of the algorithm. Then the MST of G is the union of the contracted edges from that phase and the edges in the MST of G'.
- In each contraction phase, the number of vertices in the graph is reduced by at least a half. Hence there will only be $O(\log |V|)$ phases in all.
- Each phase can be implemented in $O(|V| + |E|)$ time and so we obtain yet another MST finding algorithm. Total running time for this algorithm is $O(|E| \log |V|)$.

5.4.2 *Heavy Edges*

We've already seen the definition of a light edge. Now we examine edges which are not light.

Let F be any spanning tree of G (in particular, F may or may not be a minimum spanning tree). Consider any two vertices, say u and v, of G and there is a unique path $P(u, v)$ between them in F. Let $w_F(u, v)$ be the edge with the maximum weight along this path.

We say an edge $\{u, v\}$ is F-heavy if the weight of this edge is larger than the weight of each of the edges on the unique path between u and v in F. More formally, we define an edge $\{u, v\} \in E$ to be F-heavy if $w(u, v) > w_F(u, v)$, otherwise it is F-light. Observe that by this definition, all edges of F are F-light (every edge of a path in F is at most as heavy as the heaviest edge in that path of F).

Lemma 5.4.1 *Let F be a spanning tree of G which is not necessarily minimum. If an edge of G is F-heavy then it does not lie in the Minimum Spanning Tree of G.*

Proof. It is left for you to prove it formally. The proof proceeds by contradiction. Assume that the edge $e = \{u, v\} \in E$ is F-heavy, that T is an MST of G, and that $e \in T$ (so, we are talking about two trees here, T, which is known to be an MST, and F, which may be one as well). Consider the edges on the path $P(u, v)$ in F. Add all these edges to T and remove e from T, to obtain a graph G' (that is, $G' = T \cup P(u, v) \setminus \{e\}$, but since it may have some cycles, we cannot call it a tree). G' is still connected. Remove edges from $P(u, v)$ one-by-one from G' until G' is once again a tree. Observe that this new tree is still a spanning tree, but it must have weight lower than that of T, contradicting the minimality of T. ∎

It is very important to note that because of the way F-light is defined, an edge which is F-light may or may not be in the MST. For example, if we construct F such that the heaviest edge of E is in F, that edge will be counted as F-light, even though it may not be present in any minimum spanning tree.

5.4.3 *Randomized Algorithm*

Algorithm 5.5 gives the main steps of the randomized algorithm we have been discussing. The analysis is based on Timothy Chan's paper[6].

[6] Timothy M. Chan. Backwards analysis of the Karger-Klein-Tarjan algorithm for minimum spanning. *Inf. Process. Lett.*, 67(6):303–304, 1998

Theorem 5.4.2 *Algorithm 5.5 correctly computes the MST of G in $O(|V| + |E|)$ time.*

Proof. The correctness of the algorithm is straightforward. To estimate the complexity, the crux is in estimating the size of the set E_3, i.e., the size of the set of F_2-light edges in E_1. We will prove in the Sampling Lemma (Lemma 5.4.4) that for a random subset $R \subset E$, the expected number of edges that are light with respect to MST(R) is at most $(|E| \cdot |V_1|)/|R|$. In our case, the expected value of $|R| = |E_1|/2 \leq |E|/2$, and hence the expected number of F_2-light edges will be at most $2|V_1| \leq |V|/4$. Hence the running time of this algorithm is given by the recurrence

Algorithm 5.5: Randomized-MST
Input: Connected Graph $G = (V, E)$ with distinct edge weights
Output: A minimum spanning tree T

1 Execute 3 phases of Boruvka's algorithm (reduces number of vertices). Let the resulting graph be $G_1 = (V_1, E_1)$, where $|V_1| \leq |V|/8$ and $|E_1| \leq |E|$. Let C be the set of contracted edges. (Running time: $O(|V| + |E|)$)
2 Random Sampling: Choose each edge in E_1 with probability $p = 1/2$ to form the set E_2 and obtain the sampled graph $G_2 = (V_2 = V_1, E_2)$.
3 Compute Recursively the Minimum Spanning Tree of G_2, and let it be F_2. ($T(|V|/8, |E|/2)$)
4 Verification: Compute the set of F_2-light edges in E_1, and let this set be E_3. ($O(|V_1| + |E_1|)$ time)
5 Final MST: Compute MST, F_3, of the graph $G_3 = (V_3 = V_1, E_3)$. ($T(|V|/8, |V|/4)$)
6 **return** MST of G as $C \cup F_3$.

$$T(|V|, |E|) = O(|V| + |E|) + T(|V|/8, |E|/2) + T(|V|/8, |V|/4),$$

which magically solves to $O(|V| + |E|)$. ∎

Before we describe the Sampling Lemma here are some technicalities. Consider that we are sampling the edges of the graph $G = (V, E)$, and the sampled edges form the subgraph R.

We use the notation R for both the set of edges as well as the sampled graph. Since we are sampling the edges, it is possible that the sampled graph R of the graph G is not connected, and hence there will not be any minimum spanning tree. To ensure connectedness, we will fix any spanning tree T_0 of G, consisting of $|V| - 1$ edges and we will consider the minimum spanning tree of $R \cup T_0$, denoted as $MST(R)$.

Lemma 5.4.3 (Observation about light edges) *An edge $e \in E$ is light with respect to $MST(R)$ if and only if $e \in MST(R \cup \{e\})$.*

Proof. If $e = (u, v)$ is light then there is some edge e' on the unique path between u and v in $MST(R)$ such that its weight, $w(e') = w_{MST(R)}(u, v)$, and hence e can be added to $MST(R)$ and e' can be removed to obtain MST of $R \cup \{e\}$. Therefore $e \in MST(R \cup \{e\})$.

Now suppose $e \in MST(R \cup \{e\})$. We need to show that e is light with respect to $MST(R)$. Since e is part of the MST, by definition it is light with respect to that MST. ∎

Lemma 5.4.4 (Sampling Lemma) *For a graph $G = (V, E)$ and a random subset $R \subset E$, of edges, the expected number of edges that are light with respect to $MST(R)$ is at most $(|E| \cdot |V|)/|R|$.*

Proof. Pick a random edge $e \in E$ (this choice is independent of the edges in R). We will prove that e is light with respect to $MST(R)$ with probability at most $|V|/|R|$. From Lemma 5.4.3 we see that this is equivalent to finding the bound on the probability that $e \in MST(R \cup \{e\})$. Let $R' = R \cup \{e\}$. We will use a technique called *backward analysis*. First we analyze the probability on a fixed set R', and then we will show that the expression obtained is not dependent on the elements of R', but just the cardinality, and hence the probability holds unconditionally as well.

Instead of adding a random edge to R, we will think of deleting a random edge from R'. This is an easier proposition since we know the elements of R', having just fixed it. $MST(R')$ has $|V| - 1$ edges, and e is a random edge of R', hence the probability that e is an edge from $MST(R')$, given a fixed choice of R', is $(|V| - 1)/|R'| \leq |V|/|R|$. This bound is independent upon the choice of the set R', and holds unconditionally as well. ∎

5.5 MST Verification

This section is contributed by Gregory Bint. If I give you any tree F derived from a graph $G = (V, E)$, can you identify whether F is a minimum spanning tree of G?

A trivial method for determining this would be to run a known-correct algorithm such as Kruskal's or Prim's on G and compare the output to F, however there are two main drawbacks to this approach:

1. It is too slow
2. The MST may not be unique, making a direct comparison difficult.

What we would like is to be able to calculate this in linear time with respect to the graph. The following lemma has been shown to be very useful in this respect, and it is used by virtually every MST verification algorithm.

Lemma 5.5.1 *Let F be a spanning tree of G, then F is a minimum spanning tree of G if and only if every edge in $E \setminus F$ is F-heavy.*

Proof. Let $P(u, v)$ be the unique tree path between u and v in F, and let $w_F(u, v)$ be the weight of the heaviest edge along that path. $w(e)$ or $w(u, v)$ is the weight of the edge e having endpoints u and v.

We first show that if F is a MST, then every edge in $E \setminus F$ is F-heavy. Let e be any edge in $E \setminus F$ with endpoints u and v and assume that e is F-light. Note that $P(u,v) \cup \{e\}$ is a cycle. Let $e' = \{x,y\}$ be the edge corresponding to $w_F(u,v)$. Since e is F-light, $w(e') > w(e)$, meaning we could replace e' by e in F to obtain a lighter tree overall, contradicting the minimality of F.

For the other half of the proof, we show that if every edge in $E \setminus F$ is F-heavy, then F is a MST of G. Again, we proceed by contradiction. Suppose that F is not a MST of G, then we should be able to lower the weight of the tree by replacing some edges in F with those from $E \setminus F$. But, for every $e \in E \setminus F$ with endpoints u and v, we have that $w(u,v) > w_F(u,v)$, so exchanging e for any other edge in $P(u,v)$ will increase the weight of F. ∎

Given the above lemma, a natural idea for an algorithm would be to try to classify every edge in the graph, and then check if each non-tree edge is in fact F-heavy. This turns out to be something which *is* possible: given a graph $G = (V,E)$ and a tree F, we can partition the edges of G in two sets, the set of heavy edges and the set of light edges, with respect to F in $O(|V| + |E|)$ time.

We will see that many of the algorithms for doing so fairly complex, although there has been recent progress in simplifying it somewhat.

5.5.1 *Overview of verification algorithms*

Here we look at a brief history of the literature on MST Verification. As hinted at above, every single one of the following methods uses Lemma 5.5.1 as its underpinning. This problem can also be restated as the following:

Problem 5.5.2 (The Tree Path Maxima Problem) *Let F be a spanning tree of G, then we want to identify the cost of the heaviest edge along each tree path $P(u,v)$.*

Given an answer to Problem 5.5.2, we can perform a simple linear scan through the the edges of $G \setminus F$. For each edge $e \in G \setminus F$, we compare the cost of e with the tree path of its endpoints. If every edge e is heavier than its corresponding tree path maxima, then F is a MST of G. We look at this sort of translation of the problem in more detail in Section 5.5.6.

Here is a timeline of some results:

- In 1979, Tarjan introduced a method which uses path compression[7] of trees to achieve a near-linear time of $O(m\alpha(m,n))$ where α

[7] Robert Endre Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715, October 1979

is the Inverse Ackermann function.

- In 1984, Komlós's provided an algorithm[8] which showed that only a linear number of *comparisons* of the edge costs would be sufficient to solve the problem, however the algorithm itself has significantly more than linear overhead.
- In 1992, Dixon *et al.*[9] combine methods from Tarjan's 1979 algorithm and Komlós's 1984 algorithm to produce the first linear time MST verification algorithm. The problem is divided into one large problem and several small problems, with the larger being attacked with path compression, and the smaller with a lookup scheme which is bounded in size by Komlós's algorithm.
- In 1994, Karger *et al.* present an algorithm for computing the MST of a graph in *expected* linear time.[10] While not a verification algorithm in itself, its output could be useful to help verify another tree, or to take the place of the other tree altogether (e.g., why bother verifying a potential MST in linear time when you can just *create* one!)
- In 1995, King produced another linear time MST verification method[11] which is a great deal simpler than that of Dixon *et al.*. In this method, Boruvka's algorithm is used to reduce a general tree down to one which can be handled entirely by the full branching tree base case of Komlós's algorithm, which is simpler than his algorithm for general trees.
- In 2010, Hagerup simplifies King's method[12] even further and provides an implementation in the *D* programming language. Like King's method, Hagerup continues to use Komlós's *full branching tree* case, but eschews complex edge encoding schemes in favour of a richer logical data type.

We will walk through parts of Komlós's algorithm, Dixon *et al.*'s algorithm, King's algorithm, and finally Hagerup's algorithm as we piece together the tools needed for a reasonably simple approach to solving this problem.

[8] J. Komlos. Linear verification for spanning trees. In *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pages 201–206, 1984; and J. Komlós. Linear verification for spanning trees. *Combinatorica*, 5(1):57–65, 1985

[9] B. Dixon, M. Rauch, and R. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992

[10] David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42(2):321–328, March 1995; and Philip N. Klein and Robert E. Tarjan. A randomized linear-time algorithm for finding minimum spanning trees. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, STOC '94, pages 9–15, New York, NY, USA, 1994. ACM

[11] Valerie King. A simpler minimum spanning tree verification algorithm. In SelimG. Akl, Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 440–448. Springer Berlin Heidelberg, 1995; and V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18(2):263–270, 1997

[12] Torben Hagerup. An even simpler linear-time algorithm for verifying minimum spanning trees. In Christophe Paul and Michel Habib, editors, *Graph-Theoretic Concepts in Computer Science*, volume 5911 of *Lecture Notes in Computer Science*, pages 178–189. Springer Berlin Heidelberg, 2010

5.5.2 *Komlós's Algorithm*

In 1984, Komlós[13] gives an algorithm of sorts which can solve Problem 5.5.2 in $O(n + m)$ *comparisons*. Komlós does not provide an implementable algorithm, however, and there are other factors of overhead in the method which would drive up the actual cost of a straight implementation. Nevertheless, this method of breaking down the problem is built upon by later papers, notably Dixon *et al.* in 1992, and King, which we cover in Section 5.5.4.

[13] J. Komlos. Linear verification for spanning trees. In *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pages 201–206, 1984; and J. Komlós. Linear verification for spanning trees. *Combinatorica*, 5(1):57–65, 1985

Komlós begins by considering two special cases of spanning trees. In each case, we consider F to be a directed tree with edges oriented away from the root. Additionally, we shift the edge costs down to their lower endpoint vertices, as this simplifies the conceptual model.

The first case occurs when the tree is a path. For the path, we construct a *symmetric order heap*, H, which a tree with both the binary search property on the ordering determined by the path, and the (maximum) heap property determined by the vertex costs. The root of H represents the heaviest vertex, and the heaviest vertex of any path from u to v is found at $LCA(u, v)$. Determining the LCA of two vertices in a tree can be accomplished in several ways, see Chapter 6.

The second case is somewhat more interesting and is concerned with processing *full branching trees*. Not to be confused with full *binary* trees, a full branching tree is defined as one where every leaf is at the same level, and every internal vertex has *at least* 2 children. Let F be our full branching tree with root r and all edges directed away from r.

We need to calculate the maximum cost edge of every path through F. Given a vertex y, let $A(y)$ be the set of all paths through F which contain y. That is $A(y) = \{P(x, z) | x \geqq y \geqq z\}$ where $u \geqq v$ denotes that u is a predecessor of v (or, u may equal v). Since F is directed away from the root, this means that u is at least as close to the root than v. Note that if F was not directed, then it might be possible for $x \geqq y \geqq z$ to hold even though $y \notin P(x, y)$. Given $A(y)$, let $A^*(y)$ be the set of all paths through y, but restricted to just the interval $[r, y]$; that is, just the subpath from the root down to y.

We process F one level at a time, starting from the root, finding the maximum weights of all paths in the sets $A^*(y)$. To do this, assume that we have calculated the maximum costs in all such paths up to level i, and that we are now trying to process some vertex y on level $i + 1$. Let $\bar{y}$ be the parent of y. Since $\bar{y}$ resides on level i, we know the maximum cost of all paths in $A^*(\bar{y})$.

The key observation to make here is the following.

Property 5.5.3 *Consider two paths $P(x, \bar{y})$ and $P(x', \bar{y})$. If x is a predecessor of x' then the maximum cost in $P(x, \bar{y})$ is at least as large as the maximum cost in $P(x', \bar{y})$ (since, under these conditions, $P(x', \bar{y})$ is a subpath of $P(x, \bar{y})$).*

In $A^*(\bar{y})$, the shortest path is $P(\bar{y}, \bar{y})$ while the longest is $P(r, \bar{y})$. By the above property, the maximum costs form a non-decreasing sequence with respect to the length of the path. That is, we can order the maximum costs by considering the path length. This observation about the ordering helps us while building $A^*(y)$, as we can use a binary search insertion of $f(y)$ to compare $f(y)$ against all path cost

maximums in $A^*(\bar{y})$ simultaneously.

By now you should be asking "How are all these sets like $A(y)$ and $A^*(\bar{y})$ created, copied, and updated?" As far as Komlós's paper is concerned, the answer is "slowly". Essentially what Komlós shows us is that a linear number of *comparisons* are sufficient to determine maximum path costs, however finding those comparisons is left open.

The remainder of Komlós's paper details how these two primitive cases can be applied to any general tree, however this method is fairly complex. No implementable algorithm is given in this paper.

5.5.3 *Dixon* et al.*'s Technique*

This technique has the distinction of being the first to achieve a linear running time, requiring $O(m)$ time on a graph with n vertices and m edges. The underlying process is fairly complex, however, and involves first preprocessing the graph into a suitable form.

The preprocessing itself is interesting as it shows a method of massaging a graph into a more attractive form for the problem at hand without affecting anything about the spanning tree that we wish to verify. The preprocessing involves the following steps.

For a given graph $G = (V, E)$ and spanning tree F, not necessarily minimum, we choose an arbitrary vertex r to be the root of F. Now consider any non-tree edge $\{v, w\}$ with cost $c(v, w)$ and lowest common ancestor u. If u is not one of v or w, then this implies that v and w are not *related*; that is, the v is neither ancestor nor descendant of w. In such a case, we replace the edge $\{v, w\}$ by $\{v, u\}$ and $\{u, w\}$, each with cost $c(v, w)$. F is unchanged by this process and, more importantly, the maximum weight along $P(v, w)$ is also unchanged, which preserves the current minimality of F.

Taken over the entire graph, this will at most double the number of non-tree edges. When completed, every non-tree edge in F is a backedge.

In the second stage of preprocessing, we will mark several vertices. We can imagine these marks as subdividing the tree into edge-disjoint subtrees where a marked vertex represents a "root", and any marked descendants are ignored.

The choice of which vertices to mark is based on subtree size. Using a post-order traversal, for each vertex v we calculate $h = 1 + \sum\{s(w) | w \text{ is a child of } v\}$. Let g be a small integer, then if $h \leq g$ we assign $s(v) := h$, otherwise $s(v) := 1$ and v becomes marked. We will look more at the specific choice of g later. Note the following important properties resulting from this process:

1. The number of marked vertices, and hence the number of subtrees, is at most $(n-1)/g + 1$.

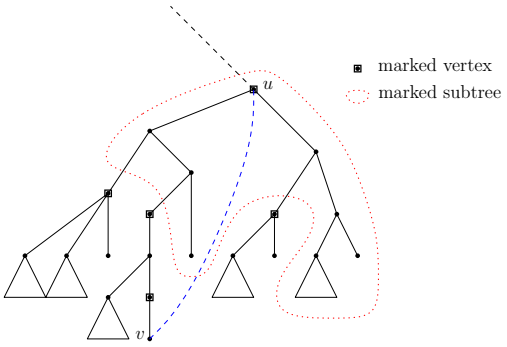


Figure 5.3: An example tree with marked vertices. The marked subtree defined by u is enclosed in red.

2. Considering any subtree, if its root (marked vertex) is deleted, along with incident edges, we get a collection of disjoint trees, each with size at most g. We call each of these a *microtree*.

Following that process, r is also marked, although it will probably not have Property 2.

A final phase of edge replacements will ensure that all backedges either span two vertices belonging to the same microtree, or span between microtrees and marked vertices only (i.e., the edge $\{u, v\}$ in Figure 5.3 will be replaced).

To accomplish this, we first build the tree F' whose vertex set consists of all of the marked vertices of F, and where, for two vertices s and t in F', s is the parent of t (i.e, there is a tree edge between them) if s is the first marked vertex that we encounter when walking from t to r. We call T' the *macrotree*.

We can now eliminate the "long" edges, like $\{u, v\}$, by doing the following. Let $p(v)$ be the nearest marked vertex to v which is a proper ancestor of v. Note that if v is marked then $p(v) \neq v$. We also assume that $p(r)$ is undefined (but it won't be needed anyway). We can calculate $p(v)$ for the entire tree using a depth-first search. For every non-tree edge $\{u, v\}$, assume w.l.o.g. that u is an ancestor of v and find $p(u)$ and $p(v)$. If $p(u) = p(v)$, then u and v are part of the same microtree (recall that the root of a microtree is *not* marked).

Otherwise, if $p(u) \neq p(v)$, then we know that there is at least one marked vertex between them. Let $r_1 = u$ if u is marked, or $r_1 = p(u)$ if u is not marked. Similarly, let $r_3 = v$ if v is marked, or $r_3 = p(v)$ if v is not marked. Let r_2 be the child of r_1 in F' (note: F', not F). We then replace $\{u, v\}$ by $\{u, r_2\}$, $\{r_2, r_3\}$, and $\{r_3, v\}$, skipping any edge that creates either a self-loop or which duplicates a tree edge. For edges which we did not skip, assign the cost $c(u, v)$. As in the first phase of edge replacements, assigning this cost preserves the current minimality of F.

With this preprocessing finished, we have now divided the prob-

lem into one large tree rooted at r with several microtrees around the periphery. The authors complete the process by using Tarjan's Path Compression on the large tree.

The microtrees are processed in a very different way. Essentially, the authors precalculate all possible minimum spanning trees on graphs containing at most g vertices. Leveraging Komlós's result, they show that for any such input, the corresponding decision tree for comparing edges and determining minimality is not too big. The choice of g is such that the total size of these precalculations is only $O(n)$, which places g in the neighbourhood of $O(\log \log n)$ [14].

[14] V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18(2):263–270, 1997

5.5.4 *King's Method*

Presented by King[15] in 1995, this method is not the first MST verification algorithm to achieve linear time (that falls to Dixon *et al.*[16]), however it is quite a bit simpler. King's method uses Boruvka's algorithm in a clever way to change any input tree into a full binary tree, which can then be entirely processed by the appropriate case presented by Komlós. This method requires linear time and space in the unit-cost RAM model with $\Theta(\log n)$ word size.

[15] Valerie King. A simpler minimum spanning tree verification algorithm. In SelimG. Akl, Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 440–448. Springer Berlin Heidelberg, 1995; and V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18(2):263–270, 1997

[16] B. Dixon, M. Rauch, and R. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992

Boruvka Tree Property

The first step is to take our input tree F and convert it to a full binary tree. This is accomplished by running Boruvka's algorithm on the tree F (we usually would run Boruvka's on an entire graph, but not in this case). As Boruvka's runs on F, we can build a new tree B which represents the *execution* of the algorithm on F, rather than a modification of F itself.

Algorithm 5.6 details the construction of B. In the first step, a leaf is added to B for each vertex of F, so we already know that $|B| \geq |F|$. In fact, B will have at most twice as many vertices as F when we are finished. The algorithm proceeds by colouring the vertices and edges to represent subtrees within F, such that any vertices connected along a coloured (blue) path is considered part of the same subtree.

Refer to Figure 5.4 for an example of the algorithm's execution. Note the following important properties which ensure that B is a full branching tree.

1. In each step of Loop 1, an edge joins two blue trees into one.
2. In each phase of the while loop, every blue tree is combined by some edge with another blue tree. Thus, from every level of B, every vertex has a parent in the next level.

For every v in F there is a vertex $f(v)$ in B, and by construction we also have that for every path $F(x, y)$ there is a path $B(f(x), f(y))$.

Algorithm 5.6: FullBranchingTree
Input: A spanning tree $F = (V, E)$ with distinct edge weights
Output: A full branching tree B satisfying Lemma 5.5.4

```
1  Initialize B as an empty tree
2  foreach vertex v of V do
3  |  Colour v blue, considering it as a singleton tree
4  |  Add the leaf f(v) to B
5  end
6  while there is more than one blue tree do
   |  // Loop "1", joins blue trees together
7  |  foreach blue tree a do
8  |  |  Select a minimum cost edge e incident to a and colour it
   |  |   blue
9  |  end
   |  // Loop "2", updates B
10 |  foreach new blue tree t do
11 |  |  Add f(t) to B
12 |  |  Let A be the set of trees joined into t in Loop 1
13 |  |  Add an edge {f(t), f(a)} for each a ∈ A
14 |  |  Set the cost of {f(t), f(a)} to that of edge selected by a
   |  |   in Loop 1 (i.e., e)
15 |  end
16 end
17 return B
```

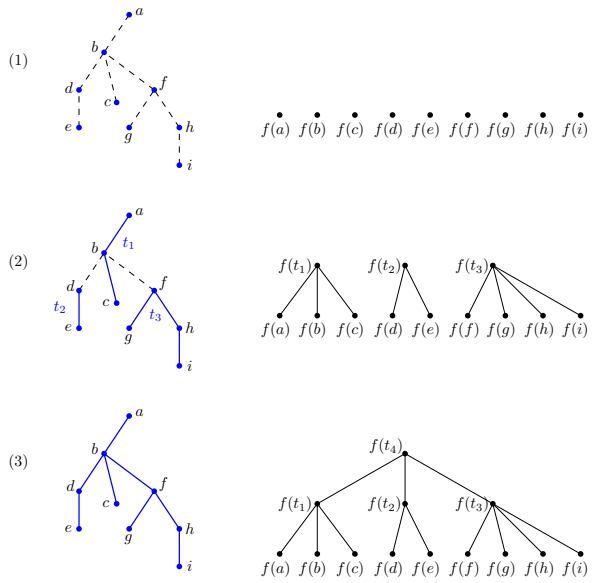


Figure 5.4: An example of the Full Binary Tree construction given by Algorithm 5.6. F is shown on the left and B on the right. Edge weights are not shown, so imagine that each tree chooses its minimum weight edge at each step.

However, to show that there is any meaningful correspondence between these paths beyond their existence, we need the following lemma, presented as Theorem 1 in King's paper.

Lemma 5.5.4 *Let F be any spanning tree and let B be the tree constructed by Algorithm 5.6. For any pair of vertices $x, y \in F$, the cost of the heaviest edge in $F(x, y)$ equals the cost of the heaviest edge in $B(f(x), f(y))$.*

Proof. Let the cost of an edge e be denoted by $w(e)$. For every edge $e \in B(f(x), f(y))$, we will show that there is an edge $e' \in F(x, y)$ such that $w(e') \geq w(e)$.

Suppose that $e = \{a, b\}$ such that a is the endpoint of e which is farthest from the root. As a is in B, $a = f(t)$ for some blue tree t, and t must contain either x or y, but not both. Similarly, $b = f(t')$ which is new blue tree consisting of $f(t)$ and others from the previous phase of the algorithm. Since $e \in B$, e was selected by t.

Let e' be the edge in $F(x, y)$ with exactly one endpoint in t. Since e' is adjacent to t, t would have considered e'. Since t ultimately chose e, it must be that $w(e') \geq w(e)$ since t chooses the edge with minimum cost.

To finish the proof we also need to show the following: The cost of the heaviest edge in $F(x, y)$ is the cost of the heaviest edge in $B(f(x), f(y))$. Let e be the heaviest edge in $F(x, y)$ (for simplicity, assume that there is a unique such edge). If e is ever selected by a blue tree which contains either x or y, then $B(f(x), f(y))$ contains an edge with the same weight.

Otherwise, assume that e is selected by some other blue tree t' not

containing x or y. We know that e is on the path from x to y in F, so t' contained at least one intermediate vertex on that path. But since F is a tree, if it contains an intermediate vertex of $F(x, y)$, it must be incident to at least two edges of $F(x, y)$. By our assumption, e is the heaviest edge on this path, so t' would have selected the other edge, giving a contradiction. ∎

The intuition with the last part of the above proof is that, since e is the heaviest edge along $F(x, y)$, any blue tree which includes part of that path, but which does not yet include x or y always has another edge to select which brings it "closer" to x or y.

King's algorithm now continues with B rather than F, which maintains the path maximum cost property for each path in F, implying that if Lemma 5.5.1 holds for B it will also hold for F.

The remainder of King's paper shows a bit-wise labeling scheme from for the vertices and edges of B which exploits Property 5.5.3 of the full binary tree case presented by Komlós. We will now jump to Hagerup's method to conclude our verification method. The paper simplifies the labeling scheme. Hagerup's algorithm simplifies the King's method from this point.

5.5.5 *Hagerup's Method*

Hagerup presents an algorithm for solving the Tree Path Maxima problem (TPM) rather than MST Verification, *per se*, but as we have mentioned, a solution to TPM implies a solution to MST Verification. A sketch of such a translation is given in the next section.

The input to Hagerup's method assumes that we are given a tree on n vertices and a list of pairs $(u_1, v_1), \ldots, (u_m, v_m)$ such that in each pair u_i is a proper ancestor of v_i. At most, this list would describe the endpoints of every root to leaf path in B, and every subpath of such a root to leaf path. Any subset is also permissible. In practice, we choose a subset equivalent to the non-tree edges of G, the graph containing the spanning tree F we are trying to verify.

The basic algorithm involves collecting several types of information about each vertex. For every vertex u in B, we store the depth $d(u)$, and, if u is not the root r, we let $w(u)$ represent the cost of the edge from u to its parent. For each u we also build the following set:

$$D_u = \{d(u_i) | u_i \text{ is a proper ancestor of } u \text{ and } v_i \text{ is a descendant of } u\}$$

Simply put, D_u stores the set of depths corresponding to proper ancestors of u such that u is in the subpath represented by some pair (u_i, v_i) from the input.

We would also like to create the set M_u for each u, which stores a subset of the ancestors of u indicated by D_u. The choice of which

ones are stored again exploits Property 5.5.3.

Consider any two successive ancestors d and d' of u which are indicated by D_u such that d is closer to the root, and d' is closer to u. Then $d \in M_u$ if the path maximum cost of the path $d \to u$ is greater than that of $d' \to u$. Put another way, we store only those ancestors of u where there is an actual increase in path maximum cost between it and the previous (closer) ancestor.

This can still work out to be a lot of entries, however, and a lot of copying between vertices, which breaks linear time. Fortunately, Hagerup was able to find an alternate, yet equivalent set representation which does satisfy our needs, and our desired running time, using the *set infix* operator. The details of this operator and its equivalence to M_u take a few pages to discuss and can be found in Hagerup's paper.

5.5.6 *Putting it all together*

One way of applying all of the tools we have seen so far to build a complete MST Verification algorithm is as follows.

Taking a graph $G = (V, E)$ with spanning tree F, let $U = E \setminus F$ be the set of non-tree edges. We use King's method of using Boruvka's method to convert F to the full branching tree B. Translate U onto B so that $\forall e = \{x, y\} \in U$ we create $e' = \{f(x), f(y)\}$ and call the resulting graph G'. We next apply Dixon *et al.*'s first preprocessing step to G' to replace all cross edges with back edges. Let U' be the set of non-tree edges in G' after all of this.

The set U' corresponds to the pairs (u_i, v_i) that we need to input into Hagerup's algorithm. After that algorithm has run, MST Verification is completed by examining every non-tree edge in G, translating it to the equivalent one or two edges in U', querying B, and determining whether the non-tree edge is costlier than the tree path maximum.

The extra steps required to find U, translate it to B, and then find U' all take time linear in the number of edges.

5.6 *Bibliographic Notes*

Kruskal's algorithm, presented in Section 5.2 makes use of a data structure known as UNION-FIND or DISJOINT-SET. A near-linear time implementation was first described by Tarjan[17].

Boruvka's algorithm is quite old[18], and not originally published in English. Nešetřil *et al.* published a translation from the original Czech in 2001 along with some comments.[19]

The way that King uses Boruvka's algorithm is first described by

[17] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, April 1975

[18] Otakar Borůvka. O jistém problému minimálním. *Práce mor. přírodověd. spol. v Brně III*, 3:37–58, 1926; and Otakar Borůvka. Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí. *Elektrotechnickỳ obzor*, 15:153–154, 1926

[19] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. Otakar borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1–3):3 – 36, 2001. <ce:title>Czech and Slovak 2</ce:title>

Tarjan in 1983[20].

[20] Robert Tarjan. *Data Structures and Network Algorithms*, volume 44, chapter 6. Minimum Spanning Trees, pages 71–83. SIAM, 1983

5.7 Exercises

5.1 *Let S=(V,T) be a minimum cost spanning tree, where* $|V| = n + 1$. *Let* $c_1 \leq c_2 \leq ... \leq c_n$ *be the costs of the edges in T. Let S' be an arbitrary spanning tree with edge costs* $d_1 \leq d_2 \leq .... \leq d_n$. *Show that* $c_i \leq d_i$, *for* $1 \leq i \leq n$.

5.2 *Assume all edges in a graph G have distinct cost. Show that the edge with the maximum cost in any cycle in G cannot be in the Minimum Spanning Tree of G. Can you use this to design an algorithm for computing MST of G by deletion of edges, and what will be its complexity?*

5.3 *Design an efficient algorithm to find a spanning tree of a connected, (positive) weighted, undirected graph* $G = (V, E)$, *such that the weight of the maximum-weight edge in the spanning tree is minimized (Justify your answer).*

5.4 *Prove that if all edge weights are distinct then the minimum spanning tree of a simple undirected graph is unique.*

5.5 *Provide a formal proof of Lemma 5.4.1.*

5.6 *Suppose all edge weights are positive integers in the range* $1..|V|$ *in a connected graph* $G = (V, E)$. *Devise an algorithm for computing Minimum Spanning Tree of G whose running time is better than that of Kruskal's or Prim's algorithm.*

5.7 *Consider a connected graph* $G = (V, E)$ *where each edge has a non-zero weight. Furthermore assume that all edge weights are distinct. Show that for each vertex* $v \in V$, *the edge incident to v with minimum weight belongs to a Minimum Spanning Tree.*

Can you use this to devise an algorithm for MST - the above step identifies at least $|V|/2$ *edges in MST - you can collapse these edges (by identifying the vertices and then recursively apply the same technique - the graph in the next step has at most half of the vertices that you started with - and so on!).*

5.8 *Which of the following algorithms result in a minimum spanning tree? Justify your answer. Assume that the graph* $G = (V, E)$ *is connected.*

1. *Sort the edges with respect to decreasing weight.*
   Set $T := E$.
   For each edge e taken in the order of decreasing weight do, if $T - \{e\}$ *is connected, then discard e from T.*
   Set $MST(G) = T$.

2. *Set* $T := \emptyset$.
   For each edge e, taken in arbitrary order do, if $T \cup \{e\}$ *has no cycles then*
   $T := T \cup \{e\}$.
   Set $MST(G) = T$.

3. *Set* $T := \emptyset$.
   For each edge e, taken in arbitrary order do
   begin
   $T := T \cup \{e\}$.
   If T has a cycle c then let e' *be a maximum weight edge on c.*
   Set $T := T - \{e'\}$.
   end
   Set $MST(G) = T$.

5.9 *A spanning tree T of a undirected (positively) weighted graph G is called a minimum bottleneck spanning tree (MBST) if the edge with the maximum cost is minimum among all possible spanning trees. Show that a MST is always a MBST. What about the converse?*

5.10 *Design a linear time algorithm to compute MBST. (Note that an edge with medium weight can be found in linear time. Consider the set of edges whose weight is smaller than the weight of the 'median edge'. What happens if this graph is connected? disconnected?*

5.11 *Consider an undirected (positively) weighted graph* $G = (V, E)$ *with a MST T and a shortest path* $\pi(s, t)$ *between two vertices* $s, t \in V$. *Will T still be an MST and* $\pi(s, t)$ *be a shortest path if*

a) Weight of each edge is multiplied by a fixed constant $c > 0$.

b) Weight of each edge is incremented by a fixed constant $c > 0$.

5.12 *Let* $G = (V, E)$ *be a weighted simple connected graph, and assume that all edge weights are distinct. Define the weight of a spanning tree to be the sum total of the weights of edges in that tree. By definition, a minimum spanning tree T of G has the smallest sum total of the weight among all possible spanning trees of G. Suppose we are not interested in minimizing the sum total of the weights, but just the weight of the heaviest edge in a spanning tree. Call such a tree a* light spanning tree *(LST). First show that any MST of G is also a LST. Next show that a LST may not always be a MST. To compute LST, we can use an algorithm to compute MST and report that MST as a LST. You are asked to think of an alternate algorithm, running in* $O(|V| + |E|)$ *time, to find a LST. (Hint: Let* e_m *be the edge with the median weight among edges in* $G = (V, E)$. *Consider the subgraph* G' *formed by all edges in E, whose weight is at most the weight of* e_m. *Can you deduce something about LST from the connectivity of* G'.*)*

5.13 *Suppose you are given n-points in the plane. We can define a complete graph G on these points, where the weight of an edge* $e = (u, v)$, *is Euclidean distance between u and v. We need to partition these points into k*

non-empty clusters, for some $n > k > 0$. The property that this clustering should satisfy is that the minimum distance between any two clusters is maximized. (The distance between two clusters A and B is defined to be the minimum among the distances between pair of points, where one point is from cluster A and the other from cluster B.) Show that the connected components obtained after running Kruskal's algorithm till it finds all but the last $k - 1$ (most expensive) edges of MST of G produces an optimal clustering.

6
Lowest Common Ancestor

We will focus on

1. Lowest common ancestors in a binary trees
2. Range minima queries
3. Reduction of LCA queries to RMQ queries
4. Reduction of RMQ queries to LCA queries

Given a rooted binary tree T on n nodes, we are asked to preprocess it in $O(n)$ time so that the following type of queries can be answered in $O(1)$ time. Given any two nodes u and v of T, report their Lowest Common Ancestor $LCA(a, b)$, i.e., among all the common ancestors of nodes a and b, find the one which is furthest from the root of T. This subproblem arises in many graph applications. Orginal algorithm is due to Harel and Tarjan [1984]. Many years later, Schieber and Vishkin [1993] proposed a new algorithm for the same problem while studying parallel algorithms. Both of these algorithms are fairly complex and are considered to be far from being implementable. Recently, Bender and Farach-Colton [2000] proposed a fairly simple algorithm for the LCA problem, and thats what we present in this chapter.

It is well known that the following Range Minima Problem (RMQ) is related to the LCA problem. Given an array $A[1...n]$ consisting of n numbers, preprocess it so that given any two indices i and j, where $1 \leq i \leq j \leq n$, report the minimum element (or its index in A) in the subarray $A[i...j]$. Next we will show the reduction of the LCA problem to RMQ problem, and then provide a solution for the RMQ problem.

6.1 $LCA \rightarrow RMQ$

Let T be the given binary rooted tree. Consider the depth first search traversal of T. Observe that the shallowest node encountered in the depth first traversal of T between u and v is the node corresponding to $LCA(u, v)$. (Recall that the main property of dfs traversal is that once it enters a subtree, then it completely visits all the nodes in the subtree - this sort of corresponds to a nice bracketing sequence.) Our aim is to find this node using the RMQs.

Corresponding to the dfs traversal of T, let E be the Euler tour. Recall that E stores the nodes of T in the same order as they are visited during the dfs traversal. The tour E consists of $2n - 1$ entries. Let the level of a node in T be its distance from the root. Corresponding to E, define a level array $L[1...2n - 1]$ which stores the level of the node $E[i]$ in $L[i]$. Furthermore, observe that a node may appear several times in Euler tour. For each node $x \in T$, we maintain an index $R(x)$ that stores the index of the first appearance of x in E. Given our notation, the nodes between $E[R(u), ..., R(v)]$ are nodes in Euler tour between the first visits of u and v. What is the shallowest node among the nodes in $E[R(u), ..., R(v)]$? For this we will look at the corresponding entries in the level array L. More precisely, we need to report what is the minimum element in the subarray $L[R(u)...R(v)]$; this returns us the index of the shallowest node (one with the smallest level) and denote this by $RMQ_L[R(u)...R(v)]$. Hence, $LCA(u, v) = E[RMQ_L[R(u)...R(v)]]$.

Lemma 6.1.1 *LCA problem on a rooted binary tree T of n nodes can be converted to the range minima query problem on an array L of size $2n - 1$ elements. The reduction takes $O(n)$ time. Moreover, LCA queries can be answered within $O(1)$ time in addition to the time required to answer the range minima queries on L.*

Proof. Notice that the depth first traversal and the construction of Euler tour of T can be done in $O(n)$ time. Within the same time bounds we can maintain the level array as well as keep track of the first appearance of each node in Euler tour. Hence the conversion can be done in linear time. Given the query, $LCA(u, v)$, we need to find the representatives $R(u)$ and $R(v)$ in E, then need to answer the query $RMQ_L[R(u)...R(v)]$ followed by one more look up in the array E to report the node corresponding to $LCA(u, v)$. This computation only requires a few pointer manipulation and hence requires $O(1)$ time in addition to answering the range minima query. ∎

6.2 Range Minima Queries

Let A be the array of length n consisting of numbers. Our task is to preprocess A so that the range minima queries $RMQ(i,j)$, $1 \leq i \leq j \leq n$, can be answered in $O(1)$ time.

6.2.1 A naive $O(n^2)$ algorithm

A simple way to achieve a constant query time is to precompute and store minima for each possible query. In all there are $O(n^2)$ possible queries of type $RMQ(i,j)$, where $1 \leq i \leq j \leq n$, and for each of them we can compute and store the minima in the range $A[i,...,j]$. It is easy to see that this computation can be done in $O(n^2)$ time and then given a query it can be answered in $O(1)$ time.

6.2.2 An $O(n \log n)$ algorithm

In place of precomputing minima for each possible query, now we precompute minima's for only $O(n \log n)$ selected types of queries. For every i between 1 and n and for every j between 1 and $\log n$, we find minimum element in the subarray $A[i,...,i+2^j]$ (we are sloppy with boundary conditions here to keep it simple) and store it in a table in location $M[i,j]$. Next we show that using dynamic programming the table M can be computed in $O(n \log n)$ time. Minima in a subarray of size 2^j is computed by looking at the minima of two constituent blocks of size 2^{j-1}. Either $M[i,j] = M[i,j-1]$ or $M[i,j] = M[i+2^{j-1}-1, j-1]$.

How do we answer a range minimum query in $O(1)$ time? Let the query be $RMQ(i.j)$, where $1 \leq i \leq j \leq n$. First compute $k = \lfloor \log_2 j - i \rfloor$. Now observe that 2^k is the largest interval, that is a power of 2, that fits in the range from i to j. Compute $RMQ(i,j)$ be finding out the minimum of two entries in the table, namely $M[i,k]$ and $M[j-2^k+1,k]$. Notice that these two table values have been precomputed and hence query can be answered in $O(1)$ time.

Lemma 6.2.1 *An array A consiting of n numbers can be preprocessed in $O(n \log n)$ time so that the range minima queries can be answered in $O(1)$ time.*

6.2.3 An $O(n)$ algorithm with ± 1 property

Consider the following special case of the array A where each element differs from its previous element either by a $+1$ or a -1 (this is especially true for the LCA problem as levels of consecutive nodes in Euler tour differs by 1). We will show that in this case A can be preprocessed in $O(n)$ time and RMQs can be answered in $O(1)$ time.

The strategy is pretty simple. First we partition array A into subarrays, where each subarray is of size $\frac{\log n}{2}$ (we are assuming that n is a nice power of 2, otherwise we have to use floors and ceilings and that will not add anything more in terms of understanding.) Within each subarray we find the minimum value and then store all these minimas in an array A'. Notice that the size of the array A' is $\frac{2n}{\log n}$ and hence it can be preprocessed in $O(n)$ time by using Lemma 6.2.1.

Consider a range minima query $RMQ(i, j)$ in array A, where $i \leq j$. It is answered as follows: Indices i and j may fall within the same subarray, therefore we need to preprocess each subarray for answering RMQs. If i and j fall in different subarrays then we compute the following three quantities:

1. Minimum value starting at index i up to the end of the subarray containing i.
2. Minimum value among the subarrays between the subarray containing i and j. This is computed using the preprocessing done for A' in constant time.
3. Minimum value from the beginning up to the index j within the subarry containing j.

Now our subproblem is reduced to solving the RMQ problem in subarrays of size $\frac{\log n}{2}$ with ± 1 property. The key observation here is that we do not have too many different kinds of these subarrays.

Claim 6.2.2 *Given two arrays of same size where each element in the first array is constant value more than the corresponding element in the second array, then the answer to RMQ queries (i.e. the index) is identical in both the arrays.*

Essentially the preprocessing and the RMQ queries work with relative order of elements in these arrays, and they do not need actual values of the elements. Hence for the two subarrays within the above claim, same preprocessing is sufficient to answer RMQ queries. We normalize each of the subarrays by first subtracting the initial value from each of the elements. Next we show that there are only $O(\sqrt{n})$ normal subarrays.

Claim 6.2.3 *There are at most $O(\sqrt{n})$ normalized subarrays. Each subarrays has length $\frac{\log n}{2}$, where the first element is a 0, and the elements in the array satisfy ± 1 property.*

Proof. Each normalized subarray can be specified by a ± 1 vector. Therefore, there are only $2^{\frac{1}{2}\log n - 1} = O(\sqrt{n})$ different types of subarrays of length $\frac{1}{2}\log n$. ∎

We preprocess each of these subarrays in $O(\log^2 n)$ time to answer RMQ queries in $O(1)$ time using the naive algorithm. The preprocessing requires in all $O(\sqrt{n}\log^2 n)$ time. We summarize the results in the following.

Lemma 6.2.4 *An array A consisting of n-numbers satisfying the ± 1 property can be preprocessed in $O(n)$ time so that the range minima queries can be answered in $O(1)$ time.*

Corollary 6.2.5 *A binary tree on n-nodes can be preprocessed in $O(n)$ time so that the lowest common ancestor queries can be answered in $O(1)$ time.*

6.3 $RMQ \rightarrow LCA$

Next we show that an instance of the RMQ problem can be converted to an instance of the LCA problem. For a linear array A of size n, the tree T for the LCA problem consists of n nodes and given a RMQ query, we perform an equivalent LCA query on T, and whose answer in turn provides the answer for the original range minima query. This will imply that the general RMQ problem (i.e., even without the ± 1 property) can be answered in $O(1)$ time by performing an $O(n)$ time preprocessing. The key to this conversion is the concept of Cartesian Tree.

Let $A[1...n]$ be the input array on which we need to perform RMQ queries. Cartesian tree T for A is defined as follows. It is a rooted binary tree and the root of T stores the index of the smallest element in A. Deleting the minimum element from A splits it into two subarrays. Left and right children of the root are recursively defined Cartesion trees for left and right subarrays of A, respectively (see Figure 6.1).

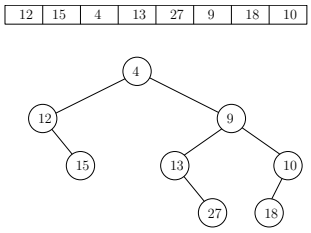


Figure 6.1: RMQ $\rightarrow$ LCA

Claim 6.3.1 *Cartesian tree T for an array A of n-numbers can be constructed in $O(n)$ time.*

Proof. We scan the array A from left to right and incrementally build the Cartesian tree $T = T_n$ as follows. Suppose so far we have built the tree T_i with respect to elements $A[1..i]$ and we want to extend it for $A[1...i+1]$ to obtain T_{i+1}, where $i < n$. Main observation is that the node storing the index $i+1$ in T_{i+1} is on the rightmost path of T_{i+1}. We start at the rightmost node of T_i and follow the parent pointers till we find the location to insert $i+1$ in Cartesian tree. Note that each comparison will either add a node or removes one from the rightmost path. Since each node can only join the rightmost path once (if it leaves it then it can't be back to the rightmost path), therefore the total time in constructing T is $O(n)$. ∎

Claim 6.3.2 *Let A be the array on n-numbers and T be the corresponding Cartesian tree storing the indices of elements in A in its node. Then $RMQ(i,j) = LCA(i,j)$.*

Proof. This follows from the recursive definition of Cartesian tree T. Let $k = LCA(i,j)$ in T. Observe that the node labeled k is the first node that separates i with j. In other words, the element $A[k]$ is the smallest element between $A[i]$ and $A[j]$, i.e. $RMQ[i,j] = k$. ∎

6.4 Summary

In this chapter, we have shown that the lowest common ancestor query in a rooted binary tree on n-nodes can be answered by solving the range minima query in an array consisting of $2n-1$ numbers satisfying the ± 1 property. Moreover, the general RMQ problem in an array can be reduced to solving LCA queries on the corresponding Cartesian tree. All our preprocessing algorithms require linear time and the queries can be answered in constant time.

6.5 Exercises

6.1 *Prove that in the LCA algorithm of Bender and Farach-Colton, why does the reduction from the LCA problem to the range-minima query works, i.e., show that in place of finding the LCA of nodes u and v in the binary tree, why does it suffice to compute the smallest level number in the level array in an interval defined by the first occurrence of the node u an v in the level array.*

6.2 *This problem is to show that an arbitrary range minima query (RMQ) problem can be solved within the same complexity as the one with the ± 1 RMQ problem. Recall that the ± 1 RMQ problem for an array of size n required $O(n)$ time to preprocess and then the queries were answered in $O(1)$ time. The idea is to reduce an arbitrary RMQ problem to the LCA problem. This reduction uses Cartesian Tree. Let A be an array consiting of n numbers (need not satisfy the ± 1 property). The Cartesian Tree C for A is defined as follows: The root of C is the minimum element of A, and it stores the position of this element in the array. Removing the root element splits the array into left and right subarrays. The left and right children of the root are recursively constructed Cartesian trees of the left and right subarrays, respectively. Prove the following:*

1. *Cartesian tree C of an array A of size n can be computed in $O(n)$ time (use incremental construction).*
2. *Show that $RMQ_A(i,j) = LCA_C(i,j)$ (Recall that in C we store the indices i and j.)*

7
Graph Partitioning

We will focus on

1. Planar Graph Partitioning
2. Planar Separator Theorems
3. Spectral Methods for Graph Partitioning
4. Graph Laplacian Matrix
5. Sparse Cuts

The first part of the chapter discusses the planar separator theorem and it is based on Kozen [1] and the paper by Lipton and Tarjan [2]. The second part of this chapter introduces the graph Laplacian matrix, spectral decomposition, and sparse cuts.

Earlier we have seen that for a binary tree on n-nodes, there exists a node such that whose removal leaves no component having more than $2(n+1)/3$ nodes. This can be extended to outerplanar graphs, where we can remove a pair of vertices such that none of the components have more that $2(n+1)/3$ nodes. Usually this phenomenon is referred to as a balanced decomposition using small size separators. This is a 'key idea' in most of the divide and conquer type algorithms on these graphs. As can be seen that the depth of recursion will be $O(\log n)$ and since the size of the separator is small, the "merge" step will be economical as well.

The organization of this chapter is as follows. Section 7.1 has some basic definitions regarding graph separators in planar graphs. Section 7.2 presents the constructive proof of the planar separator theorem. Section 7.3 has extensions of planar separator theorem for weighted graphs, edge separators, and r-division. Section 7.4 introduces the graph Laplacian matrix, its connection to graph partitioning and sparse cuts.

[1] D. Kozen. *The design and analysis of algorithms*. Springer, 1992

[2] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979

7.1 Preliminaries

Definition 7.1.1 *A graph is called* planar *if the vertices and edges can be laid out (embedded) in the plane so that no two edges intersect except at their end points. An embedded planar graph is usually referred to as a* plane graph.

Definition 7.1.2 *In an embedded plane graph, we have vertices, edges and faces. The* dual *of a plane graph G is a planar graph G^* whose vertices correspond to faces of G and two vertices in G^* are joined together if the corresponding faces in G share an edge.*

Definition 7.1.3 *A plane graph G is* triangulated *if each of its face is a triangle, i.e., it is bounded be three edges. In other words, in the dual each vertex has degree three.*

Definition 7.1.4 *A set $S \subseteq V$ for a graph $G = (V, E)$ is called a* vertex separator, *if removal of vertices (and incident edges on these vertices) from G results in two disjoint sets of vertices $A, B \subseteq V$ with no edges between them. If the sizes of the sets A and B are a constant fraction of that of the size of V, then S is called as a* balanced separator.

Definition 7.1.5 *A planar graph $G = (V, E)$ consists of at most $|E| = 3|V| - 6$ edges. This follows from Euler's relation, i.e. $|V| - |E| + |F| = 2$. You may like to check the proof at*

`http:\\www.ics.uci.edu/~eppstein/junkyard/euler/`

Definition 7.1.6 *An* outerplanar graph *is a plane graph such that all its vertices lie on a single face. This face is usually referred to as the* outerface.

Definition 7.1.7 *The dual of a triangulated outerplanar graph is a binary tree.*

We have seen that a complete graph on five vertices, K_5, and a complete bipartite graph on six vertices, $K_{3,3}$, are nonplanar. It is easy to see that a tree is planar, and outerplanar graphs are planar. Both of these graphs admit small size separators. What we will prove in this chapter is that all planar graphs satisfy a similar property.

Theorem 7.1.8 *[Lipton and Tarjan [3]] Let $G = (V, E)$ be an embedded undirected triangulated planar graph, where $n = |V|$. There exists a partition of V into disjoint sets A, B, and S, such that*

1. $|A|, |B| \leq \frac{2n}{3}$
2. $|S| \leq 4\sqrt{n}$
3. *There is no edge in E that joins a vertex in A with a vertex in B.*

[3] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979

4. Such a set S can be found in linear time.

It will turn out that the way we prove this theorem, it will lead to a linear time algorithm (i.e. $O(|V| + |E|)$) for finding such a separator. Note that if the given graph is not embedded in the plane, then there is a linear time algorithm by Hopcroft and Tarjan that embeds it. In fact that algorithm also figures out in linear time whether the given graph is planar or not, and if it is planar it finds an embedding. Also if a plane graph is not triangulated, then it can be triangulated in linear time, by inserting required number of edges on each face. Other than this essentially we will use breadth first and the concept of fundamental cycles to prove this theorem.

7.2 *Proof of the Planar Separator Theorem*

Assume that the graph $G = (V, E)$ is undirected, connected, planar, triangulated and embedded. The first step in the proof/algorithm is to do a breadth-first search starting at an arbitrary vertex, say s, in G, and assign levels to vertices. Vertex s is at level 0, vertices adjacent to s are at level 1, vertices adjacent to level 1 vertices that have not been assigned any level are level 2 vertices, and so on. Let l be the last level, and pretend that there is a level $l + 1$ which consists of no vertex (this is just required for the proof!). Let $L(t)$ denote the set of vertices that are in level t, $0 \leq t \leq l$. Recall that in BFS, no edge can span over two or more levels. All edges must connect vertices in the same level or consecutive levels. Observe that each of the level, $L(t)$, for $0 < t < l$, is a separator in its own right, although may not be of small size and may not lead to a balanced decomposition!

Number the vertices according to BFS ordering, where s gets number 1, followed by vertices in level 1, then vertices in level 2, and so on (see Figure 7.1). Let t_1 be the middle level, that is the one which contains the vertex number $n/2$ in the BFS numbering. Consider the set $L(t_1)$. Note that $|\cup_{t<t_1} L(t)| < n/2$ and $|\cup_{t\leq t_1} L(t)| \geq n/2$. If $|L(t_1)| \leq 4\sqrt{n}$, then $S = L(t_1)$ and we are done. Note that in that case we can set the set A to be all the vertices in levels 0 up to the level $t_1 - 1$. Similarly the set B can be defined as all the vertices in levels $t_1 + 1$ to l. Clearly $|A| < n/2$ and $|B| < n/2$. In general, it is not necessary that $L(t_1)$ may satisfy the requirements on the size of the separator. Here is the lemma which will be very handy in that case.

Lemma 7.2.1 *There exists levels $t_0 \leq t_1$ and $t_2 > t_1$ such that, $t_2 - t_0 \leq \sqrt{n}$, $|L(t_0)| \leq \sqrt{n}$ and $|L(t_2)| \leq \sqrt{n}$.*

Proof. Note that $|L(0)| = 1$ and $|L(l + 1)| = 0$. Let $t_0 \leq t_1$ be the largest number such that $|L(t_0)| \leq \sqrt{n}$. Let $t_2 > t_1$ be the smallest

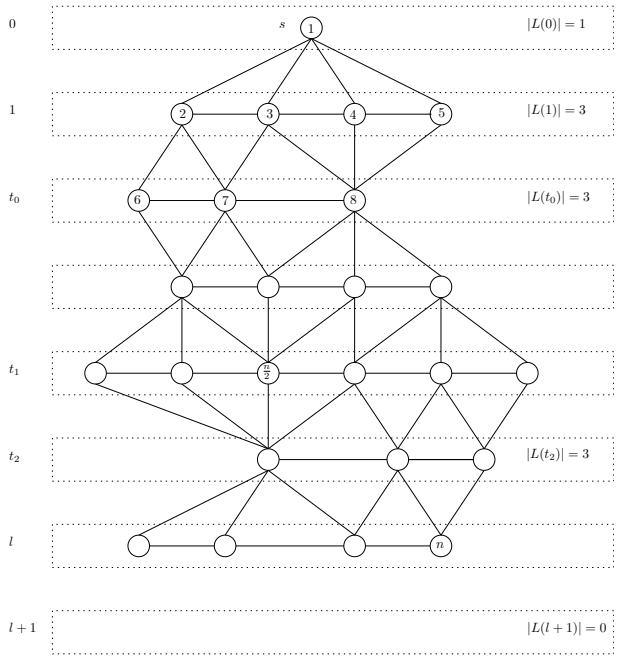


Figure 7.1: BFS and the sets $L(\cdot)$.

number such that $|L(t_2)| \leq \sqrt{n}$. Note that every level between t_0 and t_2 contains more than $\sqrt{n}$ vertices, therefore by pigeon hole principle there must be fewer than $\sqrt{n}$ levels between t_0 and t_2, otherwise G will have more than n vertices! Therefore, $t_2 - t_0 \leq \sqrt{n}$. ∎

Define three sets C, D and E as follows: $C = \cup_{t<t_0} L(t)$, $D = \cup_{t_0<t<t_2} L(t)$ and $E = \cup_{t>t_2} L(t)$. If $|D| \leq 2/3n$, then we have the required separator, by setting $S = L(t_0) \cup L(t_2)$, A the largest of C, D or E and B the union of the other two.

What if $|D| > 2/3n$? Then both the sets C and E are small, have less than $1/3n$ vertices. We will find a $\frac{1}{3} - \frac{2}{3}$ separator S_D, of D, of size at most $2\sqrt{n}$. Let D be split into D' and D'' by S_D. Then S will include the vertices in $L(t_0), L(t_2)$, and the separator vertices S_D. Set $A = \max(C, E) \cup \min(D', D'')$ and $B = \min(C, E) \cup \max(D', D'')$. Observe that S, A, and B satisfy the required size criteria.

Next we will present some ideas regarding finding the separator S_D of D. First we remove all the vertices that are not in D, except the start vertex s. We connect s to all the vertices in level $t_0 + 1$. This can be done still preserving the planarity of D, since the original graph is planar. Now we construct a spanning tree T in D, such that its diameter is at most $2\sqrt{n}$. Start with vertices in level $L(t_2 - 1)$. For each vertex in this level, choose one of the vertex in the previous level $L(t_2 - 2)$, adjacent to it as its parent. Continue this process with vertices in levels $t_2 - 2$, $t_2 - 3$, $\cdots$, to obtain the tree T. Next we

state two lemmas, that are relatively easy to prove, that will show the critical property relating the tree T, the plane graph D, its dual D^*, and the dual tree T'.

Lemma 7.2.2 *Let $G = (V, E)$ be a connected plane graph and G^* be its dual. For any $E' \subseteq E$, the subgraph (V, E') has a cycle if and only if the subgraph $(V^*, E - E')$ of G^* is disconnected.*

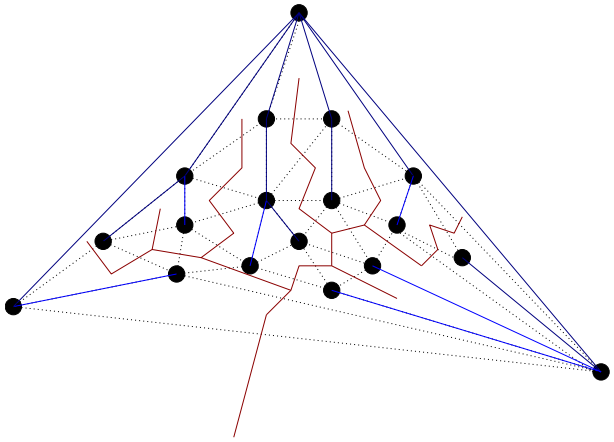


Figure 7.2: The edges of (V, E') are in blue. The edges of $(V^*, E - E')$ are in red.

Lemma 7.2.3 *Let $G = (V, E)$ be a connected plane graph with dual $G^* = (V^*, E)$ and let $E' \subseteq E$. Then (V, E') is a spanning tree of G if and only if $(V^*, E - E')$ is a spanning tree of G^* (see Figure 7.2 for an illustration).*

Let E_T be the edges of the spanning tree T, constructed by following the parents in D as stated above. Recall that the diameter of T is at most $2\sqrt{n}$. Also D is triangulated. Consider the dual D^* of D, and consider the edges in $E - E_T$. They define a spanning tree T' in D^* (by Lemma 7.2.3). Also we can orient each edge in T' away from the root. Pick a face of D (say its outer face) and choose this as the root T'. It will turn out that the required separator S_D will be defined by an edge, $e = (u, v)$ in $e \in E - E_T$, and the unique path in the tree T between u and v. In other words, e defines a unique cycle, $c(e)$, in T. The cycle $c(e)$ is referred to as a *fundamental cycle* in literature. To compute/define $c(e)$ appropriately we first perform a DFS of T' and compute the following three quantities.

1. $I(e)$= number of vertices which are in the interior of the cycle $c(e)$.
2. $|c(e)|$= number of vertices on the cycle $c(e)$.
3. Linked list representation of $c(e)$.

For each step of DFS, one of the following four cases will occur (see Figure 7.3)

Case 1: DFS visits a leaf of T' (i.e. a triangular face of D). Then $I(e) = 0$, $|c(e)| = 3$, and $c(e) = \{x, u, v\}$.

Case 2: DFS visits a triangle corresponding to an edge $e = (u, v) \in E - E_T$, its degree is two and the other edge of the triangle is $e' = (u', v) \in E - E_T$ which was visited in the previous step. Moreover $u' \in c(e)$. Then $I(e) = I(e')$, $|c(e)| = |c(e')| + 1$, and $c(e) = uc(e')$.

Case 3: DFS visits a triangle corresponding to an edge $e = (u, v) \in E - E_T$, its degree is two and the other edge of the triangle is $e' = (u', v) \in E - E_T$ which was visited in the previous step. Moreover $u' \notin c(e)$. Then $I(e) = I(e') + 1$, $|c(e)| = |c(e')| - 1$, and $c(e)$ equals to $c(e')$ except that u' is removed from the front of the list.

Case 4: DFS visits a triangle corresponding to edge $e = (u, v) \in E - E_T$ and its degree is three. The other two edges $e' = (u, y) \in E - E_T$ and $e'' = (vy) \in E - E_T$ have been already visited by the DFS. Let p be the common path between the cycles $c(e')$ and $c(e'')$. One of the end points of p is x, and the other end point is y. Then $I(e) = I(e') + I(e'') + |p| - 1$, $|c(e)| = |c(e')| + |c(e'')| - 2|p| + 1$, and $c(e)$ consists of $c'xc''$, where c' is the cycle $c(e')$ with path p removed, and similarly c'' is the cycle $c(e'')$ with path p removed.

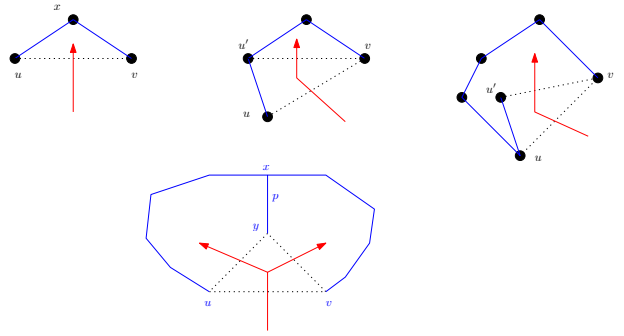


Figure 7.3: Red edges represent the DFS traversal of faces of D corresponding to the tree T'. Dashed edges corresponds to edges in $E - E_T$ and blue edges corresponds to edges in E_T, i.e. a spanning tree of D. Case 1: DFS visits a leaf, i.e. DFS visits a triangle corresponding to $e = (u, v) \in E - E_T$, its degree is one. Case 2: DFS visits a triangle corresponding to $e = (u, v) \in E - E_T$, its degree is two and the other edge of the triangle is $e' = (u', v) \in E - E_T$ which was visited in the previous step and $u' \in c(e)$. Case 3: Same as Case 2 except that $u' \notin c(e)$. Case 4: DFS visits a triangle corresponding to edge $e = (u, v) \in E - E_T$ and its degree is three.

Lemma 7.2.4 *In the above setting of the graph D, there exists an edge $e \in E - E_T$ such that $I(e) \leq 2/3n$ and $n - (I(e) + |c(e)|) \leq 2/3n$.*

Proof. Let $e \in E - E_T$ be the first edge in the leaf to root path in T' such that $I(e) + |c(e)| \geq n/3$. Then $n - (I(e) + |c(e)|) \leq 2/3n$. We will prove that $I(e) \leq 2/3n$. The edge e corresponds to one of the four cases encountered in the DFS.

1. In Case 1, $I(e) = 0 \leq 2/3n$.
2. In Case 2, $I(e) + |c(e)| = I(e') + |c(e')| + 1$, and $I(e') + |c(e')| < n/3$, and hence $I(e) + |c(e)| \leq 2/3n$.
3. In Case 3, since $I(e) + |c(e)| = I(e') + |c(e')|$, e cannot be the first edge with this property.
4. In Case 4, $I(e') + |c(e')| < n/3$ and so is $I(e'') + |c(e'')| < n/3$. $I(e) + |c(e)| = I(e') + I(e'') + |p| - 1 + |c(e')| + |c(e'')| - 2|p| + 1 \leq 2/3n - |p| \leq 2/3n$.

∎

7.3 *Generalizations of the Planar Separator Theorem*

In the previous section we saw that the planar separator theorem provides us with a procedure to separate the vertices of a planar graph $G = (V, E)$, $|V| = n$, into three sets A, B, S, where $|A|, |B| \leq 2n/3$, $|S| \leq 4\sqrt{n}$, and there exists no edge between A and B. In this section we will consider some generalizations of this theorem.

7.3.1 *Weighted Separators*

In our version of the planar separator theorem we considered all vertices to be equal. However, a common variant permits vertices to be weighted such that the sum of all weights is equal to 1 (any other set of non-negative weights can be trivially mapped to one like this). The only difference is that instead of bounding the sizes of the the sets A and B to be $\leq 2n/3$, we bound their weights to be $\leq 2/3$. The separating set S however is still bounded in terms of the number of vertices it contains, and may therefore have arbitrary weight.

To prove the weighted planar separator theorem, our proof from the previous section is sufficient. We need only change certain references to the sizes of sets to refer to the weight of the sets. The rest of the analysis largely follows unchanged.

From now on, we will assume that the planar separator theorem refers to the weighted variant of the planar separator theorem. If no weights are specified, we will assume that all vertices have equal weight, which coincides with our original definition.

7.3.2 *r-Divisions*

In this section we will use the planar separator theorem to construct a more general graph partitioning. The contents of this section are based on a paper by Frederickson [4].

[4] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987

We define a *region* to be a subset of the vertices of a graph $G = (V, E)$. An *interior vertex* of a region R is contained only in R, and adjacent only to other vertices in R. A *boundary vertex* is one that is shared between at least two regions. All vertices will be either boundary or interior. Given a parameter r, we will divide the graph into $\Theta(n/r)$ regions with $O(r)$ vertices each, and $O(\sqrt{r})$ boundary vertices each. Such a division will be called an *r-division*. Note that the planar separator theorem provides an n-division by taking the two sets $A \cup S$ and $B \cup S$.

We begin with a potential naive algorithm. Start with a single region containing all of V. While any region R contains more than r vertices, apply the planar separator theorem on R to produce A, B, S. Now replace R with $R' = A \cup S$ and $R'' = B \cup S$.

Clearly this procedure produces regions with no more than r vertices, and since it reduces the size of a region by at most $2/3$ until this bound is satisfied, it follows that each region contains $\Theta(r)$ vertices. Consequently, there must be $\Theta(n/r)$ regions. However the number of boundary vertices is more complicated. Initially we have a single region that is made of all interior vertices. Further, A and B consist entirely of interior vertices after constructing R' and R'', and S consists entirely of boundary vertices. Therefore we introduce at most $4\sqrt{n}$ boundary vertices at each recursive step.

To determine the number of boundary vertices, we define $b(v)$ for some vertex v to be one less than the number of regions it is contained in, and $B(n, r)$ to be the sum of $b(v)$ for all $v \in V$. Note that $B(n, r)$ is strictly greater than the number of boundary vertices, as $b(v) \geq 1$ for all boundary vertices, by definition. Our algorithm gives us the following recurrence:

$$\begin{aligned} B(n,r) &\leq 4\sqrt{n} + B(\alpha n + O(\sqrt{n}), r) + B((1-\alpha)n \\ &\quad + O(\sqrt{n}), r) && \text{for } n > r \\ B(n,r) &= 0 && \text{for } n \leq r, \end{aligned}$$

where $1/3 \leq \alpha \leq 2/3$. This recurrence can be solved for $B(n, r) \leq 4n/\sqrt{r} - O(\sqrt{n})$. Therefore, the number of boundary vertices produced by this algorithm is $O(n/\sqrt{r})$. However this tells us nothing about the number of boundary vertices *per region*. Indeed, some regions may have many boundary vertices. To resolve this, we perform further processing on regions with more than $c\sqrt{r}$ boundary vertices, for some constant c. Given such a region R, we set all k boundary vertices of R to have weight $1/k$, and all interior vertices of R to have weight 0. We then apply the planar separator theorem to R and replace R as before. Since only the boundary vertices have weights, the planar separator theorem will split up the boundary

vertices among the two resultant regions. Therefore, after enough iterations all regions will have few enough boundary vertices. Further, since the regions are still strictly shrinking, we cannot have violated the bounds on the size of the region. It remains to be proven that we have not violated the constraint on the maximum number of regions.

If a region has $i > c\sqrt{r}$ boundary vertices, then at most $di/(c\sqrt{r})$ splits will be performed, for some constants c and d. This will result in at most $di/(c\sqrt{r})$ new regions. If t_i is the number of regions with i boundary vertices, then the number of new regions will be at most

$$\sum_i (di/c\sqrt{r})t_i = O(n/r)$$

Therefore, our modified algorithm produces an r-division.

In our construction, the recursion tree has a depth of $O(\log(n/r))$. Further, for each level we spend $O(n)$ time. Therefore, this algorithm runs in $O(n \log(n/r))$ time.

Further processing on the graph can provide our r-division with additional properties such as regions having a constant number of neighbors, and boundary vertices being shared between at most a constant number of regions.

7.3.3 *Edge Separators*

Up until now, we have only considered vertex separators. Edge separators are exactly the same as vertex separators, except that instead of removing vertices, we wish to remove edges. Specifically, given a graph $G = (V, E)$ we wish to find a cut-set $S \subseteq E$ that separates V into two disjoint subsets A, B. Every edge in S has one endpoint in A and one endpoint in B, and every edge in $E \setminus S$ has both of its endpoints in only A or B. In general we would like to ensure that A and B are approximately the same size, and S is small.

For graphs with low (e.g. constant) maximum degree, the results for edge separators are generally very similar to those for vertex separators. However on arbitrary planar graphs, edge separators perform much worse.

For instance, consider a graph $G = (V, E)$ in which every vertex has degree 1, except for some vertex v with degree $n - 1$. This graph is a tree, and therefore planar. An excellent vertex separator for G would be $\{v\}$, as it would disconnect the entire graph, allowing us to pick any subsets of $V \setminus \{v\}$ we want for our separated sets. However an edge separator would have to remove a linear number of edges to get balanced sets.

From this example it is clear that not all results for vertex separators hold for edge-separators. In general, vertex separators are more powerful, as for every edge an edge-separator would need to remove,

a vertex separator would need to remove at most one vertex, but potentially far fewer. Equivalently, if a vertex separator includes some vertex v, an edge separator would need to include every edge of v to achieve the same result. Consequently, results on edge separators often include factors based on the maximum or average degree of the graph [61].

We conclude our look at the planar separator theorem and its generalizations with a table of separator results. There are far too many results on separators with special requirements and for special classes of graphs to adequately report here. As a result, this table is by no means comprehensive. Note that $\Delta(G) = \sum_{v \in V} deg(v)^2$, $\sigma(G) = \sum_{v \in V} c(v)^2$ and $c(v)$ is cost associated with each vertex, and $T_{SSSP}(G)$ denotes the time to compute single-source shortest paths in G.

Separator	Graph	# of Sets	Set Sizes	Separator Size	Time	Ref.
Vertex	Tree	2	$\leq 2n/3$	1	$O(n)$	[117]
Vertex	Planar	2	$\leq 2n/3$	$O(\sqrt{n})$	$O(n)$	[105]
Vertex	Planar	$\Theta(n/r)$	$O(r)$	$O(n/\sqrt{r})$	$O(n \log(n/r))$	[60]
Vertex	Genus g	-	$\leq \epsilon n$	$O(\sqrt{(g+1/\epsilon)n})$	$O(n+g)$	[3]
Vertex	Planar	-	$tw(G)$	$\leq 4\sqrt{2\sigma(G)/t}$	$O(n + T_{SSSP}(G))$	[4]
Edge	Planar	-	$tw(G)$	$\leq 4\sqrt{2\Delta(G)/t}$	$O(n + T_{SSSP}(G))$	[4]

7.4 Graph Laplacian

This section is based on [5]. First, recall eigenvalues and eigenvectors of a matrix from Chapter 4. Given an $n \times n$ matrix A, a non-zero vector v is an eigenvector of A, if $Av = \lambda v$ for some scalar λ. λ is the eigenvalue corresponding to the vector v. If A has n distinct eigenvalues, then the corresponding eigenvectors are linearly independent. Let S be a real symmetric matrix. All eigenvalues of S are real, and all the components of the eigenvectors are real. Any pair of eigenvectors of S corresponding to two different eigenvalues are orthogonal.

[5] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(2):298–305, 1973; and Daniel A. Spielman. *Spectral and Algebraic Graph Theory*. Yale University, USA, 2019

Symmetric matrix S is *positive semi-definite* if all its eigenvalues are ≥ 0. Alternatively, for all non-zero vectors $x \in R^n$, if $x^T S x \geq 0$ holds, then all the eigenvalues of S are ≥ 0.

As a warmup, consider the adjacency matrix of a complete graph on n-vertices. It consists of all 1s except that the diagonal entries are 0s. For example, for K_4, we have $A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$ and its eigenvalues are $\lambda_1 = n - 1 = 4$ and $\lambda_2 = \lambda_3 = \lambda_4 = -1$.

Definition 7.4.1 *(**Graph Laplacian Matrix**)*

Let $G = (V, E)$ be a graph with n vertices. Let A be its adjacency matrix of size $n \times n$. Let D be an $n \times n$ diagonal degree matrix, where

$$D(i,j) = \begin{cases} degree(v_i), \text{ if } i = j \\ 0, \text{ otherwise} \end{cases}$$

The graph Laplacian matrix L of the graph G is given by $L = D - A$. The set of eigenvalues of L constitutes the spectrum *of G.*

Example 7.4.2 *The Laplacian matrix for complete graph K_4 is given by*

$$L = D - A = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix}$$

We make some observations.

Observation 7.4.3 *Let $\overrightarrow{1}$ be a n-dimensional vector where all of its components are 1. The vector $\overrightarrow{1}$ is an eigenvector of L and the corresponding eigenvalue is 0.*

Observation 7.4.4 *L is a symmetric matrix and hence all of its eigenvalues will be real, and the components of all eigenvectors are real.*

Let N be the node-edge incidence matrix of dimension $|V| \times |E|$ of the graph $G = (V, E)$. Let $V = \{1, \ldots, n\}$. There is a row corresponding to each vertex - row i for vertex i in N. Similarly, there is a column corresponding to each edge $e \in E$. For each edge $e = (i, j)$, $i < j$, the entries in its column in N corresponding to the row of vertex i is 1, the entry corresponding to the row of vertex j is -1, and all other entries are 0.

Example 7.4.5 *Let $G = (V, E)$ be K_3. Let $V = \{a, b, c\}$.*

The node-edge matrix $N =$

	ab	*ac*	*bc*
a	1	1	0
b	-1	0	1
c	0	-1	-1

Observe that

$$NN^T = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} = D - A = L$$

Example 7.4.6 *Let $G = (V, E)$ be K_{13}. Let $V = \{a\} \cup \{b, c, d\}$.*

The node-edge matrix $N =$

	ab	*ac*	*ad*
a	1	1	1
b	-1	0	0
c	0	-1	0
d	0	0	-1

Observe that

$$NN^T = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} = D - A = L$$

Observation 7.4.7 $L = NN^T$.

Proof. Consider the ij-th entry of NN^T. First consider the case that $i = j$. Row i of N consists of information of all the edges incident on vertex i (encoded as +1 or -1). The number of non-zero entries in row i is equal to the degree of vertex i in G. Since the entry NN^T_{ii} is the dot product of row i with itself, and hence it equals the degree of vertex i.

Now consider when $i \neq j$. Here we want to understand what is the dot product of rows i and j of N. Each column of N has only two non-zero entries, namely $+1$ and -1, and no two columns are the same as edges connect distinct pairs of vertices. Hence the dot product is either 0 or -1. It is -1 if and only if $ij \in E$. ∎

Observation 7.4.8 *L is positive semi-definite matrix. Its smallest eigenvalue is 0 and all other eigenvalues are ≥ 0.*

Proof. In the singular value decomposition in Chapter 4, we have already seen that for a real matrix A, AA^T and A^TA are real-symmetric positive semi-definite matrices. Hence all the eigenvalues of L are ≥ 0. By Observation 7.4.3, 0 is an eigenvalue of L with respect to the eigenvector $\overrightarrow{1}$. ∎

We can also express L as sum of Laplacian matrices corresponding to each edge of G. For an edge $e = (i, j)$, define the $n \times n$ Laplacian matrix L_e, where all the entries are 0 except $L_{ii} = 1, L_{ij} = -1, L_{ji} = -1, L_{jj} = 1$. Now $L = \sum_{e \in E} L_e$. Exercise 7.12 asks you to prove that L_e and L are positive semi-definite matrices, and results in an alternate proof of Observation 7.4.8.

Let b_e be a n-dimensional column vector consisting of 1 and -1 in i and j coordinate, respectively, and the remaining coordinates are 0.

Observation 7.4.9 $L_e = b_e b_e^T$.

By Observation 7.4.8, one of the eigenvalues of L for any graph G is 0 and it corresponds to the eigenvector $\overrightarrow{1}$. Next, we show that G is connected if and only if the spectrum of L consists of eigenvalue 0 with multiplicity 1 (i.e. 0 occurs exactly once as an eigenvalue).

Lemma 7.4.10 *A graph $G = (V, E)$ is connected if and only if the eigenvalue 0 occurs with multiplicity 1 in $L(G)$.*

Proof. First, assume that G is disconnected. It consists of two or more components. Assume that it has two components G_1 and G_2. Then $L(G)$ can be expressed (possibly may require permutation of the rows) as $L(G) = \begin{bmatrix} L(G_1) & 0 \\ 0 & L(G_2) \end{bmatrix}$. If the number of vertices in the component G_1 are k, than observe that $a = (1, \ldots, 1, 0, \ldots, 0)$ and $b = (0, \ldots, 0, 1, \ldots, 1)$, where a (resp., b) consists of k 1s (resp., 0s) followed by $n - k$ 0s (resp., 1s), are eigenvectors of L with eigenvalue 0. Since a and b are linearly independent, the eigenvalue 0 occurs with multiplicity > 1.

Now assume that G is connected. Consider the following:

$$\begin{aligned} x^T L x &= x^T \left(\sum_{e \in E} L_e \right) x \\ &= \sum_{e \in E} x^T L_e x \\ &= \sum_{e \in E} x^T b_e b_e^T x \text{ (see Observation 7.4.9)} \\ &= \sum_{e=(i,j) \in E} (x_i - x_j)^2 \\ &\geq 0 \end{aligned}$$

Consider an eigenvector x corresponding to eigenvalue 0. Then $Lx = \lambda x = 0$. Thus $x^T L x = 0 = \sum_{e=(i,j) \in E} (x_i - x_j)^2$. This implies that all $x_i = x_j$ for each edge $e = (i, j)$. But G is connected, and hence it will be the case that $x = (\alpha, \alpha, \ldots, \alpha)$ for some non-zero α. Therefore x and the vector of all 1s are in the same direction. This implies that only eigenvector corresponding to eigenvalue 0 is $\overrightarrow{1}$. ∎

Now assume that given graph $G = (V, E)$ is connected. Then the second smallest eigenvalue $\lambda_2 > 0$ of its Laplacian matrix $L(G)$. Consider the eigenvector v_2 corresponding to λ_2. Since L is symmetric, v_2 is orthogonal to the eigenvector $\overrightarrow{1}$ corresponding to eigenvalue 0. This implies that $\overrightarrow{1} \cdot v_2 = 0 \implies \sum_{i=1}^{n} v_{2i} = 0$, where v_{2i} denotes the i-th coordinate of v_2. Since $v_2 \neq \overrightarrow{0}$, some of the coordinates of v_2 are positive and some are negative. We can partition the set of vertices V into two groups A and B, where A consists of all the

vertices corresponding to positive coordinates of v_2, and set $B = V \setminus A$. This typically results in a nice partition of G provided that λ_2 is small. There is an alternate way to partition the graph using the second eigenvector v_2 using the concept of graph conductance.

Definition 7.4.11 *(**Conductance**) For a graph $G = (V, E)$, define $vol(S)$, where $S \subseteq V$, to be the sum total of the degree of the vertices in S. Note that $Vol(V) = 2|E|$. Let $\delta(S)$ be the number of edges in the $cut(S, V \setminus S)$, i.e. the number of edges where one end is in S and the other end is in $V \setminus S$. Define the conductance of the $cut(S, V \setminus S)$ as*

$$\Phi(S) = \frac{\delta(S)}{vol(S)}$$

The conductance of the graph G as

$$\Phi(G) = \min_{S \subset V : vol(S) \leq |E|} \frac{|\delta(S)|}{vol(S)}$$

It is the smallest fraction with respect to the number of edges in the $cut(S, V \setminus S)$ and the volume of the set S. To be more precise, we should express

$$\Phi(G) = \min_{S \subset V : |S| \leq |V|/2} \frac{|\delta(S)|}{(\min vol(S), vol(V \setminus S))}$$

Example 7.4.12 *Conductance of a cycle on n vertices C_n is $\frac{2}{n}$. Let the vertices in order in the cycle be $1, 2, \ldots, n$ and assume n is even. We form two components $A = \{1, \ldots, n/2\}$ and $B = V \setminus A$. Now conductance of the $cut(A, B) = \frac{2}{\min(vol(A), vol(B))} = \frac{2}{n}$, as $vol(A) = vol(B) = \sum_{i=1}^{n/2} deg(i) = n$.*

Example 7.4.13 *Conductance of K_n is $\frac{1}{2}$. Partition the set of vertices in two groups of equal size. Observe that the volume of a group is $(n-1)n/2$ and the number of edges in the cut is $(n/2)^2$.*

Definition 7.4.14 *(**Sparse Cut**) A subset $S \subset V$ forms a sparse cut of G if $\Phi(S) = \frac{|\delta(S)|}{vol(S)}$ is small (for example less than 0.1). The edges connecting S with the rest of the graph are very few compared to the sum total of degrees of the vertices in S. By removing the edges on the cut, we partition G into subgraphs of smaller sizes.*

The second eigenvalue λ_2 and the corresponding eigenvector v_2 of L are used as follows to compute the sparse cut of G.

Sparsecut Heuristic

Input: A connected graph $G = (V, E)$.
Output: A Sparse Cut $(S, V \setminus S)$.

Step 1: Compute the second Eigenvector of L and let it be $v_2 \in R^n$, i.e. $Lv_2 = \lambda_2 v_2$

Step 2: Sort coordinates of v_2 and let the order be $v_{21} \geq v_{22} \geq \ldots \geq v_{2n}$. Note that v_{2i} corresponding to the value assigned to the vertex i by the eigenvector v_2.

Step 3: For $i := 1$ to n do, let $S_i = \begin{cases} \{v_1, \ldots, v_i\}, \text{ if } i \leq |V|/2 \\ \{v_{i+1}, \ldots, v_n\}, \text{ if } i > n/2 \end{cases}$

Step 4: Return $(S_i, V \setminus S_i)$ corresponding to $\min\limits_{1 \leq i \leq n} \phi(S_i)$ as the sparse cut of G.

Example 7.4.15 *Consider the graph in the margin.*

$$L = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 4 & -1 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 3 & -1 \\ 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{bmatrix}$$

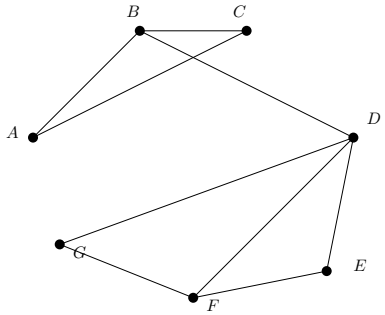


Figure 7.4: Figure for Example 7.4.15

$\lambda_2 = 0.398$ *and* $v_2 = (-1.38, -.83, -1.38, .6, 1, 1, 1)$

S_i	$\phi(S_i)$
E	*2/2*
E,F	*3/5*
E,F,G	*3/7*
D,B,C,A	*3/*min(7, 11)
A,B,C	**1/7**
A,C	*2/4*
A	*2/2*

Example 7.4.16 *Consider the graph in the margin.*

$$L = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & -1 & 0 & 0 & -1 \\ -1 & -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 4 & -1 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 3 & -1 \\ 0 & -1 & 0 & -1 & 0 & -1 & 3 \end{bmatrix}$$

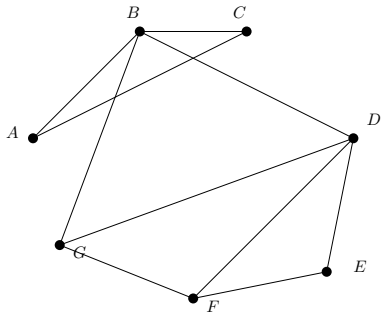


Figure 7.5: Figure for Example 7.4.16

$\lambda_2 = 0.64$ *and* $v_2 = (-2.93, -1.04, -2.93, 1.34, 2.5, 2.05, 1)$

S_i	$\phi(S_i)$
E	*2/2*
E,F	*3/5*
E,F,D	*3/9*
G,B,A,C	*3*/min(9, 11)
B,A,C	**2/8**
A,C	*2/4*
C	*2/2*

In both the above examples, the sparse cut has been obtained by the same set $S = \{A, B, C\}$. We would have got the same cut if we partitioned the vertices with respect to the positive and negative coordinates in the second eigenvector.

The sparse cut heuristic is grounded on Cheeger's inequality, stating that the conductance and second smallest eigenvalue of a connected graph G are closely related. The inequality states the following.

Lemma 7.4.17 (**Cheeger's inequality**) *Let $\mathcal{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ be the normalized adjacency matrix, and let $\mathcal{L} = I - \mathcal{A}$ be the normalized Laplacian matrix of a connected graph $G = (V, E)$. It is known that for the normalized Laplacian matrix, all its eigenvalues $\lambda_i \in [0, 2]$. Let G be a connected graph and let $\lambda_2 > 0$ be the second smallest eigenvalue of $\mathcal{L}$. Cheeger's inequality states that*

$$\lambda_2/2 \leq \Phi(G) \leq \sqrt{2\lambda_2}$$

The normalization removes the dependence on the largest degree in the Laplacian.

See Exercise 7.17

Computation of the conductance of the graph is a **NP**-hard problem. Rather than computing the conductance, the sparse cut heuristic finds the second smallest eigenvalue λ_2 in polynomial time. If $\lambda_2 \to 0$, then $\Phi(G) \to 0$ as by Cheeger's inequality $\lambda_2/2 \leq \Phi(G) \leq \sqrt{2\lambda_2}$. Moreover, the same inequality also states that if $\lambda_2 \nrightarrow 0$, then $\Phi(G) \nrightarrow 0$.

7.5 Exercises

7.1 *Let T=(V,E) be a connected undirected tree such that each vertex has degree at most 3. Let n=|V|. Show that T has an edge whose removal disconnects T into two disjoint subtrees with no more than (2n+1)/3 vertices each. Give a linear time algorithm to find such and edge; prove its correctness.*

7.2 *Provide an algorithm running in $O(n \log k)$ time to partition the binary tree on n vertices into k ($k \leq n$) subtrees, so that each of the subtree is of size at most $(2/3)^k n$. Try to see whether you can improve the running time of this algorithm (this is not easy!).*

7.3 *Prove the weighted version of the planar-separator theorem. Let $G = (V, E)$ be an embedded undirected triangulated planar graph, where $n = |V|$. Each vertex $v \in V$ has a positive weight $w(v) \geq 0$ and $\sum_{v \in V} w(v) = 1$. There exists a partition of V into disjoint sets A, B, and S, such that*

1. *$w(A), w(B) \leq \frac{2}{3}$, where $w(A)$ is the sum total of weights of all the vertices in set A*
2. *$|S| \leq 4\sqrt{n}$*
3. *There is no edge in E that joins a vertex in A with a vertex in B.*
4. *Such a set S can be found in linear time.*

7.4 *Prove Lemma 7.2.2. Let $G = (V, E)$ be a connected planar graph and G^* be its dual. For any $E' \subseteq E$, the subgraph (V, E') has a cycle if and only if the subgraph $(V^*, E - E')$ of G^* is disconnected.*

7.5 *Prove the following theorem on Geometric Separators. In 2-dimensions assume that you have n squares of arbitrary sizes. Squares are axis aligned. Moreover none of the points in the plane is inside more than k-squares. Prove that there exists either a vertical or a horizontal line which partitions the set of squares in such a way that at least $\lfloor \frac{n+1-k}{4} \rfloor$ of squares interiors lie to each side of the line. How fast you can find such a line?*

7.6 *Show that for a complete graph on n vertices, the eigenvalues of its adjacency matrix are $\lambda_1 = n - 1$, and $\lambda_2 = \ldots = \lambda_n = -1$. (Hint: Think about the eigenvectors corresponding to these eigenvalues.)*

7.7 *Define the trace of a square $n \times n$ matrix A to be the sum of its diagonal entries, i.e., $tr(A) = \sum_{i=1}^{n} A_{ii}$. Show that $tr(A) = \sum_{i=1}^{n} \lambda_i$, where $\lambda_1, \ldots, \lambda_n$ are eigenvalues of A. (Hint: Consider the polynomial equation $det(A - \lambda I) = (\lambda - \lambda_1)(\lambda - \lambda_2) \ldots (\lambda - \lambda_n)$.)*

7.8 *Show that the eigenvalues of the adjacency matrix of a bipartite graph occur in complementary pairs. If λ is one of the eigenvalues then $-\lambda$ is also an eigenvalue. In other words, the spectrum of a bipartite graph is symmetric around the origin. (Hint: Observe that the adjacency matrix of a bipartite graph can be rearranged to look like $A = \begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix}$. Now if λ is an eigenvalue and $A \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \end{bmatrix}$, then show that $\begin{bmatrix} x \\ -y \end{bmatrix}$ is also an eigenvector corresponding to eigenvalue $-\lambda$).*

7.9 *Show that for a graph $G = (V, E)$, if the eigenvalues of its adjacency matrix A come in complementary pairs, then G is a bipartite graph. (Hint: To show that G is bipartite, it is sufficient to show that it has no odd-length*

cycles. Consider matrices A^k for odd powers of k. If any of the diagonal entries of A^k is > 0, we know it has an odd cycle. Show that if eigenvalues of A come in complementary pairs than $tr(A^k) = 0$ and conclude that all diagonal entries of A^k are 0 for odd powers of k.)

7.10 *Show that for a complete bipartite graph K_{mn}, two eigenvalues of its adjacency matrix are $\sqrt{mn}$ and $-\sqrt{mn}$. Moreover, all the remaining $m + n - 2$ eigenvalues are 0. (Hint: Show that the rank of the adjacency matrix of A is 2. This implies that there are two non-zero eigenvalues. Let us call them λ_1 and λ_2. Use the fact that $tr(A) = 0$ to show that $\lambda_1 = -\lambda_2 = \alpha$. Express $det(A - \lambda I) = (\lambda - \lambda_1)(\lambda - \lambda_2)\lambda^{n+m-2} = \lambda^{n+m} - \alpha^2\lambda^{n+m-2}$. Expand $det(A - \lambda I)$, and show that $\alpha = \sqrt{mn}$. Think of what contributes to the coefficient of λ^{n+m-2} term in the expansion of $det(A - \lambda I)$.)*

7.11 *Let G be d-regular graph. Show that if v is an eigenvector of the adjacency matrix A corresponding to eigenvalue λ, than v is eigenvector of the Laplacian matrix L of G with eigenvalue $d - \lambda$. What can you say about the spectrum (the set of all eigenvalues) of A and L in this case?*

7.12 *Show that if A and B are two square symmetric positive semi-definite $n \times n$ real matrices, than $A + B$ is a square symmetric positive semi-definite matrix. Let $G = (V, E)$ be a graph and let L_e be the Laplacian $n \times n$ matrix corresponding to an edge $e \in E$. Show that L_e is positive semi-definite. Show that $L = \sum_{e \in E} L_e$ is positive semi-definite matrix.*

7.13 *Show that a graph $G = (V, E)$ has k-connected components if and only if the eigenvalue 0 occurs with multiplicity k in $L(G)$. (Hint: See proof of Lemma 7.4.10).*

7.14 *Let e_i denotes the standard basis vector in n-dimensional Euclidean space. It has 1 in the i-th coordinate and all other coordinates are 0. Let $G = (V, E)$ be a graph, and let $V = \{1, \ldots, n)$. Show that the Laplacian matrix $L(G) = \sum_{e=(i,j) \in E} (e_i - e_j)(e_i - e_j)^T$.*

7.15 *Given a graph $G = (V, E)$, where $V = \{1, \ldots, n\}$, and an n-dimensional real vector x, whose i-th coordinate corresponds to a real value assigned to vertex i, show that $x^T Lx = \sum_{e=(i,j) \in E} (x_i - x_j)^2$.*

7.16 *In the four graphs in Figure 7.6, two of the vertices have been assigned values. Find an assignment of real values to the remaining vertices so that $\sum_{e=(i,j) \in E} (x_i - x_j)^2$ is minimized.*

Following exercises are derived from Lap Chi Lau's notes.

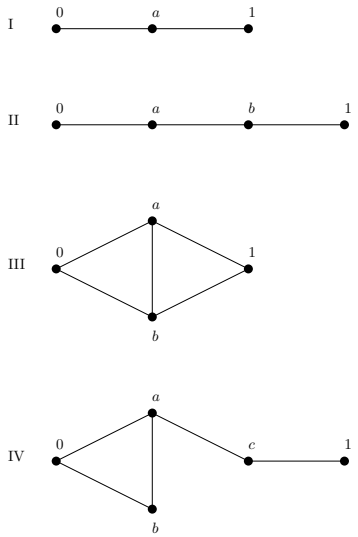


Figure 7.6: Graphs for Exercise 7.16

7.17 *Let D be a diagonal degree matrix of a connected graph G. Let $\mathcal{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ be its normalized adjacency matrix. Define $\mathcal{L} = I - \mathcal{A}$ as its normalized Laplacian matrix. Answer the following*

1. *Show that $\mathcal{L} = D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$*
2. *Show that 0 is an eigenvalue of $\mathcal{L}$ corresponding to the vector $D^{-\frac{1}{2}}\overrightarrow{1}$.*
3. *Show that $\mathcal{L}$ is positive-semi definite. (Hint: Recall that $L = \sum_{e \in E} L_e$. For positive semi-definite, try to show that $x^T\mathcal{L}x \geq 0$ for any $x \in \Re^n$.)*
4. *Show that the largest eigenvalue of $\mathcal{A}$ is ≤ 1 as eigenvalues of $\mathcal{L} = I - \mathcal{A}$ are ≥ 0.*
5. *Show that $I + \mathcal{A}$ is positive semi-definite. (Hint: Express $x^T(I + \mathcal{A})x = x^T\mathcal{L}x + 2x^T\mathcal{A}x$)*
6. *Show that the smallest eigenvalue of $\mathcal{A} \geq -1$.*
7. *Conclude that all eigenvalues of $\mathcal{L}$ are in $[0, 2]$, and all eigenvalues of $\mathcal{A}$ are in $[-1, 1]$.*

The following exercise defines the concept of Rayleigh Quotient that connects eigenvalues and eigenvectors to optimization problems.

7.18 *Let S be a real symmetric $n \times n$ matrix. Let its eigenvalues be $\alpha_1 \geq \alpha_2 \ldots \geq \alpha_n$ and the corresponding orthonormal eigenvectors are $v_1, v_2, \ldots, v_n$, respectively. First we show that*

$$\alpha_1 = \max_{x \in \Re^n, x \neq \overrightarrow{0}} \frac{x^TSx}{x^Tx}$$

We denote eigenvalues by α's to keep them distinct from eigenvalues λ's of Laplacian matrix.

Answer the following.

1. *Show that any vector $x \in \Re^n$ can be expressed as a linear combination of eigenvectors $v_1, v_2, \ldots, v_n$, i.e., $x = c_1v_1 + \ldots + c_nv_n$, where $c_1, \ldots, c_n$ are constants.*
2. *Show that $x^TSx = \sum_{i=1}^{n} c_i^2\alpha_i$.*
3. *Show that $x^Tx = \sum_{i=1}^{n} c_i^2$.*
4. *Show that $\frac{x^TSx}{x^Tx} = \frac{\sum_{i=1}^{n} c_i^2\alpha_i}{\sum_{i=1}^{n} c_i^2} \leq \alpha_1$*
5. *Show that for $x = v_1$, $\frac{x^TSx}{x^Tx} = \alpha_1$.*
6. *Conclude that $\alpha_1 = \max_{x \in \Re^n, x \neq \overrightarrow{0}} \frac{x^TSx}{x^Tx}$*

7. *Now consider all the vectors in $\Re^n$ that are orthogonal to v_1. Let the set of these vectors be T_2. Now for any vector $x \in T_2$, where $x = c_1v_1 + \ldots + c_nv_n$, we have that $c_i = x \cdot v_i$ and $c_1 = x \cdot v_1 = 0$. Show that $\alpha_2 = \max_{x \in T_2, x \neq \overrightarrow{0}} \frac{x^T S x}{x^T x}$.*

8. *In general, define T_k to be the set of vectors in $\Re^n$ that are orthogonal to $v_1, \ldots, v_{k-1}$. Show that $\alpha_k = \max_{x \in T_k, x \neq \overrightarrow{0}} \frac{x^T S x}{x^T x}$.*

9. *What can you say about smallest eigenvalues α_{n-1} and α_n in terms of Rayleigh Quotients?*

7.19 *Let $G = (V, E)$ be a d-regular graph, i.e., degree of each vertex is d. Let $V = \{1, \ldots, n\}$.*

1. *Show that for the normalized Laplacian matrix of G, $\mathcal{L}(G) = \frac{1}{d}L(G)$.*

2. *Using the Rayleigh Quotients, show that $\lambda_2 = \min_{x \perp \overrightarrow{1}} \frac{x^T \mathcal{L} x}{x^T x} = \min_{x \perp \overrightarrow{1}} \frac{\sum_{e=(ij) \in E} (x_i - x_j)^2}{d \sum x_i^2}$*

3. *Assume that the graph conductance of G is given by a set S such that $|S| = \frac{|V|}{2}$. Set $x_i = +1$ for each vertex $i \in S$ and $x_i = -1$ for each vertex in $V \setminus S$. Observe that $x \perp \overrightarrow{1}$. Show that $\lambda_2 \leq 2\Phi(S)$.*

4. *Now consider a general subset $S \subset V$ that defines the graph conductance, i.e. $\Phi(G) = \Phi(S)$. Set $x_i = \frac{1}{|S|}$ for all $i \in S$ and set $x_i = \frac{-1}{|V \setminus S|}$ for all $i \in V \setminus S$. Observe that $x \perp \overrightarrow{1}$. Show that $\lambda_2 \leq 2\Phi(S)$.*

This proves one direction of Cheeger's inequality for regular graphs.

Following exercises will help us build more intuition for the sparse cut heuristic. Let $G = (V, E)$ be a connected graph and let $V = \{1, \ldots, n\}$. Let $S = \{1, \ldots, i\} \subset V$. For each vertex $1 \leq j \leq i$, assign j an integer $s_j = +1$. Similarly, for each vertex $i + 1 \leq j \leq n$, assign the vertex j an integer $s_j = -1$. Hence, all the vertices in S are assigned $+1$ value and all the vertices in $V \setminus S$ are assigned a value of -1. Let $cut(S, V \setminus S)$ be the edges in the cut, i.e., one end of the edge is in S and the other end in $V \setminus S$. Answer the following.

1. Let $e \in E$. Show that if $e = (k, l) \in cut(S, V \setminus S)$, then $|s_k - s_l| = 2$.

2. Let $e \in E$. Show that if $e = (k, l) \notin cut(S, V \setminus S)$, $s_k - s_l = 0$.

3. Show that the number of edges in the cut, $|cut(S, V \setminus S)| = \frac{1}{4} \sum_{e=(i,j) \in E} (s_i - s_j)^2$.

4. Let A be the adjacency matrix of S. Show that $|cut(S, V \setminus S)| = \frac{1}{8} \sum_{1 \leq i,j \leq n} A_{ij}(s_i - s_j)^2$.

5. Let D be the diagonal $n \times n$ matrix where D_{ii} is the degree of vertex i, and all other entries $D_{ij} = 0$ for $i \neq j$. Show that the number of edges in the cut can be expressed as

$$|cut(S, V \setminus S)| = \frac{1}{4} \sum_{1 \leq i,j \leq n} (D_{ij} - A_{ij}) s_i s_j$$

6. Show that the size of the $cut(S, V \setminus S)$ can be expressed as

$$|cut(S, V \setminus S)| = \frac{1}{4} s^T L s,$$

where $L = D - A$ is the Laplacian matrix of G and s is the ± 1 vector indicating which vertices are in S and which ones are in the set $V \setminus S$.

7. Evaluate the size of the cut using the expression

$$\frac{1}{4} \sum_{1 \leq i,j \leq n} (D_{ij} - A_{ij}) s_i s_j$$

for the following graphs:
(a) P_4 - a path on 4 vertices and assume that S consists of the first two vertices of the path and $V \setminus S$ consists of the last two vertices,
(b) A cycle C_4 consisting of 4 vertices and assume that S consists of a pair of vertices connected by an edge, and
(c) K_4 and assume that S consists of exactly one vertex.

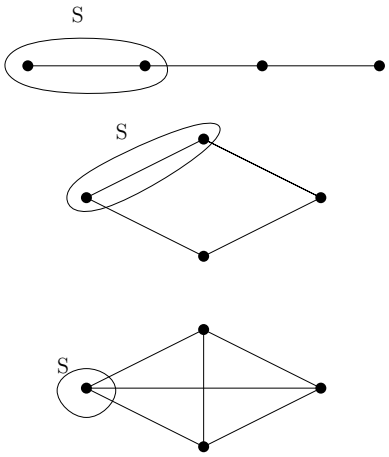


Figure 7.7: P_4, C_4, and K_4 with the set S.

7.20 *Consider the expression $\frac{1}{4} s^T L s$ that determines the number of edges in the cut, given the partition of V into S and $V \setminus S$ for a graph $G = (V, E)$. In this exercise we are looking for ways to find the best partition. Among all possible ± 1 n-dimensional vectors, let $s \in \{-1, +1\}^n$ minimizes $\frac{1}{4} s^T L s$ under the condition that $s \cdot \overrightarrow{1} = \sum_{i=1}^{n} s_i = 0$. Show that the resulting s gives us a minimum balanced partition of the graph G. (Note that $\frac{1}{4}$ can be ignored when we want to compute the best partition, as eventually we worry about the sign's of the coordinates in s.)*

7.21 *The optimization problem in the previous exercise is a discrete problem as $s \in \{-1, +1\}^n$ and $\sum_{i=1}^{n} s_i = 0$. Consider a relaxation where x is an n-dimensional vector in $\Re^n$, where $\sum x_i^2 = n$, and $\sum_{i=1}^{n} x_i = 0$ and it minimizes $\frac{1}{4} x^T L x$. Since $x \cdot \overrightarrow{1} = \sum x_i = 0$, for each $x_i \geq 0$, set $s_i = +1$ and for each $x_i < 0$, set $s_i = -1$. Show that by setting s_i's by this strategy, we obtain a partition of G.*

7.22 *Consider the problem of finding a vector $x \in \Re^n$ such that $x^T L x$ is minimized, where $\sum x_i^2 = n = x^T x$, and $x \cdot \overrightarrow{1} = \sum_{i=1}^{n} x_i = 0$. Let us use the*

Lagrange multiplier's method to find an optimum x. So we want to optimize the function $x^T Lx - \gamma(x^T x - n)$ under the constraint that $x \cdot \overrightarrow{1} = 0$. Answer the following.

We are using γ instead of λ here for Lagrange multiplier as λ is reserved for eigenvalues.

1. *Show that the partial derivative with respect to γ of the function $x^T Lx - \gamma(x^T x - n)$ is $x^T x - n$.*

2. *Compute the partial derivative with respect to x of the function $x^T Lx - \gamma(x^T x - n)$, where $x \cdot \overrightarrow{1} = 0$. Show that it corresponds to eigenvalue equation $Lx = \gamma x$.*

3. *Show that the optimal value for the optimization problem corresponds to eigenvalues and eigenvectors of the system $Lx = \gamma x$, where $x \cdot \overrightarrow{1} = 0$.*

4. *Recall that the smallest eigenvalue of L is 0 and the corresponding eigenvector is $\overrightarrow{1}$. Show that the minimum value of the objective function $\frac{1}{4}x^T Lx$ under the constraints that $x^T x = n$ and $x \cdot \overrightarrow{1} = 0$, is attained at the second smallest eigenvalue λ_2 of L.*

5. *Show that the optimum value of $\frac{1}{4}x^T Lx$ under the constraints that $x^T x = n$ and $x \cdot \overrightarrow{1} = 0$ is $\frac{n}{4}\lambda_2$. (Hint: Substitute $Lx = \lambda_2 x$ in the Lagrange equation.) Note that if $\lambda_2 \to 0$, then the size of the cut is small.*

6. *Consider the second eigenvector v_2 corresponding to λ_2. Partition the set V into S and $V \setminus S$ with respect to the sign of the coordinates in v_2. Show that $S \neq \emptyset$ and $S \neq V$, and hence $(S, V \setminus S)$ forms a proper cut of G.*

8
Locality-Sensitive Hashing

Given a collection of items we can identify duplicate items using hashing. For example, we hash items using an appropriate hash function, and the duplicate items will fall within the same bucket. Suppose, we want to identify near similar items. For example, suppose we have Boolean vectors in d dimensions, and we want to identify vectors that match in at least 98% of coordinates. Clearly, we cannot apply hashing directly as if the two vectors differ in at least one coordinate, we expect the hash function to place them in different buckets. This is the topic of study in this chapter.

The concept of *Locality-Sensitive Hashing* (LSH) is used to determine which items in a given set are similar under some well-defined similarity measure. The key idea is to hash the items using several hash functions. The hash functions have the property that the probability of collision is higher for items that are similar as compared to the items that are dissimilar. Hence the similar items are more likely to hash into the same buckets. Rather than using the naive approach of comparing all pairs of items within a set, LSH technique only compares items within a bucket, thereby reducing the number of comparisons. LSH is often used for finding similar items in very large data sets. LSH provides a method for efficient approximate nearest neighbor search and it has been used in data mining, pattern recognition, computer vision, computational geometry, data compression, spell checking, plagiarism detection, and chemical similarity.

This chapter is organized as follows. We will describe the LSH using an example of finding similar documents in a collection of documents. In Section 8.1 we explain what are shingles in a document, the set comprising of shingles in a document, and the notion of Jaccard similarity to measure the similarity between sets. In Section 8.2 we describe minhashing for summarizing sets. In Section 8.3 we describe the LSH technique for finding sets (documents) having high Jaccard similarity based on minhashing. In Section **??** we discuss finite metric spaces. Section 8.5 discusses the theory of locality sensi-

tive functions. In particular, we define a sensitive family of functions and show how to construct AND and OR families. In Section 8.6 we provide construction of various LSH families including Hamming distance, Cosine distance, Euclidean Distance, Fingerprint similarity, and Image similarities. We also discuss the properties of the similarity measure under which we can apply the theory of LSH. Section **??** consists of some problems for broadening our understanding of LSH technique and its applications. We conclude this chapter by providing some bibliographic remarks.

8.1 Similarity of Documents

We will introduce LSH using the problem of finding similar documents. This problem appears, for example, on the web when attempting to find similar, or even duplicate web pages. A search engine would use this technique to allow these similar documents to either be grouped or only shown once on a results page, so that other possible search matches could also be prominently displayed.

Problem 8.1.1 *Given a collection of web-pages, find the near duplicate web-pages.*

Clearly the content of the page is what matters, so for a preprocessing step any HTML tags are stripped away and main textual content is kept. Typically multiple white spaces are also replaced by a singe or no space during this preprocessing. As a result we are left with a document containing a string of text characters. Keeping in mind that we want to compare similarity of documents opposed to exact equality, the text is split up into a set of smaller strings by a process called *shingling*. This allows the documents to be represented as sets, where fragments of documents can match others. The shingle length k should be large enough so that the probability of any given k-shingle appearing in any given document is relatively low. But if the two documents are similar, the probability that a particular shingle will appear in both the documents is higher. For our purposes, it is sufficient to know that choosing $k = 5$ works well for electronic mail, and $k = 9$ is suitable for large text documents. In other words, we look for concatenation of strings made of k words.

Definition 8.1.2 k-shingle*: A k-shingle of a text document is defined to be any substring of length k which appears in the document.*

Example 8.1.3 *Let the document D = 'The cow jumped over the moon', and $k = 2$. Then the possible k-shingles for D are:*

$\{Thecow, cowjumped, jumpedover, overthe, themoon\}$.

For simplicity, we will assume that the document contains one long sequence made up of alphabets and we will work with k-shingles of the sequence.

Example 8.1.4 *Let the document $D = \{adbdabadbcdab\}$, and $k = 2$. Then the possible k-shingles for D are: $\{ad, db, bd, da, ab, ba, bc, cd\}$. Note that the set representing the shingles of D consists of unique shingles.*

Note that in the above example, D consists of 13 alphabets, each alphabet requires 1-byte of memory space. For $k = 2$, we need 8 shingles to represent D, each requiring 2-bytes of memory space. Thus, the shingle representation as such increases the memory requirement. However we can work instead with integers by using a hash function to map each k-shingle to an integer. For example, consider the following hash function that maps 9-byte shingles to an integer.

Example 8.1.5 *Let $k = 9$, and let $\mathcal{H}$ be a hash function that maps the set of characters to integers: $\mathcal{H} : |C|^9 \to \mathbb{Z}$. Let $|\mathbb{Z}| \leq 2^{32} - 1$.*

$\mathcal{H}$ potentially reduces the space requirements as a 9-byte shingle is converted to a 4-byte integer. However, this representation of shingles may use 4 times more memory space than the original document. A solution to overcome this is the subject of the next section.

Now that we have documents mapped to sets of k-shingles, we can use a similarity measure called *Jaccard similarity* to compare the two sets. The Jaccard similarity is defined with respect to two sets S and T, and is the ratio of the size of the intersection of S and T to the size of their union. See Figure 8.1 for an illustration.

Definition 8.1.6 Jaccard Similarity: *Let S and T be two sets. Define the Jaccard Similarity of S and T as $\mathsf{SIM}(S, T) = \frac{|S \cap T|}{|S \cup T|}$.*

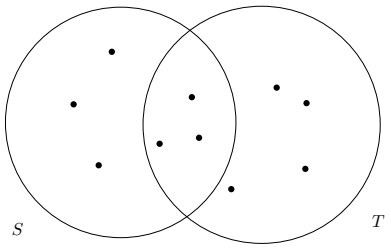


Figure 8.1: Jaccard similarity of two sets. The intersection $|S \cap T| = 3$ and the union $|S \cup T| = 10$. Therefore their Jaccard similarity is 3/10.

As a result, we redefine our problem statement as follows:

Problem 8.1.7 *Given a constant $0 \leq s \leq 1$ and a collection of sets $\mathcal{S}$, find the pairs of sets in $\mathcal{S}$ whose Jaccard similarity is greater than s.*

Refer to Example 8.1.3. Suppose we have two documents D and E, where $D =$ "The cow jumped over the moon" and $E =$ "The dog jumped over the moon", and let $k = 2$. The shingles corresponding to D and E are

$\{Thecow, cowjumped, jumpedover, overthe, themoon\}$ and

$\{Thedog, dogjumped, jumpedover, overthe, themoon\}$, respectively.

Their Jaccard similarity is 3/7. We will report that D and E are similar documents for any value of $s \leq 3/7$.

8.2 *Similarity-Preserving Summaries of Sets*

In this section, we will present a solution for the storage of sets of shingles using smaller representations called signatures. These *signatures* will also have the property that the similarity measure between any two will be approximately the same as the similarity between the two sets of shingles which they are derived from.

First, let us consider a natural representation of a set. Let U be the universe from which the elements of the set are drawn. Order the elements of U in some order. A set $S \subseteq U$ can be represented by a 0-1 vector of length $|U|$, where a 1 represents that the corresponding element from the universe is present in the set, and a 0 represents that the element is absent from the set. Similarly for a collection of sets over the universe U, we can associate a *characteristic matrix* M, where each column represents the vector corresponding to a set and each row corresponds to an element of U.

An example is given in Table 8.1, where we have four sets (representing households in a neighborhood), a universe U consisting of five elements (possible vacation destinations), and the characteristic matrix M. Observe that the set S_1 prefers cruise and safari, S_2 loves to visit resorts, S_3 loves Ski, Safari and prefers to stay at home, etc.

	S_1	S_2	S_3	S_4
Cruise	1	0	0	1
Ski	0	0	1	0
Resorts	0	1	0	1
Safari	1	0	1	1
Stay at Home	0	0	1	0

Table 8.1: A characteristic matrix M for 4 sets $\{S_1, S_2, S_3, S_4\}$. The universe U consists of 5 elements.

One effective way to compute the signature for a collection of sets is to use *minhashing*. For minhashing, the rows of the characteristic matrix are first randomly permuted. Let π be a permutation of rows. Then for each set (column in the characteristic matrix), its minhash value h is the index of the first row which is a 1 after applying the permutation π. In the previous example, suppose we permute the rows by applying the permutation $\pi : 01234 \rightarrow 40312$. The resulting table after permutation is shown as Table 8.2. Observe that the minhash values of the sets with respect to the permutation π are: $h(S_1) = 1$, $h(S_2) = 3$, $h(S_3) = 0$, and $h(S_4) = 1$. (Note that the rows are numbered 0,1,2,3,4).

	S_1	S_2	S_3	S_4
Ski	0	0	1	0
Safari	1	0	1	1
Stay at Home	0	0	1	0
Resorts	0	1	0	1
Cruise	1	0	0	1

Table 8.2: Characteristic matrix after the permutation $\pi : 01234 \rightarrow 40312$ of rows of Table 8.1.

In the following lemma we establish an important connection between the Jaccard similarity of two sets and the probability that their minhash values are the same after a random permutation of the rows of the characteristic matrix. We show that the Jaccard similarity is equal to the probability that these minhash values are the same.

Lemma 8.2.1 *For any two sets S_i and S_j in a collection of sets $\mathcal{S}$ where the elements are drawn from the universe U, the probability that the minhash value $h(S_i)$ equals $h(S_j)$ is equal to the Jaccard similarity of S_i and S_j, i.e., $Pr[h(S_i) = h(S_j)] = \mathsf{SIM}(S_i, S_j) = \frac{|S_i \cap S_j|}{|S_i \cup S_j|}$.*

Proof. Focus on the columns representing the sets S_i and S_j in the characteristic matrix before the random permutation. For any row,

the entries corresponding to these columns are either (a) both 0, (b) both 1, or (c) one is 0 and the other is 1. Let X be the number of rows of type (b) and let Y be the number of rows of type (c). Observe that the Jaccard similarity of S_i and S_j is $\mathsf{SIM}(S_i, S_j) = \frac{|X|}{|X|+|Y|}$.

Now, what is the probability that when we scan the rows from top to bottom, after the random permutation, we meet a type (b) row before a type (c) row? This is exactly $\frac{|X|}{|X|+|Y|}$, which is also precisely when $h(S_i) = h(S_j)$. ∎

Consider *minhash signature matrix* $\mathsf{SIG}(M)$. These are matrices constructed by repeatedly minhashing a characteristic matrix M of a set system $\mathcal{S}$ with universe U as follows. Pick a set of n random permutations of rows of M. For each set in $\mathcal{S}$ compute its h-value with respect to each of the n-permutations. This results in $\mathsf{SIG}(M)$ — it consists of $|\mathcal{S}|$ columns and n-rows, and the (i, j)-th entry corresponds to the signature of the j-th set with respect to the i-th permutation.

Example 8.2.2 *Shown in Table 8.3 is the minhash signature matrix* $\mathsf{SIG}(M)$ *created from the characteristic matrix of Table 8.1 using* $n = 2$ *permutations. The first permutation* π_1 *maps row* x *to* $x + 1 \bmod 5$ *and the second permutation* π_2 *maps the row* x *to* $3x + 1 \bmod 5$*. (Note that rows are numbered* $0, 1, 2, 3, 4$*.) Let us denote the minhash signatures corresponding to* π_1 *and* π_2 *by* h_1 *and* h_2*, respectively.*

	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

Table 8.3: Signature matrix $\mathsf{SIG}(M)$ for 4 sets corresponding to permutations π_1 and π_2 of characteristic matrix M in Table 8.1.

Next we discuss how to compute $\mathsf{SIG}(M)$ efficiently. Since the characteristic matrix M is typically very large, we cannot afford to permute its rows. In place of performing the permutations explicitly, we use several hash functions $h_1, \ldots, h_n$, where each $h_i : \{1, \ldots, U\} \to \{1, \ldots, U\}$, $1 \leq i \leq n$. The $i - th$ row of M is mapped to the row at index $h(i)$ after permutation. Note that h_i may not result in a valid permutation as two different rows of M may hash to the same index due to collisions. Nevertheless this is not a major issue as it avoids the need for explicitly permuting the rows of M. Shown below is an outline of the required steps to compute this signature matrix $\mathsf{SIG}(M)$. (If required, we can replace the hash functions by actual permutations, and this will not alter these steps.)

Step 1: Initialize each entry of the signature matrix $\mathsf{SIG}(M)$ to ∞.

Step 2: Pick n random hash functions $h_1, \ldots, h_n$, where $h_i : \{1, \ldots, U\} \to \{1, \ldots, U\}$.

Step 3: Execute the following steps for each row r of M.

1. Compute $h_1(r), h_2(r), \ldots, h_n(r)$.

2. For $c = 1, \ldots, |\mathcal{S}|$; if $M[r, c] = 1$ then for each $i = 1, \ldots, n$, $\mathsf{SIG}(i, c) := \min(h_i(r), \mathsf{SIG}(i, c))$.

Let us analyze the running time of the above algorithm. Step 1 requires $O(n|\mathcal{S}|)$ time. Step 2 requires time proportional to computing n hash functions. For Step 3, observe that for each non-zero entry of M, we compute n hash values, and this requires $O(n)$ time. So the total computation time is upper bounded by $O(n|\mathcal{S}||U|)$, or more precisely $O(|\mathcal{S}||U| + n|K|)$, where K is the total number of elements in all the sets. We do not need any additional memory except to store the description of n hash functions and the signature matrix $\mathsf{SIG}(M)$ and . Although the signature matrix is fairly small compared to the characteristic matrix, its size could still be large.

Just to have some perspective. Every year, Carleton admits approximately 5000 students. Each student typically takes 5 courses and each course consists of usually 4 assignments. So in all $5000 * 5 * 4 = 100,000$ assignments (i.e., documents) are generated by the students each term. If we want to find near similar documents by comparing every pair of documents, we need to evaluate $\binom{100,000}{2} = 10^{10}$ pairs of documents. If the comparison between a pair of documents requires 10^{-5} seconds, it will take $\sim$ 28-hours to find near similar documents. Instead, suppose we generate 125 signatures, each of size 4-bytes, for each of the documents. Then we have a signature matrix consisting of $125 * 100,000$ signatures of total size that equals to 50Mb. In the next section, we will introduce the Locality-Sensitive Hashing (LSH) technique and show how we can find near similar documents (without using pairwise direct comparisons) very efficiently using the signature matrix.

8.3 LSH for Minhash Signatures

First we replace each document by its shingles forming a well-defined set. From these sets and by the application of minhashing concept of the previous section, we construct the signature matrix. Following the notation of the previous section, let the signature matrix be $\mathsf{SIG}(M)$ for the set of documents $\mathcal{S}$. We partition the rows of this matrix into $b = n/r$ *bands*, where each band is made of r-consecutive rows. See Table 8.4 for an illustration. For simplicity, we assume that r divides n. For each band we define a hash function $h : \mathbb{R}^r \to \mathbb{Z}$, which takes a column vector of length r and maps it to an integer (i.e. a bucket). If we want we can even choose the same hash function for all the bands, but the buckets are kept distinct for each band. Now if two vectors of length r in any one of the bands hash to the same bucket, we declare that the corresponding sets

(documents) are potentially similar.

Band #	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}
	2	2	1	0	0	1	3	2	5	0	3
Band 1	1	3	2	0	2	2	1	4	2	1	2
	3	0	3	0	4	3	2	0	0	4	2
	0	4	3	1	5	3	3	2	3	5	4
Band 2	2	1	1	0	4	1	2	1	4	2	5
	4	2	1	0	5	2	3	2	3	5	4
	2	4	3	0	5	3	3	4	4	5	3
Band 3	0	2	4	1	3	4	3	2	2	2	4
	0	2	1	0	5	1	1	1	1	5	1
	0	5	1	0	2	1	3	2	1	5	4
Band 4	1	3	1	0	5	2	3	3	6	3	2
	0	5	2	1	5	1	2	2	6	5	4

Table 8.4: Partitioning of a signature matrix for $|\mathcal{S}| = 11$ sets, with $n = 12$ hash functions, into four bands ($b = 4$) of three rows each ($r = 3$). Note that (a) in Band 1, the sets $\{S_3, S_6\}$ are hashed into the same buckets, (b) in Band 3, $\{S_3, S_6, S_{11}\}$ are hashed into the same bucket, and also $\{S_8, S_9\}$ are hashed into the same bucket (possibly different from the bucket consisting of $\{S_3, S_6, S_{11}\}$), and (c) In Band 4, $\{S_2, S_{10}\}$ are hashed into the same bucket.

Lemma 8.3.1 *Let $s > 0$ be the Jaccard similarity of two sets. The probability that the minhash signature matrix agrees in all the rows of at least one of the bands for these two sets is $f(s) = 1 - (1 - s^r)^b$.*

Proof. The proof is straightforward and uses the following chain of simple arguments.

1. From Lemma 8.2.1, the probability that the minhash signatures for these two sets are the same in any particular row of the signature matrix is s.
2. The probability that the signatures agree in all the rows in one particular band is s^r. The probability is computed by taking AND of r independent events.
3. The probability that the signatures do not agree in at least one of the rows in this band is $1 - s^r$. This is the probability of the complementary event. Alternatively, one can think of this as the probability of an OR-event, i.e. the signatures do not agree in the first row OR the signatures do not agree in the second row OR ... OR the signatures do not agree in the last row.
4. The probability that the signatures do not agree in any of the b bands is $(1 - s^r)^b$.
5. Therefore, the probability that the signatures agree in at least one of the bands is $f(s) = 1 - (1 - s^r)^b$.

∎

Corollary 8.3.2 *In the above method, the probability that the two sets with Jaccard similarity $0 \leq s \leq 1$ are detected similar is $1 - (1 - s^r)^b$.*

In Table 8.5 we evaluate the probability function $f(s) = 1 - (1 - s^r)^b$ for different values of s, b, and r.

(b, r) $f(s) = 1 - (1 - s^r)^b \searrow$	$(4, 3)$	$(16, 4)$	$(20, 5)$	$(25, 5)$	$(100, 10)$
$s = 0.2$	0.0316	0.0252	0.0063	0.0079	0.0000
$s = 0.4$	0.2324	0.3396	0.1860	0.2268	0.0104
$s = 0.5$	0.4138	0.6439	0.4700	0.5478	0.0930
$s = 0.6$	0.6221	0.8914	0.8019	0.8678	0.4547
$s = 0.8$	0.9432	0.9997	0.9996	0.9999	0.9999
$s = 1.0$	1.0	1.0	1.0	1.0	1.0
$t = (\frac{1}{b})^{(\frac{1}{r})}$	0.6299	0.5	0.5492	0.5253	0.6309

Table 8.5: Values of the function $f(s) = 1 - (1 - s^r)^b$ for different values of s, b, and r.

The graphical representation of $f(s) = 1 - (1 - s^r)^b$ is in Figure 8.2. In the graph, x-axis represents values of s and y-axis represents the value of the probability function $f(s)$. As we can see the curve is S-shaped for different combinations of values of b and r. We observe that as $s \to 1$, the probability function $f(s) = 1 - (1 - s^r)^b \to 1$, i.e., the higher the Jaccard similarity between two sets, the probability that these two sets will map to the same bucket is high.

One important aspect of this curve is that the steepest slope occurs at the value of s which is approximately $t = (1/b)^{(1/r)}$ and this can derived as follows. To find the steepest slope, we need to compute for what values of s, $f''(s) = 0$. It turns out that $s = (\frac{r-1}{br-1})^{\frac{1}{r}}$ results in the steepest slope. For values of $br >> 1$, $s \approx (\frac{1}{b})^{\frac{1}{r}}$. In other words, if the Jaccard similarity s of the two sets is above the threshold $t = (\frac{1}{b})^{\frac{1}{r}}$, then the probability that they will be found potentially similar is very high. For example, the last row of Table 8.5 lists the thresholds. Consider the entries in the row corresponding to $s = 0.8$ and observe that most of the values for $f(s = 0.8) \to 1$ as $s > t$.

What this technique has done is to give us some idea in terms of which sets are very likely to be similar. If required, we can actually compare these potential pairs of sets (or their minhash signatures) to find out whether they are actually similar. Note that in this technique we need to choose appropriate values of parameters n, b, and r, given the value of the threshold t. Then any pairs of sets whose Jaccard similarity $s > t$ will likely be classified as similar sets. Increasing the value of n results in higher running time as we have to compute those many minhash signatures. Choosing smaller values of n results in less accurate results. See, for example, the results for $n = 12$ corresponding to the $b = 4, r = 3$ column in Table 8.5 in comparison to the results for $n = 125$ corresponding to the $b = 25, r = 5$ column or the results for $n = 1000$ corresponding to the $b = 100, r = 10$

column.

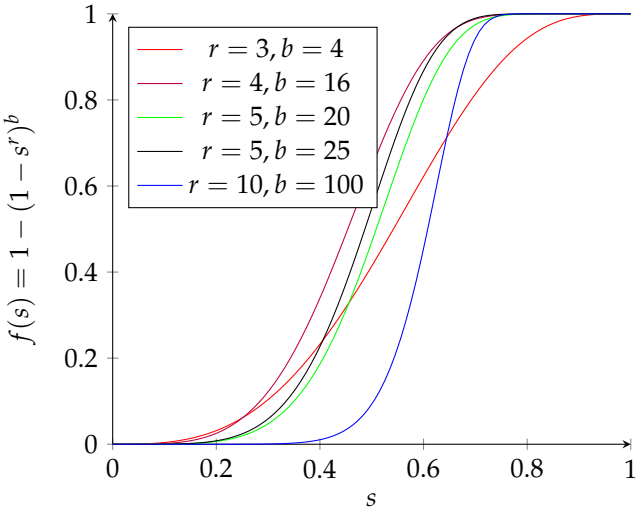


Figure 8.2: The S-curve $f(s) = 1 - (1 - s^r)^b$ for different values of b and r of Table 8.5.

At an abstract level what we have done here is to use a family of functions (the minhash functions) and the banding technique (with parameters b and r) to distinguish between pairs which are at a low distance (similar) to the pairs which are at a large distance (dissimilar). The steepness of the S curve suggests a threshold where this technique can be effective. In the next section, we will see that there are other families of functions that can be considered to separate the pairs which are at a low distance from the pairs which are at a higher distance.

8.4 Metric Space

Consider a finite set X. A *metric* or *distance measure* d on X is a function

$$d : X \times X \to [0, \infty)$$

satisfying the following properties. For all elements $u, v, w \in X$:

1. Non-negativity: $d(u, v) \geq 0$.
2. Symmetric: $d(u, v) = d(v, u)$.
3. Identity: $d(u, v) = 0$ if and only if $u = v$.
4. Triangle Inequality: $d(u, v) + d(v, w) \geq d(u, w)$.

Consider the following examples of metric spaces.

Example 8.4.1 (Euclidean Distance) *Let X be a set of n-points in plane. Euclidean distance between any two points $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$ of*

X is defined as $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. Observe that X with the Euclidean distance measure satisfies the metric properties.

Example 8.4.2 (Jaccard Distance) *Let $\mathcal{S}$ be a collection of sets. First observe that the Jaccard Similarity doesn't satisfy the metric properties as for any set $S \in \mathcal{S}$, the Jaccard similarity $\mathsf{SIM}(S, S) = 1$. This violates the identity property. Define the Jaccard distance between two sets $S_i, S_j \in \mathcal{S}$ as $\mathsf{JD}(S_i, S_j) = 1 - \mathsf{SIM}(S_i, S_j)$. Since $0 \leq \mathsf{SIM}(S_i, S_j) \leq 1$, $\mathsf{JD}(S_i, S_j) \geq 0$. Thus Jaccard distance satisfies the non-negativity property. Symmetry and Identity is obvious. To show the triangle inequality, it may be best to look at the relationship between the minhash signatures and Jaccard similarity. By Lemma 8.2.1 we know that $Pr[h(S_i) = h(S_j)] = \mathsf{SIM}(S_i, S_j)$. Since $\mathsf{JD}(S_i, S_j) = 1 - \mathsf{SIM}(S_i, S_j)$, thus $\mathsf{JD}(S_i, S_j) = Pr[h(S_i) \neq h(S_j)]$. Now consider three sets $S_i, S_j, S_k \in \mathcal{S}$. To show that $\mathsf{JD}(S_i, S_j) + \mathsf{JD}(S_j, S_k) \geq \mathsf{JD}(S_i, S_k)$, it is enough to show that $Pr[h(S_i) \neq h(S_j)] + Pr[h(S_j) \neq h(S_k)] \geq Pr[h(S_i) \neq h(S_k)]$. Observe that if $h(S_i) \neq h(S_k)$ then at least $h(S_i) \neq h(S_j)$ or $h(S_j) \neq h(S_k)$ must hold. If $h(S_i) = h(S_j)$ and $h(S_j) = h(S_k)$ then it will follow that $h(S_i) = h(S_k)$. Thus, the set $\mathcal{S}$ with the Jaccard distance measure satisfies the metric properties.*

Example 8.4.3 *Let X be a set of n elements, where the distances between any pair of elements $x, y \in X$ are defined as follows.*

$$d(x, y) = \begin{cases} 1, & \text{if } x \neq y \\ 0, & \text{otherwise} \end{cases}$$

Observe that X with distance function d satisfies the metric properties.

Example 8.4.4 (Hamming Distance) *Consider the space X of d-dimensional Boolean vectors. Consider two vectors $x, y \in X$. The* Hamming distance *$\mathsf{HAM}(x, y)$ is defined as the number of coordinates in which x and y differ. For example, let $x = 110011$ and $y = 100111$. Then $\mathsf{HAM}(x, y) = 2$, as they differ in exactly two coordinates. Observe that the Hamming distance is non-negative, symmetric, and that the distance between two identical vectors is 0. The Hamming distance between any three vectors x, y, and z also satisfies the triangle inequality $\mathsf{HAM}(x, y) + \mathsf{HAM}(y, z) \geq \mathsf{HAM}(x, z)$ since the number of components in which x differs from z cannot be larger than the sum of the number of components in which x differs from y and y differs than z. Therefore, we can use Hamming distance as a metric over the d-dimensional vectors.*

8.5 Theory of Locality Sensitive Functions

In this section we will consider a family of functions $\mathcal{F}$. The families are typically comprised of hash functions. We say that a hash function $f \in \mathcal{F}$ identifies two items x and y to be similar if f hashes them

to the same bucket. We use the notation $f(x) = f(y)$ to denote that x and y are hashed to the same bucket. For example, the minhash functions seen previously form a family of functions. Recall the definition of a distance measure from Section 8.4. Next we define a notion of *sensitive family* that encapsulates the idea that if two items are close to each other with respect to the distance measure, then the probability that they hash to the same bucket by any function $f \in \mathcal{F}$ will be high. Conversely, if the two items are far from each other, then the probability that they hash to the same bucket will be low.

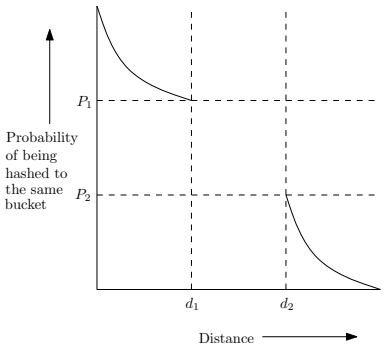


Figure 8.3: A smaller distance between items corresponds to a higher probability of similarity.

Definition 8.5.1 *Let d be a distance measure and let $d_1 < d_2$ be two distances in this measure. Let $0 \leq p_2 < p_1 \leq 1$. A family of functions $\mathcal{F}$ is said to be (d_1, d_2, p_1, p_2)-sensitive if for every $f \in \mathcal{F}$ the following two conditions hold;*

1. *If $d(x, y) \leq d_1$ then $Pr[f(x) = f(y)] \geq p_1$.*
2. *If $d(x, y) \geq d_2$ then $Pr[f(x) = f(y)] \leq p_2$.*

See Figure 8.3 for an illustration.

Example 8.5.2 *Consider the Jaccard distance measure for finding similar sets in a collection of sets $\mathcal{S}$. Let $0 \leq d_1 < d_2 \leq 1$. The family of minhash-signatures is $(d_1, d_2, p_1 = 1 - d_1, p_2 = 1 - d_2)$-sensitive and this can be argued as follows. Suppose that the Jaccard similarity between two sets is at least s. Then their Jaccard distance is at most $d_1 = 1 - s$. By Lemma 8.2.1 the probability that they will be hashed to the same bucket by minhash signatures is $\geq p_1 = 1 - d_1$. Similarly, suppose that the Jaccard similarity is at most s'. Then their Jaccard distance is at least $d_2 = 1 - s'$. The probability that the minhash signatures map them to the same bucket is at most $p_2 = 1 - d_2$.*

Next we look into amplifying sensitive families using AND and OR constructions. Suppose we have a (d_1, d_2, p_1, p_2)-sensitive family $\mathcal{F}$. We can construct a new family $\mathcal{G}$ by an *AND-construction* as follows. Each function $g \in \mathcal{G}$ is formed from a set of r independently chosen functions of $\mathcal{F}$, say $f_1, f_2, \ldots, f_r$ for some fixed value of r. Now, $g(x) = g(y)$ if and only if for all $i = 1, \ldots, r$, $f_i(x) = f_i(y)$.

Claim 8.5.3 *$\mathcal{G}$ is an (d_1, d_2, p_1^r, p_2^r)-sensitive AND family.*

Proof. Each function f_i is chosen independently. For any $0 \leq p \leq 1$, if p is the probability that any $f_i \in \mathcal{F}$ will hash two items u and v to the same bucket, then the probability that any function $g \in \mathcal{G}$ will hash u and v to the same bucket is p^r. This is the probability of all the r independent events to occur simultaneously. ∎

Similarly we can construct $\mathcal{G}$ by an *OR-construction*. Each member g in $\mathcal{G}$ is constructed by taking b independently chosen members $f_1, f_2, \ldots, f_b$ from $\mathcal{F}$. We say that $g(x) = g(y)$ if and only if $f_i(x) = f_i(y)$ for at least one of the members in $\{f_1, f_2, \ldots, f_b\}$.

Claim 8.5.4 *$\mathcal{G}$ is an $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^b)$-sensitive OR family.*

Proof. Each function f_i is chosen independently, and this is the probability of at least one of the b-events to occur. We can compute this by first finding the probability that none of the b events occur. Let p be the probability that any function $f_i \in \mathcal{F}$ hashes two items u and v to the same bucket. Then $1-p$ is probability that f_i does not hash them to the same bucket, and $(1-p)^b$ is the probability that none of the functions $\{f_1, f_2, \ldots, f_b\}$ hash them to the same bucket. Thus, $1-(1-p)^b$ is the probability that at least one of the functions $\{f_1, f_2, \ldots, f_b\}$ hash them to the same bucket. ∎

Observe that the AND-construction reflects rows within a band in Table 8.4. Moreover, the OR-construction reflects the combination of various bands in that table. Furthermore, AND construction lowers all the probabilities, and by choosing the family $\mathcal{F}$ and the parameter r, we can try to push $p_2^r \to 0$. The OR-construction increases all the probabilities, and by choosing $\mathcal{F}$ and b appropriately we can try to push $1-(1-p_1)^b \to 1$. This is the essence of the idea of amplification. Next we look into some concrete examples.

Let us play with some values of b and r to see the effect of AND and OR-constructions. We first construct an AND-family $\mathcal{F}_1$ for a certain value of r, and then construct an OR-family $\mathcal{F}_2$ for a certain value of b. Next we construct an AND-OR family $\mathcal{F}_3$ by first constructing an AND family followed by an OR family. Similarly we construct the family $\mathcal{F}_4$ by first constructing an OR-family followed by an AND-family. Let us look at the amplification of the probabilities in Table 8.6 for different values of p.

	$\mathcal{F}_1$ (AND)	$\mathcal{F}_2$ (OR)	$\mathcal{F}_3$ (AND-OR)	$\mathcal{F}_4$ (OR-AND)
p	p^r	$1-(1-p)^b$	$1-(1-p^r)^b$	$(1-(1-p)^r)^b$
0.2	0.0001	0.6723	0.0079	0.0717
0.4	0.0256	0.9222	0.1216	0.4995
0.6	0.1296	0.9897	0.5004	0.8783
0.7	0.2401	0.9975	0.7446	0.9601
0.8	0.4096	0.9996	0.9282	0.9920
0.9	0.6561	0.9999	0.9951	0.9995

Table 8.6: Illustration of four families obtained for different values of p. $\mathcal{F}_1$ is the AND family for $r = 4$. $\mathcal{F}_2$ is OR family for $b = 5$. $\mathcal{F}_3$ is the AND-OR family for $r = 4$ and $b = 5$. $\mathcal{F}_4$ is the OR-AND family for $r = 4$ and $b = 5$.

Let us try to understand the columns of Table 8.6. Let $\mathcal{F}$ be a $(0.2, 0.6, 0.8, 0.4)$-sensitive minhash function family. This means if

the distance between two sets S_i and S_j is ≤ 0.2, any function in $\mathcal{F}$ will hash them to the same bucket with probability ≥ 0.8. Similarly if the distance between them is ≥ 0.6, with probability at most 0.4 they will hash to the same bucket. Let us focus our attention on rows corresponding to $p_2 = 0.4$ and $p_1 = 0.8$ in Table 8.6. For the AND-family $\mathcal{F}_1$ we see that $p^r = p^4$ is substantially lower than p, but still $p_2^4 \to 0$ and p_1^4 is away from 0. For the OR-family $\mathcal{F}_2$ for $b = 5, 1 - (1-p)^5 \geq p$, but still the value corresponding to p_1 tends towards 1 and the value corresponding to p_2 is away from 1. More interesting are the last two columns. Notice that the AND-OR family $\mathcal{F}_3$ corresponds to the LSH for minhash signature family of Section 8.3. Here we first apply a r-way AND-construction and then a b-way OR-construction. The AND-construction converts any probability p to p^r. This, when followed by a b-way OR-construction further converts the probability to $1 - (1-p^r)^b$. As we can see from Table 8.6 the value corresponding to p_1 tends towards 1 and the value corresponding to p_2 is closer to 0. Notice that the value of the function $1 - (1-p^r)^b$ with respect to the values of p forms an S-curve. Its fixed-point $p = 1 - (1-p^4)^5$ is $p \approx 0.6672$ and its threshold value $t = (\frac{1}{b})^{(\frac{1}{r})} \approx 0.6687$.[1] This implies that for values of p significantly less than 0.6672, AND-OR family amplifies it towards 0. Similarly, values of p larger than 0.6672 are amplified to 1. Notice that this technique amplifies the probabilities in the right direction (away from threshold and fix-point), provided we can apply the function several times (e.g. 20 times in our example with $r = 4$ and $b = 5$). The OR-AND family $\mathcal{F}_4$ does not provide anything interesting. In this construction first we have applied a r-way OR-construction followed by a b-way AND-construction.

[1] Thanks to WolframAlpha

8.6 LSH Families

In the previous section we saw LSH families for the Jaccard distance measure. In this section we will construct LSH-families for various other distance measures.

8.6.1 LSH family for Hamming Distance

Consider two d-dimensional Boolean vectors x and y. Recall from Example 8.4.4 that the Hamming distance $\mathsf{HAM}(x, y)$ is defined as the number of coordinates in which x and y differ. We construct a locality-sensitive family for the d-dimensional Boolean vectors as follows. Let $f_i(x)$ denote the i-th coordinate of x. For two vectors x and y, the probability that $f_i(x) = f_i(y)$ for a randomly chosen coordinate i will equal the number of coordinate agreements out of

the total number of coordinates. Since the vectors x and y disagree in $\mathsf{HAM}(x,y)$ positions out of d positions, then they agree in $d - \mathsf{HAM}(x,y)$ positions. Hence $Pr[f_i(x) = f_i(y)] = 1 - \frac{\mathsf{HAM}(x,y)}{d}$.

Claim 8.6.1 *For any $d_1 < d_2$, $\mathcal{F} = \{f_1, f_2, \ldots, f_d\}$ is a $(d_1, d_2, 1 - d_1/d, 1 - d_2/d)$-sensitive family of hash functions.*

Proof. Recall Definition 8.5.1. Let $p_1 = 1 - d_1/d$ and $p_2 = 1 - d_2/d$. A family of functions $\mathcal{F}$ is said to be (d_1, d_2, p_1, p_2)-sensitive if for every $f_i \in \mathcal{F}$ the following two conditions hold:

1. If $\mathsf{HAM}(x,y) \leq d_1$ then $Pr[f_i(x) = f_i(y)] \geq p_1$
2. If $\mathsf{HAM}(x,y) \geq d_2$ then $Pr[f_i(x) = f_i(y)] \leq p_2$

∎

8.6.2 LSH family for similarity of vectors using dot products

Consider two $d-$*dimensional* vectors $x = (x_1, x_2, \ldots, x_d)$ and $y = (y_1, y_2, \ldots, y_d)$ with tails at the origin o. Their dot product is defined as $x \cdot y = \sum_{i=1}^{d} x_i y_i$. Note that the dot product is positive if the angle between the vectors is between 0 and $\pi/2$ and is negative if the angle is between $\pi/2$ and π. The vectors x and y define a plane, say P. Consider the intersection of any d-dimensional hyperplane H (different than P) passing through o. The intersection between H and P defines a line h passing through o. The vectors x and y may or may not be on the same side of h. Let v be the normal vector to H and passing through o in the plane P. To determine whether x and y are on the same side of h, we can compute the dot products $v \cdot x$ and $v \cdot y$. If the dot product has the same same sign, then h does not separate x from y. Whereas, if they have different sign, then h separates them, as the angle between v and one of x or y will be less than $\pi/2$, and with the other one it will be more than $\pi/2$.

For example, in Figure 8.4, the hyperplane H_1 intersects the plane containing vectors x and y in line h_1 passing through o. Note that the dot products $v_1 \cdot x$ and $v_1 \cdot y$ have different signs, where v_1 is the normal to H_1 passing through o. Furthermore, H_2 intersects the plane containing vectors x and y in the line h_2, and $v_2 \cdot x$ and $v_2 \cdot y$ have the same sign, where v_2 is the normal to H_2 passing through o.

Let us choose a random hyperplane H passing through o. We want to estimate the probability that it will separate the vectors x and y, i.e. what is the probability that x will be one side of H and y is on the other side? Let the angle between x and y be θ. It is easy to see that for H to separate x and y, the line h (and the corresponding normal vector v) have to be in a particular sector of angle θ at o. If h

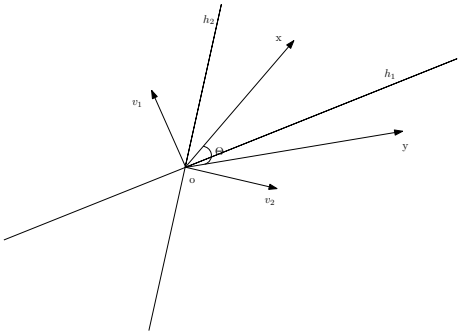


Figure 8.4: Two vectors x and y shown in a plane. The intersection of hyperplanes H_1 and H_2 with the plane containing x and y forms lines h_1 and h_2. Vectors v_1 and v_2 are normal to H_1 and H_2, respectively, and are in the plane containing x and y. Observe that $v_1 \cdot x > 0$, $v_1 \cdot y < 0$, $v_2 \cdot x > 0$, $v_2 \cdot y > 0$.

falls within this sector, then H separates x from y. Or equivalently we say that the dot products $v \cdot x$ and $v \cdot y$ have different signs, which we write as $f_v(x) \neq f_v(y)$, where $f_v(x)$ represents the sign of the dot product $v \cdot x$. Therefore, $Pr[f_v(x) \neq f_v(y)] = \theta/\pi$ and $Pr[f_v(x) = f_v(y)] = 1 - \theta/\pi$.

We construct a sensitive family of functions $\mathcal{F}$ as follows. We select a set of random $d - dimensional$ vectors v anchored at origin. These vectors constitute the set $\mathcal{F}$. Given any d-dimensional vector x, for each vector $v \in \mathcal{F}$, we compute the sign of the dot product $v \cdot x$, and store it as $f_v(x)$. From the above arguments, it is easy to see that if the angle between two vectors x and y is at most $d_1 = \theta_1$, then the probability that for any $v \in \mathcal{F}$, $Pr[f_v(x) = f_v(y)] \geq 1 - d_1/\pi$. Similarly if the angle is at least $d_2 = \theta_2$, then $Pr[f_v(x) = f_v(y)] \leq 1 - d_2/\pi$. Thus, our resulting family $\mathcal{F}$ is a $(d_1, d_2, 1 - d_1/\pi, 1 - d_2/\pi)$-sensitive family of functions.

8.6.3 LSH family for Near Neighbors in 2-dimensions

Consider a set of points P in a 2-dimensional space. We are interested in finding pairs of points which are within certain distance to each other, say Δ. Each hash function f in the family $\mathcal{F}$ will be represented by a line l with random orientation in this space. We partition l into intervals of equal size 2Δ, and orthogonally project all points of P on l. For a point $x \in P$, let $f_l(x)$ denote the interval in which x lies after the projection. If two points $x, y \in P$ lie in the same interval after projection, then $f_l(x) = f_l(y)$. Next, we show that if the distance between x and y is at most Δ, then with probability at least $1/2$, $f_l(x) = f_l(y)$, i.e. if $d(x, y) \leq \Delta$ then $Pr[f_l(x) = f_l(y)] \geq 1/2$. Moreover, if $d(x, y) > 4\Delta$, then $Pr[f_l(x) = f_l(y)] \leq 1/3$. Using this method, if two points are close to each other then there are high chances that they will project to the same interval. Conversely, if the points are far away, it is unlikely they will project to the same interval.

Without loss of generality, we assume that the line l is horizontal. Let x and y be two points in the plane. Now we show that if $d(x, y) \leq \Delta$, then $Pr[f_l(x) = f_l(y)] \geq 1/2$. Let m be the mid-point of the interval $f_l(x)$. With probability $1/2$, the projection of x lies to the left of m in $f_l(x)$. Furthermore, with probability $1/2$, the projection of y lies to the right of projection of x. In this case, since $d(x, y) \leq \Delta$, projection of y lies in $f_l(x)$ (i.e., $f_l(x) = f_l(y)$). Thus with probability $1/4$, the projections of x and y lie in $f_l(x)$ where the projection of x is to the left of m and the projection of y is to the right of the projection of x. Similarly, with probability $1/4$, projections of x and y lie in $f_l(x)$ where the projection of x is to the right of m and the projection of y is to the left of projection of x. Since the above two cases are mutually

exclusive, if $d(x,y) \leq \Delta$, then $Pr[f_l(x) = f_l(y)] \geq 1/2$.

Next we show that if $d(x,y) > 4\Delta$, then $Pr[f_l(x) = f_l(y)] \leq 1/3$. As before let l be horizontal and let θ be the angle of the line passing through x and y with respect to l. For the projections of x and y to fall in the same interval, we will need that $d(x,y)\cos\theta \leq 2\Delta$. For this to happen $\cos\theta \leq 1/2$, or the angle the line xy forms with the horizontal needs to be between 60° and 90°. Observe that there is at most 1/3rd chance that the angle between the horizontal and xy is in that range. See Figure 8.5 for an illustration.

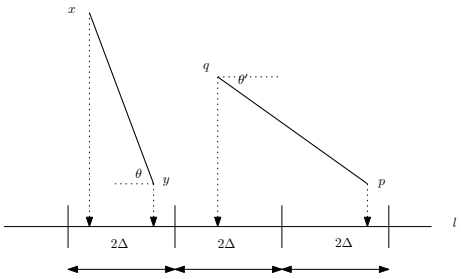


Figure 8.5: Points x and y project to the same interval on l, whereas p and q project to different intervals.

Hence, the family with respect to the projection on a random line with intervals of size 2Δ is a $(\Delta, 4\Delta, 1/2, 1/3)$-sensitive family. Again, we can amplify these probabilities by taking combination of ANDs and ORs as in Section 8.5.

8.6.4 *LSH family for answering near-neighbor queries*

Suppose we are given a set of n points P in D-dimensional Euclidean space $\mathbb{R}^D$. For a query point $q \in \mathbb{R}^D$, if there is a point $p \in P$ such that $d(q,p) = R$, then any point $x \in P$ that is within a distance of $(1+\epsilon)R$ from q, for some constant $\epsilon > 0$, is called a (R,ϵ)-*near neighbor* (denoted as (R,ϵ)-NN) of q. (Note that all the distances are Euclidean distance in this subsection.) If p happens to be the closest point of q in P, an (R,ϵ)-NN is usually referred to as an *approximate nearest neighbor*. We first sketch a construction of a family $\mathcal{F}$ of hash functions f to answer (R,ϵ)-NN queries, where for all $x \in P$ and given value of R, the following holds (with high probability):

- if x is a (R,ϵ)-NN of q, $f(q) = f(x)$.
- if x is not a (R,ϵ)-NN of q, $f(q) \neq f(x)$.

We will utilize the data structure for answering (R,ϵ)-NN queries to answer the approximate nearest neighbor queries as follows. Let $d_{\min}$ and $d_{\max}$ be the smallest and the largest inter-point distances in P, respectively. We will construct data structures for (R,ϵ)-NN queries for values of $R = d_{\min}/\epsilon, d_{\min}, (1+\epsilon)d_{\min}, (1+\epsilon)^2 d_{\min}, \ldots, d_{\max}$. For the query point q, we will perform binary search in the data structures for various values of R to find the smallest value of R for which we obtain an element $x \in P$ such that x is a (R,ϵ)-NN of q (i.e., $f(q) = f(x)$). The binary search incurs an additional cost of $O(\log \frac{D_{\max}}{D_{\min}})$.

Now we turn our attention to answering (R,ϵ)-NN queries. Pick a window size $w > 0$, where w is real number. We will hash points in P to buckets by the following procedure.

1. For $a = 1, \ldots, b$ do

   (a) Pick $r = O(\log n)$ random vectors $v_1, v_2, \ldots, v_r$ in $\mathbb{R}^D$. The

components v_{ij}, $j = 1, \ldots, D$ for the vectors v_i, $i = 1, \ldots, r$, are chosen independently from the standard normal distribution $N(0, 1)$.

(b) For each v_i, choose an offset $o_i \in (0, w]$ uniformly at random.

(c) Each point $x \in P$ is mapped to the bucket $f_\alpha(x) = (\lceil \frac{\langle v_1, x\rangle + o_1}{w} \rceil, \ldots, \lceil \frac{\langle v_r, x\rangle + o_r}{w} \rceil)$. (Note that $\langle v_i, x\rangle$ represents the dot-product of v_i and x.)

To answer the (R, ϵ)-NN query for point $q \in \mathbb{R}^D$, we execute the following steps:

1. Compute $f_1(q), f_2(q), \ldots, f_b(q)$ to identify the buckets in which point q falls.
2. Form a set $\mathsf{Candidate}(q)$ by taking union of points of P in buckets $f_1(q), f_2(q), \ldots, f_b(q)$.
3. For up to $2b$ elements of $\mathsf{Candidate}(q)$, if there exists an element $x \in \mathsf{Candidate}(q)$ such that $||x - q||_2 \leq (1 + \epsilon)R$, report x as (R, ϵ)-NN of q. Otherwise report NIL.

Before we proceed further, let us look at the vectors v_i's chosen in the above procedure. Each component of these vectors are independent identical random variable from the distribution $N(0, 1)$. Let $X_1, \ldots, X_r$ be i.i.d. random variables from $N(0, 1)$. We are interested in understanding the distribution of the linear combination of $X_1, \ldots, X_r$.

Claim 8.6.2 *Let $X = (X_1, \ldots, X_r)$, where each random variable X_i, $1 \leq i \leq r$, is from the standard normal distribution $N(0, 1)$. Let $u = (u_1, \ldots, u_r)$, where each u_i is a real number. The dot product $\langle u, X\rangle$ has the distribution $N(0, ||u||_2^2)$.*

Proof. $\mathbb{E}[\langle u, X\rangle] = \mathbb{E}[\sum_{i=1}^r (u_i X_i)] = \sum_i u_i \mathbb{E}[X_i] = 0$ by linearity of expectation. Note that for any random variable Y, its variance $Var(Y) = \mathbb{E}[Y^2] - \mathbb{E}[Y]^2$. Thus, $Var[\langle u, X\rangle] = \mathbb{E}[(\langle u, X\rangle)^2]$. Now $\mathbb{E}[(\langle u, X\rangle)^2] = \mathbb{E}[(u_1 X_1 + \cdots + u_r X_r)(u_1 X_1 + \cdots + u_r X_r)] = \sum_{i \neq j} u_i u_j \mathbb{E}[X_i X_j] + \sum_i u_i^2 \mathbb{E}[X_i^2] = \sum_{i \neq j} u_i u_j \mathbb{E}[X_i]\mathbb{E}[X_j] + \sum_i u_i^2 \mathbb{E}[X_i^2] = 0 + \sum_i u_i^2 = ||u||_2^2$, since (a) for $i \neq j$, $\mathbb{E}[X_i X_j] = \mathbb{E}[X_i]\mathbb{E}[X_j]$ as X_i and X_j are independent, and (b) $\mathbb{E}[X_i^2] = 1$, since $Var[X_i] = E[X_i^2] - E[X_i]^2 = 1$ as X_i has $N(0, 1)$ distribution. ∎

Let v be one of the vectors with offset o computed by the procedure where each of its component is i.i.d. random variable from the standard normal distribution $N(0, 1)$. For a point $x \in P$ and a query point $q \in \mathbb{R}^D$, we are interested in finding the probability that both x and q will hash to the same bucket, i.e $Pr(\lfloor \frac{\langle v, q\rangle + o}{w} \rfloor = \lfloor \frac{\langle v, x\rangle + o}{w} \rfloor)$. Observe that x and q will hash into the same bucket of width w if

(1) $|\langle v, q\rangle - \langle v, x\rangle| \leq w$, and
(2) the 'divider' does not fall between $\langle v, q\rangle$ and $\langle v, x\rangle$.

Note that $|\langle v, q\rangle - \langle v, x\rangle| \leq w$ is equivalent to $|\langle v, q - x\rangle| \leq w$. Let us apply Claim 8.6.2 to $\langle v, q - x\rangle$, where v and $q - x$ play the roles of X and u, respectively. By the claim we know that $\langle v, q - x\rangle$ has the distribution of $N(0, ||q - x||_2^2)$. Alternatively, we can say that $\langle v, q - x\rangle$ is a random variable cZ, where $c = ||q - x||_2$ and Z has $\phi(z) = N(0, 1)$ distribution. Thus the condition $|\langle v, q - x\rangle| \leq w$ can be written as $|cZ| \leq w$. The probability that the divider falls between $\langle v, q\rangle$ and $\langle v, x\rangle$ is $\frac{|\langle v, q-x\rangle|}{w}$. Thus, the probability that the divider does not fall between $\langle v, q\rangle$ and $\langle v, x\rangle$ is $1 - \frac{|\langle v, q-x\rangle|}{w}$.

Now the probability of collision

$$Pr(c) = Pr(\lfloor \frac{\langle v, q\rangle + o}{w} \rfloor = \lfloor \frac{\langle v, x\rangle + o}{w} \rfloor) = \int_{z=0}^{\leq \frac{w}{c}} \phi(z)(1 - \frac{cz}{w})dz. \tag{8.1}$$

Substituting $t = cz$, we obtain

$$Pr(c) = \int_{t=0}^{w} \frac{1}{c}\phi\Big(\frac{t}{c}\Big)\Big(1 - \frac{t}{w}\Big)dt. \tag{8.2}$$

Also it can be reasoned that the function $Pr(c)$ is monotonically decreasing in $c = ||x - q||_2$, i.e. the probability of collision decreases as the distance between x and q increases.

To answer the (R, ϵ)-NN query, we consider two critical values of c, namely $c = R$ and $c = (1 + \epsilon)R$, in Equation 8.2 and let the corresponding probabilities be $p_1 = Pr(c = R)$ and $p_2 = Pr((c = 1 + \epsilon)R)$. We know that $p_1 > p_2$ from the monotonicity. We claim the following:

Claim 8.6.3 *For a query point $q \in \mathbb{R}^D$, let $p^* \in P$ be a point such that $||q - p^*|| \leq R$. With respect to the above procedure:*

1. *For some $\alpha \in \{1, \ldots, b\}$, with probability $\geq 1/2$, $f_\alpha(p^*) = f_\alpha(q)$.*
2. *With probability $> 1/2$ the total number of elements $x \in P$ such that $f_\alpha(q) = f_\alpha(x)$ and $||x - q||_2 > (1 + \epsilon)R$ is at most $2b$.*

Proof. We prove the first statement. From Equation 8.2, we know that probability of collision between p^* and q is at least p_1. Therefore, for a fixed α, $Pr[f_\alpha(p^*) = f_\alpha(q)] \geq p_1^r$. By setting $r = \log_{\frac{1}{p_2}} n$, we have

$$p_1^r = p_1^{\left(\log_{\frac{1}{p_2}} n\right)} = n^{-\left(\frac{\log \frac{1}{p_1}}{\log \frac{1}{p_2}}\right)} = n^{-\rho},$$

where $\rho = \frac{\log \frac{1}{p_1}}{\log \frac{1}{p_2}}$. From the banding technique, we know that the probability that the collision occurs in at least one of the b bands is

$1-(1-n^{-\rho})^b$. We set $b = n^\rho$. Then this probability can be expressed as

$$1-(1-n^{-\rho})^b = 1-(1-n^{-\rho})^{n^\rho} \geq 1-\frac{1}{e} > \frac{1}{2}.$$

Next we prove the second part. Let $p \in P$ such that $||p-q|| > (1+\epsilon)R$. We know from Equation 8.2, the probability of collision for a fixed $\alpha \in \{1,\ldots,b\}$ is $\leq p_2^r = p_2^{\left(\log_{\frac{1}{p_2}} n\right)} = \frac{1}{n}$. Since the set P has n elements, the expected number of collisions per band is at most 1, and the total number of the collisions is at most b. Thus the probability that the number of collisions exceeds more than $2b$ is at most $1/2$ from Markov's inequality. ∎

Claim 8.6.4 *The queries can be answered in $O(n^\rho D \log n) = o(nD)$ time.*

Proof. To evaluate a query $q \in \mathbb{R}^D$, we need to compute $b = n^\rho$ hash functions $f(q)$. Each of them requires a computation of $r = O(\log n)$ quantities of the form $\lceil \frac{\langle v_i, q\rangle + o_i}{w} \rceil$. The dot product requires $O(D)$ time. Thus the total time required to hash q is $O(n^\rho D \log n)$. Once we compute all the buckets in which q lies, we need to evaluate the distance between q and at most $2b$ elements of P. This requires a total of $O(n^\rho D)$ time. Thus the queries can be answered in $O(n^\rho D \log n)$ time. Since $p_1 > p_2$, $\rho = \frac{\log \frac{1}{p_1}}{\log \frac{1}{p_2}} < 1$. Hence the queries require $o(nD)$ time. ∎ Furthermore, notice that the dimension D is not in the exponent, as in the Voronoi diagram based methods for finding nearest neighbors.

Claim 8.6.5 *The procedure requires $O(n^{1+\rho} + Dn^\rho \log n)$ space to store the data structures.*

Proof. We need to store the non-empty buckets, i.e. partition of points in P, for each of the $b = n^\rho$ bands. Also, we need to store the parameters for each of the hash functions f. For each band, this includes $O(r)$ vectors v's and offsets o_i's. In all this requires $O(bn + rbD) = O(n^{1+\rho} + Dn^\rho \log n)$ memory space. ∎

The material of this subsection is adapted from the notes of L. Cayton.

8.6.5 Fingerprint Matching

Fingerprint matching typically requires comparison of several features of the fingerprint pattern. These include ridge lines which form arches, loops, or circular patterns, along with minutia points and patterns which form ridges, and bifurcations (see Figure 8.6). Typically

each fingerprint is mapped to a normalized grid that takes care of the size and orientation of the fingerprint. After the normalization, it is expected that for two fingerprints of the same finger, if a grid cell of one fingerprint contains a minutia, the corresponding grid cell of the other fingerprint, with high probability, will contain that minutia. Therefore, we can abstract a fingerprint to be a set of grid points, where matching any two fingerprints amounts to matching elements in the corresponding grid cells.

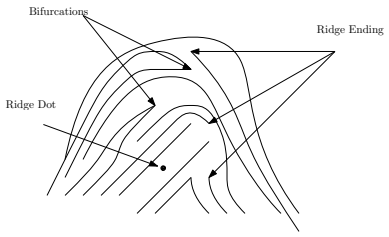


Figure 8.6: Miniutia in fingerprints.

Assume that the probability of finding a minutia in a random grid cell of any given fingerprint is 0.2. Also, assuming that we have two fingerprints of the same finger, let the probability of a minutia appearing in the same grid cell of both fingerprints given that one of them does have a minutia there be 0.85. We will define a locality-sensitive family of functions $\mathcal{F}$ as follows. Each function $f \in \mathcal{F}$ sends two fingerprints to the same bucket if they have minutia in each of the three specific grid cells. Let us estimate the chance of ending up with two matching fingerprints. First, the probability that two arbitrary fingerprints will map to the same bucket with respect to the function f is $0.2^6 = 0.000064$. Assuming that we have two fingerprints from the same finger, f maps them to the same bucket with a probability of $0.2^3 \times 0.85^3 = 0.0049$. Note that the probability that the three particular cells of the first fingerprint contains minutia is 0.2^3, and given that the two fingerprints are from the same same finger, the other fingerprint will have minutia in the same cells with probability 0.85^3. Now we can use the OR-sensitive families to amplify these probabilities.

As an example, suppose we use 1000-way OR-functions. Then two fingerprints from different fingers will map to the same bucket with a probability of $1 - (1 - 0.000064)^{1000} \approx 0.061$. Similarly, two fingerprints from the same finger will map to the same bucket with a probability of $1 - (1 - 0.0049)^{1000} \approx 0.992$.

For another example, let us now use 2000 OR-functions that are partitioned in two groups of 1000 functions each. Assume also that we have constructed buckets for each group. Given a query fingerprint, we will find the fingerprints which are potential matches using the above scheme in each group independently. We select only those fingerprints which are potential matches in both the groups. This produces a set of fingerprints that we will actually compare against the query fingerprint. Note that in this scheme actual comparisons of fingerprints occur with only a few fingerprints (those in the intersection). Hence, the probability that we will detect matching fingerprints is $(0.992)^2 \approx 0.984$. The probability of false positives (the non-matches which we will detect after making the comparison with the query fingerprint) is $0.061^2 \approx 0.00371$, which is insignificant.

Notice that in this scheme we have been able to avoid comparing the query fingerprint with all the fingerprints and with very high probability we will find matching fingerprints.

8.7 Bibliographic Notes

A method, based on shingling, for finding similar files in a large file system was proposed [110]. The k-shingles of a document and minhashing technique are formally introduced in [31, 30]. These concepts were developed as a result of the authors' work on the AltaVista web index algorithm which was used for detecting similar documents. Minwise-independent families also underpin the theory behind minhashing, as the resemblance of document-sets is shown to be equal to the probability that the min-wise permutation of two sets using random permutation functions are equal.

Locality-sensitive hashing technique was introduced in [83] where the approximate nearest-neighbor search problem was first reduced to the problem of point location in equal balls. Along with this, the distance measures for Hamming distance and the resemblance measure given in [31] were used as schemes. It is worth noting that the approximate search is deemed accurate enough for practical purposes, where the actual closest neighbor can still be found by checking all approximate near-neighbors [8]. A scheme based on p-stable distributions ($p \in (0, 2]$) under the L_p norm for locality-sensitive hashing was introduced in [46]. Further improvements are given in [8].

Locality-sensitive hashing is used in for video identification and retrieval. In this case, feature vectors are usually constructed from video frames using certain color histograms. In [90], the authors address two weaknesses of using LSH for this purpose. Responding to a non-uniform distribution of points in the space, they focus on using a hierarchical hash table to produce a more uniform distribution. In addition to this, they also attempt to partition dimensions more carefully in order to produce a more even hashing. A new scheme for video retrieval is then proposed in [82], where a color histogram is used which better handles the adverse effects of brightness & color variations. This is used in conjunction with an additional uniform distance shrinking projection which is applied to the produced feature vectors.

Locality-sensitive hashing is also used in image search. Similar to applications in video (for a single frame), images are often processed using color histograms to produce feature vectors which can be compared. This method was used in [67], with the histograms compared using the L_1 norm. Following this, techniques using χ^2 distance [70],

p-stable distributions [86], and kernelized locality-sensitive hashing [104] have been proposed. Kernelized locality-sensitive hashing has also been used as a basis for search on text images in [120].

LSH has recently been proposed for use in a wide range of other areas. One of these is the creation of hash values in P2P networks [47]. This would allow for a conceptual search, as data with similar ontologies would be located near each other. Here, data is defined by its extracted concept vectors, which are then hashed into buckets based on the cosine distance measure. Another, in [85], kernelized LSH is applied to an utterance model in order to identify speakers. In this case the Hamming distance metric is used. Other areas include use for species diversity estimation by allowing ease of grouping similar DNA sequences [129], and incremental clustering for affinity groups in Hadoop in order to store related data closer together [88]. [144] has proposed a new scheme based on entropy for LSH. The argument is that an improvement can be made for [46] such that the distribution of mapped values will be approximately uniform. [2]

[2] Parts of this section are contributed by Andrew Wylie.

8.8 Exercises

8.1 *Show that Euclidean metric satisfies triangle inequality.*

8.2 *Given two documents* $D_1 = \{$*"His brow trembled"*$\}$ *and* $D_2 = \{$*"The brown emblem"*$\}$*, compute all k-shingles for each document with* $k = 4$*.*

8.3 *Compute the Jaccard Similarity of the two sets of k-shingles for* D_1 *and* D_2*.*

8.4 *Compute the signature matrix (minhash signature) of a given permutation of characteristic matrix in Table 8.7, using* $n = 3$ *different hash functions.*

Element	S_1	S_2	S_3	S_4
a	1	0	1	1
b	1	0	1	0
c	0	1	0	1
d	1	0	1	0
e	0	0	1	0

Table 8.7: A characteristic matrix.

8.5 *Explain the reasoning behind the proof of Lemma 8.2.1.*

8.6 *Show that the minhash function family is* $(d_1, d_2, 1 - d_1, 1 - d_2)$*-sensitive.*

8.7 *Calculate hamming distance for the vectors* $v = [5,1,3,2,4]$ *and* $u = [1,1,2,2,4]$.

8.8 *Show that the AND amplification construction is* (d_1, d_2, p_1^r, p_2^r)*-sensitive.*

8.9 *When applying amplification constructions to a locality-sensitive family of functions, which order of composition is 'better', and why? Explain when you would want to use different orders of construction.*

8.10 *Show that the Jaccard Distance which is defined as* $1-$ *the Jaccard Similarity between the two sets is a metric.*

8.11 *In Section 8.6.3 we considered the problem of computing LSH families for near neighbors in 2-dimensions. This exercise extends that solution to 3-dimensions. Let* P *be a set of points in 3-dimensions. Consider a line* l *with a random orientation that is partitioned in intervals of size* 2Δ*. Project points in* P *orthogonally on* l*, and for a point* $x \in P$*, let* $f_l(x)$ *be the interval it projects on* l*. For any pair of points* $x, y \in P$*, show the following.*

1. *If* $d(x,y) \leq \Delta$, $Pr[f_l(x) = f_l(y)] \geq 1/2$.
2. *If* $d(x,y) \geq 4\Delta$, $Pr[f_l(x) = f_l(y)] < 1/2$.

8.12 *Show that the function* $1-(1-s^r)^b$ *is non-decreasing with respect to* s *for fixed values of* r *and* b.

8.13 *Assume that we have a set of points* P *with the distance function* $d(\cdot,\cdot)$ *between pairs of points of* P *that satisfies the metric property. Let* a *be a positive real number and let* $0 < s_\beta < s_\alpha < 1$*. Let* $\mathcal{F}$ *be a family of functions such that for all* $f_i \in \mathcal{F}$ *and any pair of points* $x, y \in P$ *the following holds:*

1. *If* $d(x,y) < a/2$, $Pr[f_i(x) = f_i(y)] \geq s_\alpha$.
2. *If* $d(x,y) > 2a$, $Pr[f_i(x) = f_i(y)] < s_\beta$.

Show that one can always choose positive integers b *and* r *such that the AND-OR family constructed from* $\mathcal{F}$ *with parameters* r *and* b *satisfies*

1. *If* $d(x,y) < a/2$, $1-(1-s_\alpha^r)^b \to 1$.
2. *If* $d(x,y) > 2a$, $1-(1-s_\beta^r)^b \to 0$.

Note that we are given that $0 < s_\beta < s_\alpha < 1$*, but we do not know the actual values of* s_α *and* s_β*. For example, can one always find* r *and* b *such that if* $d(x,y) < a/2$, $1-(1-s_\alpha^r)^b \geq 0.95$ *irrespective of knowing the actual value of* s_α*? Similarly, can one always find* r *and* b *such that if* $d(x,y) > 2a$, $1-(1-s_\beta^r)^b < 0.05$.

8.14 *This exercise is inspired by Section 3.2 of* [3] *and it relates to the origins of the LSH scheme. Consider the problem of finding near neighbors. First we establish some notation. Let* (X, d) *be a metric space where* X *is a set of points from a finite dimensional metric space and let* $d(\cdot, \cdot)$ *denote the distance function satisfying the metric properties between pairs of elements of* X*. Given a set* $P \subseteq X$*, we are asked to preprocess* P *such that for any query point* $q \in X$*, report a* c-approximate nearest neighbor *from* P *to* q *which is defined as follows. Let* $p \in P$ *be the point closest to* q *with respect to the distance measure* d*. Then any point of* P *which is within a distance of* $cd(q, p)$ *from* q *is called an* c*-approximate nearest neighbor, where* $c > 1$ *is a constant. Let* $B(x, \alpha)$ *denote the set of points of* X *within the distance* α *from* x*. Let* $0 < \alpha$ *and* $1 > p_1 > p_2 > 0$ *be real numbers. Suppose we have a family of hash functions* $\mathcal{F} = \{h : X \to U\}$ *that is* $(\alpha, c\alpha, p_1, p_2)$*-sensitive for* (X, d) *satisfying the following. For any* $p, q \in X$*,*

1. *If* $d(p, q) \leq \alpha$ *then* $Pr[h(x) = h(y)] \geq p_1$*.*
2. *If* $d(p, q) > c\alpha$ *then* $Pr[h(x) = h(y)] \leq p_2$*.*

Suppose we amplify the gap between 'high' probability p_1 *and 'low' probability* p_2 *by applying the banding technique where* $b = |P|^{\rho} / p_1$ *and* $r = \lceil \log_{\frac{1}{p_2}} |P| \rceil$*, where* $\rho = \log_{\frac{1}{p_1}} / \log_{\frac{1}{p_2}}$*. Using this structure, we hash each point* $p \in P$ *to the appropriate bucket for each band. Given a query point* $q \in X$*, we also hash* q *to the appropriate bucket for each band. Whenever* q *has collisions with elements of* P *already present in those buckets, we append them to a list say* L*, but terminate this process if the size of* L *reaches* $3b$*. If there exists an element* $p \in L$ *such that* $d(p, q) \leq c\alpha$ *then we report* p*, otherwise output NIL.*

8.15 *Suppose the probability of finding the minutia in a random grid cell of a random fingerprint is* 25%*. Assume that if we have two fingerprints from the same finger, given that one of them has a minutia in a grid cell, than the probability that the other one also has minutia in the same grid cell is* 90%*. Suppose you define locality sensitive hash functions with respect to four (random) grid cells. Any of the hash functions* h *declares the two fingerprints* f *and* f' *to be potentially similar, if each of the four cells (corresponding to* h*) in* f *and* f' *consists of minutia. Suppose our sensitive family of functions is composed of* 1000 *such functions. For finding similar fingerprints, we use an OR-sensitive family, i.e. if any of these* 1000 *functions identifies the two fingerprints to be same, then we report them to be similar. Estimate what is the probability of false positives and false negatives? (You don't have to evaluate the actual values - you can just leave them as expressions.)*

8.16 *Assume that input is an array* A *of* n *elements (say integers). Suppose you want to pre-process this array to answer the following queries:*

[3] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(1):321–350, 2012

1. Membership(x, A): *For a query element x, report if $x \in A$.*

2. Nearest(x, A): *Report the element of A that minimizes $|a_i - x|$, for $i = 1, \ldots, n$.*

For each of these queries, what will be the time required for preprocessing A and what will be the time of query, if you are interested to know (a) Exact Answer (b) Approximate Answer. (You need to define what approximate means to you.)

8.17 *Show that the Jaccard distance measure between sets satisfies the metric properties.*

8.18 *For the locality-sensitive hashing technique with respect to signatures of sets, we partitioned the signature matrix in b bands, each band consisting of r rows, and analyzed that the probability that the two sets with Jaccard similarity of s, will be reported similar with probability $f(s) = 1 - (1 - s^r)^b$, using the so called AND-OR construction. This analysis was based on estimating the probability that signatures for the two sets should match in all rows (constituting the AND-family) of at least one of the bands (the OR-family). Suppose, we alter our strategy, and use OR-AND construction. To be more precise, we have the same partitioning in terms of b bands and r rows, but now we say that the signatures match in a band, if they match in at least one of the rows in that band, but we declare the two sets to be similar if their signatures match in all the bands. Estimate what will be the probability that the two sets are reported similar whose Jaccard similarity is s using the OR-AND strategy. Call this estimate $f'(s)$. Furthermore, compare the two estimates, $f(s)$ and $f'(s)$, for various values of s, (you may fix $b = 20$ and $r = 5$ or to any other values).*

8.19 *Continuing the thread of the last question, and let us again consider AND-OR construction. Let the bands be $B_1, \ldots, B_b$. But now within each band we further apply the banding technique. To be more precise, for any band B_i, $i = 1, \ldots, b$, we further subdivide its rows in b' sub-bands, each sub-band consists of r' rows, where $r = b'r'$. Within each band B_i, we perform the AND-OR construction for its b' sub-bands. Derive an expression, say $f''(s)$, for the probability that two sets will be reported similar with Jaccard similarity of s. Evaluate $f''(s), f'(s), f(s)$ for a few different values of s, r, b, r', b' to reason whether is there any point in applying the banding technique within each band B_i.*

8.20 *Suppose, we have a case of fake currency bills of \$50 that are circulating in the market. The association of* ⟨FraudBusters⟩ *employs a cumbersome method that is computationally inefficient. It is based on scanning and digitizing the bill. Let us assume that the digital copy has a resolution of 2048×1536 pixels (approximately 3 million Pixels). To check whether the*

Bill is real, this organization compares each pixel of the digital copy with the corresponding pixel of a digitized copy of a real bill. It is given that if the bill is original, then the probability that any pixel will match with the corresponding pixel of the real bill is 95%, *but if the bill isn't original then the probability of the match decreases to* 60%. *Given this information, can you devise a LSH based scheme to determine most of the fake bills?*

8.21 *Let us assume that we have a large collection B of binary vectors in dimension $d = 10,000$. We are asked to compute a data structure so that the following queries can be answered efficiently. Given any query binary vector q in dimension d, we are interested to report all the binary vectors in B that are approximately* 95% *similar to q. We say that two vectors $a = a_1a_2 \ldots a_d$ and $b = b_1b_2 \ldots b_d$ are* 95% *similar if $a_i = b_i$ for at least* 95% *of indices i, $1 \leq i \leq d$. Design an algorithm that computes such a data structure and show how each query can be answered efficiently. The time to answer the query q should not exceed $O((k+1)d)$, where k is the number of vectors in B that are at least* 95% *similar to q. It is fine if you have some false positives and negatives, but their percentage shouldn't be large.*

8.22 *You want to use the locality-sensitive hashing (LSH) technique to identify similar sets from a collection $\mathcal{S}$ of sets with the following objectives. Let $X, Y \in \mathcal{S}$ and let their Jaccard similarity be $s = \frac{|X \cap Y|}{|X \cup Y|}$.*

$Pr(X$ and Y are reported similar if $s \geq 0.9) \geq 0.99$, and

$Pr(X$ and Y are reported similar if $s \leq 0.5) \leq 0.2$

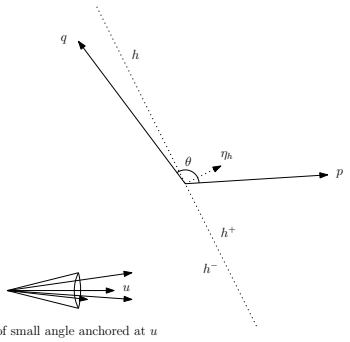
Can you achieve these objectives by using minHash signature matrix M with at most 100 *signatures, i.e., the number of rows in M is $\leq$ 100? Justify.*

8.23 *This problem is about classifying vectors that are oriented in a similar direction. Assume that we have a collection V of d-dimensional vectors from the origin and assume that d is large. If we take any two distinct vectors $p, q \in V$, there is a unique plane Π_{pq} passing through the origin that contains p and q. (You may want to visualize this first for the vectors in three dimensions.) Suppose $p = (p_1, p_2, \ldots, p_d)$ and $q = (q_1, q_2, \ldots, q_d)$. The angle θ between p and q is related to their dot-product by the expression $cos(\theta) = \frac{p \cdot q}{|p||q|}$, where $|p|$ and $|q|$ is the length of vectors p and q. For example, if $p = (1,3)$ and $q = (-3,4)$, $|p| = \sqrt{1^2 + 3^2} = \sqrt{10}$ and $|q| = \sqrt{3^2 + 4^2} = 5$ and $cos(\theta) = \frac{p \cdot q}{|p||q|} = \frac{1 \cdot -3 + 3 \cdot 4}{5\sqrt{10}} = \frac{9}{5\sqrt{10}}$, or $\theta \approx 55^\circ$.*

Choose a random vector η_h at origin and consider the (hyper)plane h (i.e. plane in 3-dimensions, or a line in 2-dimensions) normal to η_h passing through origin. Any hyperplane h partitions the space into two half spaces, usually referred to as h^+ and h^-. Assume that none of the vectors p and q are incident on h. The vectors p and q may lie in the same half-space with respect to h or one resides in h^+ and the other in h^-. It should also be clear that if the angle between p and q is small, then there is more chance that p

and q will reside in the same half-space of h, as compared to when the angle is large. Observe that if the dot products $p \cdot \eta_h$ and $q \cdot \eta_h$ have the same sign, then p and q are within the same half-space of h. Otherwise, they are in different half-spaces.

Suppose we want to identify vectors that are close together, i.e., they point approximately in the same direction. (For example, if we take a cone of small-angle centred in the direction of vector u with the apex at origin, we want to find all the vectors that are within this cone. All these vectors are pointing in (approximately) the same direction as u. See the Figure.) For a random vector η_h and its corresponding normal hyperplane h, we assign a signature $h(u) \in \{+, -\}$ to each vector $u \in V$ based on the sign of $u \cdot \eta_h$. For any two vectors $u, v \in V$, we say $h(u) = h(v)$ if sign of $u \cdot \eta_h$ is the same as the sign of $v \cdot \eta_h$. Show that $Pr(h(u) = h(v)) = 1 - \frac{\theta}{\pi}$, where $\cos(\theta) = \frac{u \cdot v}{|u||v|}$. Observe that if the angle between u and v is small, there is high chance that $h(u) = h(v)$. We can construct a family of hash functions by choosing several random hyperplanes $h_1, \ldots, h_n$, and use the banding technique to amplify the probability of classifying the similar vectors.

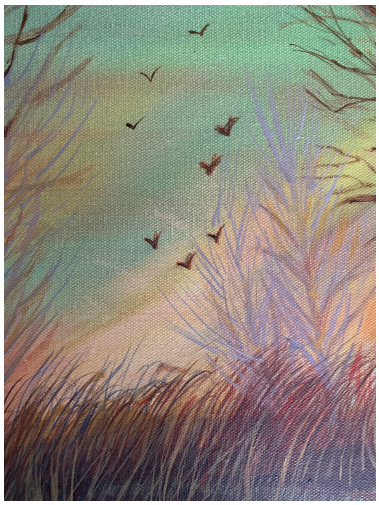


Cone of small angle anchored at u

9
Data Streams

We will focus on

1. Heavy Hitters: Finding a majority element (if it exists) in an array using $O(1)$ memory in linear time.
2. Count Min Sketch: Finding frequent elements in a data stream.
3. Bloom Filters: Detecting membership in a set.
4. Flajolet-Martin algorithm for estimating distinct elements in a stream.
5. Counting in sliding windows.



9.1 Heavy Hitters

Let A be an array consisting of n elements (positive integers) and suppose A consists of a majority element, i.e. an element that occurs $> n/2$ times. We want to locate the majority element by performing a linear scan of A.

Notice that in the above algorithm, each non-majority element can 'cancel' at most one majority element. Since the number of majority elements are at least $\lfloor \frac{n}{2} \rfloor + 1$, one copy of the majority element will be stored in the *current*. Observe that the above algorithm runs in $O(n)$ time and uses only two variables, c and *current*, to maintain the majority element. In the i-th step, the element at location $A[i]$ is compared against the *current* and the value of the counter is used to update the status of the current. We summarize our discussion in the following claim.

Claim 9.1.1 *Let A be a data stream of size n that contains a majority*

Algorithm 9.1: Finding the majority element in A
Input: Array A of size n consisting a majority element
Output: The majority element

```
1  c ← 0
2  for i = 1 to n do
3  |  if c = 0 then
4  |  |  current ← A[i]
5  |  |  c ← c + 1
6  |  end
7  |  else
8  |  |  if A[i] = current then
9  |  |  |  c ← c + 1
10 |  |  end
11 |  |  else
12 |  |  |  c ← c − 1
13 |  |  end
14 |  end
15 end
16 return current
```

element. This element can be reported in $O(n)$ time using $O(1)$ memory space.

Next we turn to the following variant. Given a parameter $0 < k < n$, report all the elements in A that occur at least n/k times. As can be seen, this has applications in finding what are the most frequent items bought in a store, what are the most frequent search queries, etc. Here, to gain efficiency in terms of computation time and the required memory space, in place of reporting all the elements that occur at least n/k times, we will report all the elements that occur at least $n/k - \epsilon n$ times for some constant $0 < \epsilon < 1$. For example, if $\epsilon = 1/3k$, then we report all the elements that occur at least $0.667\frac{n}{k}$ times instead of the elements that occur at least n/k times. We use a *count min sketch* (**CMS**), a two dimensional table with r rows and b columns, that keeps track of approximate counts for each element in A. It utilizes r hash functions $h_1, h_2, \ldots, h_r$, where each h_i maps natural numbers to one of the possible b buckets, i.e., $h_i : \mathbb{N} \rightarrow \{1, \ldots, b\}$. We will assume that h_i's map any natural number to any of the possible b buckets uniformly at random. We compute the CMS table as follows:

If we assume that each hash value $h_j(A[i])$ can be computed in $O(1)$ time then it is easy to see that the CMS table can be computed in $O(nr)$ time. Next let us see how we can use this table to estimate

Algorithm 9.2: Computation of Count Min Sketch
Input: An array A consisting of n natural numbers and r hash functions $h_1, \ldots, h_r$, where $h_i : \mathbb{N} \to \{1, \ldots, b\}$
Output: $CMS[\cdot, \cdot]$ table consisting of r rows and b columns

```
for i = 1 to r do
    for j = 1 to b do
        CMS[i, j] ← 0
    end
end
for i = 1 to n do
    for j = 1 to r do
        CMS[j, h_j(A[i])] ← CMS[j, h_j(A[i])] + 1
    end
end
return CMS[·, ·]
```

the frequency of an element x in A. Let f_x^* be the true frequency of x in A. Let $f_x = \min\{CMS[1, h_1(x)], \ldots, CMS[r, h_r(x)]\}$. We claim the following.

Claim 9.1.2 *Let $b = \frac{2}{\epsilon}$. Then $Pr[|f_x - f_x^*| \geq \epsilon n] \leq \frac{1}{2^r}$.*

Proof. First observe that for any $1 \leq j \leq r$, $CMS[j, h_j(x)] \geq f_x^*$ as each copy of x increments the counter $CMS[j, h_j(x)]$. Moreover, due to the collisions in the hash function, it is possible that h_j may 'hash' other elements to the same location where it hashes x (and thus increments the count). Note that since each value is hashed uniformly at random, we expect about n/b elements of A to be hashed in the same bucket as that of x. Why?

Let us establish an indicator random variable I_y corresponding to each value $y \in A$ as follows:

$$I_y = \begin{cases} 1 & \text{if } h_j(y) = h_j(x) \\ 0, & \text{otherwise} \end{cases}$$

Let V represents the set of all distinct values in A. Note that $x \in V$. For any element $v \in V$, let f_v^* represents the true frequency of the value v in A. We have

$$CMS[j, h_j(x)] = f_x^* + \sum_{\substack{y \in V \\ y \neq x}} I_y * f_y^* \tag{9.1}$$

Since we have assumed that the hash function maps any value to any of the buckets uniformly at random, $Pr(I_y = 1) = 1/b$ and its

expected value $E[I_y] = 1/b$. Thus,

$$E[CMS[j, h_j(x)]] = f_x^* + \sum_{\substack{y \in V \\ y \neq x}} f_y^*/b \leq f_x^* + n/b \tag{9.2}$$

By setting $b = \frac{2}{\epsilon}$, we obtain

$$E[CMS[j, h_j(x)]] \leq f_x^* + n/b = f_x^* + \epsilon n/2 \tag{9.3}$$

If we define a random variable X_j that equals the difference of the value of the counter in $CMS[j, h_j(x)]$ and f_x^*, then it is easy to see that $E[X_j] \leq n/b = \epsilon n/2$. Thus, by Markov's inequality, $Pr(X_j > 2(\epsilon n/2)) \leq 1/2$. This also holds for each value of $j = 1, \ldots, r$. Furthermore, X_j is independent of X_k as the corresponding hash functions h_j and h_k are independent for any $k \neq j$ and $1 \leq k, j \leq r$. Therefore, we have that for $f_x = \min\{CMS[1, h_1(x)], \ldots, CMS[r, h_r(x)]\}$, the probability $Pr[|f_x - f_x^*| \geq \epsilon n] \leq \frac{1}{2^r}$. ∎

Markov's inequality states that the probability that a random variable deviates from its expectation by a factor of c is at most $1/c$.

Now let us see how to report all the elements in A that occur frequently. Let us choose $\epsilon = 1/3k$. Thus $b = 2/\epsilon = 6k$. The size of the CMS table will be $b \times r = 6kr$. We will scan the array A, starting at the location 1, and update the CMS table by the above procedure. In addition, we maintain a set of $O(k)$ items that occur most frequently among all the elements in A scanned so far. Note that these are the heavy hitters! The items are stored in a heap data structure and their key is their frequency. Assume that we have so far scanned $i - 1$ items and have updated the CMS table and the heap accordingly, and now consider the i-th item. First we update the counts in the CMS table by executing $CMS[j, h_j(A[i])] \leftarrow CMS[j, h_j(A[i])] + 1$, for $j = 1$ to r. Let $x = A[i]$. If the count returned for x is $\geq i/k$, we need to perform the following heap operations: If x is present in the heap, we delete x and re-insert it again with the updated count value. If x was not present in the heap, then we insert it in the heap, but remove all the elements (Extract-min) whose count is less than i/k.

Next we do some complexity analysis. First assume that the count value in the CMS table for each element is exact, i.e. equals the frequency in A. Then, observe that the heap contains all the heavy hitters, i.e. all the items that occur at least n/k times. The size of the heap is at most k, and for each element we perform $O(r)$ computation for updating the CMS table, and at most one insertion and one deletion in the heap, which costs at most $O(\log k)$ time.

Since our counts are not accurate, let us try to figure out what actually is in the heap. Clearly all the elements whose frequency is at least n/k are in the heap since the count in CMS table is at least the frequency. But for some elements, the count in the CMS table is an overestimate of their frequency. By how much? We know by the

above claim, for any element the probability that the count exceeds the frequency by ϵn is small ($\leq 1/2^r$). Thus, all the elements in the heap will have frequency at least $\frac{n}{k} - \epsilon n = \frac{n}{k} - \frac{n}{3k} = 0.667\frac{n}{k}$. Thus with a heap of size $O(k)$, we can report all the elements in A that occur at least $0.667\frac{n}{k}$ times. The total memory usage is $O(6kr)$ and the probability of error is $\leq 1/2^r$. If we choose $r = c \log_2 n$, then this translates to $O(k \log n)$ space and the probability of error $\leq 1/2^r = 1/n^c$.

9.2 *Bloom Filters*

Suppose we have a set S consisting of a large number of elements from a universe U, and we want to repeatedly query S to know whether any query element $x \in U$ is a member of S. Bloom filters are a simple (randomized) data structures that can answer these queries extremely fast, though may incur false positives. For example, the set S may represent non-spam e-mail addresses, and we may want to know whether a new mail received is from a non-spam e-mail address to not to qualify it as a Junk mail. It is fairly important that the membership can be decided quickly, though it may be possible to wrongly classify some of the junk mails as regular mails.

The Bloom filter B is a binary array of size m. We can think of B as a bit vector of length m. Initially all bits of B are initialized to 0. In addition we have k hash functions, say $h_1, h_2, \ldots, h_k$, that map each element of the universe to one of the locations in $[1, \ldots, m]$. For each element $x \in S$, we set $B[h_1(x)] = B[h_2(x)] = \cdots = B[h_k(x)] = 1$. For testing the membership of an element y of the universe in B, we check whether each of the locations $B[h_i(y)] = 1$ for $1 \leq i \leq k$. It is easy to see that each element in S can set the bits in the filter in $O(k)$ time, and the membership test takes $O(k)$ time. Moreover, if $y \in S$, our structure always returns the correct result, but if $y \notin S$, it may wrongly classify (false-positive) that $y \in S$ due to the collisions in the hashing. The main challenge in the design of the Bloom filters is to decrease the false positive rate.

Let us evaluate the probability that a particular bit of B is set to 1. Let us focus on the i-th bit and we want to evaluate the probability that $Pr(B[i] = 1)$. Let $n = |S|$. Then any of the kn hash's can set $B[i]$ to 1. What is the probability that none of these hash values happen to be i? This is exactly $(\frac{m-1}{m})^{kn} = (1 - \frac{1}{m})^{kn}$ as each value has a choice of $m - 1$ other locations out of m. Thus, $p = Pr(B[i] = 1) = 1 - (1 - \frac{1}{m})^{kn}$. Now for a false-positive to occur, all of the k locations (may not be distinct) corresponding to the hash values $h_1(y), \ldots, h_k(y)$ must be 1. One may guess that the probability of false-positives to occur is p^k, but as it turns out that the analysis is not that simple. First we

answer, by an example, why p^k is not the right value.

Example 9.2.1 *Consider the scenario when $n = 1, k = 2$, and $m = 2$. Let $S = \{x\}$ and the two hash functions be h_1 and h_2. We will see that the false-positive rate is $10/16$, whereas $p^k = p^2 = 9/16$. Observe that $h_1(x)$ and $h_2(x)$ can set either one or both of the locations of B to 1. We have the three cases:*

Case A: Only the first location is set to 1. This happens with probability $1/4$.

Case B: Only the second location is set to 1. This happens with probability $1/4$.

Case C: Both the locations are set to 1. This happens with probability $1/2$.

Now on querying for an element y, where $y \neq x$, let us see what is the probability of the false-positive.

In Case A, the probability is the product of (a) the probability of false-positive given that we are in Case A ($= 1/4$), and (b) the probability of being in Case A ($= 1/4$). This is given by $1/4 * 1/4 = 1/16$.

By symmetry, for Case B, the probability of false-positive is $1/4 * 1/4 = 1/16$.

In Case C the probability of false-positive is $1/2 * 1 = 1/2$.

Thus, the false-positive rate is $1/16 + 1/16 + 1/2 = 10/16$, whereas $p = 1 - (1 - \frac{1}{m})^{kn} = 1 - (1 - \frac{1}{2})^2 = \frac{3}{4}$ and $p^2 = 9/16$.

It turns out that the derivation of the actual expression for the false positive rate is technical. We state the result, without derivation, and the interested reader can consult the research article.

Theorem 9.2.2 [1] *Let $p_{k,n,m}$ be the false-positive rate for a Bloom filter that stores n elements of a set S in a bit-vector of size m using k hash functions.*

1. *We can express $p_{k,n,m}$ in terms of the Stirling number of second kind as follows:*

$$p_{k,n,m} = \frac{1}{m^{k(n+1)}} \sum_{i=1}^{m} i^k i! \binom{m}{i} \left\{ {kn \atop i} \right\}$$

2. *Let $p = 1 - (1 - 1/m)^{kn}$, $k \geq 2$ and $\frac{k}{p}\sqrt{\frac{\ln m - 2k \ln p}{m}} \leq c$ for some $c < 1$. Upper and lower bound on $p_{k,n,m}$ are given by*

$$p^k < p_{k,n,m} \leq p^k \left(1 + O\left(\frac{k}{p}\sqrt{\frac{\ln m - 2k \ln p}{m}}\right)\right)$$

In the theorem we use Stirling number of second kind. It is the number of ways to partition a set of a objects into b non-empty subsets and is denoted by $\left\{ {a \atop b} \right\}$.

[1] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel H. M. Smid, and Yihui Tang. On the false-positive rate of bloom filters. *Inf. Process. Lett.*, 108(4):210–213, 2008

9.3 Flajolet-Martin Algorithm

In this section we will discuss the algorithm of Flajolet and Martin's [2] for estimating the frequency moments in a data stream. We adapt

[2] Philippe Flajolet and G. Nigel Martin. Probabilistic counting. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 76–82. IEEE Computer Society, 1983

the description and notation from the paper by Alon, Matias and Szegedy [3].

Let $A = (a_1, \ldots, a_m)$ be a sequence (data stream) of m-elements, where each $a_i \in N = \{1, \ldots, n\}$. For simplicity we assume that $n = 2^d$, and hence $d = \log n$-bits are sufficient to represent any number in N. We define frequency moments using the quantity m_i that represents the numbers of i's in A, i.e. $m_i = |\{j | a_j = i\}|$. We define k-th *frequency moments* F_k, $k \geq 0$, by

$$F_k = \sum_{i=1}^{n} m_i^k$$

Note that F_0 equals the number of distinct elements in A (assuming $0^0 = 0$), F_1 equals the number of elements in A, and F_2 represents the surprise number. Low surprise number refers to an even distribution of data in the stream. It is straightforward to see that if we can store all the elements in A in memory then we can compute any of the frequency moments F_k's exactly as follows: Sort A and compute m_i's for each $i \in \{1, \ldots, n\}$, and evaluate $F_k = \sum_{i=1}^{n} m_i^k$. Next, we will show the following theorem due to Flajolet and Martin's [59].

Theorem 9.3.1 *We can design an algorithm that uses only $O(\log n)$ bits of memory space and computes an approximation $\widehat{F_0}$ of F_0 such that $\frac{1}{c} \leq \frac{\widehat{F_0}}{F_0} \leq c$, with probability at least $1 - \frac{2}{c}$, for $c > 2$.*

Before we describe the algorithm, we assume that we have a perfect hash function $h : N \to N$ that maps any $x \in N$ uniformly at random to any element of N. In the algorithm, for each element $x \in A$, we will be interested in the location of the rightmost 1 in the binary representation of $h(x)$. Note that the least-significant bit is at location 1 and the most significant bit is at location d. Here is the algorithm: It is obvious that the above algorithm only requires $O(\log n)$ bits of memory to store the value of R. Moreover, the running time is $O(m)$, assuming that we can evaluate $h(\cdot)$ and the rightmost 1 in the binary representation of $h(\cdot)$ in $O(1)$ time. Next, we show that $\frac{1}{c} \leq \frac{\widehat{F_0}}{F_0} \leq c$ with probability at least $1 - \frac{2}{c}$, for $c > 2$. We make the following observations.

Observation 9.3.2 *Consider $h(a_i)$ for some $a_i \in A$. The probability that the location of the rightmost 1 in the binary representation of $h(a_i)$ is at least r, where $r \in \{1, \ldots, d\}$, equals $\frac{1}{2^r}$.*

Proof. If $h(a_i)$ maps to any number such that its $r-1$ least significant bits are all 0s, then the location of the rightmost 1 in $h(a_i) \geq r$. By assumption, $h(a_i)$ maps a_i uniformly at random to any of the 2^d

[3] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999 This paper won the Gödel Prize.

Algorithm 9.3: Compute $\widehat{F_0}$, an approximation to the number of distinct elements in A

Input: Array A of size m where each $a_i \in N = \{1, \ldots, n\}$
Output: $\widehat{F_0}$

```
R ← 0
for i = 1 to m do
    Compute binary representation of h(a_i)
    Let r be the location of the rightmost 1 in h(a_i)
    if r > R then
        R ← r
    end
end
return F̂_0 ← 2^R
```

values. Thus the probability that the binary representation of $h(a_i)$ has $r-1$ least significant bits that are all 0s is $\frac{2^{d-r}}{2^d} = \frac{1}{2^r}$. ∎

Define an indicator random variable I_x^r for each element $x \in A$ as follows:

$$I_x^r = \begin{cases} 1, & \text{if rightmost 1 in } h(x) \text{ is at location} \geq r \\ 0, & \text{otherwise.} \end{cases}$$

Define A' to be a set containing exactly one copy (with no duplicates) of all the elements in A. Note that $|A'| = F_0$. Define

$$Z^r = \sum_{x \in A'} I_x^r.$$

Observation 9.3.3 *Following are the expectations and variances of the random variables I_x^r and Z^r.*

1. $E[I_x^r] = \frac{1}{2^r}$
2. $Var[I_x^r] = \frac{1}{2^r}(1 - \frac{1}{2^r})$
3. $E[Z^r] = \frac{F_0}{2^r}$
4. $V[Z^r] < E[Z^r]$

Proof.

1. Note that $E[I_x^r] = 1 \cdot Pr(I_x^r = 1) + 0 \cdot Pr(I_x^r = 0) = \frac{1}{2^r}$.
2. $Var[I_x^r] = E[(I_x^r)^2] - E[I_x^r]^2 = \frac{1}{2^r} - (\frac{1}{2^r})^2 = \frac{1}{2^r}(1 - \frac{1}{2^r})$.
3. $E[Z^r] = E[\sum_{x \in A'} I_x^r] = \sum_{x \in A'} E[I_x^r] = \frac{F_0}{2^r}$.

4. $Var[Z^r] = Var[\sum_{x \in A'} I_x^r]$. This equals $\sum_{x \in A'} Var[I_x^r]$ as I_x^r's are independent. Therefore, $Var[Z^r] = F_0 \frac{1}{2^r}(1 - \frac{1}{2^r}) < F_0 \frac{1}{2^r} = E[Z^r]$.

∎

Observation 9.3.4 *If $2^r > cF_0$ then $Pr(Z^r > 0) < \frac{1}{c}$.*

Proof. Recall Markov's inequality (see Theorem 2.5.1) that states that for a non-negative discrete random variable X and $s > 0$ be a constant, $Pr(X \geq s) \leq E[X]/s$. We have, $Pr(Z_r > 0) = Pr(Z_r \geq 1) \leq \frac{E[Z_r]}{1} \leq \frac{F_0}{2^r} < \frac{1}{c}$. ∎

Observation 9.3.5 *If $c2^r < F_0$ then $Pr(Z_r = 0) < \frac{1}{c}$.*

Proof. Recall Chebyshev's inequality (see Exercise 2.24) that states the following. Let X be a random variable with mean μ and variance $Var[X] = \sigma^2$. Let $s > 0$ be a constant. Then $P(|X - \mu| \geq s) \leq \frac{\sigma^2}{s^2}$. We know that $Var[Z^r] < E[Z^r]$. Note that $Pr[Z^r = 0] \leq Pr(|Z^r - E[Z^r]| \geq E[Z^r]) \leq \frac{Var[Z^r]}{E[Z^r]^2} < \frac{1}{E[Z^r]} = \frac{2^r}{F_0} < \frac{1}{c}$. ∎

Algorithm 9.3 is correct if $\frac{1}{c} \leq \frac{\widehat{F_0}}{F_0} \leq c$, where $\widehat{F_0} = 2^R$. By Observation 9.3.4 if $\frac{2^R}{F_0} > c$, then $Pr(Z^R > 0) < \frac{1}{c}$ and by Observation 9.3.5 if $\frac{2^r}{F_0} < \frac{1}{c}$ then $Pr(Z_r = 0) < \frac{1}{c}$. Thus, by the union bound, with probability at most $\frac{2}{c}$, $\frac{2^R}{F_0} > c$ or $\frac{1}{c} > \frac{2^R}{F_0}$. Hence, with probability at least $1 - \frac{2}{c}$, $\frac{1}{c} \leq \frac{2^R}{F_0} \leq c$.

The above shows that if $c = 6$, then the probability of success of the algorithm is $\geq \frac{2}{3}$. Next we see how we can further improve the success probability. We will execute Algorithm 9.3 s times, with different hash functions. Assume the s runs of the algorithm evaluates the location of the rightmost 1 at locations $R_1, \ldots, R_s$ (possibly not distinct). To compute $\widehat{F_0}$, we take the median value R among $\{R_1, \ldots, R_s\}$, and report $\widehat{F_0} = 2^R$. We will show that if we choose $s = O(\log \frac{1}{\epsilon})$, then for any $c > 4$, with probability at least $1 - \epsilon$, $\frac{1}{c} \leq \frac{\widehat{F_0}}{F_0} \leq c$. First we make the following observations.

Observation 9.3.6 *Two runs of the algorithm are independent, i.e. values of R_i and R_j do not depend on each other but only on the chosen hash functions.*

Define an indicator random variable X_i, $1 \leq i \leq s$, as follows:

$$X_i = \begin{cases} 0, & \text{if } \frac{1}{c} \leq \frac{\widehat{2^{R_i}}}{F_0} \leq c. \\ 1, & \text{otherwise.} \end{cases}$$

Intuitively, X_i is 1 if and only if the i-th run is not a "success". Note that $Pr(X_i = 1) \leq \frac{2}{c}$. We define $\beta = \frac{2}{c}$. Since we have assumed that $c > 4$, $\beta < \frac{1}{2}$. Define $X = \sum_{i=1}^{s} X_i$. Thus $E[X] \leq s\beta < \frac{s}{2}$.

Observation 9.3.7 *If $X < \frac{s}{2}$, then $\frac{1}{c} \leq \frac{\widehat{F_0}}{F_0} \leq c$.*

Proof. If $X < \frac{s}{2}$, then the algorithm fails at most half of the times, i.e. $\frac{2^{R_i}}{F_0} \notin (\frac{2}{c}, c)$ for at most half of the runs. Suppose we failed in the i-th run of the algorithm. That implies either the value of R_i is much lower or much higher than the "right" value. But the median value of $\{R_1, \ldots, R_s\}$ has to be the right value, otherwise more than half the values will be wrong values, contradicting the fact that $X < \frac{s}{2}$. ∎

Observation 9.3.8 *For any $\epsilon > 0$, if $s = O(\log \frac{1}{\epsilon})$, $Pr(X < \frac{s}{2}) \geq 1 - \epsilon$.*

Proof. We will show that $Pr(X \geq \frac{s}{2}) < \epsilon$. Since $E[X] < \frac{s}{2}$, we have that

$$\begin{aligned} Pr(X \geq \frac{s}{2}) &= Pr(X - E[X] \geq \frac{s}{2} - E[X]) \\ &= Pr(X - E[X] \geq \frac{s}{2} - s\beta) \\ &= Pr(X - E[X] \geq \frac{\frac{1}{2} - \beta}{\beta} s\beta) \\ &= Pr(X - E[X] \geq \frac{\frac{1}{2} - \beta}{\beta} E[X]) \\ &= Pr(X \geq (1 + \frac{\frac{1}{2} - \beta}{\beta}) E[X]) \end{aligned}$$

Recall Chernoff bounds (see Section 2.5) that states that for a random variable X that is sum of independent identically distributed $0-1$ random variables and $0 \leq \delta \leq 1$, we have that $Pr(X \geq (1+\delta)E[X]) \leq \exp(-\frac{\delta^2 E[X]}{3})$. Applying Chernoff bounds, where $\delta = \frac{\frac{1}{2}-\beta}{\beta} \in (0,1)$, we have that $Pr(X \geq \frac{s}{2}) \leq \exp(-\frac{1}{3}(\frac{\frac{1}{2}-\beta}{\beta})^2 E[X])$. We want $\exp(-\frac{1}{3}(\frac{\frac{1}{2}-\beta}{\beta})^2 E[X]) \leq \epsilon$. Since $E[X] \leq s\beta$, this is equivalent to finding for what value of s, $\exp(-\frac{1}{3}(\frac{\frac{1}{2}-\beta}{\beta})^2 s\beta) \leq \epsilon$. This reduces to $s > 3\frac{\beta}{(\frac{1}{2}-\beta)^2} \log \frac{\gamma}{\epsilon}$, for some constant γ. Thus, if $s = O(\log \frac{1}{\epsilon})$, we have that $Pr(X \geq \frac{s}{2}) < \epsilon$. ∎

Now look closely at the median value R of $\{R_1, \ldots, R_s\}$ that were computed from the s runs of the algorithm. From the above observations we conclude that when we set $\widehat{F_0} = 2^R$, with probability at least $1 - \epsilon$, $\frac{1}{c} \leq \frac{\widehat{F_0}}{F_0} \leq c$. Therefore, the probability of success is much better but now we need to store $s = O(\log \frac{1}{\epsilon})$ values each of which is $\log n$ bits long.

9.4 Counting in Sliding Windows

In this section we discuss the Basic Counting algorithm of [4] on maintaining statistics over a data stream. In particular, we are interested in answering queries over the last $N >> 0$ data items under the constraint that there is not sufficient space to store them in the memory. The input consists of an endless stream of binary bits. At any time, among the last N bits received, we are interested in queries that seek an approximate count of the number of 1's in the stream among the last k bits, where $k \leq N$. For answering these queries the Basic Counting maintains a simple data structure of $O(\frac{1}{\epsilon}\log^2 N)$ bits, for some constant $\epsilon > 0$. For each new bit, the time to update the structure is $O(\log N)$ and the count reported for number of 1's is within a factor of $1+\epsilon$. In the exercises, we will see that this can be generalized to the case where the stream consists of positive numbers and our task is to report the approximate sum of the numbers and its variations.

[4] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002

Let us discuss the Basic Counting algorithm of [45] for reporting an approximate count of the number of 1's in the stream of binary bits among the last k bits, where $k \leq N$. Observe that it is not possible to report the exact count of 1's among the last N bits of the stream by using only $o(N)$ space. If we don't know the exact locations of 1's, on the arrival of the new data bit we need to know whether the N-th latest bit is a "1" or "0" as this will influence the exact count of number of 1's. Similarly, when the next bit arrives, we need to know whether the $(N-1)$-th latest bit is a "1" or "0", ...

To maintain the approximate count we employ the following data structure. We will maintain (implicitly) the *time stamp* of each of the latest N bits. Each new bit in the stream gets a time stamp of 1 and the time stamps of all other (older) bits are incremented by one. We create $O(\log N)$ buckets. The 1's among the latest N bits are partitioned among these buckets. The number of 1's in a bucket will be a power of 2, except possibly one bucket. Each 1-bit is assigned to exactly one bucket and a 0-bit may or may not be assigned to any bucket. There are at most two buckets of a given *size*. The size of the bucket defined as the number of 1s in it. Let B_i denotes a bucket that holds 2^i number of 1-bits, where $i \in \{0, \ldots, \log N\}$. Each bucket also stores the time stamp of its most recent bit. As we will see that the most recent bit of any bucket will be the bit whose value 1. On receiving a new bit in the data stream, the following updates are done depending on whether the value of this bit is 0 or 1.
0**-bit**: We increment the time stamp of each of the buckets by 1, and if any of the buckets time stamp exceeds N, we discard that bucket.
1**-bit**: We create a bucket B_0 consisting of the newest 1-bit with a time stamp of 1. Now we scan the list of buckets in order of increasing

size. As a result of creating the bucket B_0, we may now have up to three buckets of size 1. If that is not the case, we increment the time stamps of each of the buckets as before, and possibly discard buckets whose time stamps exceed N. Otherwise, we have three buckets of type B_0. Let their time stamps be j, i, and 1, where $N \leq j < i < 1$. We merge the two oldest B_0 buckets, i.e. the buckets with time stamps i and j, to form a new bucket B_1 with time stamp i. This new bucket B_1 consists of all the bits from time stamp j up to the time stamp i. Therefore, this bucket includes exactly two 1-bits, and possibly many 0-bits. As a result of this process, we have one bucket of type B_0. But now we may have three buckets of type B_1. So we repeat the process for the two oldest buckets of type B_1 and replace them by a new bucket of type B_2. If this results in creating three buckets of type B_2, we repeat. This process can cascade at most $O(\log N)$ times. At the end, we will have at most two buckets of each type.

If we visualize the stream to be bits coming on a horizontal axis from right, then we have buckets of type B_0, followed by buckets of type B_1, then $B_2, \ldots$, as we traverse the stream from right to left. Moreover, for each type we have at most two buckets, and the last bucket (of the largest size) may overlap partially with the bits in the window of interest. See Figure 9.1 for an illustration.

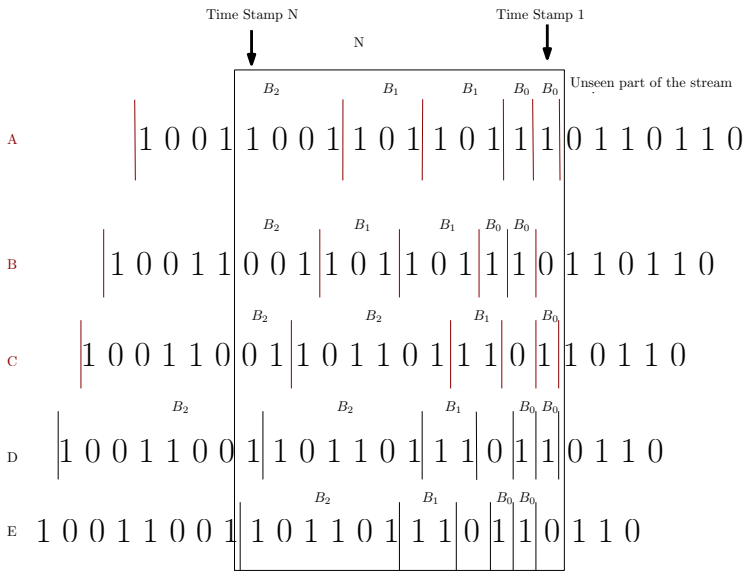


Figure 9.1: Illustration of Basic Counting. (A) Configuration of buckets in the window of size N of interest. It consists of two buckets of type B_0, each consisting of one 1-bit with time stamps 1 and 2 (time stamps of buckets are not shown). This is followed by two buckets B_1, each consisting of two 1-bits with time stamps 3 and 6, and finally a bucket B_2 which overlaps partially with the window of interest. (B) Buckets after receiving a 0 bit. Buckets remain the same. Their time stamps are incremented by 1. (C) Updated buckets after receiving a 1-bit. Note that as a result of creation of a new bucket B_0 consisting of the newest 1-bit, we have three buckets of type B_0. We merge the two older buckets of type B_0 to form a new bucket B_1, and that in turn created three buckets of type B_1. Two oldest of them are merged to form a new bucket B_2. (D) A new bucket B_0 is created with time stamp 1 after receiving the next 1-bit. (E) The last bucket is discarded as it falls outside of the window of interest.

Next we analyze the resources required in storing the data structure and also the time to update. We maintain $O(\log N)$ buckets as the size of window is N and the bucket of type B_i stores 2^i 1-

bits. For each bucket we need to store its time stamp and its size (or type). The time stamps requires $O(\log N)$ bits. Since the bucket B_i, for some $i \in \{0, \ldots, \log N\}$, stores 2^i 1-bits, it is sufficient to store i with bucket B_i for its size. As $0 \leq i \leq \log N$, i can represented using $O(\log \log N)$ bits. Therefore, the data structure requires $O(\log N(\log N + \log \log N)) = O(\log^2 N)$ bits of space. Now let us evaluate the time to update. On receiving a 0-bit, we update the time stamps of each of the $O(\log N)$ buckets. This requires $O(\log N)$ time. On receiving a 1-bit, not only we have to update the time stamps, but potentially merge and cascade buckets if required. It is easy to see that time to merge and cascade is proportional to the number of buckets and can be performed in $O(\log N)$ time.

Finally, let us see how to answer the queries. For any query value $k \in \{1, \ldots, N\}$, we want to report an approximate count of the number of 1's among the latest k bits of the stream. We initialize a count variable, say count, to 0. Following the convention used in Figure 9.1, we start traversing the buckets from right to left. For each bucket of type B_i that is encountered in the traversal, we can easily find out whether all of its bits are completely within the window of size k by looking at the time stamp of the next bucket. If B_i is completely contained in the window, we increment the count by 2^i. If a bucket B_i is completely outside the window, its contribution to the count is 0. This leaves us potentially with at most one bucket, say of type B_j, that partially overlaps the window. For this bucket, we know that its last bit is a 1-bit and it is within the window. But we don't know location of any of its other 1-bits and how many of them are within the window. Thus for this bucket, we only add half of its size, i.e. $\frac{2^j}{2}$ to the count. This may overestimate or underestimate the true count. We claim that the accumulated value of count is within a factor of two of the true count. In the following we justify the approximation factor.

Except of the bucket of type B_j that is partially in the window of size k, we know that all buckets of type $B_0, B_1, \ldots, B_{j-1}$ are completely within the window. For those buckets, the count of the number of 1-bits is $\sum_{i=0}^{j-1} 2^i \geq 2^j - 1$. Thus the true count (and the approximate count) value is at least 2^j (as the last bit of the bucket labelled B_j is in the window of interest). For the bucket B_j that overlaps partially with the window, the number of bits that can be in the true count can be anywhere from 0 upto $2^j - 1$. But we only took a contribution of 2^{j-1} in the reported value count. Thus the ratio of the true count to the reported count is within a factor of $(\frac{1}{2}, 2)$.

Next consider the following minor variation of the data structure. Let $r \geq 2$ be an integer parameter. In place of maintaining two

buckets of type B_i let us maintain r or $r-1$ copies of B_i for each $i \geq 1$. Note that the buckets of type B_0 may be less than $r-1$. Updates are as before, and at any time we exceed r copies of any type of buckets, we take the oldest two buckets and merge them to form a new bucket of the next size. As before, for the query assume that the bucket labelled B_j is only partially overlapping the query window. Now we know that at least $1 + \sum_{i=1}^{j-1}(r-1)2^i$ 1-bits are in the query window. We want to argue that true count and the reported value count are within a factor of $1 \pm \frac{1}{r-1}$. By setting $r = 1 + \frac{1}{\epsilon}$, we obtain a data structure of size $O(\frac{1}{\epsilon}\log^2 N)$ that can approximate the count of the number of 1s within a factor of $1 \pm \epsilon$.

9.5 Bibliographic Notes

A nice description of Heavy Hitters is in Gries and Misra [118]. Count-Min Sketch was proposed by Cormode and Muthukrishnan [39]. Bloom filters have been sketched by Bloom in [16] and a refined analysis is given by Bose et al. in [24]. Flajolet-Martin's algorithm for estimating distinct elements in a stream is given in [59]. A more detailed discussion and its generalization, the concept of median of means, and several exercises have been adapted from the paper by Alon, Matias and Szegedy [6]. The concept of counting in sliding windows was introduced in a paper by Datar, Gionis, Indyk, and Motwani [45]. Exercise 9.24 is also from their paper.

9.6 Exercises

9.1 *Suppose we want to ensure that we report all the elements that occur with a frequency of at least* 2% *in a stream with probability* ≥ 0.97. *Try to come up with reasonable values of b and r for the Count-Min Sketch (CMS) table. Justify your choice.*

9.2 *This problem is based on the Count-Min Sketch data structure. Suppose we want to ensure that we report all the elements that occur with a frequency of at least* 3% *in a data stream with probability* ≥ 0.95. *Try to come up with reasonable values of b and r for the Count-Min Sketch (CMS) table and justify your choice.*

9.3 *Consider a stream of IP-addresses that are being routed through a switch on a given day. Let* $h_1, \ldots, h_r$, *be* $r > 1$ *hash functions, where each function maps any IP-address to one of the possible b buckets uniformly at random. Assume that you are interested in performing some statistics on frequencies of IP-addresses on two consecutive days that use the same switch. Let* n_1 *be the number of packets that used the switch on the first*

day and the corresponding Count-Min-Sketch table be CMS_1 of size $r \times b$. Let n_2 be the number of packets that used the switch on the second day and let CMS_2 be the corresponding count-min-sketch table. For both the days we use the same set of hash-functions $(h_1, \ldots, h_k)$, though the tables are kept separate. In the class we did the analysis with respect to estimating the frequency count for an IP-address x with respect to a single table. Now suppose we want to estimate the combined frequency of x for the two days. Note that only thing which we have at the end of these two days are the two CMS tables and the count of total number of packets on each of the days. Let $f_x = \min\{CMS_1[1, h_1(x)] + CMS_2[1, h_1(x)], CMS_1[2, h_2(x)] + CMS_2[2, h_2(x)], \ldots, CMS_1[r, h_r(x)] + CMS_2[r, h_r(x)]\}$. How good is this estimate? In other words, estimate $Pr(f_x - f_x^* > \epsilon(n_1 + n_2))$, for a constant $\epsilon > 0$. Justify your answer.

9.4 *This problem refers to Count-Min Sketch. Suppose we have designed a beautiful CMS table with b columns and r rows for a streaming application that needs to be launched today afternoon. You worked extremely hard leading up to now to figure out the exact values of b and r. But, just now your manager informed you that you can actually increase the size of the CMS table by 50%. Should you add more rows? More columns? A combination of both? Justify your answer.*

9.5 *This is similar to the previous exercise. Recall that in the count-min sketch, we constructed a table* CMS *of size $r \times b$, and showed that for $b = \frac{2}{\epsilon}$, $Pr[f_x - f_x^* \geq \epsilon n] \leq \frac{1}{2^r}$, where f_x is an approximation to the true frequency f_x^* of an element x in the stream consisting of n elements. The* CMS *table uses a total of br space. Suppose, we are told that we can use twice the space. Given that now we can have $2br$ space, should we*

1. *Double the number of rows of the* CMS *table?*
2. *Double the number of columns of the* CMS *table?*
3. *Have two tables* CMS_1 *and* CMS_2*, each of size br. Both the tables use independent hash functions. We populate the entries in both the tables by following the same scheme as before. We set $f_x = \min\{f_x^1, f_x^2\}$, where f_x^1 and f_x^2 are the approximate frequency of x in* CMS_1 *and* CMS_2*, respectively.*

Justify your answer.

9.6 *Consider a stream S consisting of n natural numbers. We establish a count-min sketch table of size $r \times b$ to estimate the frequency of each element in S. We are interested in reporting all the elements that occur at least $n/1000$ times in S with a probability of at least 0.95. Can we achieve this objective with a CMS table where we set $b = 50$ and $r = 8$? Justify your answer. You can assume n is sufficiently large.*

9.7 *Here is an example of a tiny Bloom filter. It uses an array B consisting of 2 bits and two independent hash functions f and g that maps elements of a universe U uniformly at random to the indices of B. Initially both the bits of B are set to 0, i.e. $B = \boxed{0 \mid 0}$. Let $S = \{a, b\} \subset U$ be a set of two elements. We set up the Bloom filter for the membership in S using the elements of S as follows: $B[f(a)] = 1$, $B[f(b)] = 1$, $B[g(a)] = 1$, and $B[g(b)] = 1$. After this step, the array B has one of the following configurations:*

$$\boxed{0 \mid 1} \qquad \boxed{1 \mid 0} \qquad \boxed{1 \mid 1}$$

To test whether an element x from the universe is in S, we compute $f(x)$ and $g(x)$, and we say that $x \in S$ if both $B[f(x)] = 1$ and $B[g(x)] = 1$. Assume $x \notin S$. What is the probability of a false positive, i.e. the probability of saying that $x \in S$?

9.8 *Here is an example of a tiny Bloom filter. It uses an array B consisting of 3 bits and two independent hash functions h_1 and h_2 that maps elements of the universe U uniformly at random to the indices of B. Initially the bits of B are set to 0,*

i.e. $B = \boxed{0 \mid 0 \mid 0}$. Let $S = \{x, y\} \subset U$ be a set of two elements. We set up the Bloom filter for the membership in S using the elements of S as follows: $B[h_1(x)] = 1$, $B[h_2(x)] = 1$, $B[h_1(y)] = 1$, and $B[h_2(y)] = 1$. After this step, the array B has one of the following seven configurations:

$$\boxed{0 \mid 0 \mid 1} \quad \boxed{0 \mid 1 \mid 0} \quad \boxed{1 \mid 0 \mid 0}$$
$$\boxed{0 \mid 1 \mid 1} \quad \boxed{1 \mid 1 \mid 0} \quad \boxed{1 \mid 0 \mid 1}$$
$$\boxed{1 \mid 1 \mid 1}$$

To test whether an element α from the universe is in S, we compute $h_1(\alpha)$ and $h_2(\alpha)$, and we say that $\alpha \in S$ if both $B[h_1(\alpha)] = 1$ and $B[h_2(\alpha)] = 1$. Consider an element $z \in U$ such that $z \notin S$. What is the probability of false positive, i.e. saying that $z \in S$? Please present details of your calculations.

9.9 *Here is an example of a tiny Bloom filter. It uses an array of 9 bits and two independent hash functions f and g. We want to test membership in a set S of three elements, so we hash each of the three elements using both f and g, and we set to 1 any bit that any of the three elements is hashed to by either of the hash functions. When a new element x arrives, we compute $f(x)$ and $g(x)$, and we say x is in the set S if both $f(x)$ and $g(x)$ are 1. Assume x is not in the set S. What is the probability of a false positive; i.e., the probability of saying that x is in S.*

9.10 *Let A be a data stream. Without loss of generality assume that $a_1, a_2, \ldots, a_k$ are the most frequent k elements with frequencies $f_1 \geq f_2 \geq$*

$\cdots \geq f_k$, *respectively. We sample each element of* A *uniformly at random to construct a multi-set* A'. *We are interested to know how many times we need to sample* A *so that the most frequent* k *elements have a representative in* A' *with high probability. In other words, what should be the size of* A' *so that with probability* $\geq 1 - \epsilon$, *for* $\epsilon > 0$, *so that* $a_1, \ldots, a_k \in A'$.

Hint: Let us assume that $s = |A'|$. *Estimate first the probability that if we choose* s *elements from* A, *each uniformly at random with replacement, what is the probability that* $a_k \notin A'$? *What is an upper bound on the probability that none of* $a_1, a_2, \ldots, a_k$ *are in* A'? *Show that by choosing* $s = O(\frac{1}{f_k} \log \frac{k}{\epsilon})$, *with probability* $\geq 1 - \epsilon$, $a_1, \ldots, a_k \in A'$.

9.11 *This exercise is taken from* [5]. *Assume that we have a vector* $\mathbf{a}$ *of dimension* n *and its current state at time* t *is given by* $\mathbf{a}(t) = [a_1(t), a_2(t), \ldots, a_n(t)]$. *Initially, at time* $t = 0$, *for all* $1 \leq i \leq n$, $a_i(0) = 0$. *Updates to the elements of* $\mathbf{a}$ *are presented over time as a stream consisting of a pair of integers. At the* t*-th time instance, an update of the form* (i_t, c_t) *results in* $a_{i_t}[t] := a_{i_t}[t-1] + c_t$, *and for all* $j \neq i_t$, $a_j[t] := a_j[t-1]$. *Assume that* $c_t \geq 0$. *Show that using the count-min sketch we can answer the following:*

[5] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005

1. *Point Query: At any time* t, *return an approximation to* $a_i(t)$.
2. *Range Query: At any time* t *and for two indices* $1 \leq l < r \leq n$, *return an approximation to* $\sum_{i=l}^{r} a_i(t)$. *Hint: Note that we can construct* $O(n \log n)$ *ranges of type* (x, y), *where* $1 \leq x \leq y \leq n$ *and* $y - x = 2^k$ *for some integer* k. *These are called the dyadic ranges. For each* k, *we have* $O(n)$ *dyadic ranges and we can maintain a CMS table for each* k. *Observe that any range* (l, r) *can be expressed as the union of* $O(\log n)$ *disjoint precomputed sub-ranges. This can be deduced by looking at the binary representation of* $r - l + 1$.
3. *Approximate Median: Let* $\epsilon > 0$. *Let the number of elements seen by time* t *in the data stream be* m_t. *At any time* t, *report an index* $1 \leq j \leq n$ *such that* $\sum_{i=1}^{j} a_i(t) \geq \frac{m_t}{2} - \epsilon m_t$ *and* $\sum_{i=1}^{j-1} a_i(t) \leq \frac{m_t}{2} + \epsilon m_t$.
4. *Inner Product Query: Suppose we have two* n*-dimensional vectors* $\mathbf{a}$ *and* $\mathbf{b}$. *At any time* t, *report an approximation of the dot product* $\mathbf{a} \cdot \mathbf{b}$.

9.12 *Assume that a very large data stream* S *consists of elements from a universe* U. *Each element in* S *has the property that it may occur at most twice. Let* s_1 *be the count of the number of elements in* S *that occur exactly once. Similarly, let* s_2 *is the total count of the elements that occur exactly twice in* S. *To count the number of distinct elements in the stream* S *you may take a sample* $S' \subset S$, *say each element of* S *is chosen uniformly at random with probability* $0 < p < 1$. *Let the count of the number of elements in* S' *that occur exactly once be* s_1' *and the number of elements that occur exactly twice be* s_2'. *Derive an expression for expected value of* $E[s_1']$ *and* $E[s_2']$. *Is it true that* $\frac{s_2}{s_1+s_2} = \frac{s_2'}{s_1'+s_2'}$?

9.13 *This exercise is based on the paper of Alon et al. [6]. The following set of questions will provide us an estimate on the second frequency moment F_2 in a data stream $A = (a_1, \ldots, a_m)$, where each $a_i \in N = \{1, \ldots, n\}$. Recall that $F_2 = \sum_{i=1}^{n} m_i^2$, and m_i is the number of elements in A that are equal to i, for $1 \leq i \leq n$. Let h be a hash function that maps elements of N independently and uniformly at random to $\{-1, +1\}$. Consider the following algorithm:*

Algorithm 9.4: Computation of $\widehat{F_2}$

Input: Array A of size m where each $a_i \in N = \{1, \ldots, n\}$
Output: $\widehat{F_2}$

```
R ← 0
for i = 1 to m do
    R ← R + h(a_i)
end
F̂_2 ← R^2
return F̂_2
```

1. *Show that $R^2 = (\sum_{i=1}^{n} h(i)m_i)^2$.*
2. *Show that for any $i \in N$, $E[h(i)] = 0$ and $h(i)^2 = E[h(i)^2] = 1$.*
3. *For any $i \in N$, evaluate expected values of $h(i)^3$ and $h(i)^4$.*
4. *Show that $E[R^2] = F_2$.*
5. *Show that $E[R^4] = \sum_{i=1}^{n} m_i^4 + 6 \sum_{1 \leq i < j \leq n} m_i^2 m_j^2 \leq 3F_2^2$.*
6. *Show that $Var[R^2] \leq 2F_2^2$.*
7. *Modify the algorithm by taking s random hash functions $h_1, \ldots, h_s$, that are independent of each other and each of them maps elements of N independently and uniformly at random to $\{-1, +1\}$. We initialize s different counts: $R_1 = R_2 = \cdots = R_s = 0$. For $1 \leq i \leq s$ and $1 \leq j \leq m$, set $R_i := R_i + h(a_j)$. Report $X = \frac{1}{s} \sum_{i=1}^{s} R_i^2$ as an estimate for F_2. Show that $E[X] = F_2$ and $Var[X] \leq \frac{2}{s} F_2^2$.*
8. *Using Chebyshev's inequality, show that $Pr(|X - F_2| \geq \gamma F_2) \leq \frac{2}{s\gamma^2}$, for some positive constant γ.*
9. *Show that if we choose $s = \frac{2}{\gamma^2 \epsilon}$ in the previous exercise, then $Pr(|X - F_2| \leq \gamma F_2) \geq 1 - \epsilon$.*
10. *How much memory Algorithm 12 requires for execution?*

9.14 *This exercise is about the power of medians of means. Assume that we want to compute a value $\mathcal{X}$ using a randomized algorithm. In the analysis of our algorithm we use a random variable X that estimates $\mathcal{X}$, i.e. $E[X] = \mathcal{X}$. To have a good estimation, we take $k \times s$ independent random variables that have identical distribution as that of X, where $s = O(\log \frac{1}{\epsilon})$ and $k = \frac{cVar[X]}{\gamma^2 E[X]^2}$ for some positive constants c, γ, and ϵ. We denote them by $\{X_{11}, \ldots, X_{1k}, X_{21}, \ldots, X_{2k}, \ldots, X_{s1}, \ldots, X_{sk}\}$. Now we use the method of [6], where we define $Y_i = \frac{1}{k} \sum_{j=1}^{k} X_{ij}$, $1 \leq i \leq s$, and Z as the median value of $\{Y_1, \ldots, Y_s\}$. Show the following.*

1. *For $i \in \{1, \ldots, s\}$, $E[Y_i] = \mathcal{X}$.*
2. *$E[Z] = \mathcal{X}$.*
3. *$Var[Y_i] = \frac{1}{k} Var[X]$.*
4. *Using Chebyshev's inequality show that $Pr(|Y_i - \mathcal{X}| \geq \gamma\mathcal{X}) \leq \frac{1}{c}$.*
5. *Using the ideas from Observation 9.3.7 and the Chernoff bounds, show that $Pr(|Z - \mathcal{X}| \geq \gamma\mathcal{X}) \leq \epsilon$.*

9.15 *This exercise is similar to the previous exercise where we make some assumptions on $Var[X]$. Let $E[X] = \mathcal{X}$ and $Var[X] < c\mathcal{X}^2$. By using the median of means with $s = 3 \log \frac{2}{\epsilon}$ and $k = 8\frac{c}{\gamma^2}$, show that $Pr(|Z - \mathcal{X}| > \gamma\mathcal{X}) < \epsilon$. As a hint, try to answer the following questions.*

1. *For $i \in \{1, \ldots, s\}$, $E[Y_i] = \mathcal{X}$ and $E[Z] = \mathcal{X}$.*
2. *$Var[Y_i] = \frac{1}{k} Var[X]$.*
3. *Using Chebyshev's inequality show that $Pr(|Y_i - \mathcal{X}| \geq \gamma\mathcal{X}) < \frac{1}{8}$.*
4. *For each Y_j define an indicator random variable I_j given by*

$$I_j = \begin{cases} 1, & \text{if } |Y_j - \mathcal{X}| > \gamma\mathcal{X} \\ 0, & \text{otherwise.} \end{cases}$$

   Show that $E[I_j] < \frac{1}{8}$ and $E[\sum_{j=1}^{s} I_j] < \frac{s}{8}$.

5. *Recall that Z is the median of $Y_1, \ldots, Y_s$. Show that if $Z \notin ((1-\gamma)\mathcal{X}, (1+\gamma)\mathcal{X})$, then $\sum_{j=1}^{s} I_j > \frac{s}{2}$. Show, using the Chernoff bounds, that $Pr(|Z - \mathcal{X}| \geq \gamma\mathcal{X}) < 2\exp(-\frac{s}{3}) = \epsilon$.*

9.16 *(see Theorem 2.2. in [6]) By combining Exercises 9.13 and 9.14, show that for any $\epsilon > 0$ and $\gamma > 0$, we can compute an estimate $\widehat{F}_2$ of the second frequency moment F_2 for a data stream $A = (a_1, \ldots, a_m)$ in one pass, where*

each $a_i \in N = \{1, \ldots n\}$, using only $O(\frac{\log \frac{1}{\epsilon}}{\gamma^2}(\log n + \log m))$ memory bits and $Pr(|F_2 - \widehat{F}_2| \geq \gamma F_2) \leq \epsilon$. (Hint: See whether the choice of $s = 2 \log \frac{1}{\epsilon}$ and $k = \frac{16}{\gamma^2}$ will suffice in Exercise 9.14.)

9.17 *This exercise estimates the k-th frequency moment F_k. It is based on Section 2.1 of [6]. Let k be a positive integer. Let $A = (a_1, \ldots, a_m)$ be a data stream of m elements, where each $a_i \in N = \{1, \ldots, n\}$. Choose an index $p \in \{1, \ldots, m\}$ uniformly at random. Define r to be the number of times the element a_p occurs in the stream among the elements $(a_p, a_{p+1}, \ldots, a_m)$. Define the random variable $X = m(r^k - (r-1)^k)$.*

1. *Show that it is sufficient to maintain $O(\log n + \log m)$ bits to compute X.*
2. *Let $A = (1, 2, 2, 3, 1, 1)$. Evaluate F_k and $E[X]$.*
3. *Show that $E[X] = F_k$.*
4. *Show that if $a > b > 0$, then $a^k - b^k \leq (a-b)ka^{k-1}$.*
5. *Show that $E[X^2] \leq kmF_{2k-1} = kF_1F_{2k-1}$.*
6. *Assume that for n positive numbers $m_1, \ldots, m_n$, where each $m_i \geq 0$, the following inequality is true:*

$$\sum_{i=1}^{n} m_i \sum_{i=1}^{n} m_i^{2k-1} \leq n^{1-\frac{1}{k}} \left(\sum_{i=1}^{k} m_i^k \right)^2.$$

   Show that $Var[X] \leq E[X^2] \leq kn^{1-\frac{1}{k}} F_k^2$.

7. *Apply the framework of Exercise 9.16 by constructing sufficient number of estimates similar to X. Define $Y_1, \ldots, Y_s$, where each Y_i is average of some random variables, and show that with high probability the median Y value doesn't deviate from F_k significantly.*
8. *Estimate the space used by the algorithm.*

9.18 *Let $S = \{x_1, \ldots, x_n\}$ be a set of n distinct numbers. We are interested in finding an approximate median element of S. Define the rank of an element $y \in S$ as the number of elements in S that are $\leq y$, i.e. $rank(y) = |\{x \in S | x \leq y\}|$. An element $y \in S$ is an approximate median of S, if $\frac{n}{2} - \epsilon n \leq rank(y) \leq \frac{n}{2} + \epsilon n$ for some $\epsilon \leq \frac{1}{6}$. We employ the following strategy to find an approximate median element. We sample s elements from S, each independently and uniformly at random with replacement. Let $S' \subset S$ be the set of sampled elements. We set y to be the median of the sampled elements. Define the three subsets of S as follows.*

$$L = \{x \in S : rank(x) < \frac{n}{2} - \epsilon n\}$$
$$U = \{x \in S : rank(x) > \frac{n}{2} + \epsilon n\}$$
$$M = \{x \in S : \frac{n}{2} - \epsilon n \leq rank(x) \leq \frac{n}{2} + \epsilon n\}$$

Answer the following.

1. *Show that the probability that a sampled element is from the set L is $\frac{1}{2} - \epsilon$.*
2. *Let $X = |L \cap S'|$. Show that $E[X] = (\frac{1}{2} - \epsilon)s$.*
3. *Show that if $|L \cap S'| > \frac{s}{2}$, then y is not an approximate median. Same holds if $|R \cap S'| > \frac{s}{2}$.*
4. *Show that $Pr(X > \frac{s}{2}) \leq Pr(X \geq (1+\epsilon)E[X])$.*
5. *Using Chernoff bounds and by setting $s = \frac{9}{\epsilon^2} \log \frac{2}{\delta}$ show that $Pr(X \geq (1+\epsilon)E[X]) \leq exp(-\frac{\epsilon^2}{3}E[X]) \leq \frac{\delta}{2}$.*
6. *Show that if $|L \cap S'| \leq \frac{s}{2}$ and $|R \cap S'| \leq \frac{s}{2}$, then y is an approximate median.*
7. *Show that if we draw $s = \frac{9}{\epsilon^2} \log \frac{2}{\delta}$ samples, $Pr(\frac{n}{2} - \epsilon n \leq rank(y) \leq \frac{n}{2} + \epsilon n)) \geq 1 - \delta$.*
8. *How many samples we need to draw if $\epsilon = 0.1$ and we want to succeed with probability at least $3/4$?*

9.19 *Continuing with the previous exercise, suppose input is a data stream A of unknown size where we are allowed to perform one pass in order to find an approximate median. We do not have enough space to store all the elements of A, but have enough space to store the s sampled elements of A. This exercise is about how to sample s elements from a data stream A, uniformly at random, where we do not know the size of A in advance and we can only afford to store $O(s)$ elements. Let S be the set of s sampled elements of A that we wish to report. We employ the following strategy: Store the first s elements of A in S. For each successive element of A, say the i-th element ($i > s$), we toss a coin where the probability of the favourable outcome is s/i. If the outcome is favourable, the i-th element replaces one of the elements S, selected uniformly at random. Show that when the algorithm has terminated, each element of A has a probability of $\frac{s}{|A|}$ being in S. (Hint: Think first of the simpler cases where $s = 1$ or $s = 2$.)*

9.20 *Let us look at the Count Sketch algorithm, an alternate method for estimating the frequency of elements in a stream [35], that came before the Count-Min-Sketch. Similar to CMS, we have a table consisting of r rows and b columns. In addition to having the hash functions $h_1, \ldots, h_r$, we also have hash functions $s_1, \ldots, s_r$, where each $s_i : \mathbb{N} \to \{-1, +1\}$. The algorithm is similar to CMS except that for each element of the stream $A[i]$, we increment/decrement the value in the j-th row of the table based on the outcome of $s_j(A[i])$. The details are sketched in Algorithm 9.5. To estimate the frequency η_x of an element $x \in A$, we report the median value of $\{s_1(x) \cdot CS[1, h_1(x)], s_2(x) \cdot CS[2, h_2(x)], \ldots, s_r(x) \cdot CS[r, h_r(x)]\}$. For*

Algorithm 9.5: Computation of Count Sketch Table
Input: An array A consisting of n natural numbers and $2r$ hash functions $h_1, \ldots, h_r, s_1, \ldots, s_r$, where each $h_i : \mathbb{N} \to \{1, \ldots, b\}$ and each $s_i : \mathbb{N} \to \{-1, +1\}$
Output: $CS[\cdot, \cdot]$ table consisting of r rows and b columns

```
for i = 1 to r do
    for j = 1 to b do
        CS[i, j] ← 0
    end
end
for i = 1 to n do
    for j = 1 to r do
        CS[j, h_j(A[i])] ← CS[j, h_j(A[i])] + s_j(A[i])
    end
end
return CS[·, ·]
```

an element $x \in A$ and $i \in \{1, \ldots, r\}$ and $j \in \{1, \ldots, b\}$, define $B_i[h_i(x)]$ to be the set of elements other than x that are mapped to the same bucket where x is mapped in the i-th row of the CS table. Let K be the set of k most frequent elements in A and similarly $\bar{K}$ to be the set of non-frequent items in A, i.e. $\bar{K} = \mathbb{N} \cap (A \setminus K)$. Define $B_i^{>k}[h_i(x)] = B_i[h_i(x)] \cap \bar{K}$. Let $c > 1$ be a constant. Answer the following questions.

1. *Show that for any element $x \in A$ and for any $i \in \{1, \ldots, r\}$, $s_i(x) \cdot CS[i, h_i(x)] = \eta_x + s_i(x) \sum_{y \in B_i[h_i(x)]} s_i(y) \cdot \eta_y$.*
2. *Show that for any element $x \in A$ and for any $i \in \{1, \ldots, r\}$, $E[s_i(x) \cdot CS[i, h_i(x)]] = \eta_x$.*
3. *Show that $Var[s_i(x) \cdot CS[i, h_i(x)]] = \sum_{y \in B_i[h_i(x)]} \eta_y^2$.*
4. *Show that $E[\sum_{y \in B_i^{>k}[h_i(x)]} \eta_y^2] = \frac{1}{b} \sum_{z \in \bar{K}} \eta_z^2$.*
   Using Markov's inequality, show that $Pr(\sum_{y \in B_i^{>k}[h_i(x)]} \eta_y^2 \leq \frac{c}{b} \sum_{z \in \bar{K}} \eta_z^2) \geq 1 - \frac{1}{c}$.
5. *Let $b \geq ck$. Show that $Pr(B_i[h_i(x)] \cap K = \emptyset) \geq 1 - \frac{1}{c}$.*
6. *Show that for any element $x \in A$, $Pr((s_i(x) \cdot CS[i, h_i(x)] - \eta_x)^2 \leq cVar[s_i(x) \cdot CS[i, h_i(x)]]) \geq 1 - \frac{1}{c}$*
7. *Combining the previous three exercises, show that $Pr(\sum_{y \in B_i^{>k}[h_i(x)]} \eta_y^2 \leq$*

$\frac{c}{b}\sum_{z\in\bar{K}}\eta_z^2) \wedge Pr(B_i[h_i(x)] \cap K = \emptyset) \wedge Pr((s_i(x)\cdot CS[i,h_i(x)] - \eta_x)^2 \leq cVar[s_i(x)\cdot CS[i,h_i(x)]]) \geq 1 - \frac{3}{c}$. *Furthermore, suppose for some* $x \in A$ *and* $i \in \{1,\ldots,r\}$, *the above three probability statements are true. Then show that* $Pr(|s_i(x)\cdot CS[i,h_i(x)] - \eta_x| \leq c\sqrt{\frac{\sum_{z\in\bar{K}}\eta_z^2}{b}}) \geq 1 - \frac{3}{c}$.

8. *To estimate the frequency* η_x *of* $x \in A$, *we return the median value* $\widehat{\eta_x}$ *of* $\{s_1(x)\cdot CS[1,h_1(x)], s_2(x)\cdot CS[2,h_2(x)],\ldots,s_r(x)\cdot CS[r,h_r(x)]\}$. *Let* $r = \Omega(\log\frac{n}{\delta})$ *and* $c \geq 8$. *Show that the expected number of indices in* $\{1,\ldots,r\}$ *that satisfy* $Pr(|s_i(x)\cdot CS[i,h_i(x)] - \eta_x| \leq 8\sqrt{\frac{\sum_{z\in\bar{K}}\eta_z^2}{b}})$ *is* $\geq \frac{5}{8}r$. *Using Chernoff bounds show that with high probability at least* $\frac{r}{2}$ *indices in* $\{1,\ldots,r\}$ *satisfy* $Pr(|s_i(x)\cdot CS[i,h_i(x)] - \eta_x| \leq 8\sqrt{\frac{\sum_{z\in\bar{K}}\eta_z^2}{b}})$. *Conclude that* $Pr(|\eta_x - \widehat{\eta_x}| \leq 8\sqrt{\frac{\sum_{z\in\bar{K}}\eta_z^2}{b}}) \geq 1 - \frac{\delta}{n}$, *as* $\widehat{\eta_x}$ *is the median*

9. *While executing the Algorithm 9.5 we can maintain a heap of k-elements that have the k highest median values. When the next element* $x \in A$ *is considered, if it is already in the heap its count is incremented. Otherwise, if its median value is greater than the smallest median value in the heap, then x is added to the heap and the element with the smallest median value is removed. Let* η_k *be the frequency of the k-th most frequent element in A. Let* $b = \max(8k, \frac{256\sum_{z\in\bar{K}}\eta_z^2}{(\epsilon\eta_k)^2})$, *for some* $\epsilon > 0$. *Show that when the algorithm has terminated all the elements whose frequency is at least* $(1+\epsilon)\eta_k$ *are in the heap. Furthermore, show that no element whose frequency is less that* $(1-\epsilon)\eta_k$ *will be in the heap.*

9.21 *In the* Basic Counting *algorithm why we need to take multiple copies of the buckets of type* B_i *for* $i \geq 0$? *What will happen in the analysis if we only take at most one copy of each of the bucket types?*

9.22 *This question is about Stream Statistics Over Sliding Windows. You need to determine the value of r, where we use up to* $r \geq 2$ *buckets of type* B_i *for* $i \geq 0$, *so that the count of the number of* 1*s reported by the algorithm is within* 5% *of the actual count of* 1*s among the last N bits seen in a data stream. Justify your choice of r. Furthermore, analyze the total space used by the data structure that maintains all the required buckets to achieve the desired accuracy?*

9.23 *Consider a stream consisting of positive numbers, where each number is represented using d-bits. We are interested in answering queries among the last N numbers received. The query consists of a value* $k \in \{1,\ldots,N\}$, *and we want to know the (approximate) sum of the last k numbers in the stream. Modify the* Basic Counting *algorithm's data structure so that now*

in place of counting 1*'s in the bit stream, it can approximate the sum. Note that you only have memory to store* $o(N)$ *numbers as before.*

Hint: Consider d-streams, where the k-th stream represents the k-th most-significant bit of the numbers.

9.24 *Assume that we have a stream consisting of numbers from the set* $\{-1, 0, +1\}$ *and we are interested in maintaining the sum of last N bits of the stream. In this exercise we will show that it will require* $\Omega(N)$ *bits to maintain an approximate sum that is within a constant factor of the exact sum. Suppose we have an algorithm* $\mathcal{A}$ *that maintains the approximate sum within a constant factor on the input consisting of stream of* $\{-1, 0, +1\}$ *. Assume that we have a bit string X consisting of* $\frac{N}{2}$*-bits composed of only* 0*s and* 1*s. We form an input of N bits for Algorithm* $\mathcal{A}$ *as follows: We replace each* 0*-bit of X by a pair of bits* $(1, -1)$ *and each* 1*-bit of X by the pair* $(-1, 1)$*. Now this sequence of N-bits is presented to our algorithm* $\mathcal{A}$*. Note that the exact sum of these N-bits is* 0*. In addition to these N bits derived from stream X, the next set of N bits that will be presented to* $\mathcal{A}$ *consists of only* 0*-bits and we will show that we will recover completely the original bit vector X. Answer the following:*

1. *Show that if the* $(N+1)$*-st bit in the stream for* $\mathcal{A}$ *is* 0*, the output to the exact sum query on receiving this bit will be* $+1$ *(respectively* -1*) if and only if the* 1*st bit in the stream of X was a* 1 *(respectively,* 0*). Moreover, on receiving this* $(N+1)$*st* 0 *bit, Algorithm* $\mathcal{A}$ *will output a positive number (resp. negative number) if and only if the* 1*st bit in the stream of X was a* 1 *(respectively,* 0*)*

2. *For a positive integer* $i < \frac{N}{2}$*, show that after receiving the* $(N + 2i - 1)$*-th* 0 *bit, the output to the approximate sum query algorithm* $\mathcal{A}$ *is a positive number (respectively a negative number) if and only if the i-th bit in the stream X was a* 1 *(respectively,* 0*).*

3. *Show that after receiving the* $2N$*-th* 0 *bit by* $\mathcal{A}$ *, we would have completely recovered all the bits of the stream X (and therefore the first N bits of the stream A).*

4. *Conclude that to estimate the approximate sum within a constant factor in a sliding window of size N in a stream of (positive and negative) numbers we need to store* $\Theta(n)$ *bits.*

9.25 *Consider the following snapshot in the DGIM algorithm, where we want to count the number of* 1*s in a sliding window. The table below gives a snapshot of the last* 100 *bits received. (Assume that the most recent bit has an end time of 100, i.e. the new bits enter the table from the right. Note that the meaning of the entry in the column corresponding to eighty is that there are exactly eight 1s in the locations from sixty-six (inclusive) to eighty (inclusive), and the location eighty is 1.)*

End Time	*65*	*80*	*87*	*92*	*95*	*98*	*100*
Size (#1s)	*8*	*8*	*4*	*2*	*2*	*1*	*1*

Answer the following:

1. *What will be the estimate of the number of 1s in the most recent 21 bits and in the most recent 40 bits?*
2. *What will be the minimum and maximum number of 1s that are possible in the most recent 21 and 40 bits, respectively.*
3. *Construct the table after the following four bits (in order) are added: 1 0 1 1.*
   (To remove any confusion, 101st bit is 1, 102nd bit is 0, 103rd bit is 1, and 104th bit is 1.)

9.26 *Consider an endless stream of binary bits. Consider the following snapshot in the DGIM algorithm, where we want to count the number of* 1s *in a sliding window. Recall that we choose at most two buckets that consist of the same number of 1s. Moreover, the number of* 1s *in a bucket is some power of* 2*. The table below gives a snapshot of the last 100 bits received. (Assume that the most recent bit has an end time of 100, i.e. the new bits enter the table from the right. Note that the meaning of the entry in the column corresponding to sixty-five is that there are precisely eight 1s in the locations from forty-one (inclusive) to sixty-five (inclusive), and the location sixty-five is 1.)*

End Time	*40*	*65*	*72*	*82*	*90*	*97*	*100*
Size (#1s)	*8*	*8*	*4*	*2*	*2*	*1*	*1*

Answer the following:

1. *What will be the estimate of the number of 1s in the most recent 45 bits reported by the DGIM algorithm?*
2. *What will be the minimum and the maximum number of 1s possible in the most recent 45 bits in the above setting.*
3. *Construct the table after the following four bits (in order) are added in the stream: 1 1 0 1 (Note that: 101st bit is 1, 102nd bit is 1, 103rd bit is 0, and 104th bit is 1).*

9.27 *Consider tracking the most popular movies from the sale of movie tickets sold worldwide. Let* $c = 10^{-6}$ *and* $\tau = 1/2$*. We maintain decaying scores for movies whose threshold is at least* τ*. For each new ticket sale for a movie, say without loss of generality this is for the movie M, perform the following steps.*

1. *For each movie whose score is being maintained, its new score is reduced by a factor of* $(1 - c)$*. (To be precise, if the score of a movie was* s*, the new score is* $s := s(1 - c)$*.)*

2. *If we have the score of M, add 1 to that score. Otherwise, create a new score for M and initialize it to 1.*

3. *Remove any score that falls below τ.*

Answer the following questions:

1. *What is the sum of all scores at any point in time?*

2. *How many scores are maintained at any given time?*

3. *If $\tau = 1/3$ instead of $1/2$, what will be the number of scores maintained and the sum of all the scores at any point in time?*

9.28 *Suppose you have a stream S consisting of 55 numbers, where each number i, $1 \leq i \leq 10$, occurs i times. (To clarify, 1 occurs once, 2 occurs twice, 3 occurs thrice, ..., 10 occurs ten times.) Answer the following questions:*

1. *Compute the frequency moment F_0 of S.*

2. *Execute the following algorithm to evaluate the estimate $\hat{F}_0$ for the given input stream S.*

   Step 1: Initialize $R := 0$

   Step 2: For each element $i \in S$ do:

   (a) Compute binary representation of i

   (b) Let r be the location of the rightmost 1 in the binary representation

   (c) if $r > R$, $R := r$

   Step 3: Return $\hat{F}_0 = 2^R$

3. *Evaluate the second frequency moment F_2 of S.*

4. *Execute the following algorithm to evaluate the estimate $\hat{F}_2$ for the input stream S. You can assume that the hash function h maps every even integer to -1 and every odd integer to $+1$.*

   Step 1: Initialize $Y := 0$.

   Step 2: For each element $i \in S$, $Y := Y + h(i)$

   Step 3: Return $\hat{F}_2 = Y^2$

9.29 *Here is an alternative simple method to find an estimate on F_0, i.e., the number of distinct elements, in a stream A consisting of n elements. Elements in A are drawn from a universe U. Let $h : U \rightarrow (0, 1]$ be a random hash function. The steps in the algorithm are as follows:*

Step 1: $Y := 0$.

Step 2: On the arrival of element x in the stream A, compute $h(x)$.
If $Y > h(x)$, $Y := h(x)$.

Step 3: Return $\hat{F_0} = \lceil \frac{1}{\bar{Y}} \rceil$.

Answer the following.

1. *Show that* $E[Y] = \frac{1}{1+F_0}$.
   Hint: Show that $E[Y] = \int_0^1 Pr(Y \geq x)dx = \int_0^1 (1-x)^{F_0} dx$.

2. *Show that* $Var[Y] \leq \left(\frac{1}{1+F_0}\right)^2$.
   Hint: Show that $E[Y^2] = \int_0^1 x^2 F_0 (1-x)^{F_0-1} dx = \frac{2}{(1+F_0)(2+F_0)}$.

Recall probability density functions, cumulative probability density functions, expected value and variance of continuous random variables.

Instead of using one hash function, let us use k independent hash functions $h_1, \ldots, h_k : U \to (0,1]$, *and compute the corresponding* $Y_1, \ldots, Y_k$. *Define* $\bar{Y} = \frac{1}{k} \sum_{i=1}^{k} Y_i$. *Answer the following.*

1. *Show that* $E[\bar{Y}] = \frac{1}{1+F_0}$.

2. *Show that* $Var[\bar{Y}] \leq \left(\frac{1}{k(1+F_0)}\right)^2$.

3. *By applying Chebyshev's inequality show that*
   $Pr(|\bar{Y} - \frac{1}{1+|F_0|}| \geq \frac{\epsilon}{1+F_0}) \leq \frac{1}{k\epsilon^2}$.

4. *Show that with probability* $\geq 1 - \frac{1}{k\epsilon^2}$,
   $(1 - O(\epsilon))F_0 \leq \lceil \frac{1}{\bar{Y}} \rceil - 1 \leq (1 + O(\epsilon))F_0$.

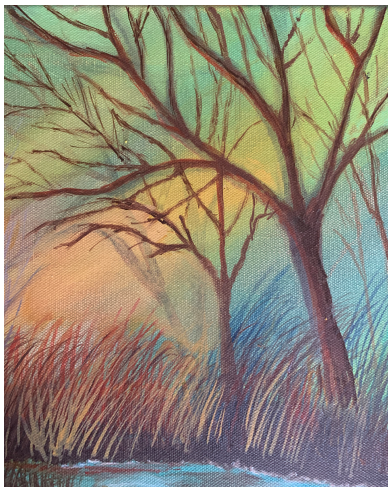
5. *Show that if* $k > \frac{4}{\epsilon^2}$, *with probability* $\geq \frac{3}{4}$, $\bar{Y} \in \left[\frac{1-\epsilon}{1+|F_0|}, \frac{1+\epsilon}{1+|F_0|}\right]$.

10
Online Algorithms

We will focus on

1. Online algorithm for bipartite matching.
2. WATERLEVEL algorithm for fractional bipartite matching.
3. RANKING randomized algorithm for bipartite matching.
4. BALANCE algorithm for b-matching.

Keywords: LP, LP Duality, competitive ratio, matching, online algorithms, WATERLEVEL Algorithm, BALANCE algorithm, RANKING algorithm.



An algorithmic solution to a problem consists of efficiently transforming the given input to the desired output. Typically, the whole input is presented before the algorithm starts. Whereas in an online algorithm the input items arrive over time. When the new input item arrives, the online algorithm has to make an irreversible decision on what to do with the new item. Therefore, once the decision is being made, it cannot be reversed or altered on the arrival of future items. In this chapter we look at some of the recent algorithms related to online fractional bipartite matching problem and its implications to web advertising. We will also look at some classical results in online learning theory related to regret minimization and its applications to zero-sum games. This chapter is based on [48, 92, 98, 111, 131].

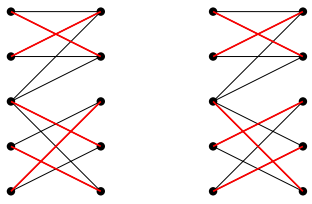


Figure 10.1: Red edges forms matching. Maximum matching may not be unique.

10.1 Online Bipartite Matching

In this section we discuss an online algorithm for finding a matching in a bipartite graph. Let $G = (V = L \cup R, E)$ be a bipartite graph where the vertex set V consists of the sets L and R (referred to as

'left' and 'right' sets) and a set E of edges (v, w) where $v \in L$ and $w \in R$. The set $M \subseteq E$ is a matching in G if no two edges in M share a vertex.

Graph G is presented in an online manner. All the vertices in the set L are known in advance, but the vertices in R and the edges are presented over time. At each time instant $t \in \{1, 2, 3, \dots\}$, a new vertex $r_t \in R$ and all its incident edges arrive. The online matching algorithm needs to decide among all the currently unmatched neighbors of r_t in the set L to which vertex (if any) r_t should be matched. The vertex r_t remains matched to that vertex (if any) for the rest of the algorithm.

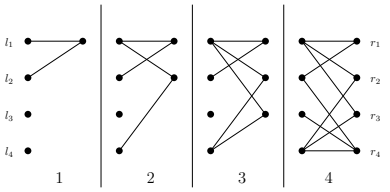


Figure 10.2: Online arrival of vertices in R with their incident edges over 4 time steps.

Our task is to come up with an online algorithm that maximizes the size of the matching M reported by the online algorithm. By size, we mean the number of edges in M. To understand the quality of our solution we use the widely popular notion of the competitive analysis where we compare the size of M against the size of the maximum matching M^* in G. This comparison seems to be unfair as the online algorithm doesn't have the full knowledge of G and an adversary may choose a permutation of vertices of R that is possibly the worst for making matching decisions at each time stance. Even in this adversarial setting competitive bipartite matching algorithms have been proposed. We will present a straightforward simple greedy strategy and show that it is $\frac{1}{2}$-competitive using the LP-duality framework.

First let us see that a deterministic algorithm can't achieve better than $\frac{1}{2}$-competitive ratio.

Example 10.1.1 *Consider a bipartite graph on* 4 *vertices, where* $L = \{l_1, l_2\}$ *and* $R = \{r_1, r_2\}$. *At the first time step the algorithm is presented with the vertex* r_1 *and the two incident edges* (r_1, l_1) *and* (r_1, l_2). *Let us say that the online algorithm decides to add the edge* (r_1, l_1) *to* M. *At the next time step, the algorithm receives* r_2 *and only one edge* (r_2, l_1). *Since* l_1 *is already matched to* r_1, r_2 *remains unmatched. So the size of* M *is* 1, *whereas the optimal matching for* G *is* $(r_1, l_2), (r_2, l_1)$ *of size* 2.

Depending on the action of r_1, *the adversary can decide which edge to present as an incident edge to* r_2 *in the second step and therefore the deterministic online algorithm can't do better.*

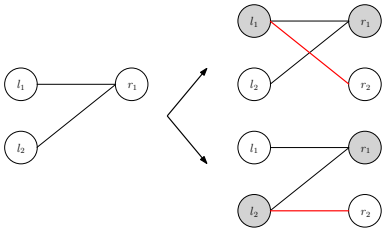


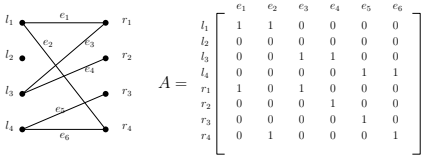
Figure 10.3: Example where the competitive ratio is $\frac{1}{2}$

We first define a few quantities before presenting the linear programming (LP) formulation. For each edge $e \in E$ let x_e to be a non-negative variable taking a real value. Let $Adj(v)$ refers to the edges incident to the vertex $v \in L \cup R$. We will also express the LP using the standard matrix-vector notation, where

1. $c = \begin{pmatrix} 1 \\ 1 \\ \dots \\ 1 \end{pmatrix}$ is a vector of length $|E|$.

2. $b = \begin{pmatrix} 1 \\ 1 \\ \dots \\ 1 \end{pmatrix}$ is a vector of length $|V|$.

3. $x = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_{|E|} \end{pmatrix}$ is the vector of variables corresponding to the edges.



4. A is a $|V| \times |E|$ matrix and its ij-th entry is 1 if the edge corresponding to the column j is incident on the vertex corresponding to the row i, otherwise 0.

The Primal Linear Program can be stated as follows.

Primal LP:

Objective function:

$\max \sum_{e \in E} x_e$	$\max c^T x$

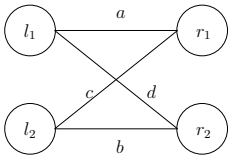
Subject to:

$\sum_{e \in Adj(v)} x_e \leq 1$, for all $v \in L \cup R$	$Ax \leq b$
$x_e \geq 0$, for all $e \in E$	$x \geq 0$.

In the above formulation the variables can take fractional values, but it is also known that there is an integral solution that achieves an optimal value. (We know this from the existence of maximum matching. Alternatively, we can argue as follows. If an edge e takes a fractional value $x_e > 0$, then it will have a neighboring edge taking a fractional value, and so on. This forms a cycle where each edge on the cycle is taking a fractional value. Then one can move around the smallest fractional value to other edges on the cycle without altering the value of the objective function. This results in one fewer edge taking the fractional value. Continuing this process on the finite graph we can eventually show that each edge (i.e. x_e) takes an integral value. The optimal value of the Primal LP is the size of the maximum matching $|M^*|$ in G and the edges taking the value $x_e = 1$ constitute M^*.

Example 10.1.2 *Consider the complete bipartite graph where* $|L| = |R| = 2$. *Assume that* $L = \{l_1, l_2\}$ *and* $R = \{r_1, r_2\}$. *It is easy to see that for this graph the maximum value of the objective function of the Primal LP is* 2.

For example, we can achieve an optimum value of 2 *by setting* $x_{l_1,r_1} = x_{l_2,r_2} = x_{l_1,r_2} = x_{l_2,r_1} = \frac{1}{2}$. *We can also achieve the optimal value by setting* $x_{l_1,r_1} = x_{l_2,r_2} = 1$ *and* $x_{l_1,r_2} = x_{l_2,r_1} = 0$.



Primal	Dual
$\max c^T x$	$\min b^T y$
$Ax \leq b$	$A^T y \geq c$
$x \geq 0$	$y \geq 0$.

What is a Dual LP?

Consider the following Linear Program:

$$\begin{aligned} \max\, & x_1 + x_2 \\ & x_1 + 2x_2 \leq 4 \\ & 2x_1 + x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Let us try to find an upper bound to the value of the objective function. Given that $x_1, x_2 \geq 0$, from the first constraint we have $x_1 + x_2 \leq x_1 + 2x_2 \leq 4$ and from the second constraint we have $x_1 + x_2 \leq 2x_1 + x_2 \leq 6$. Thus the constraints state that the value of the objective function cannot be more than 4. But we can also consider linear combinations of the constraints. For example, consider $\alpha(x_1 + 2x_2 \leq 4) + \beta(2x_1 + x_2 \leq 6)$, where $\alpha, \beta \geq 0$. The combination $4\alpha + 6\beta$ can be an upper bound to the objective function of LP provided that $\alpha + 2\beta \geq 1$ (corresponding to x_1) and $2\alpha + \beta \geq 1$ (corresponding to x_2). Suppose we set $\alpha = \frac{1}{2}$ and $\beta = \frac{1}{4}$. This results in taking the linear combination $\frac{1}{2}(x_1 + 2x_2) + \frac{1}{4}(2x_1 + x_2) = x_1 + \frac{5}{4}x_2 \geq x_1 + x_2$. Since this choice of α and β also satisfies $\alpha + 2\beta \geq 1$ and $2\alpha + \beta \geq 1$, $\frac{1}{2}4 + \frac{1}{4}6 = \frac{7}{2}$ is another upper bound to the value of objective function. We may also consider $\alpha = \beta = 1/3$. This choice satisfies the constraints and results in an upper bound of $\frac{10}{3}$, bit better than $\frac{7}{2}$. We can see that $x_1 = \frac{8}{3}$ and $x_2 = \frac{2}{3}$ satisfies the constraints of the LP and results in the objective value of $\frac{10}{3}$. Thus the upper bound using the linear combination that we obtained is the optimal value! Moreover, finding the right upper bound can also be written as a (dual) linear program:

$$\begin{aligned} \min\, & 4\alpha + 6\beta \\ & \alpha + 2\beta \geq 1 \\ & 2\alpha + \beta \geq 1 \\ & \alpha, \beta \geq 0 \end{aligned}$$

In general, given the Primal Linear Program $\max c^T x$ subject to $Ax \leq b$, $x \geq 0$, its Dual LP is expressed as $\min b^T y$, subject to $A^T y \geq c$,

$y \geq 0$. Looking closely at the Primal-Dual pair, we observe that

1. For each variable in the Primal we have a constraint in the Dual.
2. For each constraint in the Primal we have a variable in the Dual.
3. Maximization becomes a Minimization problem.
4. If x and y are feasible solutions to the Primal and Dual LPs, respectively, then $c^T x \leq (A^T y)^T x = y^T (Ax) \leq y^T b = b^T y$. Note that if x and y are feasible then $x, y \geq 0$ and are bounded and therefore we can apply the above substitution to obtain that $c^T x \leq b^T y$. This is called the Weak Duality.

The Strong Duality Theorem states that if x and y are optimal values for the Primal and Dual LPs, respectively, then $c^T x = b^T y$.

Let us consider the Dual Linear Program to the maximum matching LP. We will introduce $|V|$ variables corresponding to each vertex constraint of the primal. We label them $p_1, p_2, \ldots, p_{|V|}$ and let $p = (p_1, p_2, \ldots, p_{|V|})^T$. Recall that the value of the objective function of the Dual LP is an upper bound to the value of the objective function of the Primal LP.

Dual LP:

Objective function:

$$\min \sum_{v \in V} p_v \quad \Big| \quad \min b^T p$$

Subject to:

$$p_v + p_w \geq 1, \text{for all } e = (v, w) \in E \quad \Big| \quad A^T p \geq c$$
$$p_v \geq 0, \text{for all } v \in V \quad \Big| \quad p \geq 0.$$

Now we have all the tools necessary to show that the following Greedy Online Algorithm is $\frac{1}{2}$-competitive.

Greedy Online Matching Strategy: At time step t:
Match r_t to any of the unmatched neighbors in the set L.

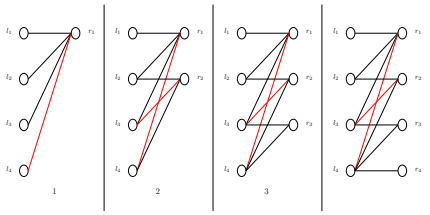


Figure 10.4: Execution of Greedy Online Algorithm. Red edges in greedy matching.

Example 10.1.3 *Consider the graph in Figure 10.4 where* $|L| = |R| = n$. *Assume that* $L = \{l_1, l_2 \ldots, l_n\}$ *and* $R = \{r_1, r_2, \ldots, r_n\}$. *Let* $E = \{(l_i, r_j) | i \geq j, \text{ for all } i, j \in \{1, \ldots, n\}\}$. *It is easy to see that for this graph the maximum value of the objective function of the Primal LP is* n. *For example, we can achieve the optimal value by setting all the variables* x_e*'s*

corresponding to the edges $e = (l_i, r_i)$ to 1, for $i = 1, \ldots, n$, and all other variables to 0. The set of edges $\{(l_1, r_1), (l_2, r_2), \ldots, (l_n, r_n)\}$ forms a perfect matching and it satisfies all the constraints of the LP.

Now consider the execution of the greedy online algorithm where the vertices of the set R and their incident edges come in increasing order of their indices. Assume that the greedy algorithm matches the vertices $r_1, r_2, \ldots, r_{\lfloor \frac{n}{2} \rfloor}$ to $l_n, l_{n-1}, \ldots, l_{\lceil \frac{n}{2} \rceil}$, respectively. But any of the remaining vertices r_j where $j > \lfloor \frac{n}{2} \rfloor$ cannot be matched as there are no free vertices left in L that are adjacent to them. Thus the size of the greedy matching is $\approx \frac{n}{2}$.

We make the following observation.

Lemma 10.1.4 *The matching M computed by the Greedy Online Algorithm in $G = (V = L \cup R, E)$ is maximal.*

For the purpose of analysis, for each vertex $v \in V$ we introduce a quantity q_v that is a real number and it is initialized to zero. Whenever we find an edge $e = (v, w)$ in the matching during the execution of the greedy algorithm, we set $q_v = \frac{1}{2}$ and $q_w = \frac{1}{2}$. Clearly the size of the matching M reported by the greedy algorithm is

$$|M| = \sum_{v \in V} q_v$$

After the execution of the greedy algorithm consider the dual LP where we set $p_v = 2q_v$ for all $v \in V$.

Observe that $p_v \geq 0$ for all $v \in V$. Moreover, for each edge $e = (v, w) \in E$, $p_v + p_w = 2q_v + 2q_w \geq 1$. Otherwise, both the endpoints of e aren't matched. That contradicts the fact that the greedy algorithm computes a maximal matching. The value of the objective function of the dual is given by

$$\sum_{v \in V} p_v = 2 \sum_{v \in V} q_v = 2|M|.$$

Using the fact the the value of the objective function of the Dual LP is an upper bound to the value of the objective function of Primal we obtain that $2|M| \geq |M^*|$, or equivalently $\frac{|M|}{|M^*|} \geq \frac{1}{2}$.

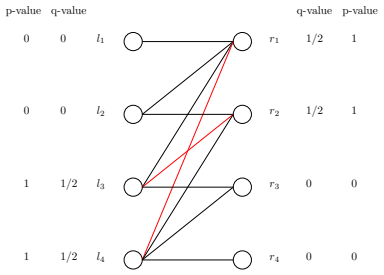


Figure 10.5: Assignment of p and q values to vertices.

10.2 *Fractional Online Bipartite Matching* - WATERLEVEL

In this section we discuss the fractional matching problem on the bipartite graph $G = (V = L \cup R, E)$. The vertices in L are known in advance and each has a unit capacity. The vertices in R come in an online fashion along with its incident edges. Each vertex in R has a unit amount of information to handout. At each time instant t, we need to transmit the information from the current vertex r_t to its neighboring vertices in the set L so that the following conditions are met:

1. Sum total of the information that is transmitted from r_t to its neighbors in L is at most 1.
2. One or more neighbors of r_t may receive the information provided they do not exceed their capacity of 1. I.e. the sum total of the information received by any particular vertex $v \in L$ over the entire execution of the algorithm is at most 1.
3. Once the information is transmitted from r_t to its neighbors it cannot be reversed in the online algorithm.

Let x_e represent the amount of information that travels on an edge $e \in E$. The above two conditions imply that for any vertex $v \in V$, $\mathsf{Level}(v) = \sum_{w \in N(v)} x_{vw} \leq 1$, where $N(v)$ denotes the neighbors of v in G. The objective is to maximize the total information received by the vertices in the set L over the entire execution of the algorithm, or equivalently maximize $\sum_{e \in E} x_e$. Observe that for the (static) graph G the LPs stated in the previous section also apply to this problem formulation as x_e's can take fractional values. Moreover, the value of the objective function is the size of the maximum matching in G.

The following algorithm, so called the WATERLEVEL algorithm that generalizes the greedy matching strategy discussed previously, is proposed for the fractional online bipartite matching. We will show that it is $1 - \frac{1}{e} \approx 0.63$-competitive.

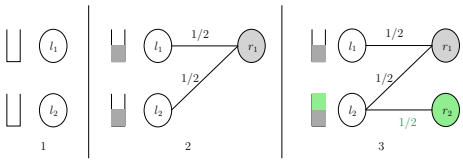


Figure 10.6: Execution of WATERLEVEL Algorithm. Cost of fractional matching is $\frac{3}{2}$, and the competitive ratio is $\frac{3}{4}$.

WATERLEVEL Algorithm

At any time step t:

Drain the water (information) from r_t to its neighbors where the preference is always given to the neighbor with the largest residual capacity remaining till

Case 1: All neighbors of r_t are saturated, or

Case 2: r_t transmits all its information.

More precisely the steps involved in the computation on the arrival of the vertex r_t are as follows:

1. Initialize for all $v \in N(r_t)$, $x_{vr_t} = 0$.
2. Recall that $\mathsf{Level}(v) = \sum_{w \in N(v)} x_{vw}$ denotes the current level of any vertex $v \in L \cup R$. On the arrival of r_t we compute the quantity NLevel given by

$$\sum_{v \in N(r_t)} \max\{\mathsf{NLevel}, \mathsf{Level}(v)\} = 1 + \sum_{v \in N(r_t)} \mathsf{Level}(v)$$

and set $\mathsf{NLevel} = \min\{1, \mathsf{NLevel}\}$.

3. We raise the level of each vertex $v \in N(r_t)$ to NLevel unless it was already above it. I.e., for all $v \in N(r_t)$ set

$$\begin{aligned} x_{vr_t} &= \max\{\mathsf{NLevel}, \mathsf{Level}(v)\} - \mathsf{Level}(v) \\ \mathsf{Level}(v) &= \mathsf{Level}(v) + x_{vr_t} \end{aligned}$$

Claim 10.2.1 *On the completion of the processing for r_t, $\mathsf{Level}(v)$ for all neighbors of r_t is at least* NLevel.

Example 10.2.2 *Consider the execution of the* WATERLEVEL *algorithm on the graph in Figure 10.4. Assume that the vertices in R arrive in the order of increasing indices. Let us walk through the execution of the algorithm on the arrival of each of the vertices in R. Initially, for all $v \in L$, $\mathsf{Level}(v) = 0$ and for all edges $e \in E$, $x_e = 0$.*

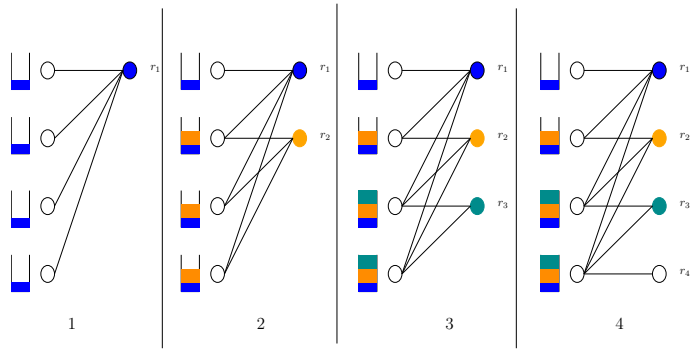


Figure 10.7: Fractional Flow Value $= 4 \times \frac{1}{4} + 3 \times \frac{1}{3} + 2 \times \frac{5}{12} = \frac{17}{6} > 2$. Total flow value received at vertices in L are $\mathsf{Level}(l_1) = \frac{1}{4}$; $\mathsf{Level}(l_2) = \frac{1}{4} + \frac{1}{3} = \frac{7}{12}$; and $\mathsf{Level}(l_3) = \mathsf{Level}(l_4) = \frac{1}{4} + \frac{1}{3} + \frac{5}{12} = 1$.

Computation for r_1: r_1 is adjacent to l_1, l_2, l_3, and l_4 and their current Level*'s are 0. Therefore,* NLevel $= \frac{1}{4}$ *as it satisfies* $\sum_{v \in N(r_1)} \max\{\frac{1}{4}, 0\} = 1 + \sum_{v \in N(r_1)} 0$. *Moreover,* $x_{l_1 r_1} = x_{l_2 r_1} = x_{l_3 r_1} = x_{l_4 r_1} = \frac{1}{4}$ *and* $\mathsf{Level}(l_1) = \mathsf{Level}(l_2) = \mathsf{Level}(l_3) = \mathsf{Level}(l_4) = \frac{1}{4}$.

Computation for r_2: r_2 is adjacent to l_2, l_3, and l_4 and their current Level*'s are $\frac{1}{4}$. Therefore,* NLevel $= \frac{7}{12}$ *as it satisfies* $\sum_{v \in N(r_2)} \max\{\frac{7}{12}, \frac{1}{4}\} = 1 + \sum_{v \in N(r_2)} \frac{1}{4}$. *Moreover,* $x_{l_2 r_2} = x_{l_3 r_2} = x_{l_4 r_2} = \frac{7}{12} - \frac{1}{4} = \frac{1}{3}$ *and* $\mathsf{Level}(l_1) = \frac{1}{4}$, $\mathsf{Level}(l_2) = \mathsf{Level}(l_3) = \mathsf{Level}(l_4) = \frac{7}{12}$.

Computation for r_3: r_3 is adjacent to l_3 and l_4 and their current Level*'s are $\frac{7}{12}$. Since $\frac{13}{12}$ satisfies* $\sum_{v \in N(r_3)} \max\{\frac{13}{12}, \frac{7}{12}\} = 1 + \sum_{v \in N(r_3)} \frac{7}{12}$, NLevel $= \min\{1, \frac{13}{12}\} = 1$. *Moreover,* $x_{l_3 r_3} = x_{l_4 r_3} = 1 - \frac{7}{12} = \frac{5}{12}$ *and* $\mathsf{Level}(l_1) = \frac{1}{4}$, $\mathsf{Level}(l_2) = \frac{7}{12}$, $\mathsf{Level}(l_3) = \mathsf{Level}(l_4) = 1$.

Computation for r_4: r_4 is adjacent only to l_4 and l_4 is already saturated. Thus, NLevel $= \min\{1, 2\} = 1$, *and* $x_{l_4 r_4} = 1 - 1 = 0$. *The algorithm finishes with* $\mathsf{Level}(l_1) = \frac{1}{4}$, $\mathsf{Level}(l_2) = \frac{7}{12}$, $\mathsf{Level}(l_3) = \mathsf{Level}(l_4) = 1$.

Therefore the total weight of the fractional matching is $1+1+\frac{7}{12}+\frac{1}{4} \approx 2.83$ *and the optimal matching is of size* 4. *The competitive ratio of the* WATERLEVEL *algorithm is* $> \frac{1}{2}$ *for this example.*

Example 10.2.3 *Consider the graph* $G = (V = L \cup R, E)$ *of Example 10.1.3 where* $L = \{l_1, l_2, \ldots, l_n\}$, $R = \{r_1, r_2, \ldots, r_n\}$, *and* $E = \{(l_i, r_j) | i \geq j,$ *for all* $i, j \in \{1, \ldots, n\}\}$. *In the previous example, we considered the case when* $n = 4$.

Let us execute the WATERLEVEL *algorithm on* G *where vertices in the set* R *come in an online manner in increasing order of their indices. Let* j *be the first index at which there is no further flow of information from vertices in* $r_t \in R$ *for* $t > j$. *This implies that all the vertices* $l_{j+1}, \ldots, l_n$ *are saturated (i.e. for any* l_i, $i > j$, $\sum_{w \in R} x_{l_i w} = 1$).

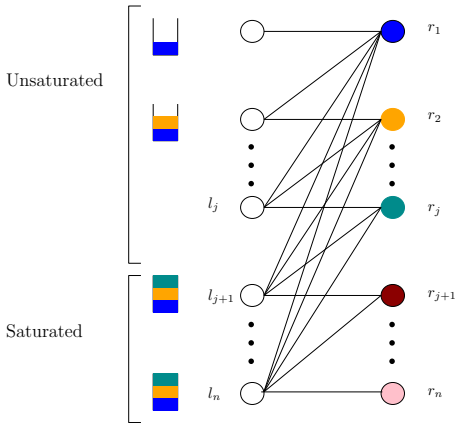
The index j *must satisfy*

$$\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{n-j+1} \geq 1$$

Recall that the n-th Harmonic number $H_n = \sum_{k=1}^{n} \frac{1}{k} \approx \ln n$. *Thus, we want to determine for what value of* j,

$$\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{n-j+1} = H_n - H_{n-j} \approx \ln \frac{n}{n-j} \geq 1.$$

If $j = n(1-\frac{1}{e})$, $\ln \frac{n}{n-j} = \ln \frac{n}{n-n(1-\frac{1}{e})} = \ln e = 1$. *This implies that the vertices* $r_1, \ldots, r_{j-1}$ *are able to send all of their information to vertices of* L, *whereas* $r_{j+1}, \ldots, r_n$ *aren't able to send any. Thus the weight of the fractional matching computed by the* WATERLEVEL *algorithm for this example is* $\approx j \approx n(1-\frac{1}{e})$. *Hence the competitive ratio is* $\approx (1-\frac{1}{e})$ *as* G *has a perfect matching.*



Next, we analyze the WATERLEVEL algorithm using the primal-dual LP framework. LPs are for the competitive analysis only - the algorithm never executes any LP! As remarked earlier the Primal-Dual LPs of the previous section are valid here. In the analysis of the Dual LP we introduced the quantity q_v for each vertex $v \in L \cup R$ and for each edge $e = vw \in E$ that is added to the matching M we assigned $q_v = q_w = \frac{1}{2}$. By setting $p_v = 2q_v$ for all $v \in V$, all the Dual LP constraints were satisfied and the greedy matching algorithm was shown to be $\frac{1}{2}-$competitive.

We can try to mimic the similar idea for the fractional matching analysis as follows. For all $v \in V$, we intialize $q_v = 0$. After the execution of the WATERLEVEL algorithm, for each edge $e = vw \in E$ we set $q_v = q_v + \frac{1}{2}x_{vw}$ and $q_w = q_w + \frac{1}{2}x_{vw}$. We have that the size of the fractional matching $|M| = \sum_{v \in V} q_v$. But to satisfy the Dual LP

constraints we still need to set $p_v = 2q_v$ and this can be justified as follows. Consider the edge l_4r_4 in the graph in Example 10.2.2. All the edges e incident to the vertex r_4 has $x_e = 0$. For any of those edges the sum total of the q values of their end points is at most $\frac{1}{2}$. Therefore, we need an alternate way to devise values for q_v's so that by setting $p_v = cq_v$ for some value $c < 2$ we can satisfy all the Dual LP constraints and obtain a competitive ratio $\frac{1}{c} > \frac{1}{2}$.

The main idea is that instead of splitting the value of the flow x_e on each edge $e = vw \in E$ between its endpoints evenly, split in such a way that $q_v + q_w \geq 1 - \frac{1}{e}$. Thus by setting $p_v = \frac{e}{e-1}q_v$ we can satisfy the constraints of the Dual LP and the resulting competitive ratio will be $\geq 1 - \frac{1}{e}$.

In the WATERLEVEL algorithm when we route the information from the vertex $w = r_t \in R$ to its neighbors $v \in L$, one of the following two scenarios take place.

Case 1: Vertex v after receiving information from w gets saturated. i.e. $\mathsf{Level}(v) = \sum_{z \in R} x_{vz} = 1$.

Case 2: Vertex v didn't get saturated but w runs out of all of its information to be handed out, i.e. $\sum_{v \in L} x_{vw} = 1$.

Consider Case 2. Assume that after the algorithm has terminated, the vertex v isn't saturated. Recall Claim 10.2.1. Let the information content that v has received during the entire execution of the algorithm equals $\mathsf{Level}(v) < 1$. Moreover, assume that $vw \in E$. Now consider the step in the online algorithm when $w \in R$ was revealed. In that step w routed the information to its neighbors (including v) in L whose Level's were at most $\mathsf{Level}(v)$. (This follows from the water-filling analogy since v finished with $\mathsf{Level}(v)$ at the termination and w can only send information to its neighbors up to $\mathsf{Level}(v)$ upon its arrival.)

Let us initialize $q_v = 0$ for all $v \in L$ and $q_w = 0$ for all $w \in R$. Let $f(x) = e^{x-1}$ be defined for $x \in [0, 1]$. Consider the execution of the WATERLEVEL algorithm on $G = (L \cup R, E)$ at the time instance when $w \in R$ appears in the online algorithm. Let $\mathsf{Level}(v) < 1$ for some vertex $v \in L$ before this time instance and v is one of the neighbors of w. Assume that there is a very small amount of information dx that flows from w to v on the edge vw at this time instance. We partition the increase $x_{vw} = dx$ among q_v and q_w by using the function f as follows:

$$q_v = q_v + f(\mathsf{Level}(v))dx \quad \textit{and} \quad q_w = q_w + (1 - f(\mathsf{Level}(v)))dx$$

Observe that the increase in the value of $q_v + q_w$ is dx and if $\mathsf{Level}(v)$ is ≈ 1 then a large proportion of dx is assigned to q_v as $f(\mathsf{Level}(v)) = e^{\mathsf{Level}(v)-1} \approx 1$. This is the main difference between the partitioning

of the increase x_{vw} using the function f as compared to splitting evenly among q_v and q_w.

Let us execute the WATERLEVEL algorithm and on its termination we make a determination of the q values of the vertices of G. Consider any edge $e = vw \in E$ in the final graph. We know that the processing on the arrival of the vertex $w \in R$ resulted in $\mathsf{Level}(v) = \sum_{z \in R} x_{vz} = 1$ (Case 1) or $\sum_{v \in L} x_{vw} = 1$ (Case 2). If both were < 1, then there was no reason for w to not send more information to its unsaturated neighbor v as w is not completely drained out. Next we analyze the sum $q_v + q_w$ for both the cases:

Consider Case 1: After termination we are given that v is saturated, i.e. $\mathsf{Level}(v) = 1$. During the course of the algorithm its Level went from 0 to 1. Thus for the edge vw,

$$q_v + q_w \geq q_v = \int_0^1 f(x)dx = \int_0^1 e^{x-1}dx = 1 - \frac{1}{e}$$

Consider Case 2: We know that w has sent all of its information to its neighbors including v. It is possible that v may or may not be saturated when the algorithm terminated. Suppose $\mathsf{Level}(v) = X$, where $0 \leq X \leq 1$, at the termination of the algorithm. By our observation we know that when w was sending information to its neighbors all of their Level's were at most X. Thus using the fact that f is increasing (therefore, $1 - f$ is decreasing), we get that

$$q_w \geq \int_0^1 (1 - f(X))dx = (1 - e^{X-1}) \int_0^1 dx = 1 - e^{X-1}$$

Thus

$$q_v + q_w \geq \int_0^X f(x)dx + 1 - e^{X-1} = e^{X-1} - \frac{1}{e} + 1 - e^{X-1} = 1 - \frac{1}{e}.$$

Note that in this case the lower bound on the value of $q_v + q_w$ is independent of value of $\mathsf{Level}(v)$ at the termination of the algorithm. It used only the fact that w has sent all of its information to its neighbors. Therefore, in summary, in both the cases we have that for any edge $e = (vw) \in E$, $q_v + q_w \geq 1 - \frac{1}{e}$.

Set $p_v = \frac{e}{e-1} q_v$ for all $v \in L \cup R$. This ensures that the all the constraints of the Dual LP are satisfied. We know that $\sum_{e=vw \in E} (q_v + q_w) = |M|$ and the objective value of the Dual LP is an upper bound to the objective value of the Primal LP. Since the optimal value of the Primal is the size of the optimal fractional matching M^*, we obtain

$$\sum_{e=vw \in E} p_v + p_w = \frac{e}{e-1} \sum_{e=vw \in E} q_v + q_w = \frac{e}{e-1} |M| \geq |M^*|$$

10.3 *Randomized Online Bipartite Matching* - RANKING

As before let the bipartite graph be $G = (V = L \cup R, E)$. The vertices in $L = \{l_1, \ldots, l_n\}$ are known in advance and the vertices in $R = \{r_1, \ldots, r_n\}$ come in an online fashion along with its incident edges in increasing order of their indices. In this section we will discuss the randomized algorithm called RANKING of [92] for matching in G. Now an edge is either in the matching or it isn't. The analysis of RANKING will use the Primal Dual LPs of Section 10.1 and the function $f(x) = e^{x-1}$ of Section 10.2. This analysis is based on the paper by [48]. The RANKING algorithm on the bipartite graph $G = (L \cup R, E)$ is as follows.

RANKING Algorithm

Step 1: For each vertex $v \in L$:
Assign a rank (i.e. a real number) $\mathsf{rank}(v)$ selected uniformly at random from $[0, 1]$.

Step 2: For each vertex $w \in R$ in order of its appearance:
Match w to its lowest ranked unmatched neighbor (if any) in L.

Recall the Primal-Dual LPs of Section 10.1. We will construct a Dual LP solution that is randomized (as before, the Dual is only for the analysis purpose - we only execute the RANKING algorithm). The constraints of the Dual LP may not be satisfied. We will show that they hold in expectation, i.e. $\sum_{e=(v,w)\in E} E[p_v + p_w] \geq 1$. Thus, on expected the value of the dual solution is at least the size of an optimum matching $|M^*|$ in G as the objective value of the Dual LP upper bounds the objective value of the Primal LP (and that equals $|M^*|$).

Bad choice of notation - E stands for edges and Expected Value!

Consider the execution of the RANKING. Let $e = (l_i r_j) \in E$. When the vertex $r_j \in R$ is considered by RANKING it may or may not be matched to $l_i \in L$ as that depends on whether (a) l_i is unmatched at that moment and (b) among all the unmatched neighbors of r_j, $\mathsf{rank}(l_i)$ is the lowest. Consider the set $L' = L \setminus \{l_i\}$ and the graph $G' = (L' \cup R, E')$, where E' is obtained from E by excluding the edges incident on l_i. Assume that when RANKING was executed on G', the ranks assigned to each vertex in L' is the same as the ranks assigned to the full set L. Suppose RANKING when executed on G' matches r_j to $l_{i'} \in L$. Let $\Gamma = \mathsf{rank}(l_{i'})$. (Note: For good reasons if r_j is not matched to any vertex in G', we set $\Gamma = 1$.) Next we state and prove

some claims from [48].

Claim 10.3.1 *If* $\mathsf{rank}(l_i) < \Gamma$, *the vertex* $l_i \in L$ *is matched in the execution of* RANKING *to some vertex of* R.

Proof. If l_i is already matched in G before the vertex r_j is processed by RANKING, then there is nothing to prove. For the rest of the proof we assume that l_i is not matched even after r_j has been processed by RANKING. By the assumption that the ranks of each vertex in L' is the same as that in L it follows that the (partial) matching computed by RANKING in G' and G are identical till the vertex r_j is considered. We know that in G' RANKING matches r_j to $l_{i'}$. This implies that in G, RANKING will match r_j to l_i as $\mathsf{rank}(l_i) < \mathsf{rank}(l_{i'}) = \Gamma$. Hence l_i is matched. ∎

For the rest of this section we fix the rank of each vertex in $L' = L \setminus \{l_i\}$ to be same as the rank of the corresponding vertices in L (as generated by RANKING in Step 1), and we assume that $l_i r_j$ is an edge in G.

Claim 10.3.2 *Let us execute* RANKING *on the graphs* $G' = (L' \cup R, E')$ *and* $G = (L \cup R, E)$ *in parallel. The set of unmatched vertices in* L' *is subset of the set of unmatched vertices in* L *at the start of any step of the algorithm.*

Proof. This is true at the start as the set of matched vertices is empty and $L' \subset L$. Assume that it holds true when RANKING considered the vertices $r_1, r_2, \ldots, r_{j-1}$. Consider the step when RANKING is going to consider r_j. We ask the following question: For two distinct vertices $l_k (\neq l_i)$ and $l_{k'}$ that are among the set of unmatched vertices for both L and L' before r_j was considered, can r_j be matched to l_k in G and to $l_{k'}$ in G' by RANKING? It is easy to see that this cannot occur. Before r_j was considered, $l_{k'}$ and l_k are among the set of unmatched vertices for both L and L'. If $l_{k'}$ is chosen by RANKING in G' as a match for r_j, then $\mathsf{rank}(l_{k'}) < \mathsf{rank}(l_k)$. But for G, as $l_{k'}$ was available as an unmatched vertex when r_j was considered by RANKING, there is no reason to match it to l_k which is a higher ranked vertex than $l_{k'}$. In this step in G either r_j gets matched to l_i or to $l_{k'}$. ∎

As in the previous section, we define q_v and q_w values for each vertex $v \in L$ and $w \in R$. We initialize them to 0. If an edge $e = (vw \in E)$, where $v \in L$ and $w \in R$, is identified to be in the matching by RANKING, we set $q_v = f(\mathsf{rank}(v)) = e^{\mathsf{rank}(v)-1}$ and $q_w = 1 - q_v$. Recall that $\Gamma = \mathsf{rank}(l_{i'})$ is the rank of the vertex $l_{i'} \in L'$ that is matched to w in the graph G'.

Claim 10.3.3 *Let the execution of* RANKING *on* G *matches* $r_j \in R$ *to some vertex* $v \in L$. *Then* $q_{r_j} = 1 - e^{\mathsf{rank}(v)-1} \geq 1 - e^{\Gamma-1}$.

Proof. Consider the step when RANKING considers r_j. As discussed in Claim 10.3.2, before r_j is considered, the set of unmatched vertices in L' is a subset of the set of unmatched vertices in L. This implies that r_j has a unmatched neighbor in G whose rank is at most Γ. Thus r_j will be matched to a vertex $v \in L$ (may be l_i) with a rank at most Γ. Since f is an increasing function (and $1 - f$ is decreasing), $q_{r_j} = 1 - f(\mathsf{rank}(v)) \geq 1 - f(\Gamma)$. ∎

Next we show that by setting $p_v = \frac{e}{e-1} q_v$ for all vertices $v \in L \cup R$, in expectation, all the Dual LP constraints are satisfied. It is obvious that $p_v \geq 0$ for all $v \in \{L \cup R\}$. Now we show that for each edge $e = (vw)$, where $v \in L$ and $w \in R$, $E[p_v + p_w] \geq 1$. There are two cases. Either e is in the matching reported by RANKING or it isn't.

Suppose e is in matching. Then $q_v = e^{\mathsf{rank}(v)-1}$ and $q_w = 1 - q_v$. Then $q_v + q_w = 1$ and therefore $p_v + p_w = \frac{e}{e-1} \geq 1$. Moreover, this also establishes that the competitive ration is $\frac{e}{e-1}$ (in expectation) as the cost of the Dual LP is an upper bound to the cost of the Primal.

Now consider the case where $e = (vw)$ is not in the matching. The analysis is analogous to Case 2 of the WATERLEVEL algorithm. We need to show that $E[p_v + p_w] \geq 1$. Consider the sets L and L' and the parameter Γ used in Claim 10.3.1. Assume $l_i = v$ and $r_j = w$. We know that if $\mathsf{rank}(v) < \Gamma$ then v is matched by RANKING. Therefore $E[q_v] \geq \int_0^\Gamma e^{x-1} dx = e^{\Gamma-1} - \frac{1}{e}$. By Claim 10.3.3 we know that $q_w \geq 1 - e^{\Gamma-1}$. Thus $E[q_v + q_w] = E[q_v] + E[q_w] \geq e^{\Gamma-1} - \frac{1}{e} + 1 - e^{\Gamma-1} = 1 - \frac{1}{e}$. Therefore $E[p_v + p_w] = \frac{e}{e-1} E[q_v + q_w] \geq 1$.

10.4 BALANCE *Algorithm*

In this section we present the BALANCE algorithm by [89] for the online b-matching problem. Its analysis is based on the Adwords paper by [115]. As before consider a bipartite graph $G = (L \cup R, E)$ where the vertices in R come in an online manner along with the edges incident to them. The parameter b is a fixed positive integer. When a vertex $w \in R$ is revealed to the algorithm, our task is to possibly match it one of its neighbors $v \in L$ provided that the number of vertices matched to v so far by the algorithm is $< b$. Whatever decision that we make for w cannot be altered on the arrival of future vertices of R. Note that $b = 1$ corresponds to the classical bipartite online matching problem that was addressed using the RANKING algorithm in the previous section. The BALANCE algorithm is as follows:

BALANCE Algorithm

For each vertex $w \in R$ in order of its appearance:
Among all the neighbors of w in L that have been matched $< b$ times, match w to that neighbor (if any) that is matched to the fewest.

We will show that the competitive ratio of BALANCE is $1 - \frac{1}{e}$ for large values of b. It will be better to think of this problem (termed as the *AdWords* problem [115]) in terms of advertisers and user keyword queries in an online setting. Assume that the set of vertices in $L = \{1, 2, \ldots, N\}$ are advertisers where each of them have a daily budget of \$1. These advertisers bid a small amount $\epsilon > 0$ for a set of keywords of their liking. For example, an advertiser may bid for 'collector coins' and 'hockey cards' whereas another advertiser may bid for 'Hot Sauce'. The set R comprises of keyword queries that arrive in an online manner. Each query keyword needs to be assigned to an advertiser (if any) who has bid for that keyword and has some remaining budget $\geq \epsilon$. If the query is assigned to an advertiser, its budget is decreased by ϵ and we generate a revenue of ϵ. In particular, the BALANCE algorithm assigns the query to the advertiser who has (a) bid for that keyword (b) has remaining budget $\geq \epsilon$, and (c) among all those advertisers has the largest remaining budget. The objective is to maximize the revenue generated by the algorithm, i.e. the sum total of the budget spent by the advertisers.

We assume that the budget of each advertiser is 'quantized' by an integer parameter $k >> 0$. I.e., each advertiser's budget is discretized in k equal slabs, where each slab represents $\frac{1}{k}$-th fraction of the amount. It is assumed that the advertisers spend their budgets in increasing order of their slabs. First from slab 1, followed by slab 2, ... Further assume that an optimal assignment of queries to advertisers consumes all of their budgets and its revenue is $1 * |L| = N$ and each query can be completely paid by the amount within a single slab. (Otherwise, if we assume $\frac{1}{k^2} \geq \epsilon$, we can sacrifice $\frac{1}{k}$ from the revenue of each advertiser to account for the possibility that ϵ may span two consecutive slabs. Note that for an advertiser we may incur a loss of $\leq \frac{1}{k^2}$ per slab and over its k slabs the total loss is at most $\frac{1}{k}$.) We will show that BALANCE achieves a revenue of $\geq (1 - \frac{1}{e})N$ yielding a competitive ratio of $1 - \frac{1}{e}$.

To facilitate our analysis we say an advertiser is of Type i if the fraction of the total amount that it spends during the entire execution of BALANCE is in the range $(\frac{i-1}{k}, \frac{i}{k}]$, where $i \in \{1, \ldots, k\}$. We may assume that if a bidder spends nothing then it is considered to be of Type 1, i.e. the fraction of budget spent by Type 1 advertisers is in

the range $[0, \frac{1}{k}]$. Let us ask ourselves the following question: If in an optimal assignment a query keyword q is assigned to an advertiser of Type i, where $i < k$, then from which slab the revenue with respect to q will be generated by BALANCE ? We answer this question as follows.

We are given that q is assigned to a Type i advertiser in an optimal assignment and its budget isn't completely consumed by BALANCE as $i < k$. In BALANCE q can't be paid by any slab $> i$ since the queries are assigned to potential advertisers who have consumed the smallest amount of their budget. Therefore the contribution to the revenue comes from a slab $\leq i$. We have the following observation.

Observation 10.4.1 *All the query keywords that are assigned by optimal to a Type i advertiser, for some $i < k$, are 'paid' by slabs $\leq i$ in* BALANCE.

Consider the execution of BALANCE. For $i = 1, \ldots, k$, we say x_i represents the numbers of advertisers of Type i. Let β_j represent the total amount spent from slab j of all the advertisers by BALANCE for $j = 1, \ldots, k$. The following observation follows from the definitions and the fact that each advertiser has a budget of \$1 to spend.

Observation 10.4.2 $\beta_1 = \frac{|L|}{k} = \frac{N}{k}$, *and* $\beta_j = \frac{N}{k} - \sum_{i=1}^{j-1} \frac{x_i}{k}$.

Lemma 10.4.3 *For* $1 \leq i \leq k-1$, $\sum_{j=1}^{i} x_j \leq \sum_{j=1}^{i} \beta_j$.

Proof. First consider $i = 1$. We need to show that $x_1 \leq \beta_1$. We know that $\beta_1 = \frac{N}{k}$. All the queries that are assigned to Type 1 advertisers in an optimal assignment need to be paid by slab 1 of the advertisers according to Observation 10.4.1. The total revenue of queries assigned to Type 1 advertisers in an optimal assignment is x_1 (initial budget of \$1 times the number of Type 1 advertisers) and this need to be paid by β_1 (= the total amount in Slab 1). Thus, $x_1 \leq \beta_1$.

Consider $k - 1 \geq i \geq 2$. We need to show that $x_1 + x_2 + \cdots x_i \leq \beta_1 + \beta_2 + \cdots \beta_i$. This follows from the fact that all the queries that are assigned to Types $1, 2, \ldots, i$ advertisers in an optimal assignment need to be paid by Slabs $1, 2, \ldots, i$. ∎

Lemma 10.4.4 *The revenue generated by* BALANCE *is* $\geq N(1 - \frac{1}{k}) - \sum_{i=1}^{k-1} \frac{k-i}{k} x_i$.

Proof. The revenue of BALANCE comes from advertisers of various types. An advertiser of Type i, where $i < k$, generates a revenue of $\frac{i}{k}$. There are x_i such advertisers and thus the total revenue from

Type i advertisers is $\frac{i}{k}x_i$. Also we obtain a revenue of $N - \sum_{i=1}^{k-1} x_i$ from the Type k advertiser. But we may loose a revenue of $\frac{1}{k}$ for each advertiser due to ϵ spanning consecutive slabs. Putting all this together, the revenue of BALANCE is $\geq N - \sum_{i=1}^{k-1} x_i - \frac{N}{k} + \sum_{i=1}^{k-1} \frac{i}{k}x_i = N(1-\frac{1}{k}) - \sum_{i=1}^{k-1} \frac{k-i}{k}x_i$. ∎

Our task is to establish a lower bound on the revenue $N(1-\frac{1}{k}) - \sum_{i=1}^{k-1} \frac{k-i}{k}x_i$. Since the quantity $N(1-\frac{1}{k})$ is fixed, an estimate on the upper bound on $\sum_{i=1}^{k-1} \frac{k-i}{k}x_i$ will help us in deriving bounds for BALANCE. So our task is to solve the following Linear Program:

Primal LP

Maximize $\sum_{i=1}^{k-1} \frac{k-i}{k}x_i$

Subject to:

For all $i \in \{1,\ldots,k-1\}$: $\sum_{j=1}^{i} x_j \leq \sum_{j=1}^{i} \beta_j$ (Lemma 10.4.3)

For all $i \in \{1,\ldots,k\}$: $x_i \geq 0$

Observe that the condition $\sum_{j=1}^{i} x_j \leq \sum_{j=1}^{i} \beta_j$ can be expressed as follows using Lemma 10.4.3 and Observation 10.4.2:

$$\begin{aligned}
\sum_{j=1}^{i} x_j &\leq \sum_{j=1}^{i} \beta_j \\
&\leq \sum_{j=1}^{i} \left(\frac{N}{k} - \sum_{l=1}^{j-1} \frac{x_l}{k}\right) \\
&= \frac{i}{k}N - \sum_{j=1}^{i}\sum_{l=1}^{j-1} \frac{x_l}{k} \\
&= \frac{i}{k}N - \sum_{j=1}^{i} \frac{i-j}{k}x_j
\end{aligned}$$

Equivalently,

$$\sum_{j=1}^{i}\left(1+\frac{i-j}{k}\right)x_j \leq \frac{i}{k}N$$

Thus, we can express the Primal LP as follows:

Primal LP

Maximize $\sum_{i=1}^{k-1} \frac{k-i}{k} x_i$

Subject to:

For all $i \in \{1, \ldots, k-1\}$: $\sum_{j=1}^{i} (1 + \frac{i-j}{k}) x_j \leq \frac{i}{k} N$

For all $i \in \{1, \ldots, k\} : x_i \geq 0$

The Primal LP is of the form $\max c \cdot x$, where $Ax \leq b$ and $x \geq 0$. Its Dual LP will be of the form, $\min b \cdot y$, where $A^T y \geq c$ and $y \geq 0$. More precisely, the corresponding Dual LP is:

Dual LP

Minimize $\sum_{i=1}^{k-1} (\frac{i}{k} N) y_i$

Subject to:

For all $i \in \{1, \ldots, k-1\}$: $\sum_{j=i}^{k-1} (1 + \frac{j-i}{k}) y_j \geq \frac{k-i}{k}$

For all $i \in \{1, \ldots, k-1\} : y_i \geq 0$

For example, consider the Dual LP constraint with respect to the Primal LP variable x_1. We will need that $y_1 + y_2(1 + \frac{1}{k}) + y_3(1 + \frac{2}{k}) + \cdots + y_{k-1}(1 + \frac{k-2}{k}) \geq \frac{k-1}{k}$. This can be expressed as $\sum_{j=1}^{k-1} (1 + \frac{j-1}{k}) y_j \geq \frac{k-1}{k}$. In general, for the i-th variable x_i, we have the Dual LP constraint $\sum_{j=i}^{k-1} (1 + \frac{j-i}{k}) y_j \geq \frac{k-i}{k}$. We will consider a feasible solution for both Primal and Dual LP and show that it is also optimal using complementary slackness. It states that if we have feasible solutions x and y to Primal and Dual LP's respectively and if certain equations are satisfied then they are also the optimal. To motivate this, we will look at an example, and then get back to the competitive ratio of BALANCE.

Complementary Slackness

Consider the following Primal LP:

$$\begin{aligned} \text{Maximize } & x_1 + x_2 + x_3 \\ 2x_1 + 3x_2 + x_3 &\leq 6 \\ x_1 + x_2 - 7x_3 &\leq 4 \\ 3x_1 - x_2 + 5x_3 &\leq 10 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

A feasible solution for Primal LP is $x = (0, \frac{5}{4}, \frac{9}{4})$ giving the objective value of $\frac{7}{2}$. Its Dual LP is:

$$\begin{aligned} \text{Minimize } & 6y_1 + 4y_2 + 10y_3 \\ 2y_1 + y_2 + 3y_3 &\geq 1 \\ 3y_1 + y_2 - y_3 &\geq 1 \\ y_1 - 7y_2 + 5y_3 &\geq 1 \\ y_1, y_2, y_3 &\geq 0 \end{aligned}$$

A feasible solution for Dual LP is $y = (\frac{3}{8}, 0, \frac{1}{8})$ giving an objective value of $\frac{7}{2}$.

Complementary Slackness conditions state that if feasible solutions x and y to Primal LP ($\max cx, Ax \leq b, x \geq 0$) and Dual LP ($\min by, A^T y \geq c, y \geq 0$) satisfy $\forall i : (b_i - \sum_j a_{ij} x_j) y_i = 0$ and $\forall j : (\sum_i a_{ij} y_j - c_j) x_j = 0$ then they are also optimal:

Substitute the Primal LP's feasible assignment $x = (0, \frac{5}{4}, \frac{9}{4})$ into its constraints. We observe that the inequalities 1 and 3 are tight as $2x_1 + 3x_2 + x_3 = 6$ and $3x_1 - x_2 + 5x_3 = 10$, whereas there is a slack in the inequality 2 as $x_1 + x_2 - 7x_3 < 4$. Since $y_2 = 0$, $(b_2 - \sum_j a_{2j} x_j) y_2 = (4 - (x_1 + x_2 - 7x_3)) y_2 = 0$. Similarly, for a feasible $y = (\frac{3}{8}, 0, \frac{1}{8})$ for the Dual LP, inequalities 2 and 3 are tight, but the inequality 1 has a slack as $2y_1 + y_2 + 3y_3 > 1$. As $x_1 = 0$, $((2y_1 + y_2 + 3y_3) - 1) x_1 = 0$. As both $x = (0, \frac{5}{4}, \frac{9}{4})$ and $y = (\frac{3}{8}, 0, \frac{1}{8})$ are feasible and satisfy the complementary slackness conditions, they are optimal.

Let us consider an assignment to variables x_j that makes the following constraints of Primal LP feasible. For all $i \in \{1, \ldots, k-1\}$: $\sum_{j=1}^{i} (1 + \frac{i-j}{k}) x_j \leq \frac{i}{k} N$ and for all $i \in \{1, \ldots, k\} : x_i \geq 0$. We set $x_1 = \frac{N}{k}$, $x_2 = \frac{N}{k}(1 - \frac{1}{k})$, $x_3 = \frac{N}{k}(1 - \frac{1}{k})^2, \ldots, x_i = \frac{N}{k}(1 - \frac{1}{k})^{i-1}, \ldots,$ $x_k = \frac{N}{k}(1 - \frac{1}{k})^{k-1}$. These are derived by setting $\sum_{j=1}^{i} (1 + \frac{i-j}{k}) x_j = \frac{i}{k} N$ and solving for x_i for $i = 1, 2, \ldots, k-1$. Moreover, each $x_i \geq 0$. Thus

the assignment $x_i = \frac{N}{k}(1-\frac{1}{k})^{i-1}$ is a feasible solution for Primal LP.

Now consider the Dual LP constraints and try to find a feasible solution. For all $i \in \{1,\ldots,k-1\}$: $\sum_{j=i}^{k-1}(1+\frac{j-i}{k})y_j \geq \frac{k-i}{k}$ and $i \in \{1,\ldots,k-1\} : y_i \geq 0$. Again we solve for y_i's by setting $\sum_{j=i}^{k-1}(1+\frac{j-i}{k})y_j = \frac{k-i}{k}$. We obtain $y_{k-1} = \frac{1}{k}$, $y_{k-2} = \frac{1}{k}(1-\frac{1}{k})$, $y_{k-3} = \frac{1}{k}(1-\frac{1}{k})^2$, $\ldots$, $y_{k-i} = \frac{1}{k}(1-\frac{1}{k})^{i-1}$, $\ldots$, $y_1 = \frac{1}{k}(1-\frac{1}{k})^{k-2}$. All y_i's are feasible and are ≥ 0.

In the above assignment of x and y, all the Primal and Dual constraints are satisfied. Since all the inequalities are equalities, there is no slack, and thus the complementary slackness conditions hold. This implies that not only x and y are feasible, but they are also optimal solutions for Primal and Dual LPs. Let us evaluate the value of the objective function by substituting the value of x (or y) in the Primal (respectively, Dual) LP.

$$\begin{aligned}
\sum_{i=1}^{k-1}\left(\frac{k-i}{k}\right)x_i &= \sum_{i=1}^{k-1}\left(\frac{k-i}{k}\right)\left(\frac{N}{k}\right)\left(1-\frac{1}{k}\right)^{i-1} \\
&= \frac{N}{k^2}\left[\sum_{i=1}^{k-1}k\left(1-\frac{1}{k}\right)^{i-1} - \sum_{i=1}^{k-1}i\left(1-\frac{1}{k}\right)^{i-1}\right] \\
&= \frac{N}{k^2}\left[k\left(\frac{1-\left(1-\frac{1}{k}\right)^{k-1}}{1-\left(1-\frac{1}{k}\right)}\right) - \frac{k^2}{k-1}\left(\left(1-\frac{1}{k}\right)^{k} - k\left(2\left(1-\frac{1}{k}\right)^{k}-1\right)-1\right)\right] \\
&= \frac{N}{k^2}\left[k^2\left(1-\left(1-\frac{1}{k}\right)^{k-1}\right) - \frac{k^2}{k-1}\left((1-2k)\left(1-\frac{1}{k}\right)^{k}+k-1\right)\right] \\
&= N\left[\frac{(k-1)\left(1-\left(1-\frac{1}{k}\right)^{k-1}\right) - (1-2k)\left(1-\frac{1}{k}\right)^{k} - k + 1}{k-1}\right] \\
&= N\left[\frac{-(k-1)\left(1-\frac{1}{k}\right)^{k-1} - (1-2k)\left(1-\frac{1}{k}\right)^{k}}{k-1}\right] \\
&= N\left[\frac{-k\left(1-\frac{1}{k}\right)^{k} - (1-2k)\left(1-\frac{1}{k}\right)^{k}}{k-1}\right] \\
&= N\left(1-\frac{1}{k}\right)^{k}
\end{aligned}$$

As $k \to \infty$, $\left(1-\frac{1}{k}\right)^{k} \to \frac{1}{e}$. This implies that the upper bound on the value of $\sum_{i=1}^{k-1}\frac{k-i}{k}x_i = \frac{N}{e}$. Therefore the revenue of BALANCE by Lemma

10.4.4 is atleast $N(1-\frac{1}{k}) - \sum_{i=1}^{k-1} \frac{k-i}{k} x_i \geq N(1-\frac{1}{k}) - \frac{N}{e} \approx N(1-\frac{1}{e})$ for large values of k.

10.4.1 A Lower Bound Example

Let the set L has N vertices (advertisers) where budget of each of them is \$1. There are a total of N keywords $K_1, \ldots, K_N$ and the advertiser i bids for the keywords $\{1, \ldots, i\}$. Assume $\epsilon = \frac{1}{N}$. Each advertiser can pay for at most N queries. The online query sequence consists of N^2 queries, where the first N queries are for the keyword K_1, next N queries are for the keyword K_2, ..., and last N queries are for the keyword K_N. An optimal solution assigns N queries of type K_i to the advertise i, for $i = 1, \ldots, N$ generating a total revenue of N. This is maximum possible and each advertisers budget is completely exhausted. Let us see how BALANCE will assign these queries and what will be its revenue? The first N queries corresponding to the keyword K_1 will be distributed evenly among all the advertisers. The next N queries corresponding to the keyword K_2 will be distributed among the advertisers $2, \ldots, N$, as advertiser 1 doesn't bid for K_2. The 2nd advertiser is assigned two queries of Type K_2 and the advertisers 3 to N will get one query each. (We can assume that if two advertisers have the same remaining budget and bid for the same keyword, the query on that keyword will be assigned to the advertiser with lower vertex number.) In general, N queries for the keyword K_i will be distributed evenly among advertisers $i, \ldots, n$ provided that they have sufficient remaining budget. Observe that in this scheme, the first advertiser only receives $1(k_1)$ query, the second advertiser receives $3(1k_1 + 2k_2)$ queries, third advertiser receives $4(1k_1 + 1k_2 + 2K_3)$ queries, ... For an illustration see Figure 10.8. We want to estimate the revenue of BALANCE.

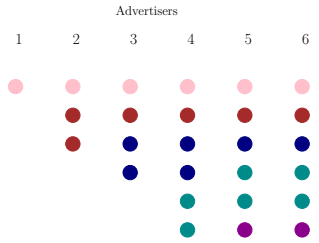


Figure 10.8: BALANCE with 6 advertisers numbered 1 to 6. Each has a budget of \$1 and can pay for 6 queries. Advertiser i bids for keywords $\{K_1, \ldots, K_i\}$. Thirty-six online queries arrive: first 6 for K_1 (pink dots), followed by next 6 for K_2 (dark red),... Revenue of BALANCE is 26 whereas optimal revenue is 36.

Consider the set of queries assigned to the advertiser N. It receives at least one query of type K_1, at least one query of type K_2, and in general at least $\lfloor \frac{N}{N-i} \rfloor$ queries of type K_i. The maximum number of queries that it can receive is at most N. We want to find the maximum index i such that some queries of type K_i can be sent to the advertiser N. We can estimate the value of i by using the following inequalities:

$$\begin{aligned} N &\leq \left\lfloor \frac{N}{N} \right\rfloor + \left\lfloor \frac{N}{N-1} \right\rfloor + \cdots + \left\lfloor \frac{N}{N-i} \right\rfloor \\ &\leq \frac{N}{N} + \frac{N}{N-1} + \cdots + \frac{N}{N-i} \\ &= N\left(\frac{1}{N} + \frac{1}{N-1} + \cdots + \frac{1}{N-i}\right) \end{aligned}$$

Note that the n-th Harmonic number $H_n = \sum_{i=1}^{n} \frac{1}{i} \approx \ln n$. Therefore the above inequality reduces to $H_n - H_{n-i} \geq 1$. Equivalently $\ln \frac{n}{n-i} \geq 1$, or $\frac{n}{n-i} \geq e$. On simplifying we obtain that $i \geq n - \frac{n}{e}$. This implies that BALANCE can only handle queries with respect to keywords K_1, K_2, $\ldots$, $K_{N-\frac{N}{e}}$. Hence the maximum revenue that it can get on this query sequence is $N(1 - \frac{1}{e})$ and that results in a competitive ratio of $1 - \frac{1}{e}$. Therefore BALANCE is an optimal online algorithm for the b-matching problem.

10.5 Exercises

10.1 *Show that any online deterministic algorithm for the fractional bipartite matching cannot have a competitive ratio better than $1 - \frac{1}{e}$ on the graph in Example 10.1.3.*

10.2 *Consider the following linear program.*

$$\begin{aligned} \min x_1 + x_2 \\ x_1 + 2x_2 \geq 4 \\ 2x_1 + x_2 \geq 6 \\ x_1, x_2 \geq 0 \end{aligned}$$

Find the best lower bound for the minimum value of the objective function by taking linear combinations of the constraints. Express this as a Dual LP and what values of x_1 and x_2 satisfy the constraints and minimize the objective value?

10.3 *Suppose $x \in \Re^n$ is a feasible solution for the Primal LP ($\max cx, Ax \leq b, x \geq 0$) and let $y \in \Re^m$ be a feasible solution for the Dual LP ($\min by, A^T y \geq c, y \geq 0$), where A is a $m \times n$ matrix.*

1. *Show that $\sum_{j=1}^{n} c_j x_j \leq \sum_{i=1}^{m} b_i y_i$. (Hint: Use the fact that y is feasible for the Dual LP and x are non-negative to show that $\sum_{j=1}^{n} c_j x_j \leq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} A_{ij} y_i \right) x_j$. Similarly $\sum_{i=1}^{m} b_i y_i \geq \sum_{i=1}^{m} \left(\sum_{j=1}^{n} A_{ij} x_j \right) y_i$. Observe that $\sum_{j=1}^{n} \left(\sum_{i=1}^{m} A_{ij} y_i \right) x_j = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} A_{ij} x_j \right) y_i$.) This is the weak-duality.*
2. *Show that x and y are optimal if and only if the complementary slackness conditions $\forall i : \left(b_i - \sum_j a_{ij} x_j \right) y_i = 0$ and $\forall j : \left(\sum_i a_{ij} y_j - c_j \right) x_j = 0$ are satisfied. (Hint: Consider $\sum_i \left[\left(b_i - \sum_j a_{ij} x_j \right) y_i \right] + \sum_j \left[\left(\sum_i a_{ij} y_j - c_j \right) x_j \right]$*

and use the fact that the optimal cost for Primal and Dual is the same when both of them have feasible solutions.)

10.4 *Show that* $\sum_{i=1}^{k-1} i\left(1-\frac{1}{k}\right)^{i-1} = \frac{k^2}{k-1}\left(\left(1-\frac{1}{k}\right)^k - k\left(2\left(1-\frac{1}{k}\right)^k - 1\right) - 1\right)$. *(Hint: Show that* $\sum_{i=1}^{j} i\left(1-\frac{1}{k}\right)^{i-1} = k\left(k - \left(1-\frac{1}{k}\right)^j (j+k)\right)$ *by induction on* j *and then substitute* $j = k-1$.

10.5 *Consider the following bipartite graph* $G = (V = L \cup R, E)$ *where* $L = \{l_1, \ldots, l_n\}$, $R = \{r_1, \ldots, r_n\}$, *and* $E = \{(l_i, r_i) | 1 \leq i \leq n\} \cup \{(l_i, r_j) | \frac{n}{2} + 1 \leq i \leq n \text{ and } 1 \leq j \leq \frac{n}{2}\}$. *Assume that the vertices in* L *are known in advance and the vertices in* R *come in increasing order of their indices. The online algorithm (called* GREEDY RANDOM *in [92]) matches the next vertex* $r_j \in R$ *to any of its unmatched neighbors in* L *(if there is any) uniformly at random. Show that the expected size of the matching computed by* GREEDY RANDOM *is* $\frac{n}{2} + \log n$. *(Hint: For* $1 \leq j \leq \frac{n}{2}$, *show that with probability at most* $\frac{1}{\frac{n}{2}-j+1}$ *the vertex* r_j *will be matched to* l_j.*)*

10.6 *Show that for* $\eta \in [0, \frac{1}{2}]$, $-\eta - \eta^2 \leq \ln(1-\eta) \leq -\eta$.
Hint: Recall that $\ln(1-x) = -\sum_{k=1}^{\infty} \frac{(-1)^k(-x)^k}{k}$ *for* $|x| < 1$.

10.7 *Show that for* $\epsilon \in [0,1]$: *(a)* $(1-\epsilon)^x \leq 1 - \epsilon x$ *if* $x \in [0,1]$ *and (b)* $(1+\epsilon)^{-x} \leq 1 - \epsilon x$ *if* $x \in [-1,0]$.

10.8 *Let* $G = (V, E)$ *be a simple undirected graph. A subset* $S \subseteq V$ *is said to* hit E, *if for each edge* $e = (u,v) \in E$, $u \in S$ *or* $v \in S$. *A set* $S^* \subseteq V$ *is said to be an* optimal hitting set *if it has the smallest cardinality among all hitting sets of* G.

The graph G *is presented in an online manner as follows. We know all its vertices* $V = \{1, \ldots, n\}$ *in advance, and the edge set is empty. At each time instance* $t = 1, \ldots, |E|$, *an edge* e_t *between a pair of vertices is added to* G. *Thus, at each time instance* t, *we have the graph* $G_t = (V, E_t = \bigcup_{i=1}^{t} e_i)$.

We need to devise an online algorithm that maintains a hitting set $S_t \subseteq V$ *of* G_t, *for* $t = 1, \ldots, |E|$, *whose cardinality is at most two times the cardinality of an optimal (offline) hitting set. I.e., if* S_t^* *is an optimal (offline) hitting set of* G_t, $|S_t| \leq 2|S_t^*|$, *for* $t = 1, \ldots, |E|$. *Explain your online algorithm and analyze its competitive ratio.*
Note: This is an online algorithm. Therefore, whichever vertices are added to S_t, *at time* t, *will remain in the hitting set till the end of the algorithm. I.e.* $S_1 \subseteq S_2 \subseteq \cdots \subseteq S_{|E|}$.
Hint: When the edge $e_t = (uv)$ *arrives and it isn't hit with the current set of vertices in* S_{t-1}, *think of setting* $S_t = S_{t-1} \cup \{u, v\}$.

The following exercises will develop an offline algorithm for the minimum cost perfect matching in weighted bipartite graphs. These

are based on lecture notes of Tim Roughgarden. Let $G = (V = L \cup R, E)$ be a bipartite graph, $L = \{1, \ldots, n\}$ and $R = \{1, \ldots, n\}$, and for each edge $e = (i, j)$, $c_e \geq 0$ is it costs, where $i \in L$ and $j \in R$. We are interested in finding a perfect matching in G whose cost is minimum. The cost of a matching M is the sum total of the costs of the edges forming M. We can assume that G is a complete bipartite graph as we can always add the missing edges with cost $+\infty$.

First, we define the concept of cycles with respect to a given matching M. Let M be a matching in G, and let C be a cycle in G, such that every alternate edge of C is in M. We call such cycles *M-alternating cycles*. We say that an M-alternating cycle C is a *negative cycle* if the total cost of edges in $C \cap M$ is more than the cost of edges in $C \setminus M$.

10.9 *Answer the following:*

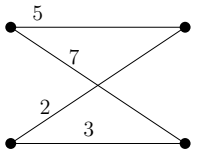
1. *Show that if a matching M contains an M-alternating negative cycle C in G, it can't be a minimum cost perfect matching.*
2. *Assume that M is a perfect matching in G and contains no negative M-alternating cycles. Let $M' \neq M$ be any other perfect matching in G. Answer the following*
   (a) *Consider the symmetric difference between M and M'. It consists of edges present in one but not the other, i.e. $F = M \oplus M' = (M \setminus M') \cup (M' \setminus M)$. Show that degree of each vertex in F is either 0 or 2.*
   (b) *Show that F is a collection of vertex disjoint cycles.*
   (c) *Show that each cycle in F is an M-alternating cycle (and also an M'-alternating cycle).*
   (d) *Show that cost of M' is at least the cost of M.*
3. *Conclude that M is a min-cost perfect matching if and only if G doesn't contain an M-alternating negative cycle.*

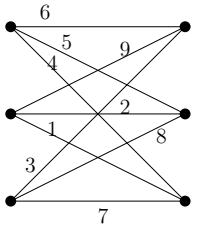
We define reduced cost of each edge $e = (i, j)$ in G to be $w(e) = c(e) - p_i - p_j$, where p_i and p_j are real numbers associated to vertices i and j, respectively. The quantity p_v associated to each vertex v is referred to as its price. An edge e of G is said to be *tight* if $w(e) = 0$. During the course of the algorithm we maintain the following invariant:

Invariant:

Non-negative Reduced Costs: For each edge $e \in E$, $w(e) \geq 0$.

Tightness: For each edge e in the (partial) matching M computed at any step of the algorithm, $w(e) = 0$.





10.10 *For the graphs in the margin, find min-cost perfect matchings. Furthermore, find an assignment of prices to each vertex so that both the invariants are satisfied.*

10.11 *Assume that M is a perfect matching that satisfies the invariant. Prove that M is a min-cost perfect matching. (Hint: Show that all M-alternating cycles in G are non-negative.)*

Assume that M is not a perfect matching (i.e., (i.e., $|M| < n$), but it satisfies the invariant (all edges in G have non-negative reduced weights, and all edges in M are tight). Consider a path Π in G that satisfies the following conditions:

1. All edges on the path are tight.
2. The number of edges on the path is odd.
3. The path starts at an unmatched vertex in L and ends at an unmatched vertex in R.
4. Alternate edges on the path are from M.

Such alternating paths, that starts and ends at an unmatched vertex, and have alternate edges from matching and not from matching are usually referred to as *augmenting paths*.

10.12 *Answer the following*

1. *Show that $M' = M \oplus \Pi$ is a matching of size $|M| + 1$ in G.*
2. *Show that if we replace M by M', the invariant still holds.*

To find such a path Π, we perform a breadth-first search starting at an unmatched vertex v in L, where the search is restricted among tight edges. Let us assume v is at Level 0 in BFS-tree. Vertices in Level 1 will be all the vertices $w \in R$ that are adjacent to v, and vw is a tight edge. Vertices in Level 2 are all the vertices $u \in L$ that are adjacent to Level 1 vertices w, where $uw \in M$. Now, vertices in Level 3 are all the vertices in $x \in R$ that are adjacent to some Level 2 vertex u and ux is a tight edge. We repeat this process and stop when an unmatched vertex in R is encountered, or we cannot make any progress in BFS. If we find an unmatched vertex $r \in R$ at an odd level, then Π is the path starting at v and terminating at r, following the breadth-first tree level-by-level.

10.13 *Perform a BFS traversal on the tight edges in the figure in the margin starting at the unmatched vertex l_1 in L to find a path Π terminating at the unmatched vertex r_3 in R. Once a path Π is found, construct $M' = M \oplus \Pi$. Now in the resulting graph, construct a BFS traversal starting at the unmatched vertex l_3 with respect to M'. Does the traversal gets stuck?*

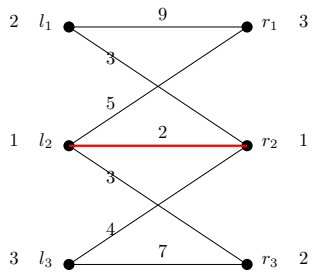


Figure 10.9: $M = \{l_2r_2\}$. Tight edges are $l_1r_2, l_2r_2, l_2r_3, l_3r_2$. Prices for vertices are listed next to the vertex.

Assume that the BFS traversal gets stuck (i.e. doesn't reach an unmatched vertex in R) with respect to a matching M and assume that M is not a perfect matching. Note that the graph G with respect to M satisfies the invariant. Define $S \subseteq L$ to be the set of vertices that are at the even level, and let $N(S) \subset R$ to be the collection of vertices such that for each vertex $r \in N(S)$, there exists some vertex $l \in S$, such that lr is a tight edge.

10.14 *Show that* $|S| > |N(S)|$.

Consider all the vertices in $R \setminus N(S)$ that are adjacent to vertices in S, and define $\Delta = \min_{\{lr | l \in S, r \in R \setminus N(R)\}} w(lr)$.

10.15 *Show that for each edge e with one end-point in S and the other end-point in $R \setminus N(S)$, $w(e) > 0$.*

10.16 *Show that $\Delta > 0$.*

10.17 *Suppose we adjust the price of each vertex in $l \in S$ by $p_l \leftarrow p_l + \Delta$, and each vertex $r \in N(S)$ by $p_r \leftarrow p_r - \Delta$. For all the remaining vertices the prices are left unchanged. Show that the new prices satisfy the invariant. (Hint: Consider any edge $e = lr$ in the graph. There are four cases depending on whether $l \in S, l \notin S, r \in N(S), r \notin N(S)$. Argue that in each of the cases the invariant is maintained after the price adjustments.)*

10.18 *Show that after the price adjustment as above, at least one of the edges that wasn't tight earlier has become tight. (Hint: Think about the edge that defined Δ.)*

10.19 *Perform the price update in Exercise 10.13 with respect to matching M'.*

10.20 *Answer the following:*

1. *When we are stuck in a BFS traversal, we add a new edge in the collection of tight edges (we may lose some). In the new subgraph on tight edges, we execute the BFS procedure again, and we may be stuck again. Show that among $|V|$ successive BFS traversals, there is always a traversal where we find the path Π, i.e., the traversal doesn't get stuck.*
2. *Show that by executing the BFS procedure at most $O(|V|^2)$ times, we can find the min-cost perfect matching in G.*
3. *Show that in $O(|V|^4)$ time we can find the minimum-cost perfect matching in G.*

The above algorithm is called the Hungarian algorithm for min-cost matching in a bipartite graph. Next, we establish the primal-dual

linear program for the min-cost bipartite matching problem and show that the prices p_v associated to vertices $v \in L \cup R$ are the variables in dual LP. The invariant in the Hungarian algorithm ensures that the dual solution is feasible. Moreover, the objective function in the dual provides a lower bound to the cost of any perfect matching in G. The requirement on the tight edges will correspond to maintaining the complementary slackness conditions in the Primal-Dual LP framework.

For each edge $e \in E$, let x_e be an indicator variable defining the presence of e in a matching M, i.e., $x_e = \begin{cases} 1, \text{ if } e \in M \\ 0, \text{ otherwise} \end{cases}$

10.21 *Primal LP: Show that the following LP corresponds to the min-cost matching problem in bipartite graphs.*

Objective function: $\min \sum_{e \in E} c_e x_e$
Subject to: $\sum_{e \in Adj(v)} x_e = 1$, *for all* $v \in L \cup R$, $x_e \geq 0$, *for all* $e \in E$.

(Note: x_e can possibly take any positive fractional values, though it turns that either $x_e = 0$ or $x_e = 1$.)

10.22 *Suppose the prices on vertices satisfy the invariant ($w_e = c_e - p_i - p_j \geq 0$, where e=(ij)). Then for any perfect matching M, show that the following holds*

$$\sum_{(i,j) \in M} c_{ij} \geq \sum_{v \in L \cup R} p_v$$

10.23 *Conclude that $\sum_{v \in L \cup R} p_v$ is a lower bound on the cost of any perfect matching M in G.*

10.24 *Dual LP: Show that the following LP gives the best lower bound on the min-cost perfect matching.*

Objective function: $\max \sum_{v \in L \cup R} p_v$
Subject to: For all edges $(ij) \in E : p_i + p_j \leq c_{ij}$,
and for all $v \in \{L \cup R\} : p_v \in \Re$

Recall that in the Hungarian algorithm, we explicitly enforced the invariant $w_{ij} = c_{ij} - p_i - p_j \geq 0$ for all the edges $(ij) \in E$, and this ensures that the dual solution is feasible.

10.25 *Show that Primal-Dual LP for the min-cost bipartite matching satisfies the complementary slackness conditions. (Hint: Observe that the inequality in Primal LP is equality. In the Hungarian algorithm, all the edges $e = (ij)$ in the matching M are tight, thus $c_{ij} = p_i + p_j$.)*

10.26 *Conclude that the min-cost perfect matching M reported by the Hungarian algorithm satisfies,*

$$\sum_{(i,j)\in M} c_{ij} = \sum_{v\in L\cup R} p_v$$

The following set of exercises will help us design an $O(|E|\sqrt{|V|})$ algorithm for computing a maximum cardinality matching in an unweighted bipartite graph $G = (V = L \cup R, E)$. The algorithm is due to Hopcroft and Karp [1].

[1] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973

Let M be a matching in G. Recall that a simple path Π (without repeated vertices) is an augmenting path with respect to M if the first and the last vertex of the path are unmatched vertices and the edges in Π alternate between $E \setminus M$ and M. Let M_1 and M_2 be two matchings in G. The symmetric difference between M_1 and M_2 is given by $M_1 \oplus M_2 = (M_1 \setminus M_2) \cup (M_2 \setminus M_1)$.

10.27 *Let M be a matching in G and let Π be an augmenting path in G with respect to M. Show that $M' = \Pi \oplus M$ is a matching in G and $|M'| = |M| + 1$.*

10.28 *Let M_1 and M_2 be two matching in G, where $|M_1| > |M_2|$. Show the following*

1. *Consider the graph $G' = (V, M_1 \oplus M_2)$. Show that each vertex in G' has degree at most 2.*
2. *Show that each component of G' is either (a) an isolated vertex, (b) a cycle where edges alternate between $M_1 \setminus M_2$ and $M_2 \setminus M_1$, or (c) a path where edges alternate between $M_1 \setminus M_2$ and $M_2 \setminus M_1$.*
3. *Define the surplus of a component $C_i = (V_i, E_i)$ of G', $\delta(C_i) = |E_i \cap M_1| - |E_i \cap M_2|$, i.e., the difference in the number of edges in E_i that are from M_1 with respect to the number of edges from M_2. Show that $\delta(C_i) \in \{-1, 0, +1\}$.*
4. *Show that $\sum_i \delta(C_i) = |M_1| - |M_2|$.*
5. *Show that each component C_i of G' for which $\delta(C_i) = +1$ is an augmenting path with respect to M_2.*
6. *Show that G' has at least $|M_1| - |M_2|$ vertex disjoint augmenting paths with respect to M_2.*

Let M^* be a maximum cardinality matching in G. Let M be a matching in G and assume that $|M| < |M^*|$.

10.29 *Show that there exists an augmenting path of length at most $1 + 2\left\lfloor \frac{|M|}{|M^*|-|M|} \right\rfloor$ in G with respect to M. (Hint: How many vertex disjoint*

augmenting paths are there in $M^* \oplus M$ with respect to M? Use the pigeonhole principle to show that at least one of these augmenting paths has at most $\left\lfloor \frac{|M|}{|M^*|-|M|} \right\rfloor$ edges from M.)

10.30 *Let M be a matching in G and let Π be a shortest (i.e. minimum number of edges) augmenting path in G with respect to M. Let Π' be an augmenting path in G with respect to $M \oplus \Pi$. Answer the following:*

1. *Let $N = (M \oplus \Pi) \oplus \Pi'$. Show that N is a matching in G consisting of $|M| + 2$ edges.*
2. *Show that $N \oplus M$ contains two augmenting paths Π_1 and Π_2 with respect to M.*
3. *Show that $|N \oplus M| = |\Pi \oplus \Pi'| \geq |\Pi_1| + |\Pi_2|$.*
4. *Show that $|\Pi_1| \geq |\Pi|$ and $|\Pi_2| \geq |\Pi|$, and conclude that $|\Pi \oplus \Pi'| \geq 2|\Pi|$.*
5. *Show that $|\Pi'| \geq |\Pi| + |\Pi \cap \Pi'|$ by using the fact that $|\Pi \oplus \Pi'| = |\Pi| + |\Pi'| - |\Pi \cap \Pi'|$.*

Consider the following incremental procedure. Let $M_0 = \emptyset$. In each successive step construct larger matchings $M_1, M_2, \ldots$ as follows. Let Π_i be a shortest augmenting path with respect to M_i. Set $M_{i+1} = M_i \oplus \Pi_i$.

10.31 *Show that for each i, $|\Pi_{i+1}| \geq |\Pi_i|$.*

10.32 *Suppose for two indices $1 \leq i < j$, $|\Pi_i| = |\Pi_j|$. Show the following:*

1. *Let $k < l$ be two indices such that $i \leq k < l \leq j$ such that Π_k and Π_l are not vertex disjoint, but for each m, where $k < m < l$, Π_m is vertex disjoint from both Π_k and Π_l. (Note that if $l = k + 1$, there is no feasible value for m.) Show that Π_l is an augmenting path with respect to $M_k \oplus \Pi_k$.*
2. *Show that $|\Pi_l| \geq |\Pi_k| + |\Pi_k \cap \Pi_l|$. Show that Π_k and Π_l cannot share an edge.*
3. *Show that Π_k and Π_l are vertex disjoint.*
4. *Conclude that Π_i and Π_j are vertex disjoint, i.e. if the two shortest augmenting paths at steps i and j have the same length, they are vertex disjoint.*

10.33 *Let M^* be a maximum cardinality matching in G. Answer the following.*

1. *For any matching M in G, show that $|M^*| - |M| \leq |V|/k$, where k is the length of the shortest augmenting path in G with respect to M.*
2. *Define $r = \left\lfloor |M^*| - \sqrt{|M^*|} \right\rfloor$. Show that in the sequence of matchings $M_0, M_1, \ldots$, where $M_{i+1} = M_i \oplus \Pi_i$ and Π_i is a shortest augmenting path with respect to M_i, there exists a matching M_r, such that $|M_r| = r$.*
3. *From Exercise 10.29, conclude that $|\Pi_r| \leq 2\lfloor\sqrt{|M^*|}\rfloor + 1$.*
4. *Show that for $i \in [1, r]$, $|\Pi_i| \leq 2\lfloor\sqrt{|M^*|}\rfloor + 1$.*
5. *Show that $|\Pi_{r+1}|, \ldots, |\Pi_{|M^*|}|$ can be at most $\sqrt{|M^*|}$ distinct numbers.*
6. *Conclude that $|\Pi_1|, \ldots, |\Pi_{|M^*|}|$ forms at most $2\sqrt{|M^*|} + 1$ distinct numbers (recall that $|\Pi_i|$ is odd).*

The main steps of the Hopcroft-Karp's maximum cardinality matching algorithm are as follows.

Input: A bipartite graph $G = (V = L \cup R, E)$
Output: A maximum cardinality matching in G.

Step 1: Set $M \leftarrow \emptyset$

Step 2: While there exists augmenting paths in G with respect to M:

1. Let l be the length of the shortest augmenting path in G with respect to M.
   Find a maximal collection of vertex disjoint augmenting paths of length l with respect to M. Let the paths be $Q_1, Q_2, \ldots, Q_r$.
2. $M \leftarrow M \oplus Q_1 \oplus \ldots \oplus Q_r$

Step 3: Return(M)

10.34 *Show that the matching returned by the algorithm is a maximum cardinality matching in G.*

10.35 *Show that after each iteration of the While-loop, the length of a shortest augmenting path increases by at least 2 as compared to the previous iteration. (I.e. if the length of the shortest augmenting path is l in the current iteration, it will be $\geq l + 2$ in the next iteration.) Conclude that after $\sqrt{|V|}$ iterations, the length of shortest augmenting paths will be $\geq 2\sqrt{|V|}$.*

10.36 *Show that after executing the While-loop for $\sqrt{V}$ iterations, there are at most $\frac{1}{2}\sqrt{V}$ augmenting paths left to be processed. Conclude that the While loop iterates, in all, at most $\frac{3}{2}\sqrt{V}$ times.*

10.37 *Show that the length of the shortest augmenting path with respect to M in each iteration of the While-loop can be computed by performing a*

(modified) breadth-first traversal in G in $O(|V| + |E|)$ time as follows. In the BFS, Level 0 consists of all the unmatched vertices in L. Level 1 consists of all their neighbors in R. Level 2 contains the matched neighbors of vertices in Level 2. Level 3 consists of neighbors of vertices in Level 3 in the set R that haven't been discovered so far, and so on. We stop the BFS traversal when we encounter an unmatched vertex in R (at an odd level) for the first time and return the level number as the length of the shortest augmenting path. Note that in the BFS traversal, the edges from level L_i to level L_{i+1} satisfy the following: If i is even, the edges are from the set $E \setminus M$. If i is odd, the edges are from M.

10.38 *Given the BFS traversal corresponding to an iteration of the While-loop, show that we can find a maximal set of vertex-disjoint augmenting paths (of shortest length l) in $O(|V| + |E|)$ time. Think of starting from the last level (Level l) in BFS where unmatched vertices in R were discovered, and walk backwards to Level 0.*

10.39 *Conclude that the Hopcroft-Karp's algorithm finds a maximum cardinality matching in a bipartite graph $G = (V = L \cup R, E)$ in $O(|E|\sqrt{|V|})$ time.*

11
Multiplicative-Weight Update Method

We will focus on

1. Multiplicative weights algorithm for regret minimization in online learning.
2. Applications to Linear Programming using the Set Cover LP.
3. Applications to zero-sum games.

We will look at some classical results in online learning theory related to regret minimization and its applications to zero-sum games. This chapter is based on [9, 131].

11.1 Multiplicative Weight Update Algorithm

This section is based on the survey paper by Arora, Hazan and Kale [1]. This is also influenced by lecture notes of Gabor Lugosi, Tim Roughgarden, and Umesh Vazirani.

[1] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012

Suppose we are a naive investor in the stock market and are interested in seeing the behaviour of Dow Jones Industrial Average (DJIA) at the end of each day of trading. We have access to n experts (newspapers, stock briefs, ...) and based on their advise we need to make a prediction whether DJIA will go up or down at the end of each day. Our prediction costs us 0 if it is correct and costs us 1 if it is wrong. We need to devise an algorithm that helps us in making prediction for each day. Suppose we are at day t, where $t \in \{1, \ldots, T\}$. Our algorithm can use our predictions as well as that of all the experts for all of the previous $t-1$ days. At the end of T days we want to be competitive with respect to the best expert, i.e. our cost is not significantly higher than the cost of any expert (including the best expert). In this section we devise algorithms for

this problem and its variants. As a warmup we first consider some special cases.

11.1.1 *Real Experts*

Suppose we are told that among all the n experts there is at least one expert that never misses the mark. The predictions of this expert are always correct. But we do not know who is that expert. We can devise a fairly straightforward algorithm that makes at most $O(\log n)$ wrong predictions as follows.

Let the set of experts be $E = \{1, \ldots, n\}$.

For each day $t := 1$ to T do:

Step 1: Among all the remaining experts in E, poll them to find the prediction of the majority of them for that day. Record that as the prediction of the algorithm.

Step 2: Observe the true outcome at the end of the day. Discard all those experts that predicted wrong from E from future considerations.

Observe that for each mistake (i.e., a wrong prediction) that the algorithm makes, the size of the set E is reduced by at least a half. Since we know that there is at least one expert that never makes mistakes, the set E is non-empty during the entire execution of the algorithm. Thus the number of wrong predictions are bounded by $O(\log n)$. An alternate way to view this algorithm is that each expert i has an associated weight w_i that is initialized to 1. If during any step (i.e. the day t) of the algorithm, if an expert makes a mistake its weight is set to 0 and the experts with weight 0 are not considered for the rest of the algorithm.

11.1.2 *Expert with at most m wrong predictions*

Now suppose that we do not have any perfect expert but let us assume that the best expert makes at most m mistakes over the entire period of T days. We do not know who is the best expert. Can we use this knowledge to devise a competitive algorithm to predict. Some aspects of the analysis will be useful in understanding other algorithms that we will study in the rest of this section. Our first weighted majority update algorithm works as follows:

Let the set of experts be $E = \{1, \ldots, n\}$.
For each expert i, set its weight $w_i^1 = 1$.

For each day $t := 1$ to T do:

Step 1: Find the weighted majority prediction of the experts. To be precise, sum total the weights of all the experts that predict "UP". Similarly, sum total the weights of all the expert that predict "DOWN" . Whichever of the two sums is higher, record that as the prediction of the algorithm for day t.

Step 2: Observe the true outcome at the end of the day t.

Step 3: For all experts i that predicted correctly, their weight for the next day is set to $w_i^{t+1} = w_i^t$. For all the experts i that predicted wrongly, their weight is set to $w_i^{t+1} = w_i^t/2$.

Define the potential function Φ^t for day $t \in \{1, \ldots, T\}$ to be the sum total of the weights of all the experts at the start of day t, i.e. $\Phi^t = \sum_{i=1}^{n} w_i^t$. Note that $\Phi^1 = n$.

The key to the analysis of all the multiplicative weight update algorithms is this potential function.

Observation 11.1.1 *If the algorithms makes a wrong prediction on day t, $\Phi^{t+1} \leq \frac{3}{4}\Phi^t$.*

Proof. Since the algorithm follows the weighted majority and it has made a mistake, that implies that the weight for the next day is decreased by at least $\frac{1}{4}$-th of the total weight at the start of day t. Thus $\Phi^{t+1} \leq \Phi^t - \frac{1}{4}\Phi^t = \frac{3}{4}\Phi^t$. ∎

If the algorithm has made M mistakes in T days, its total weight at the end of day T,

$$\Phi^{T+1} \leq \left(\frac{3}{4}\right)^M \Phi^1 = \left(\frac{3}{4}\right)^M n \tag{11.1}$$

Let us assume that the best expert is i (note that we don't know its identity and just using it to establish the bounds). At the end of day T its weight is at least $\left(\frac{1}{2}\right)^m$ since it makes at most m mistakes. Since Φ^{T+1} is the sum total of the weights of all the experts at the end of day T, we know that

$$\Phi^{T+1} \geq \left(\frac{1}{2}\right)^m \tag{11.2}$$

Putting both the equations together, we obtain:

$$\left(\frac{1}{2}\right)^m \leq \Phi^{T+1} \leq \left(\frac{3}{4}\right)^M n$$

and taking log's we obtain:

$$\begin{aligned} -m &\leq M \log\left(\frac{3}{4}\right) + \log n \\ -M \log\left(\frac{3}{4}\right) &\leq m + \log n \\ M \log\left(\frac{4}{3}\right) &\leq m + \log n \\ M &\leq 2.41(m + \log n) \end{aligned}$$

So this ensures that the bound on the number of wrong predictions made by the algorithm M isn't that much off compared to the best expert who makes at most m mistakes. This bound has a multiplicative and an additive term.

What was so special about reducing the weight for the wrong experts by $\frac{1}{2}$? We can replace that by a factor $\eta \in (0, \frac{1}{2}]$. Then, we can reduce the weight of an expert by evaluating $w_i^{t+1} = (1-\eta)w_i^t$. Next we will see that this will result in $M \leq 2(1+\eta)m + \frac{2}{\eta}\log n$. Before we dive into the analysis, let us state some mathematical inequalities.

Observation 11.1.2 *1. For $\eta \in [0, \frac{1}{2}]$, $-\eta - \eta^2 \leq \ln(1-\eta) \leq -\eta$.*

2. For $\epsilon \in [0,1]$, $(1-\epsilon)^x \leq 1 - \epsilon x$ if $x \in [0,1]$.

3. For $\epsilon \in [0,1]$, $(1+\epsilon)^{-x} \leq 1 - \epsilon x$ if $x \in [-1,0]$.

Now with each mistake by the algorithm at least half of the total weight decreases by a factor of $(1-\eta)$. Suppose on day t the algorithm made the wrong prediction. Then the sum total of weights at the end of day t is given by $\Phi^{t+1} \leq \frac{1}{2}\Phi^t + \frac{1}{2}(1-\eta)\Phi^t = (1 - \frac{\eta}{2})\Phi^t$. Following the previous analysis, at the end of day T with the new update rule $w_i^{t+1} = (1-\eta)w_i^t$, the best expert will have weight at least $(1-\eta)^m$. The potential function Φ^{T+1} after M mistakes will be at most $(1 - \frac{\eta}{2})^M n$. Thus we have

$$(1-\eta)^m \leq \Phi^{T+1} \leq (1 - \frac{\eta}{2})^M n \tag{11.3}$$

We take log's and use Observation 11.2(1) to simplify.

$$\begin{aligned} m\ln(1-\eta) &\leq M \ln(1 - \frac{\eta}{2}) + \ln n \\ -m(\eta + \eta^2) &\leq -M\frac{\eta}{2} + \ln n \\ M\frac{\eta}{2} &\leq \ln n + (\eta + \eta^2)m \\ M &\leq \frac{2}{\eta}\ln n + 2(1+\eta)m \end{aligned}$$

Next we look into ways to get rid of the factor 2 in the above analysis using randomization.

11.2 *Randomized Multiplicative Weight Update Algorithm*

Assume that the costs are real numbers in the interval $[0, 1]$. For every expert $i \in \{1, \ldots, n\}$ and for every day $t \in \{1, \ldots, T\}$, we associate a cost of $m_i^t \in [0, 1]$ on day t. We want our algorithm to be competitive against the cost of the best expert. If M^t is the expected cost that the algorithm incurs on day t, and let us assume that i is the best expert, then we will see that the following randomized algorithm will ensure that $\sum_{i=1}^{T} M^t \leq \frac{\ln n}{\eta} + (1+\eta) \sum_{i=1}^{T} m_i^t$.

Let the set of experts be $E = \{1, \ldots, n\}$.
Let η to be any real number in $(0, \frac{1}{2}]$.
For each expert i, set its weight $w_i^1 = 1$.

For each day $t := 1$ to T do:

Step 1: Define $\Phi^t = \sum_{i=1}^{n} w_i^t$. For each expert i, compute $p_i^t = \frac{w_i^t}{\Phi^t}$.

Step 2: Choose an expert based on their probabilities and predict according to the chosen expert.

Step 3: Update Weights: For each expert i set $w_i^{t+1} = w_i^t(1 - \eta m_i^t)$.

Our analysis follows the same method. We use the potential function Φ^t to establish lower and upper bounds and then take log's to establish the desired bound. Let us first evaluate the expected loss M^t that the algorithm incurs on day t. By definition of expected value, it is given by

$$M^t = \sum_{i=1}^{n} p_i^t m_i^t = \langle p^t \cdot m^t \rangle, \tag{11.4}$$

where $p^t = (p_1^t, p_2^t, \ldots, p_n^t)$ and $m^t = (m_1^t, m_2^t, \ldots, m_n^t)$ are treated as vectors. Their dot product is expressed as $\langle p^t \cdot m^t \rangle$.

Consider Φ^{t+1}. We have the following:

$$\begin{aligned}
\Phi^{t+1} &= \sum_{i=1}^{n} w_i^{t+1} \\
&= \sum_{i=1}^{n} w_i^t (1 - \eta m_i^t) \\
&= \sum_{i=1}^{n} w_i^t - \eta \sum_{i=1}^{n} w_i^t m_i^t \\
&= \Phi^t - \eta \sum_{i=1}^{n} \Phi^t p_i^t m_i^t \text{ (as } p_i^t = \tfrac{w_i^t}{\Phi^t}) \\
&= \Phi^t - \eta \Phi^t \sum_{i=1}^{n} p_i^t m_i^t \\
&= \Phi^t (1 - \eta \langle p^t \cdot m^t \rangle) \\
&\leq \Phi^t e^{-\eta \langle p^t \cdot m^t \rangle} \text{ (as } 1 - x \leq e^{-x}) \\
&= \Phi^t e^{-\eta M^t}
\end{aligned}$$

Using induction on t, we obtain

$$\begin{aligned}
\Phi^{T+1} &\leq \Phi^0 e^{-\eta \sum_{t=1}^{T} M^t} \\
&= n e^{-\eta \sum_{t=1}^{T} M^t}
\end{aligned}$$

Since all $m_i^t \in [0,1]$, $w_i^t \geq 0$. This implies that $\Phi^{t+1} \geq w_i^{t+1}$ for any individual weight as $\Phi^{t+1} = \sum_{i=1}^{n} w_i^{t+1}$. Using the update rule of w_i^{t+1}, we can conclude that $\Phi^{T+1} \geq w_i^{T+1} = w_i^1 \prod_{t=1}^{T} (1 - \eta m_i^t) \geq (1-\eta)^{\sum_{t=1}^{T} m_i^t}$. Putting both the upper and lower bounds for Φ^{T+1} we obtain

Recall that for $\epsilon \in [0,1]$, $1 - \epsilon x \geq (1-\epsilon)^x$ if $x \in [0,1]$.

$$n e^{-\eta \sum_{t=1}^{T} M^t} \geq \Phi^{T+1} \geq (1-\eta)^{\sum_{t=1}^{T} m_i^t} \tag{11.5}$$

Now we take log's and divide by η and obtain:

$$\frac{\ln n}{\eta} - \sum_{t=1}^{T} M^t \geq \frac{\ln(1-\eta)}{\eta} \sum_{t=1}^{T} m_i^t \tag{11.6}$$

This is equivalent to

$$\sum_{t=1}^{T} M^t \leq \frac{\ln n}{\eta} - \frac{\ln(1-\eta)}{\eta} \sum_{t=1}^{T} m_i^t \tag{11.7}$$

Now apply Observation (1) that states that for $\eta \in [0, \frac{1}{2}]$, $-\eta - \eta^2 \leq \ln(1-\eta)$ and we obtain:

$$\sum_{t=1}^{T} M^t \leq \frac{\ln n}{\eta} + \frac{\eta + \eta^2}{\eta} \sum_{t=1}^{T} m_i^t \tag{11.8}$$

or

$$\sum_{t=1}^{T} M^t \leq \frac{\ln n}{\eta} + (1+\eta) \sum_{t=1}^{T} m_i^t \tag{11.9}$$

On the left we have the expected cost of our algorithm and on the right we have the cost of any of the experts. The cost of the algorithm is at most an additive factor $\frac{\ln n}{\eta}$ and a multiplicative factor $(1+\eta)$ away from the cost of any of the experts (including the best expert).

11.2.1 *Multiplicative Weight Update Algorithm With Costs in* $[-1, 1]$

Only difference from the previous subsection is that now the costs of each expert can be positive or negative, i.e. $m_i^t \in [-1, 1]$. The algorithm remains unchanged and it is restated in the following:

Let the set of experts be $E = \{1, \ldots, n\}$.
Let η to be any real number in $[0, \frac{1}{2}]$.
For each expert i, initialize its weight $w_i^1 = 1$.

For each day $t := 1$ to T do:

Step 1: Define $\Phi^t = \sum_{i=1}^{n} w_i^t$. For each expert i, compute $p_i^t = \frac{w_i^t}{\Phi^t}$.

Step 2: Choose an expert based on their probabilities and predict according to the chosen expert.

Step 3: Update Weights: For each expert i set $w_i^{t+1} = w_i^t(1 - \eta m_i^t)$.

Let us mimic the analysis of the previous subsection and take into account that the costs may be negative. From the definition of expected value, the expected cost of our algorithm on day t is given by $M^t = \sum_{i=1}^{n} p_i^t m_i^t = \langle p^t \cdot m^t \rangle$, where $p^t = (p_1^t, p_2^t, \ldots, p_n^t)$ and $m^t = (m_1^t, m_2^t, \ldots, m_n^t)$ and we take their dot product $\langle p^t \cdot m^t \rangle$. The upper bound for Φ^{t+1} follows the same analysis and we obtain $\Phi^{t+1} = \sum_{i=1}^{n} w_i^{t+1} = \sum_{i=1}^{n} w_i^t(1 - \eta m_i^t) \leq \Phi^t e^{-\eta M^t}$. And using induction on t we have $\Phi^{T+1} \leq n e^{-\eta \sum_{t=1}^{T} M^t}$. Since for all $i \in \{1, \ldots, n\}$ and $t \in \{1, \ldots, T\}$, $m_i^t \in [-1, 1]$, we have $1 - \eta m_i^t \geq 0$. Thus, $w_i^t \geq 0$. This implies that $\Phi^{t+1} \geq w_i^{t+1}$ for any individual weight. Using the update rule of w_i^{t+1}, we can conclude that $\Phi^{T+1} \geq w_i^{T+1} = \prod_{t=1}^{T}(1 - \eta m_i^t)$. This can be expressed by grouping for each day the

Recall that for $\epsilon \in [0, 1]$, $(1-\epsilon)^x \leq 1 - \epsilon x$ if $x \in [0, 1]$ and $(1+\epsilon)^{-x} \leq 1 - \epsilon x$ if $x \in [-1, 0]$.

positive m_i^t's and the negative m_i^t's.

$$\Phi^{T+1} \geq (1-\eta)^{\sum_{m_i^t \geq 0} m_i^t} (1+\eta)^{-\sum_{m_i^t < 0} m_i^t} \tag{11.10}$$

Putting both the upper and lower bounds for Φ^{T+1} we obtain

$$ne^{-\eta \sum_{t=1}^{T} M^t} \geq (1-\eta)^{\sum_{m_i^t \geq 0} m_i^t} (1+\eta)^{-\sum_{m_i^t < 0} m_i^t} \tag{11.11}$$

Now we take log's and divide by η and obtain:

$$\frac{\ln n}{\eta} - \sum_{t=1}^{T} M^t \geq \frac{\ln(1-\eta)}{\eta} \sum_{m_i^t \geq 0} m_i^t - \frac{\ln(1+\eta)}{\eta} \sum_{m_i^t < 0} m_i^t \tag{11.12}$$

Rearranging the terms we obtain

$$\sum_{t=1}^{T} M^t \leq \frac{\ln n}{\eta} - \frac{\ln(1-\eta)}{\eta} \sum_{m_i^t \geq 0} m_i^t + \frac{\ln(1+\eta)}{\eta} \sum_{m_i^t < 0} m_i^t \tag{11.13}$$

This is equivalent to

Recall that for $\eta \in [0, \frac{1}{2}]$ $\eta + \eta^2 \geq -\ln(1-\eta)$ and $\ln(1+\eta) \geq \eta - \eta^2$.

$$\sum_{t=1}^{T} M^t \leq \frac{\ln n}{\eta} + (1+\eta) \sum_{m_i^t \geq 0} m_i^t + (1-\eta) \sum_{m_i^t < 0} m_i^t \tag{11.14}$$

On expanding we obtain

Note that $(\eta - \eta^2) \sum_{m_i^t < 0} m_i^t \geq \ln(1+\eta) \sum_{m_i^t < 0} m_i^t$ because of negative values!

$$\sum_{t=1}^{T} M^t \leq \frac{\ln n}{\eta} + \eta \sum_{t=1}^{T} |m_i^t| + \sum_{t=1}^{T} m_i^t \tag{11.15}$$

Since $|m_i^t| \leq 1$, we have

$$\sum_{t=1}^{T} M^t \leq \frac{\ln n}{\eta} + \eta T + \sum_{t=1}^{T} m_i^t \tag{11.16}$$

On the left we have the expected cost of our algorithm and on the right we have the cost of any of the experts. The cost of the algorithm is at most an additive factor $\frac{\ln n}{\eta} + \eta T$ away from the cost of any of the experts including the best expert. We make the following observation.

Observation 11.2.1 *By setting $\eta = \sqrt{\frac{\ln n}{T}}$ in Equation 11.16, we obtain $\sum_{t=1}^{T} M^t \leq 2\sqrt{T \ln n} + \sum_{t=1}^{T} m_i^t$.*

That is the cost of our algorithm is off by an additive factor that is proportional to the square root of the product of the number of days and the number of experts as compared to the best expert. We

may also look at the average error on each day. This can be done by dividing the inequality in the observation by T and we obtain

$$\frac{1}{T}\sum_{t=1}^{T} M^t \leq 2\sqrt{\frac{\ln n}{T}} + \frac{1}{T}\sum_{t=1}^{T} m_i^t$$

Observe that as T increases the average error drops down. Therefore, a simple multiplicative weight strategy is able to learn from experts reasonably well when executed over a number of days. This is the power of this method.

11.3 *An Application of Multiplicative Weight Update Algorithm*

Let us look at the fractional set cover linear program (LP) and see how we can use the Multiplicative Weight Update Algorithm to approximate the objective value of the LP without actually solving the LP.

In a set cover problem we are given a universe U consisting of n elements and a set of m subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of U such that $\cup_{i=1}^{m} S_i = U$. The set cover problem requires us to find minimum number of subsets of $\mathcal{S}$ such that their union covers all elements of U (i.e, their union is U). This problem can be formulated as a simple Integer Linear Program (ILP). We use a $0-1$ indicator variable x_S for each set $S \in \mathcal{S}$, where $x_S = 1$ if and only if S is included in the set cover. The ILP formulation is:

$$\begin{aligned} \min \sum_{S \in \mathcal{S}} x_S \\ \forall u \in U : \sum_{u \in S} x_S \geq 1 \\ \forall S \in \mathcal{S} : x_S \in \{0,1\} \end{aligned}$$

The relaxed Linear Program (LP) is where we replace the integrality constraint $x_S \in \{0,1\}$ by $0 \leq x_S \leq 1$ as now we allow x_S's to take on real values. Moreover, we can drop the constraint that $x_S \leq 1$ as this is a minimization problem and what we require is $\sum_{u \in S} x_S \geq 1$. Lastly, we will convert this optimization problem to the feasibility problem by realizing that the value of the objective function is one of the numbers $\{1, \ldots, m\}$ for this problem instance. Suppose we guess that the optimal value is $\beta \in \{1, \ldots, m\}$. If the following feasibility inequality can be satisfied, we know that the optimal value is at most β and using this knowledge we can perform a binary search to find

the true optimal value. The feasibility inequalities are

$$\forall u \in U : \sum_{u \in S} x_S \geq 1$$
$$\sum_{S \in \mathcal{S}} x_S \leq \beta$$
$$\forall S \in \mathcal{S} : x_S \geq 0$$

Note that the constraints $\sum_{S \in \mathcal{S}} x_S \leq \beta$ and $\forall S \in \mathcal{S} : x_S \geq 0$ define a convex region. Let us denote this region by $\mathcal{P} = \{x \in \Re^m | \sum_{i=1}^{m} x_i \leq \beta \wedge \forall i \in \{1, \ldots, m\}, x_i \geq 0\}$. We can express the feasibility problem succinctly as follows:

Fractional Set Cover Feasibility Problem:

Report $x \in \mathcal{P}$ such that $\forall u \in U : \sum_{u \in S} x_S \geq 1$,
Otherwise report infeasibility

Let us define the following abstract approximate feasibility problem. Later we will see how to solve this problem using the Multiplicative Weight Update Method. We state the problem in terms of the notation of general linear program (feasibility) formulation where A is $n \times m$ matrix, b is a vector of length m, and $\mathcal{P}$ is the convex region as defined above.

Approximate Abstract Feasibility Problem:

Let $\epsilon \geq 0$ be an error parameter.
If $x \in \mathcal{P}$ and $Ax \geq b$ is feasible then
    report $x \in \mathcal{P}$ such that $A_i x \geq b_i - \epsilon$, $\forall i \in \{1, \ldots, n\}$,
Otherwise report infeasibility

Following the abstract formulation the approximate feasibility version for the set cover problem can be stated as follows. The matrix A is a $0-1$ matrix of size $n \times m$. It represents elements of U as rows and subsets in $\mathcal{S}$ as columns. The element in A corresponding to row $u \in U$ and column $S_i \in \mathcal{S}$ is 1 if and only if $u \in S_i$.

Approximate Set Cover Feasibility Problem:

For a universe U of size n and m-subsets of U, we have
- characteristic matrix A of size $n \times m$,
- vector b of length n consisting of 1's,

- $\mathcal{P} = \{x \in \Re^m | \sum_{i=1}^{m} x_i \leq \beta \wedge \forall i \in \{1, \ldots, m\}, x_i \geq 0\}$.
- Error parameter $\epsilon \geq 0$.

If $x \in \mathcal{P}$ and $Ax \geq b$ is feasible then
report $x \in \mathcal{P}$ such that $A_i x \geq 1 - \epsilon$, $\forall i \in \{1, \ldots, n\}$,
Otherwise report infeasibility.

Suppose we have an instance of the set cover feasibility problem with a given choice of β that is feasible. We know that for this instance, the approximate set cover feasibility problem will return us an $x \in \mathcal{P}$ such that $\forall u \in U : \sum_{u \in S} x_S \geq 1 - \epsilon$. Now by setting $\bar{x} = \frac{x}{1-\epsilon}$, we can obtain a feasible solution $\bar{x}$ for the fractional set cover problem whose value of the objective function is at most $(1 + O(\epsilon))\beta$. Observe that $\bar{x}$ satisfies the constraints $\bar{x}_i \geq 0$ for $i = 1, \ldots, m$ and $\forall u \in U : \sum_{u \in S} \bar{x}_S \geq 1$. Next we see how we can use multiplicative weight update method to solve the Approximate Set Cover Feasibility Problem.

We will also require an additional algorithm, so called the ρ-bounded oracle, that will be used by the multiplicative weight algorithm to solve the feasibility problem. The ρ-bounded oracle takes as input a probability distribution $p = (p_1, \ldots, p_n)$, where $\sum_{i=1}^{n} p_i = 1$, on the rows of A (i.e. on the elements of U) and returns the following.

ρ-bounded oracle

If $x \in \mathcal{P}$ and $p^T Ax \geq p^T b$ is feasible,
return $x^* \in \mathcal{P}$ such that $\forall i : |A_i x^* - b_i| \leq \rho$.
Otherwise, return that the system is infeasible.

Note that $p^T Ax \geq p^T b$ is a single inequality. It is composed of a linear combination of rows of A given by the vector p. Thus finding $x \in \mathcal{P}$ that satisfies this inequality seems to be easier than satisfying n constraints of the fractional set cover feasibility problem. Let us now construct the ρ-bounded oracle for the set cover.

First note that for the set cover $p^T b = 1$. Therefore, we want $x \geq 0$, $\sum_{S \in \mathcal{S}} x_S \leq \beta$, and $p^T Ax = \sum_{u \in U} p_u \left(\sum_{u \in S} x_S \right) = \sum_{S \in \mathcal{S}} x_S p(S) \geq 1$, where $p(S)$ denotes the sum of the probabilities associated to the elements in S. Let us first understand the two equalities in the above equation. We are interpreting the product $p^T Ax$ in two different ways. Assume that the universe is $\mathcal{U} = \{u_1, \ldots, u_n\}$, where the i-th row of A corresponds to the element u_i, and let the sets are $\mathcal{S} =$

$\{S_1, \ldots, S_m\}$ representing the columns of A. As already mentioned the entry $A_{ij} = 1$ if and only if $u_i \in S_j$, otherwise it is 0. Think of each element $u_i \in \mathcal{U}$ has an associated probability p_i. In the first interpretation, the product Ax is a vector of dimension n, where its i-th entry is the number of sets in $\mathcal{S}$ that contain the element u_i (i.e, $\sum_{u \in S} x_S$). The product $p^T Ax$ is the dot-product of vectors p^T and Ax, where the i-th entry in Ax is multiplied by the probability p_i. Therefore, $p^T Ax$ is the sum of the products of the probability p_i of element u_i times the number of occurrences of u_i in $\mathcal{S}$. In the second interpretation, for each set $S \in \mathcal{S}$, we first sum up the probabilities associated to each element in that set. That is the quantity $p(S)$. We take the sum $p(S)$ over all sets so that for each element we take into account the number of times it occurs in the sets of $\mathcal{S}$. Therefore, $p^T Ax = \sum_{u \in U} p_u \left(\sum_{u \in S} x_S \right) = \sum_{S \in \mathcal{S}} x_S p(S)$.

Given p we want to find an $x \in \mathcal{P}$ that satisfies $\sum_{S \in \mathcal{S}} x_S p(S) \geq 1$. To do so, we will find the set $S \in \mathcal{S}$ that maximizes $p(S)$ for the given p. Suppose the set $S^* \in \mathcal{S}$ maximizes this value. Set $x_{S^*} = \beta$ and for every other set $S \neq S^*$ set $x_S = 0$. Observe that the vector x^* has 0's in all the coordinates except the coordinate corresponding to S^* where it is equal to β, i.e. $x^* = (0, 0, \ldots, \beta, 0, \ldots, 0)$. Furthermore, $x^* \in \mathcal{P}$. If $\sum_{S \in \mathcal{S}} x^*_S p(S) \geq 1$, we have the right value x^*. But if $\sum_{S \in \mathcal{S}} x^*_S p(S) < 1$, then no other $x \in \mathcal{P}$ can satisfy this inequality. Why? Note that under the constraints ($x \geq 0$, $\sum_{S \in \mathcal{S}} x_S \leq \beta$) the choice of x that maximizes the expression $\sum_{S \in \mathcal{S}} x_S p(S)$ didn't satisfy the inequality.

To complete the design of the ρ-bounded oracle, let us evaluate the value of ρ. We want to find the smallest value ρ such that for all $i \in \{1, \ldots, n\}$, $|A_i x^* - b_i| \leq \rho$. Due to the choice of x^* and matrix A being a $0-1$ matrix, the product $A_i x^*$ is either 0 or β. Thus $|A_i x^* - b_i| = |A_i x^* - 1| \leq |\beta - 1| \leq \beta \leq m$. Note that for this problem $\beta \geq 1$. So we can choose $\rho = \beta$ or $\rho = m$.

Finally, let us see how we can use this ρ-bounded oracle in the multiplicative weight update algorithm to solve the Approximate Set Cover Feasibility Problem. Recall from the previous subsection that to design the multiplicative weight algorithm we need experts, their costs, their weights, and the probabilities. We will have n experts, where the i-th expert is associated to the i-th row of A. We modify the multiplicative weight update algorithm as follows:

Multiplicative Weight Update Algorithm for Approximate Fractional Set Cover

Step 1: Fix an $\eta \in [0, \min(\frac{1}{2}, \frac{\epsilon}{2\rho})]$.

Step 2: Set $w^1 = (1, \ldots, 1)$.

Step 3: For $t = 1$ to $T = 4\frac{\rho^2 \ln n}{\epsilon^2}$ days do:

1. Compute $\Phi^t = \sum_{i=1}^{n} w_i^t$.
2. Compute the probability vector $p^t = (\frac{w_1^t}{\Phi^t}, \ldots, \frac{w_n^t}{\Phi^t})$.
3. Execute the ρ-bounded oracle for the probability vector p^t. It either returns that the system is infeasible and we STOP or returns the vector x^t.
4. Compute the costs of each expert i by evaluating $m_i^t = \frac{1}{\rho}(A_i x^t - b_i)$. (Observe that $m_i^t \in [-1, 1]$.)
5. Update weights for the next day for each expert i by executing $w_i^{t+1} = w_i^t(1 - \eta m_i^t)$.

Step 4: If we didn't report infeasibility during the T days of execution, return $\bar{x} = \frac{1}{T}\sum_{t=1}^{T} x^t$ as the answer to the Approximate Set Cover Feasibility Problem.

Before we analyze this algorithm, let us make a few remarks. Note that if $A_i x^t \geq b_i$, $m_i^t \geq 0$, and the i-th constraint is satisfied. But if $A_i x^t < b_i$, then $m_i^t < 0$. For the rows of A for which the constraints are satisfied their weights will be smaller compared to the rows for which the constraints are not satisfied. Hence, in the next round the unsatisfied rows (experts) will get higher probabilities compared to the satisfied rows. The more unsatisfied the row is higher is its probability. That is the key to this algorithm. In each step x^t is in the convex region $\mathcal{P}$. Their average $\bar{x}$ will also be in $\mathcal{P}$ due to convexity.

From the analysis of the previous section (see Equation 11.16) we know that the expected cost of this algorithm is bounded with respect to the cost of any expert i by

$$\sum_{t=1}^{T} M^t = \sum_{t=1}^{T} \langle p^t \cdot m^t \rangle \leq \frac{\ln n}{\eta} + \eta T + \sum_{t=1}^{T} m_i^t$$

Claim 11.3.1 $\sum_{t=1}^{T} M^t \geq 0$.

Proof. Since $m^t = \frac{1}{\rho}(Ax^t - b)$, we have

To be precise, we should write $(p^t)^T \cdot x^t$ for the dot-product, where $(p^t)^T$ is the transpose of the probability vector. But we may get confused as we use T to denote the number of days and not the transpose. Therefore we will drop the T for the transpose unless we really need it.

$$\begin{aligned} M^t &= \langle p^t \cdot m^t \rangle \\ &= \frac{1}{\rho}\langle p^t \cdot (Ax^t - b) \rangle \\ &= \frac{1}{\rho}(\langle p^t \cdot Ax^t \rangle - \langle p^t \cdot b \rangle) \\ &\geq 0 \end{aligned}$$

The last inequality holds as the system is satisfied, i.e. $(p^t)^T Ax^t \geq (p^t)^T b$. ∎

For $t = 1, \ldots, T$, all of $M^t \geq 0$, we have

$$\frac{\ln n}{\eta} + \eta T + \sum_{t=1}^{T} m_i^t \geq \sum_{t=1}^{T} M^t \geq 0$$

Substitute $m_i^t = \frac{1}{\rho}(A_i x^t - b_i)$, we obtain:

$$\frac{\ln n}{\eta} + \eta T + \frac{1}{\rho}\sum_{t=1}^{T}(A_i x^t - b_i) \geq 0$$

This is equivalent to:

$$\frac{\ln n}{\eta} + \eta T + \frac{1}{\rho}\sum_{t=1}^{T} A_i x^t - \frac{T}{\rho} b_i \geq 0$$

Now multiply by $\frac{\rho}{T}$ and use $\bar{x} = \frac{1}{T}\sum_{t=1}^{T} x^t$. We obtain:

$$\frac{\rho \ln n}{T\eta} + \rho\eta + A_i\bar{x} - b_i \geq 0$$

Now substitute $T = 4\frac{\rho^2 \ln n}{\epsilon^2}$ and $\eta \in [0, \min(\frac{1}{2}, \frac{\epsilon}{2\rho})]$ we obtain

$$\epsilon + A_i\bar{x} - b_i \geq 0$$

Observe that in this substitution both the terms $\frac{\rho \ln n}{T\eta}$ and $\rho\eta$ are upper bounded by $\frac{\epsilon}{2}$. In fact this is how the bounds on T and η are determined. Therefore, we have what we wanted, i.e. $A_i\bar{x} \geq b_i - \epsilon$.

Now we briefly address the computational complexity. We run the algorithm for T days. For each day we make a call to the ρ-bounded oracle. So the overall time complexity is bounded by the time it takes to run $O(\frac{\rho^2 \ln n}{\epsilon^2})$ calls to the oracle.

Let us recall what we did in this subsection. We wanted to solve the set cover problem. We described the ILP formulation and then formulated a relaxed LP that may use fractional values. We converted the LP to the fractional set cover feasibility problem as we can perform a binary search to find what is the size of the minimum set cover. Then we discussed an approximate version of the feasibility

problem. We concluded that if we answer the approximate feasibility problem then we can find an assignment that satisfies all the constraints and is within a $1 + O(\epsilon)$ factor of optimal. To answer the approximate feasibility we take help of a ρ-bounded oracle. This is an easier problem as it has a very few constraints. This was used within the multiplicative weight algorithm to find successive x's that are within the convex region $\mathcal{P}$. If the approximate feasibility problem has a solution, the average of various x^t's that are computed in the entire run of the multiplicative weight update method over T days yields an approximation. Otherwise we report infeasibility and adjust the guess on the optimal value and restart.

11.4 Exercises

11.1 *What will be the competitive ratio of the following method for predicting the trend of the stock market? Suppose we have n experts, and each day we evaluate which of the experts have predicted correctly on that day. We select any one of those experts and follow that experts advise for the next day.*

11.2 *What will be the competitive ratio of the following method for predicting the trend of the stock market? Suppose we have n experts. Each day our decision is the prediction of the majority of experts. (Note that we don't discard any of the experts.)*

11.3 *Suppose there are only two possible actions $\{\uparrow, \downarrow\}$ of Dow Jones Index at the end of each day. Answer the following questions for the different scenarios.*

1. *Each morning the algorithm chooses the actions based on some smart scheme. If the algorithm chooses $\uparrow$ with probability $\geq \frac{1}{2}$, the adversary assigns the reward of -1 for choosing the action $\uparrow$ and a reward of $+1$ for choosing $\downarrow$. If the algorithm chooses $\downarrow$ with probability $\geq \frac{1}{2}$, the adversary assigns a reward of $+1$ to the action $\uparrow$ and a reward of -1 to $\downarrow$. Over a run of T days, show that the expected reward of the algorithm is at most 0. How does this compares with the reward of the adversary if it somehow choose an optimal action for each day? (Remark: This exercise shows that the algorithm (even a randomized scheme) has no match for the adversary that chooses an optimal action on each day.)*

2. *Each morning the algorithm chooses one of the two actions by following some deterministic strategy. If the choice of our action for that day matches DJI we get a reward of $+1$, otherwise we get a reward of 0. Show that in a run of T days, an adversary can design the outcomes for each day in such a way that the reward that our algorithm gets is 0, whereas there is a fixed action if chosen for all the days will generate a revenue of*

at least $\frac{T}{2}$. If there are n actions, show that the algorithms revenue can be zero, whereas there is a fixed action that can generate a revenue of at least $T(1-\frac{1}{n})$. Conclude that no deterministic algorithm can ensure a positive reward. (Note that the problem is that we don't know which action will generate that kind of revenue till we have observed the behaviour of all the actions for T days.)

3. *For each day, an adversary tosses a fair coin. If the outcome is Heads, it assigns a reward of -1 to $\uparrow$ and $+1$ to $\downarrow$. Whereas, if the outcome is Tails, the rewards are flipped, i.e. reward of $+1$ to $\uparrow$ and -1 to $\downarrow$. Show that the expected reward of any algorithm is 0 over a run on T days. In fact this holds at the end of any of the t days, where $t = 1, \ldots, T$. Show that at least one of the fixed actions has a reward of $c\sqrt{T}$ for some constant c. Argue that if a fair coin is flipped T times, we expect to get $\frac{T}{2}$ Heads, but there is a standard deviation of $\frac{\sqrt{T}}{2}$. (Variance for getting a head in a single flip is $\frac{1}{4}$ and among n independent flips is $\frac{n}{4}$.)*

11.4 *Consider Observation 11.2.1 that states that by setting $\eta = \sqrt{\frac{\ln n}{T}}$ in Equation 11.16, we obtain $\sum_{t=1}^{T} M^t \leq 2\sqrt{T \ln n} + \sum_{t=1}^{T} m_i^t$. Consider the time-averaged version of the above equation by dividing by T and we obtain $\frac{1}{T}\sum_{t=1}^{T} M^t \leq 2\sqrt{\frac{\ln n}{T}} + \frac{1}{T}\sum_{t=1}^{T} m_i^t$. Show that after $T \geq \frac{4 \ln n}{\epsilon^2}$ number of days, the expected time-averaged cost of MWU algorithm is off by at most ϵ with respect to the best expert, where $\epsilon > 0$ is a constant.*

11.5 *In Subsection 11.2.1 m_i^t's were the losses of experts on day t. They can take any values in the interval $[-1, 1]$. Instead of thinking of m_i^t's as the loss of expert i on day t, assume that it is the gain of the expert. In that section we wanted to establish that our online strategy doesn't incur significantly more loss than the best expert. Show what changes you need to make in the multiplicative weight update method if m_i^t's are gains. Show that the expected gain of the algorithm is $\sum_{t=1}^{T} M^t \geq \sum_{t=1}^{T} m_i^t - \frac{\ln n}{\eta} - \eta \sum_{t=1}^{T} |m_i^t|$, where $\sum_{t=1}^{T} m_i^t$ is the gain of the best expert.*
(Hint: Can we think of the loss vector as $-m^t$ and use the same algorithm as in Subsection 11.2.1?)

11.6 *Consider the following problems:*

1. *Let r and s be two numbers and $\alpha, \beta \in [0, 1]$ such that $\alpha + \beta = 1$. Show that for any choice of α and β, $\alpha r + \beta s \leq \max(r, s)$.*

2. *Let $x_1, \ldots, x_n$ be numbers, and $\alpha_1, \ldots, \alpha_n \in [0, 1]$, such that $\sum_{i=1}^{n} \alpha_i = 1$. Show that for any choice of $\alpha_1, \ldots, \alpha_n$, $\sum_{i=1}^{n} \alpha_i x_i \leq \max(x_1, \ldots, x_n)$.*

3. *Observation 11.2.1 states that by setting* $\eta = \sqrt{\frac{\ln n}{T}}$ *in Equation 11.16, we have* $\sum_{t=1}^{T} M^t \leq 2\sqrt{T \ln n} + \sum_{t=1}^{T} m_i^t$. *That is, the expected regret/loss of MWU algorithm is at most* $2\sqrt{T \ln n}$, *where* n *is the number of experts and* T *is the number of days, with respect to the loss of the best expert* $(\sum_{t=1}^{T} m_i^t)$. *Show that we can extend the analysis where we replace the best expert by any (fixed) probability distribution over the experts. Formally, the quantity* $\sum_{t=1}^{T} m_i^t$ *will be replaced by* $\sum_{t=1}^{T}\sum_{i=1}^{n} p_i^* m_i^t$ *in Observation 11.2.1, where* p^* *is a fixed probability distribution over the experts. Show that the best expert is as good as any fixed probability distribution over the experts,* $\sum_{t=1}^{T}\sum_{i=1}^{n} p_i^* m_i^t \leq \max_{i=1}^{n} \sum_{t=1}^{T} m_i^t$.

Following exercises are adapted from Tim Roughgarden's lecture on Applications of MW Update Method.

11.7 *(Zero-Sum games)*

Consider the following two-person zero-sum game called matching pennies. *Two rational players A (the row player) and B (the column player), hold a penny each. In each round of the game, each of them decides, using whatever strategy they want, whether their penny will show heads or tails when placed on the table. In each round, both the players place their pennies simultaneously on the table. A wins this round and gets B's penny, if both the pennies are heads or both the pennies are tails. Otherwise, B gets A's penny in this round. The game is called the zero-sum game, as one players gain is the same as the other player's loss. These games are represented using the payoff matrix A with respect to the payoff for the row player. Row player A has two plays - play Head (Row 1) or play Tails (Row 2). Similarly, Column player B has two plays - play Head (Column 1) or play Tail (Column 2). The payoff matrix of the row player is given in Figure 11.7. The payoff for the column player is the same as the row player, except that each of the entries has an opposite sign (as the gain of one player is the loss of the other player).*

Assume that the above game is repeated for several rounds. If a player always plays the same row/column, we say it is playing a pure *strategy. If the player varies the rows/columns, say according to some probability distribution overs rows/columns, we say it is playing a* mixed *strategy. Observe that if any of the players figure out that the opponent is playing a pure strategy, then this player has an advantage. As will be clear from the following exercises, it will be better for the players to use a mixed strategy to be unpredictable.*

See https://en.wikipedia.org/wiki/Matching_pennies

		Player B	
		H	T
Player A	H	+1	-1
	T	-1	+1

Figure 11.1: Payoff matrix U for the row player A

1. *What will be the revenue of A if A employs the pure strategy of always playing Heads?*

2. *What will be the best strategy for player B if A plays the mixed strategy of playing Heads and Tails with equal probability? What will be its expected revenue?*

3. *What will be the best strategy and the expected revenue of B if A plays Heads with probability* 0.7 *and Tails with probability* 0.3.

4. *Show that the best strategy for both the players is to choose heads and tails with equal probability.*

5. *Show that the expected payoff of both the players is* 0, *if they choose Heads/Tails with probability* $= 1/2$. *I.e., show that* $\sum_{i=1}^{2}\sum_{j=1}^{2} p_i q_j A[i,j] = 0$, *where* $p_j = \frac{1}{2}$ *represents the probability of choosing row* i *by Player* A, $q_j = \frac{1}{2}$ *represents the probability of choosing column* j *by Player* B, *and* $A[i,j]$ *is the entry in* i*-th row and* j*-th column of the payoff matrix of the row player.*

6. *What is the expected payoff of Player A if A chooses each row with probability* $\frac{1}{2}$, *and B can choose any possible mixed strategy.*

7. *Assume that the row player plays first and chooses the rows* 1 *and* 2 *with probabilities* p_1 *and* $p_2 = 1 - p_1$, *and furthermore assume that it informs its mixed strategy vector* $p = (p_1, p_2)$ *to the column player. Show that the best strategy for the column player is to deterministically play one of the columns - the column that minimizes the value* $p^T Aq$, *where* $q = (1,0)$ *or* $q = (0,1)$. *In general, express* $q = (q_1, q_2)$, *where* $q_i \geq 0$ *and* $q_1 + q_2 = 1$. *Show that the optimum payoff to the column player is at least* $\max_p \left(\min_q -p^T Aq \right)$ *- first let the row player choose a strategy* p, *and then the column player minimizes over the various choices for* q.

8. *Show that if the column player plays first and the row player plays second, then the optimum payoff for the row player is at least* $\min_q \left(\max_p p^T Aq \right)$.

9. *For the matching pennies game, show that* $\max_p \left(\min_q p^T Aq \right) = \min_q \left(\max_p p^T Aq \right) = 0$.

11.8 *What will be the best mixed strategy for the row and column players in the rock-paper-scissors two person zero-sum game, where the payoff matrix of the row player is given in Figure 11.8.*

Note that both the players play simultaneously. Rock beats scissors, scissors beats paper, and paper beats rock. The outcomes of this game are draw, A wins or A looses. If A plays Rock and B plays Scissor, than A wins and gets $1 from B. Whereas if both A and B play scissors, it's a draw. Note that the payoff to the row player $= \sum_{i=1}^{3}\sum_{j=1}^{3} p_i q_j A[i,j]$, *where*

		B		
		Rock	Scissor	Paper
A	Rock	0	+1	-1
	Scissor	-1	0	+1
	Paper	1	-1	0

Figure 11.2: Payoff Matrix for the row player in rock-paper-scissor game

https://en.wikipedia.org/wiki/Rock_paper_scissors

(p_1, p_2, p_3) (resp. (q_1, q_2, q_3)) represents the probabilities of choosing rows (columns) by the row (column) player.

Consider a generic two person zero-sum game between a row and a column player, where A is the $n \times m$ matrix representing the payoff to the row player, given that the row player has n possible strategies and the column player has m possible strategies. The payoff to row player is given by the equation $\sum_{i=1}^{n}\sum_{j=1}^{m} p_i q_j A[i,j] = p^T A q$, where p_i is the probability of choosing row i by the row player and q_j is the probability of choosing column j by the column player, and let $p = (p_1, \ldots, p_n)$ and $q = (q_1, \ldots, q_m)$ be the corresponding probability vectors. Suppose the row player commits to the mixed strategy $p = (p_1, \ldots, p_n)$ first, i.e. it plays first. Than the column player wants to optimize the function $\min_q p^T A q$. The interpretation is as follows. Column player wants to ensure that the row player gets the smallest possible value once it fixes its strategy p. Row player wants to choose that p which achieves $\max_p \left(\min_q p^T A q \right)$. Now consider the scenario when the column player chooses the mixed strategy q first. Now the row player, using a similar reasoning, will like to optimize $\max_p -p^T A q$. The famous *minimax theorem* states that if both players play optimally than $\max_p \left(\min_q p^T A q \right) = \min_q \left(\max_p p^T A q \right)$, and this value is called the value of the game. The following exercise, using the multiplicative weight update method, will help us to derive a proof of the minimax theorem.

11.9 *Consider a two person zero-sum game where the row player payoff matrix is A, and assume that each entry $A[i,j] \in [-1,1]$. Assume that the row player moves first - it fixes its mixed strategy for day $t = 1, \ldots, T$ using the MWU method of Section 11.2.1. Assign experts to each row, and using the MWU method, choose probability vector p^t for rows for time t. Assume that the column player responds optimally with the best strategy q^t (which is deterministic given p^t). Now for row i, if the column player chose column j, than the cost incurred by expert i for day t is $m_i^t = A[i,j]$. The algorithm of Section 11.2.1 is reproduced in the margin. Answer the following:*

1. *Show that on each day t, the column player can choose an optimal strategy q^t to counter row players' mixed strategy p^t for the day t. Note that we assumed that the row player plays first on each of the days.*

2. *Define $\hat{p} = \frac{1}{T}\sum_{t=1}^{T} p^t$ as the average mixed strategy of the row player over T days. Let q^* be an optimal response by the column player to the row players strategy $\hat{p}$. Show the following:*

Let the set of experts be $E = \{1, \ldots, n\}$.
Let η to be any real number in $[0, \frac{1}{2}]$.
For each expert i, initialize its weight $w_i^1 = 1$.

For each day $t := 1$ to $T = 4\frac{\ln n}{\epsilon^2}$ do:

Step 1: Define $\Phi^t = \sum_{i=1}^{n} w_i^t$. For each expert i, compute $p_i^t = \frac{w_i^t}{\Phi^t}$.

Step 2: Choose an expert based on their probabilities and predict according to the chosen expert.

Step 3: Update Weights: For each expert i set $w_i^{t+1} = w_i^t(1 - \eta m_i^t)$.

(a) *Show that* $\max_p \left(\min_q p^T A q \right) \geq \min_q \hat{p}^T A q$.

(b) *Show that* $\min_q \hat{p}^T A q = \hat{p}^T A q^*$.

(c) *Show that* $\hat{p}^T A q^* = \frac{1}{T} \sum_{t=1}^{T} (p^t)^T A q^*$.

(d) *Use the fact that* q^t *is an optimal response to* p^t *for day* t *and conclude that* $(p^t)^T A q^* \geq (p^t)^T A q^t$.

(e) *Show that* $\max_p \left(\min_q p^T A q \right) \geq \frac{1}{T} \sum_{t=1}^{T} (p^t)^T A q^t$

(f) *Conclude that if the column player plays second on each day, its expected loss can't be more than* $\max_p \left(\min_q p^T A q \right)$.

3. *Define* $\hat{q} = \frac{1}{T} \sum_{t=1}^{T} q^t$. *Show the following:*

(a) *Show that the average payoff of the row player over* T *days in the MWU method is given by* $\frac{1}{T} \sum_{t=1}^{T} (p^t)^T A q^t$.

(b) *See Exercise 11.4. Note that we ran the MWU method for* $T = 4\frac{\ln n}{\epsilon^2}$ *days. Using the fact that MWU method ensures that the expected payoff of the row player is within* ϵ *if it either used a fixed strategy or used a fixed probability distribution* p *over its rows (see Exercise 11.6), show that* $\frac{1}{T} \sum_{t=1}^{T} (p^t)^T A q^t \geq \max_p \left(\frac{1}{T} \sum_{t=1}^{T} p^T A q^t \right) - \epsilon$.

(c) *Use the definition of* $\hat{q}$ *to show that* $\max_p \left(\frac{1}{T} \sum_{t=1}^{T} p^T A q^t \right) = \max_p p^T A \hat{q}$.

(d) *Show that* $\max_p p^T A \hat{q} \geq \min_q \left(\max_p p^T A q \right)$.

(e) *Conclude that the expected payoff of the row player* $\frac{1}{T} \sum_{t=1}^{T} (p^t)^T A q^t \geq \min_q \left(\max_p p^T A q \right) - \epsilon$.

4. *Recall that if the column player plays first, the revenue of the row player is at most* $\min_q \left(\max_p p^T A q \right)$. *Using the lower and upper bounds on* $\frac{1}{T} \sum_{t=1}^{T} (p^t)^T A q^t$ *and setting* $\epsilon \to 0$, *conclude the statement of the mini-max theorem* $\max_p \left(\min_q p^T A q \right) = \min_q \left(\max_p -p^T A q \right)$.

11.10 *Let* $x = (x_1, \ldots, x_n)$ *be a probability vector, i.e.,* $x_i \geq 0$ *and* $\sum_{i=1}^{n} x_i = 1$. *Let* A *be a* $n \times m$ *matrix where each entry* $A_{ij} \in [-1, 1]$, *and* b *a vector of length* n *of real numbers. The task is to determine whether there is a*

probability vector x such that $Ax \leq b$, or report none exists? In place of finding whether $Ax \leq b$, we will answer approximately. I.e., if there is no x such that $Ax \leq b$, report no such x exists, else, report an x such that $Ax \leq b + \epsilon$, for some $\epsilon > 0$. We employ the MWU algorithm as follows:

Let the set of experts be $E = \{1, \ldots, m\}$, corresponding to variables $x_1, \ldots, x_m$.
Let η to be any real number in $(0, min(\frac{1}{2}, \frac{\epsilon}{2})]$.
For each expert $i \in [m]$, set its weight $w_i^1 = 1$.

For each day $t := 1$ to $T(= \ln n/\epsilon^2)$ do:

Step 1: *Let $\Phi^t = \sum_{i=1}^{n} w_i^t$.*

For each expert i, compute $x_i^t = \frac{w_i^t}{\Phi^t}$.
Set $x^t = (x_1^t, \ldots, x_n^t)$.

Step 2: *If $Ax^t \leq b + \epsilon$, return $x = x^t$ and STOP.*
Else, there is a constraint $i \in [n]$, where $A_i x^t > b_i + \epsilon$.
Update Weights: For each $j \in [m]$, $w_j^{t+1} = w_j^t(1 - \eta A_{ij})$.

Step 3: *Report, no x exists that satisfies $Ax \leq b$.*

Suppose that there is a probability vector x^* such that $Ax^* \leq b$ and the above algorithm fails to return an x after $T = \ln n/\epsilon^2$ days. Answer the following.

1. *Show that the loss of the above algorithm for each day t is at least $A_i x^t > b_i + \epsilon$ for some $i \in [n]$, whereas $A_i x^* \leq b_i$.*
2. *Show that the total loss of the above algorithm for T days is $> \epsilon T$, i.e., $\sum_{t=1}^{T} \max_{i=1}^{n} |A_i x^t - A_i x^*| > \epsilon T$.*
3. *Using Equation 11.9, conclude that the above algorithm will always report an x such that $Ax \leq b + \epsilon$, whenever there exists an x^* such that $Ax^* \leq b$.*
4. *Suppose $A_{ij} \in [-\rho, \rho]$ instead of $A_{ij} \in [-1, 1]$. What changes in the algorithm we need to make to ensure that if there is some x^* that satisfies $Ax^* \leq b$, we report an x such that $Ax \leq b + \epsilon$, otherwise report no such x exists.*

11.11 *Consider the following linear program (LP): $\min c^T x$, subject to $Ax \geq b$ and $x \geq 0$, where A is $n \times m$ matrix and c and b are vectors of length m and n, respectively, of real numbers. Define the* feasibility linear program *for a real number $F \geq 0$ as: $c^T x \leq F$, $Ax \geq b$ and $x \geq 0$.*

Answer the following.

1. *Show that the constraints $c^T x \leq F$, $Ax \geq b$ and $x \geq 0$ defines a polyhedron $\mathcal{P}$.*

2. *Show that if $\mathcal{P}$ is non-empty, the optimal value of LP is at most F.*

Let $w = (w_1, \ldots, w_n)$ be a vector of non-negative real numbers. Define an oracle polyhedron *$\mathcal{O}$ as $c^T x \leq F$, $w^T Ax \geq w^T b$ and $x \geq 0$.*

Answer the following.

1. *Show that $w^T Ax \geq w^T b$ is a single constraint.*

2. *Show that any feasible $x \in \mathcal{P}$ is also feasible for $\mathcal{O}$ for any non-negative vector w.*

Next we will show that if $x \in P$, then there exists an $x^* = (0, \ldots, 0, F/c_j, 0, \ldots, 0) \in O$. It will be easier to think in terms of the setup of the set cover problem. Columns of A correspond to sets $S_1, \ldots, S_m$, and rows correspond to elements $u_1, \ldots, u_n$ from an universe U. $A_{ij} = 1$ if $u_i \in S_j$, else it is 0. Weight w_i corresponds to the weight of the element u_i. Let $w(S_i)$ represent the sum of the weights of elements in the set S_i. Let j^* be the index of the set in $\{S_1, \ldots, S_m\}$ that maximizes $w(S_i)$. Let $x^* = (0, \ldots, 0, F/c_{j^*}, 0, \ldots, 0)$. I.e. j^*-th coordinate of x^* is F/c_{j^*}, and all other coordinates are 0. Answer the following.

1. *Show that $w^T Ax = \sum_{i \in [m]} x_i w(S_i)$.*

2. *Show that $c^T x^* \leq F$ and $x^* \geq 0$.*

3. *Show that $w^T Ax^* \geq w^T b$.*

4. *Conclude that $x^* \in O$.*

11.12 *Let $G = (V, E)$, $n = |V|$, $m = |E|$, be an undirected flow graph with unit capacities on edges and two distinguished vertices $s, t \in V$. Our task is find the maximum flow that can be routed from s to t respecting the flow and unit capacity constraints in G. We will use the MWU framework to design an approximation algorithm for the max flow problem. Let $\mathcal{P}$ denote the set of all possible paths between s and t. Note that any path in $\mathcal{P}$ can transmit a unit flow from s to t. Thus the flow problem reduces to finding maximum number of edge disjoint paths between s and t and we can express this problem as a linear program:*

$\max_{\pi \in \mathcal{P}} f_\pi$

For each edge $e \in E$, $\sum_{e \in f_\pi} f_\pi \leq 1$,

For all $\pi \in \mathcal{P}$, $f_\pi \geq 0$.

Note that f_π is an indicator variable indicating whether the path $\pi \in \mathcal{P}$ is selected or not for carrying the flow. The condition $\sum_{e \in f_\pi} f_\pi \leq 1$ indicates that the edge e can carry at most a flow of 1.

Following the previous exercise, the feasibility LP will be whether the flow value is at least F, i.e.

$\sum_{\pi \in \mathcal{P}} f_\pi \geq F$, $\sum_{e \in f_\pi} f_\pi \leq 1, \forall e \in E$, *and* $f_\pi \geq 0, \forall \pi \in \mathcal{P}$.

We can also define the Oracle LP by having a set of non-negative weights $w_1, \ldots, w_m$, *such that* $\sum_{\pi \in \mathcal{P}} f_\pi \geq F$, $\sum_{e \in E} w_e \sum_{e \in f_\pi} f_\pi \leq \sum_{e \in E} w_e$, *and* $f_\pi \geq 0, \forall \pi \in \mathcal{P}$. *Answer the following.*

1. *Find a shortest path (with respect to weights) $\pi \in \mathcal{P}$ from s to t in G. Set $f_\pi = F$ and for all other paths $\pi' \in \mathcal{P} \setminus \{\pi\}$, $f_{\pi'} = 0$. Show that these f-values of the path in $\mathcal{P}$ satisfies the oracle LP as long as oracle LP is feasible.*

2. *Show that we can design a polynomial time algorithm to satisfy the Oracle LP or report that it is infeasibile.*

3. *Assume that the feasibility LP has a solution. Note that the maximum flow that can go through an edge e is at most F in the Oracle LP. Show that by setting $\rho = F$, we can devise a MWU algorithm for finding flows over paths in $\mathcal{P}$ such that $\sum_{\pi \in \mathcal{P}} f_\pi = F$, $\sum_{e \in f_\pi} f_\pi \leq 1 + \epsilon, \forall e \in E$, and $f_\pi \geq 0, \forall \pi \in \mathcal{P}$.*

4. *What is the running time of the MWU algorithm?*

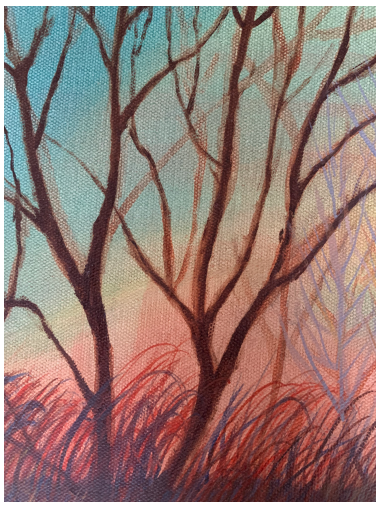
12
Dimensionality Reduction

We will focus on

1. Metric Spaces
2. Isometric Embeddings
3. Distortion
4. Universality of L_∞-normed spaces
5. Johnson-Lindenstrauss Lemma.

Keywords: Dimensionality Reduction, Metric Space, L_1, L_2 and L_∞-norm, Distortion, Isometric Embeddings, Johnson-Lindenstrauss Lemma.

In Chapter 4 we have seen various ways to express a matrix in terms of simpler matrices. The methods included how to express square matrices with linearly independent eigenvectors as $A = X\Lambda X^{-1}$, real symmetric matrices S as $Q\Lambda Q^T$, singular value decomposition of any matrix A as $U\Sigma V^T$, approximating any matrix A as a sum of tensor products, and the exercises included the PCA and its connection to SVDs. Note that matrix is an abstract mathematical structure that captures many realistic scenarios. For example, a matrix may represent users as rows and columns as movies, and the ij-th entry in the matrix represents the i-th users rating of the j-th movie. For example, as of December 2019, Netflix has about 160 million users (rows) and over 6000 movies (columns). This amounts to about 1 trillion entries in the matrix! Rather than working with this giant matrix, it is much better to work with its SVD and its variants (e.g. CUR decomposition). We can view these techniques as reducing the dimensionality of the problem. But in this chapter, we study a different type of dimensionality reduction centred around metric embeddings.



The material of this chapter is from Dasgupta and Gupta [44],

Dubhashi & Panconesi [53], Matousek [113, 112], Bourgain [25], and the lecture notes of Harold Racke on a course on metric embeddings offered at IIIT Chicago a while ago. We will touch upon the following topics: metric spaces, metric embedding, isometric and nonisometric embeddings via the concept of distortion - contraction and expansion. We will show that any n-point metric space $\langle X, d\rangle$ can be embedded isometrically into n-dimensional space with L_∞-norm. We will also show that metric spaces cannot be embedded isometrically always in the plane endowed with the Euclidean distance norm. We will conclude this chapter by sketching a proof of the Johnson-Lindenstrauss theorem [87] on embedding in Euclidean spaces.

12.1 *Preliminaries: Metric spaces and embeddings*

In this chapter the following three distance measures between a pair of points $p = (p_1, \ldots, p_k)$ and $q = (q_1, \ldots, q_k)$ in $\Re^k$ will be used.

1. L_2-norm (Euclidean): $||p - q||_2 = \sqrt{\sum_{i=1}^{k} (p_i - q_i)^2}$
2. L_1-norm (Manhattan): $||p - q||_1 = \sum_{i=1}^{k} |p_i - q_i|$
3. L_∞-norm: $||p - q||_\infty = \max\{|p_1 - q_1|, \ldots, |p_k - q_k|\}$

For example, for two points $p = (3, 5)$ and $q = (-2, 7)$ in $\Re^2$, we have

$||p - q||_1 = ||(3,5) - (-2,7)||_1 = |3 - (-2)| + |5 - 7| = 7.$
$||p - q||_2 = ||(3,5) - (-2,7)||_2 = \sqrt{(3 - (-2))^2 + (5 - 7)2} = \sqrt{29}.$
$||p - q||_\infty = ||(3,5) - (-2,7)||_\infty = \max\{|3 - (-2)|, |5 - 7|\} = 5$

Definition 12.1.1 *Let X be a set of n-points and let d be a distance measure associated with pairs of elements in X. We say that $\langle X, d\rangle$ is a finite metric space if the function d satisfies metric properties, i.e. (a) $\forall x \in X, d(x, x) = 0$, (b) $\forall x, y \in X, x \neq y, d(x, y) > 0$, (c) $\forall x, y \in X, d(x, y) = d(y, x)$ (symmetry), and (d) $\forall x, y, z \in X$, $d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality).*

For example, a finite set of points in $\Re^k$ where distances are measured with respect to L_2-norm (Euclidean) forms a metric space. A finite set of k-dimensional Boolean vectors where the distances are measured with respect to Hamming distance forms a metric space.

Definition 12.1.2 *Let $\langle X, d\rangle$ and $\langle X', d'\rangle$ be two metric spaces. A map $f : X \to X'$ is called an embedding. It is said to be isometric (i.e., distance preserving) if for all $x, y \in X$, $d(x, y) = d'(f(x), f(y))$. The contraction of f is defined to be maximum factor by which the distances*

shrink and it equals $\max_{x,y \in X} \frac{d(x,y)}{d'(f(x),f(y))}$. *Similarly the expansion of* f *is the maximum factor by which the distances are stretched and it equals* $\max_{x,y \in X} \frac{d'(f(x),f(y))}{d(x,y)}$. *Distortion of an embedding is defined to be the product of its expansion and contraction factor. For an isometric embedding, its contraction factor is* 1, *its expansion factor is* 1 *and therefore its distortion is* 1. *There are other equivalent definitions of distortion.*

We will try a couple of examples for embedding very specific metric spaces.

Example 12.1.3 *Consider the metric space defined by a complete graph on four vertices* $X = \{a, b, c, d\}$, *where the distance between every pair of distinct vertices is* 1. *Can we embed* X *in 3-dimensional Euclidean space isometrically?*

Map the points to the following coordinates $(0,0,0), (1,0,0), (1/2, \sqrt{3}/2, 0)$, *and* $(1/2, 1/2\sqrt{3}, \sqrt{2/3})$ *in 3-dimensional space and observe that the distance between every pair of points is* 1. *Therefore,* X *can be embedded isometrically in 3-dimensional Euclidean space.*

Example 12.1.4 *Let* $X = \{a, b, c, d\}$ *be a set of 4-points. Let* $d(a,b) = d(b,c) = d(c,d) = d(d,a) = 1$ *and* $d(a,c) = d(b,d) = 2$. *Is it possible to embed* X *isometrically in Euclidean space in any dimension?*

12.2 A Motivating Example

Assume that we have a set X of n-points in k-dimensional space, where $n >> 2^k$. We want to report a pair of points of X that maximizes the L_1-distance, i.e. we want to find the L_1-diameter of X.

A straightforward solution is as follows. Compute the distance between every pair of points and find the pair with the largest distance. Distance between a pair of points $p = (p_1, \ldots, p_k)$ and $q = (q_1, \ldots, q_k)$ requires the computation of $|p_1 - q_1| + \cdots + |p_k - q_k|$ and it can be computed in $O(k)$ time. There are in all $O(k\binom{n}{2})$ pairs of points in X. Thus, we can easily compute the L_1-diameter of X in $O(kn^2)$ time. Next, we will outline an algorithm using isometric embedding that takes $O(2^k n)$ time. This algorithm is very efficient if $n >> 2^k$.

We will define an isometric embedding f from points in $\Re^k$ to points in $\Re^{2^k}$. Let $x = (x_1, \ldots, x_k) \in X$. Note that $||x||_1 = \sum_{i=1}^{k} |x_i| = \sum_{i=1}^{k} \text{sign}(x_i) x_i = \text{sign}(x) \cdot x$, where $\text{sign}(x)$ is the ± 1 vector of length k denoting the sign of each coordinate of x.

Claim 12.2.1 *For any* ± 1 *vector* $y = (y_1, \ldots, y_k)$ *of length* k $||x||_1 = sign(x) \cdot x \geq y \cdot x$. *Moreover,* $||x||_1 = \max\{x \cdot y | y \in \{-1, 1\}^k\}$.

Example: For each possible ± 1 vector y in 3-dimensional space, the following table gives the dot product between y and $x = (-2, -3, 4)$. Observe from the table that $||x||_1 = |-2| + |-3| + |4| = (-1, -1, 1) \cdot (-2, -3, 4) = 9$

$y \cdot x$	$y \cdot x$
$(-\mathbf{1}, -\mathbf{1}, \mathbf{1}) \cdot (-\mathbf{2}, -\mathbf{3}, \mathbf{4}) = \mathbf{9}$	$(-1, -1, -1) \cdot (-2, -3, 4) = 1$
$(-1, 1, 1) \cdot (-2, -3, 4) = 3$	$(-1, 1, -1) \cdot (-2, -3, 4) = -5$
$(1, -1, 1) \cdot (-2, -3, 4) = 5$	$(1, -1, -1) \cdot (-2, -3, 4) = -3$
$(1, 1, 1) \cdot (-2, -3, 4) = -1$	$(1, 1, -1) \cdot (-2, -3, 4) = -9$

For each ± 1 vector y, define $f_y : X \to \Re$ by $f_y(x) = y \cdot x$. For example, $f_{(1,-1,1)}((-2, -3, 4)) = (1, -1, 1) \cdot (-2, -3, 4) = 5$

Isometric Embedding: Define $f : X \to \Re^{2^k}$ to be the concatenation of f_y's for all possible 2^k y's. For our example $x = (-2, -3, 4)$, $f(x) = (9, 3, 5, -1, 1, -5, -3, -9)$ corresponding to $2^3 = 8$ possible values for 3-dimensional vector y. Similarly for $x' = (2, 3, -2)$, $f(x') = (-7, -1, -3, 3, -3, 3, 1, 7)$.

Observe that $||f(x) - f(x')||_\infty = \max_y\{|f_y(x) - f_y(x')|\} = \max(|9 - (-7)|, |3 - (-1)|, |5 - (-3)|, |-1-3|, |1-(-3)|, |-5-3|, |-3-1|, |-9-7|) = 16 = ||x - x'||_1$. We formalize this in the following lemma.

Lemma 12.2.2 *Under the mapping $f : X \to \Re^{2^k}$ given by the concatenation of f_y's for all possible 2^k y's, where $f_y(x) = y \cdot x$, we have that for any two points $x, x' \in X$, $||f(x) - f(x')||_\infty = ||x - x'||_1$*

Proof.

$$\begin{aligned} ||f(x) - f(x')||_\infty &= \max_y\{|f_y(x) - f_y(x')|\} \\ &= \max_y\{|y \cdot x - y \cdot x'|\} \\ &= \max_y\{|y \cdot (x - x')|\} \\ &= ||x - x'||_1, \end{aligned}$$

because by Claim 12.2.1, $||x||_1 = \max\{y \cdot x | y \in \{-1, 1\}^k\}$. ∎

In place of finding the furthest pair of points in X with respect to L_1 metric we have the following new problem:

New Problem: Given n points in 2^k dimensional space X', find the furthest pair of points in X' with respect to L_∞ metric.

Observe the following:

$$\begin{aligned}\max_{u,v\in X'} ||u-v||_\infty &= \max_{u,v\in X'} \max_{i=1}^{2^k} |u_i - v_i| \\ &= \max_{i=1}^{2^k} \max_{u,v\in X'} |u_i - v_i|\end{aligned}$$

Now $\max_{u,v\in X'} |u_i - v_i|$, for a fixed i, can be computed in $O(n)$ time. Thus, $\max_{i=1}^{2^k} \max_{u,v\in X'} |u_i - v_i|$ can be computed in $O(2^k n)$ time. We conclude with the following theorem.

Theorem 12.2.3 *Given a set X of n points in $\Re^k$, by using the isometric embedding $f : L_1^k \to L_\infty^{2^k}$, we can compute the furthest pair of points in X with respect to L_1-metric by computing the furthest pair of points in the embedding with respect to L_∞-metric in $O(2^k n)$ time.*

12.3 Universal Space L_∞

Let $\langle X, d\rangle$ be any finite metric space, where $n = |X|$. In this section we show that X can be isometrically embedded into L_∞-metric space of dimension n. This shows that L_∞-metric space is *universal*, i.e. any metric space can be embedded isometrically in L_∞-metric space of an appropriate dimension.

Let $X = \{x_1, \ldots, x_n\}$. For each point $x \in X$, define $f(x) = (d(x,x_1), \ldots, d(x,x_n))$.

For the example in Figure 12.1, we have

$f(a) = (d(a,a), d(a,b), d(a,c), d(a,d)) = (0,2,1,3)$

$f(b) = (d(b,a), d(b,b), d(b,c), d(b,d)) = (2,0,3,5)$

$f(c) = (d(c,a), d(c,b), d(c,c), d(c,d)) = (1,3,0,3)$

$f(d) = (d(d,a), d(d,b), d(d,c), d(d,d)) = (3,5,3,0)$

Note that $d(b,d) = ||f(b) - f(d)||_\infty = 5$ and

$d(a,d) = ||f(a) - f(d)||_\infty = 3$.

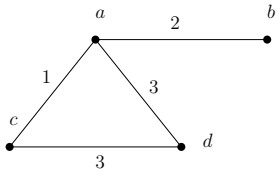


Figure 12.1: Metric Space $X = \{a,b,c,d\}$ with shortest path distances in the graph

Lemma 12.3.1 *Let $X = \{x_1, \ldots, x_n\}$. For each point $x \in X$, consider the mapping $f(x) = (d(x,x_1), \ldots, d(x,x_n))$. For any pair of points $u, v \in X$, we have $d(u,v) = ||f(u) - f(v)||_\infty$ and hence conclude that this mapping is universal.*

Proof. $||f(u) - f(v)||_\infty = \max_{x\in X} ||d(u,x) - d(v,x)|| \leq d(u,v)$ by triangle inequality.

But, $\max_{x\in X} ||d(u,x) - d(v,x)|| \geq ||d(u,u) - d(v,u)|| = d(u,v)$.

$\implies ||f(u) - f(v)||_\infty = d(u,v)$.

Thus, the mapping of elements of $x \in X$ given by $f(x) = (d(x,x_1), \ldots, d(x,x_n))$ under L_∞-norm is universal as it preserves the distances. ∎

12.4 *Embeddings into L_∞-normed spaces*

This section covers a theorem due to Matousek, see [1], using the proof technique of Bourgain [2]. Let $\langle X, d\rangle$ be a metric space where X is a set of n-points and let d satisfies the metric properties. We show that each point in X can be embedded in $k = O(Dn^{\frac{2}{D}} \log n)$-dimensional space such that the following holds. Let $x, y \in X$ and let $f(x), f(y)$ be their embedding in the k-dimensional space, respectively. We measure the distances in the k-dimensional space using the L_∞-norm. We show that the distances get distorted by a factor of at most $D \geq 1$. In fact, in this case, the mapping is a contraction, and the maximum amount that the distances can shrink is at most D. Succinctly this is specified as

$$\langle X, d\rangle \overset{D}{\hookrightarrow} L_\infty^{k=O(Dn^{\frac{2}{D}} \log n)}.$$

For the special case, $D = O(\log n)$ and $k = O(\log^2 n)$, i.e. $\langle X, d\rangle \overset{O(\log n)}{\hookrightarrow} L_\infty^{O(\log^2 n)}$

One may wonder why we should even worry about these embeddings. There are several applications. One easy application is the following. For an n-element set X, to represent the pair-wise distances, we will require $\Omega(n^2)$ space. In the embedding, each point requires space proportional to the dimension, i.e. $O(k)$. Thus the total storage requirement is $O(kn)$. For example, when $D = O(\log n)$, the space required is $O(n \log^2 n)$ instead of $O(n^2)$. Note that this is at the cost of replacing exact distances with approximate distances. Let us try to prove the following theorem in this section.

Theorem 12.4.1 $\langle X, d\rangle \overset{D}{\hookrightarrow} L_\infty^{k=O(Dn^{\frac{2}{D}} \log n)}$.

The proof is constructive and leads to a randomized algorithm for finding an embedding. First, we define the concept of a distance of a point $x \in X$ to a set $S \subseteq X$, as the distance of x to the nearest point in S. More formally,

Definition 12.4.2 *Let $S \subseteq X$. For $x \in X$, define $d(x, S) = \min_{z \in S} d(x, z)$.*

Claim 12.4.3 *Let $x, y \in X$. For all $S \subseteq X$, $|d(x, S) - d(y, S)| \leq d(x, y)$.*

Proof.
Proof uses triangle inequality. See Figure 12.2 for an illustration. Let x' be the point closest to x in S. Similarly, let y' be the point closest to y in S. Then $d(x, S) = d(x, x')$ and $d(y, S) = d(y, y')$. Assume that $d(x, x') \geq d(y, y')$. Then

$$d(x, x') - d(y, y') \leq d(x, y') - d(y, y') \leq d(x, y).$$

[1] Jiří Matoušek. On the distortion required for embedding finite metric spaces into normed spaces. *Israel Journal of Mathematics*, 93(1):333–344, 1996; and Jiri Matousek. *Lectures on Discrete Geometry*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002

[2] J. Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985

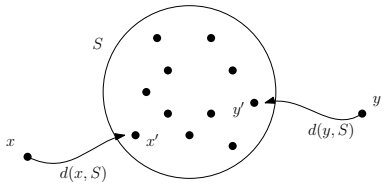


Figure 12.2: Illustration of $|d(x, S) - d(y, S)| \leq d(x, y)$.

Similarly, assume that $d(y,y') \geq d(x,x')$. Then

$$d(y,y') - d(x,x') \leq d(y,x') - d(x,x') \leq d(x,y).$$

Thus, $|d(x,x') - d(y,y')| \leq d(x,y)$. ∎

Definition 12.4.4 *(**Mapping**) Let $x \in X$. Let $S_1, S_2, \cdots, S_k \subseteq X$. The mapping f maps x to the point*

$$f(x) = \{d(x,S_1), d(x,S_2), \cdots, d(x,S_k)\}.$$

Claim 12.4.5 *Let $S_1, S_2, \cdots, S_k \subseteq X$. For $x, y \in X$, $||f(x) - f(y)||_\infty \leq d(x,y)$.*

Proof. Follows from the above claim, as for each $1 \leq i \leq k$, $|d(x,S_i) - d(y,S_i)| \leq d(x,y)$. ∎

Next we are going to construct a set of $k = O(Dm)$ subsets of X by a simple randomized selection process, where $m = O(n^{\frac{2}{D}} \log n)$. For any pair of points $x, y \in X$, we will show that there is at least one subset in this set for which $||f(x) - f(y)||_\infty \geq \frac{d(x,y)}{D}$. By Claim 12.4.5, we know that $||f(x) - f(y)||_\infty \leq d(x,y)$. Thus, this construction will show that the mapping has a distortion of at most D. Let us first present Algorithm 12.1. This algorithm returns the required mapping for each point $x \in X$.

Algorithm 12.1: Compute a set of $O(Dm)$ subsets of X and a mapping of each point $x \in X$

Input: Set X consisting of n-elements and a distortion parameter $D \geq 1$

Output: A set of $O(Dm)$ subsets of X

```
p ← min(1/2, n^{-2/D})
m ← O(n^{2/D} log n)
for j ← 1 to ⌈D/2⌉ do
    for i ← 1 to m do
        Choose set S_ij by sampling each element of X
        independently with probability p^j
    end
end
For each x ∈ X return its mapping f(x) as the point
(d(x,S_11), ··· d(x,S_m1), d(x,S_12), ··· , d(x,S_m2), ··· , d(x,S_{1⌈D/2⌉}), ··· , d(x,S_{m⌈D/2⌉}))
```

Before we show that the subsets produced by Algorithm 12.1 satisfies $||f(x) - f(y)||_\infty \geq \frac{d(x,y)}{D}$ for any pair of points $x, y \in X$, we make the following observation.

Observation 12.4.6 *Let x, y be two distinct points of X. Let $B(x,r)$ be the set of points of X that are within a distance of r from x (think of $B(x,r)$ as a ball of radius r centred at x). Similarly, let $B(y, r+\Delta)$ be the set of points of X that are within a distance of $r+\Delta$ from y. Consider a subset $S \subset X$ such that $S \cap B(x,r) \neq \emptyset$ and $S \cap B(y, r+\Delta) = \emptyset$. Then $|d(x,S) - d(y,S)| \geq \Delta$.*

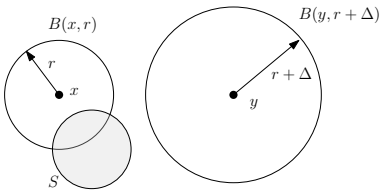


Figure 12.3: Illustration of Observation 12.4.6.

Proof. $d(x,S) \leq r$ as $S \cap B(x,r) \neq \emptyset$ and $d(y,S) > r+\Delta$ as $S \cap B(y, r+\Delta) = \emptyset$. ∎

Lemma 12.4.7 *Let x, y be two distinct points of X. There exists an index $j \in \{1, \cdots, \lceil \frac{D}{2} \rceil\}$ such that if S_{ij} is as chosen in Algorithm 12.1, than*

$$Pr\big[||f(x) - f(y)||_\infty \geq \frac{d(x,y)}{D}\big] \geq \frac{p}{12}.$$

First, let us see why the above lemma implies Theorem 12.4.1. For a fix, $x, y \in X$, the probability that none of the m trials for that particular j are good has probability of at most $(1 - \frac{p}{12})^m \leq e^{-\frac{pm}{12}} \leq \frac{1}{n^2}$. Since there are in all $\binom{n}{2}$ pairs in X, the probability that we fail to choose a good set for any of the pairs, by the union bound, is strictly less than 1. Now let us prove Lemma 12.4.7.

Proof. (Proof of Lemma 12.4.7) Set $\Delta = \frac{d(x,y)}{D}$. For $i = 0, \cdots, \lceil \frac{D}{2} \rceil$, define balls of radius $i\Delta$ as follows. Let $B_0 = \{x\}$. Let B_1 be the ball of radius Δ centred at y. Then B_2 is the ball of radius 2Δ centred at x. B_3 is the ball centred at y of radius 3Δ and so on. Hence, all balls with even i's are centred at x and at odd i's are centred at y. Since $i \leq D/2$, no even ball overlaps with an odd balls. For even (odd) i, let $|B_i|$ denote the number of points of X that are within a distance of at most $i\Delta$ from x (respectively, y). Next we claim that, there is an index $t \in \{0, \cdots, \lceil \frac{D}{2} \rceil - 1\}$, such that $|B_t| \geq n^{\frac{2t}{D}}$ and $|B_{t+1}| \leq n^{\frac{2(t+1)}{D}}$.

Claim 12.4.8 *There is an index $t \in \{0, \cdots, \lceil \frac{D}{2} \rceil - 1\}$, such that $|B_t| \geq n^{\frac{2t}{D}}$ and $|B_{t+1}| \leq n^{\frac{2(t+1)}{D}}$.*

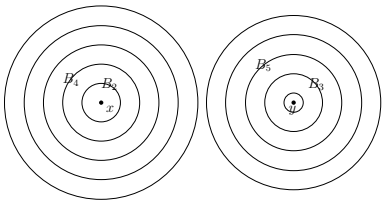


Figure 12.4: Illustration of Claim

Proof. Proof is by contradiction. $|B_0| = 1$ by construction. Thus $|B_1| > n^{\frac{2}{D}}$, otherwise the claim holds.

Since $|B_1| > n^{\frac{2}{D}}$, $|B_2| > n^{\frac{4}{D}}$, otherwise the claim holds.

Since $|B_2| > n^{\frac{4}{D}}$, $|B_3| > n^{\frac{6}{D}}$, otherwise the claim holds.

Continuing this way, for the last possible value of $t = \lceil \frac{D}{2} \rceil - 1$, we obtain $|B_{t+1}| > n^{\frac{2(t+1)}{D}} \geq n$. Since $|X| = n$, this is impossible, and hence there exists an index t that satisfies the statement of the claim. ∎

Consider the index t as stated in this claim. We have two balls, B_t and B_{t+1}, such that $|B_t| \geq n^{\frac{2t}{D}}$ and $|B_{t+1}| \leq n^{\frac{2(t+1)}{D}}$. We will next show

that for $j = t+1$, in Algorithm 12.1, the set S_{ij} chosen by the algorithm will have a non-empty intersection with B_t with probability at least $p/3$, and it will avoid B_{t+1} with probability at least $1/4$. Formally, consider the following two events:

Let E_1 be the event that $S_{ij} \cap B_t \neq \emptyset$.

Let E_2 be the event that $S_{ij} \cap B_{t+1} = \emptyset$.

Let us first analyze E_1.

$$Pr[E_1] = 1 - Pr[S_{ij} \cap B_t = \emptyset] \tag{12.1}$$
$$= 1 - (1-p^j)^{|B_t|} \tag{12.2}$$
$$\geq 1 - (1-p^j)^{n^{\frac{2(j-1)}{D}}} \tag{12.3}$$
$$\geq 1 - e^{-p^j n^{\frac{2(j-1)}{D}}} \tag{12.4}$$
$$= 1 - e^{-p} \tag{12.5}$$

If $p \leq 1/2$, $1 - e^{-p} \geq p/3$.

Similarly,

$$Pr[E_2] = Pr[S_{ij} \cap B_{t+1} = \emptyset] \tag{12.6}$$
$$= (1-p^j)^{|B_{t+1}|} \tag{12.7}$$
$$\geq (1-p^j)^{n^{\frac{2j}{D}}} \tag{12.8}$$
$$= (1-p^j)^{\frac{1}{p^j}} \tag{12.9}$$

If $p^j \leq 1/2$, $(1-p^j)^{\frac{1}{p^j}} \geq 1/4$. Note that the function $(1-p^j)^{\frac{1}{p^j}}$ achieves its minimum value in the interval $0 \leq p^j \leq 1/2$, at the ends of the interval, i.e. at $p^j = 0$ or at $p^j = 1/2$. At both the extremes, its value is at least $1/4$.

Next we need to estimate what is the $Pr[E_1 \wedge E_2]$. Note that events E_1 and E_2 are independent as the balls B_t and B_{t+1} are disjoint and do not share any points. Thus $Pr[E_1 \wedge E_2] \geq p/12$. ∎

By setting $D = \Theta(\log n)$, in Theorem 12.4.1, we obtain the following corollary.

Corollary 12.4.9 $\langle X, d \rangle \overset{\Theta(\log n)}{\hookrightarrow} L_\infty^{O(\log^2 n)}$.

Next we show the following

Lemma 12.4.10 $\langle X, d \rangle \overset{\log^2 n}{\hookrightarrow} L_1^{O(\log^2 n)}$.

Proof. Let $k = O(\log^2 n)$ be the dimension of embedding in Corollary 12.4.9. Observe that for the same embedding as in Corollary 12.4.9, for a pair of points $x, y \in X$, we have

$$||f(x) - f(y)||_1 \leq kd(x,y).$$

This follows from the fact that for each of the coordinates, $|f(x)_i - f(y)_i| \leq d(x,y)$ by Claim 12.4.3. In the proof of Theorem 12.4.1, for a pair $x, y \in X$, we know that there is at least one set (as constructed in Algorithm 12.1) which is good, i.e., with probability at least $1 - 1/n^2$ for which $||f(x) - f(y)||_\infty \geq \frac{d(x,y)}{\Theta(\log n)}$. We can extend the machinery in the Theorem to show that with high probability there are $\log n$ sets which are good. This will require to choose slightly larger value for m, but still of order of $O(\log n)$. If this is the case, then

$$||f(x) - f(y)||_1 \geq \log n \frac{d(x,y)}{\Theta(\log n)} = \Theta(d(x,y)).$$

Thus we have

$$\Theta(d(x,y)) \leq ||f(x) - f(y)||_1 \leq kd(x,y),$$

and hence we have a mapping with distortion $O(\log^2 n)$. ∎

Next we show,

Corollary 12.4.11 $\langle X, d \rangle \overset{\log^{1.5} n}{\hookrightarrow} L_2^{O(\log^2 n)}$.

Proof. Let $k = O(\log^2 n)$ be the dimension of embedding in Corollary 12.4.9. Observe that for the same embedding as in Corollary 12.4.9, for a pair of points $x, y \in X$, we have

$$||f(x) - f(y)||_2 = \sqrt{\sum (d(x, S_{ij}) - d(y, S_{ij}))^2} \leq \sqrt{k} d(x,y).$$

With similar arguments as in the proof of Lemma 12.4.10,

$$||f(x) - f(y)||_2 = \sqrt{\sum (d(x, S_{ij}) - d(y, S_{ij}))^2} \geq \sqrt{\log n (\frac{d(x,y)}{\Theta(\log n)})^2} \geq \frac{d(x,y)}{\Theta(\sqrt{\log n})}.$$

This results in a total distortion of $O(\log^{1.5} n)$. ∎

12.5 *Johnson and Lindenstrauss Theorem*

First we recall some facts about normal distribution from Chapter 2. A random variable X has a normal distribution $\mathcal{N}(\mu, \sigma^2)$, with mean μ and standard deviation $\sigma > 0$, if its probability density function is of the form $f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}, -\infty < x < \infty$. The distribution $\mathcal{N}(0,1)$, with pdf $\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$, is referred to as the *standardized normal distribution*.

If X has a Normal distribution $\mathcal{N}(\mu, \sigma^2)$, than $aX + b$ has a Normal distribution $\mathcal{N}(a\mu + b, a^2\sigma^2)$, for constants a, b. Let X and Y be independent r.v. with Normal distributions $\mathcal{N}(\mu_1, \sigma_1^2)$ and $\mathcal{N}(\mu_2, \sigma_2^2)$. Let

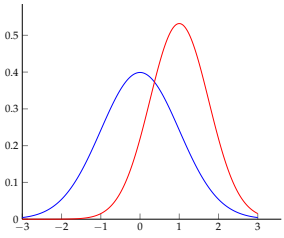


Figure 12.5: Plot of $\mathcal{N}(0,1)$ and $\mathcal{N}(1, 0.75)$

random variable $Z = X + Y$. It is well-known that Z has a Normal distribution $\mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$, i.e., sum of independent Normal distributions is a Normal distribution.

Next we state the dimensionality reduction theorem of Johnson and Lindenstrauss [3] from 1984.

[3] William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (in honor of Professor Shizuo Kakutani, held on June 8-11, 1982, at Yale University, New Haven, Connecticut)*, pages 189–206. 1984

Theorem 12.5.1 *Let V be a set of n points in d-dimensions. A mapping $f : \mathbb{R}^d \to \mathbb{R}^k$ can be computed, in randomized polynomial time, so that for all pairs of points $u, v \in V$,*

$$(1 - \epsilon)||u - v||^2 \leq ||f(u) - f(v)||^2 \leq (1 + \epsilon)||u - v||^2,$$

where $0 < \epsilon < 1$ and n, d, and $k \geq 4(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3})^{-1} \ln n$ are positive integers.

Intuitively, the theorem says that points in d-dimensional space can be mapped to a k dimensional space, where k is significantly less than d, and the interpoint Euclidean distance between a pair of points in V is preserved up to a factor of $(1 \pm \epsilon)$. The function f is defined in terms of a matrix $A_{k \times d}$ with entries from Normal distribution $\mathcal{N}(0, \frac{1}{k})$. A point $x \in \Re^d$ is mapped to the point $x' = Ax$. Note that Ax represents the product of $k \times d$ matrix with a vector x of dimension d and the product results in a vector $x' \in \Re^k$. For every pair of points $u, v \in V$, we consider the vector $x = u - v$, and apply the mapping f to x. Next we show that the expected squared length of the vector $||Ax||^2$ is $||x||^2$.

Lemma 12.5.2 *Let $A_{k \times d}$ be a matrix, where each of its entry is chosen independently from the Normal distribution $\mathcal{N}(0, \frac{1}{k})$. For any vector $x \in \Re^d$, we have $E[||Ax||^2] = ||x||^2$.*

Proof. Assume $z = Ax$, where $z = (z_1, \ldots, z_k) \in \Re^k$. We want to show that $E[||z||^2] = ||x||^2$. Note that $||z||^2 = \sum_{i=1}^{k} z_i^2$. Consider the first coordinate z_1 of z. Note that $z_1 = \sum_{i=1}^{d} A_{1i} x_i$. What is the distribution of r.v. z_1?

Recall that if a random variable X has a Normal distribution $\mathcal{N}(0, \sigma^2)$, aX has a Normal distribution $\mathcal{N}(0, a^2\sigma^2)$, for a constant a. Moreover, the sum of two independent r.v. with Normal distributions $\mathcal{N}(0, \sigma_1^2)$ and $\mathcal{N}(0, \sigma_2^2)$ has a Normal distribution $\mathcal{N}(0, \sigma_1^2 + \sigma_2^2)$.

Since each A_{1i} is distributed independently by $\mathcal{N}(0, \frac{1}{k})$. The distribution of $z_1 = \sum_{i=1}^{d} A_{1i} x_i$ is the same as the sum of d independent Normal distributions (where each of them have an associated scalar x_i). Thus, z_1 has $\mathcal{N}(0, \frac{\sum_{i=1}^{d} x_i^2}{k}) = \mathcal{N}(0, \frac{||x||^2}{k})$ distribution.

Consider $||z||^2 = ||Ax||^2 = z_1^2 + \ldots + z_k^2$, where z_i has $\mathcal{N}(0, \frac{||x||^2}{k})$ distribution. Now, by linearity of expectation and the independence of each entry in A, we have $E[||z^2||] = E[z_1^2 + \ldots + z_k^2] = kE[z_1^2]$.

By definition: $Var[z_1] = E[z_1^2] - E[z_1]^2$.

But z_1 has $\mathcal{N}(0, \frac{||x||^2}{k})$ distribution.

$\implies Var[z_1] = \frac{||x||^2}{k}$ and $E[z_1] = 0$.

$\implies E[z_1^2] = Var[z_1] = \frac{||x||^2}{k}$ ∎

Next we show that $E[||Ax||^2]$ is concentrated around $||x||^2$. We will estimate $Pr(||Ax||^2 \geq (1+\epsilon)||x||^2)$ and $Pr(||Ax||^2 \leq (1-\epsilon)||x||^2)$, for $\epsilon \in (0,1)$.

We know that $Pr(||Ax||^2 \geq (1+\epsilon)||x||^2) = Pr(\sum_{i=1}^{k} z_i^2 \geq (1+\epsilon)||x||^2)$, where z_i is a random variable with distribution $\mathcal{N}(0, \frac{||x||^2}{k})$. Set $Y_i = \frac{\sqrt{k}}{||x||} z_i$. Since z_i has distribution $\mathcal{N}(0, \frac{||x||^2}{k})$, Y_i has distribution $\mathcal{N}(0,1)$. In the expression $Pr(\sum_{i=1}^{k} z_i^2 \geq (1+\epsilon)||x||^2)$, divide by $\frac{||x||^2}{k}$, and we obtain $Pr(\sum_{i=1}^{k} Y_i^2 \geq (1+\epsilon)k)$.

Now our problem reduces to estimating $Pr(\sum_{i=1}^{k} Y_i^2 \geq (1+\epsilon)k)$, where Y_i has a $\mathcal{N}(0,1)$ distribution. Next we prove the following

Lemma 12.5.3 *Let $Y_1, \ldots, Y_k$ are k-independent random variables, where each Y_i has a $\mathcal{N}(0,1)$ distribution. The following holds*

These are Chernoff style bounds.

(a) $Pr(\sum_{i=1}^{k} Y_i^2 \geq (1+\epsilon)k) \leq e^{-\frac{k}{4}(\epsilon^2 - \epsilon^3)}$.

(b) $Pr(\sum_{i=1}^{k} Y_i^2 \leq (1-\epsilon)k) \leq e^{-\frac{k}{4}(\epsilon^2 - \epsilon^3)}$.

We will prove the first part and leave the second part as an exercise. First we state a useful identity, see [4], before proceeding with the proof of the lemma.

[4] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Struct. Algorithms*, 22(1):60–65, 2003; and D.P. Dubhashi and A. Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009

Lemma 12.5.4 *(A useful identity) Let X be a random variable distributed $\mathcal{N}(0,1)$ and $\lambda < \frac{1}{2}$ be a constant. Then, $E\left[e^{\lambda X^2}\right] = \frac{1}{\sqrt{1-2\lambda}}$*

Proof. Recall that the probability density function of the standard normal distribution is $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$.

By definition, $E[H(x)] = \int_{-\infty}^{+\infty} H(x)f(x)dx$. Thus,

$$\begin{aligned} E\left[e^{\lambda X^2}\right] &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{\lambda x^2} e^{-\frac{x^2}{2}} dx \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-(1-2\lambda)\frac{x^2}{2}} dx \end{aligned}$$

Substitute $y = x\sqrt{1-2\lambda}$, and we obtain

$$E\left[e^{\lambda X^2}\right] = \frac{1}{\sqrt{1-2\lambda}} \left[\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\frac{y^2}{2}} dy\right]$$

But, $\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\frac{y^2}{2}} dy = 1$, as this is the area under the Normal distribution curve. ∎

Proof. (of Part (a) of Lemma 12.5.3)

$$\begin{aligned} Pr(\sum_{i=1}^{k} Y_i^2 \geq (1+\epsilon)k) &= Pr(e^{\lambda \sum_{i=1}^{k} Y_i^2} \geq e^{(1+\epsilon)\lambda k}) \text{ (for } \lambda > 0) \\ &\leq \frac{E\left[e^{\lambda \sum_{i=1}^{k} Y_i^2}\right]}{e^{(1+\epsilon)\lambda k}} \text{ (applying Markov's Inequality)} \\ &= \frac{E\left[e^{\lambda Y_1^2}\right]^k}{e^{(1+\epsilon)\lambda k}} \text{ (Independence of } Y_i\text{'s)} \\ &= e^{-(1+\epsilon)k\lambda} \left(\frac{1}{\sqrt{1-2\lambda}}\right)^k \text{ (using the identity from Lemma 12.5.4)} \end{aligned}$$

Set $\lambda = \frac{\epsilon}{2(1+\epsilon)}$, and we have

$$\begin{aligned} Pr(\sum_{i=1}^{k} Y_i^2 \geq (1+\epsilon)k) &\leq e^{-(1+\epsilon)k\lambda} \left(\frac{1}{\sqrt{1-2\lambda}}\right)^k \\ &= e^{-\frac{\epsilon}{2}k} (1+\epsilon)^{\frac{k}{2}} \\ &= \left((1+\epsilon)e^{-\epsilon}\right)^{\frac{k}{2}} \\ &\leq e^{-\frac{k}{4}(\epsilon^2-\epsilon^3)} \text{ (as } 1+\epsilon \leq e^{\epsilon - \frac{\epsilon^2-\epsilon^3}{2}}) \end{aligned}$$

For Part (b) of the proof we have,

$$
\begin{aligned}
Pr(\sum_{i=1}^{k} Y_i^2 \leq (1-\epsilon)k) &= Pr(e^{-\lambda \sum_{i=1}^{k} Y_i^2} \geq e^{-(1-\epsilon)\lambda k}) \text{ (for } \lambda > 0) \\
&\leq \frac{E\left[e^{-\lambda \sum_{i=1}^{k} Y_i^2}\right]}{e^{-(1-\epsilon)\lambda k}} \text{ (applying Markov's Inequality)} \\
&= \frac{E\left[e^{-\lambda Y_1^2}\right]^k}{e^{-(1-\epsilon)\lambda k}} \text{ (Independence of } Y_i\text{'s)} \\
&= e^{(1-\epsilon)k\lambda}\left(\frac{1}{\sqrt{1+2\lambda}}\right)^k \text{ (from Lemma 12.5.4, where } E\left[e^{\lambda X^2}\right] = \frac{1}{\sqrt{1+2\lambda}})
\end{aligned}
$$

Now use $\lambda = \frac{\epsilon}{2(1-\epsilon)}$ and the inequality that $\ln(1-x) < -x - \frac{x^2}{2}$ and derive the proof of Part (b). ∎

Corollary 12.5.5 *If $k = c\frac{\ln n}{\epsilon^2}$, for some constant c,*

$$Pr((1-\epsilon)k \leq \sum_{i=1}^{k} Y_i^2 \leq (1+\epsilon)k) \geq 1 - \frac{1}{n^3}$$

Proof. From Lemma 12.5.3 we have that

$Pr(\sum_{i=1}^{k} Y_i^2 \geq (1+\epsilon)k) \leq e^{-\frac{k}{4}(\epsilon^2-\epsilon^3)}$ and $Pr(\sum_{i=1}^{k} Y_i^2 \leq (1-\epsilon)k) \leq e^{-\frac{k}{4}(\epsilon^2-\epsilon^3)}$.

Hence $Pr\left((\sum_{i=1}^{k} Y_i^2 \geq (1+\epsilon)k) \vee (\sum_{i=1}^{k} Y_i^2 \leq (1-\epsilon)k)\right) \leq 2e^{-\frac{k}{4}(\epsilon^2-\epsilon^3)}$ (by Union Bound)

Thus, $Pr((1-\epsilon)k \leq \sum_{i=1}^{k} Y_i^2 \leq (1+\epsilon)k) \geq 1 - 2e^{-\frac{k}{4}(\epsilon^2-\epsilon^3)}$

Substituting, $k = c\frac{\ln n}{\epsilon^2}$ we have that $Pr((1-\epsilon)k \leq \sum_{i=1}^{k} Y_i^2 \leq (1+\epsilon)k) \geq 1 - \frac{1}{n^3}$. ∎

Now we restate the Johnson and Lindenstrauss theorem and complete its proof.

Theorem 12.5.6 *Let V be a set of n points in d-dimensions. A mapping $f : \mathbb{R}^d \to \mathbb{R}^k$ can be computed, in randomized polynomial time, so that for all pairs of points $u, v \in V$,*

$$(1-\epsilon)||u-v||^2 \leq ||f(u)-f(v)||^2 \leq (1+\epsilon)||u-v||^2,$$

where $0 < \epsilon < 1$ and n, d, and $k \geq 4(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3})^{-1} \ln n$ are positive integers.

Proof. By choosing matrix $A_{k\times d}$ consisting of independent values from $\mathcal{N}(0, \frac{1}{k})$, we show that $\forall u, v \in V\ Pr((1-\epsilon)||u-v||^2 \leq ||Au - Av||^2 \leq (1+\epsilon)||u-v||^2) \geq 1 - \frac{1}{n}$

By Corollary 1, we know that for any vector $x \in R^d$, $Pr((1-\epsilon)||x||^2 \leq ||Ax||^2 \leq (1+\epsilon)||x||^2) \geq 1 - \frac{1}{n^3}$. Consider any pair of points $u, v \in V$. Set $x = u - v$. Then $Pr((1-\epsilon)||u-v||^2 \leq ||A(u-v)||^2 \leq (1+\epsilon)||u-v||^2) \geq 1 - \frac{1}{n^3}$.

There are in all $\binom{n}{2}$ pairs of points in V.

By union bound, we have that $\forall u, v \in V$ $Pr((1-\epsilon)||u-v||^2 \leq ||Au - Av||^2 \leq (1+\epsilon)||u-v||^2) \geq 1 - \frac{1}{n}$. ∎

We make some concluding remarks:

1. Choice of the matrix A doesn't depend on points in V.
2. We required that the matrix A satisfy $E[||Ax||^2] = ||x||^2$.
3. Note that A is a very dense matrix, and thus the computation of Av takes a lot of computation time.
4. We can choose entries of A from $\{-1, 1, 0\}$ with probabilities 1/6,1/6, and 2/3, respectively, and normalize. It turns out that this choice of A also works, see [5]. Note that in this case, the number of non-zero entries of A is approximately 1/3rd of the total number of entries.

[5] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003. Special Issue on PODS 2001

12.6 Exercises

12.1 *Show that for any embedding f of a metric space (X, d) to another metric space (X', d'), the distortion of f is at least 1.*

12.2 *Fill in the missing details to prove Corollary 12.4.11.*

12.3 *Adapt proof of Lemma 12.5.4 to show the following: Let X be a random variable distributed $\mathcal{N}(0,1)$ and $\lambda \geq 0$ be a constant. Then, $E\left[e^{-\lambda X^2}\right] = \frac{1}{\sqrt{1+2\lambda}}$.*

12.4 *Complete the proof of the second part of Lemma 12.5.3.*

12.5 *Let $X = \{a, b, c, d\}$ be a set of 4-points. Let $d(a,b) = d(b,c) = d(c,d) = d(d,a) = 1$ and $d(a,c) = d(b,d) = 2$. Is it possible to embed X isometrically in Euclidean space in any dimension?*

12.6 *Let $X = \{a, b, c, d\}$ be a set of 4-points. Let $d(a,b) = d(a,c) = d(a,d) = 1$ and $d(b,c) = d(b,d) = d(c,d) = 2$. Is it possible to embed X isometrically in Euclidean space in any dimension?*

12.7 *Let $T = (V, E)$ be a tree on $N \geq 3$ vertices. Answer the following*

1. *Show that there exist subtrees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ that share a single vertex ($|V_1 \cap V_2| = 1$ and $V_1 \cup V_2 = V$) and no edges ($E_1 \cap E_2 = \emptyset$), such that $|V_1| \leq 1 + \frac{2}{3}n$ and $|V_2| \leq 1 + \frac{2}{3}n$.*

2. *Consider the metric space $\langle X, d\rangle$ defined by the tree T, where $X = V$ and for any pair of vertices $u, v \in V$, $d(u, v)$ is the number of edges in the (unique) shortest path between u and v in T. Show that $\langle X, d\rangle$ can be embedded isometrically in $O(\log n)$ dimensional L_∞-normed space (i.e., $\langle X, d\rangle \overset{1}{\hookrightarrow} L_\infty^{O(\log n)}$).*
   Hint: The isometric embedding will associate $O(\log n)$ coordinates to each vertex of T. Consider using the distances from the vertex common to V_1 and V_2 in Part (a) to vertices in T as one of the coordinates.
   You may assume T is binary, though the statement holds also for non-binary trees.

12.8 *Suppose we are given a set S of n-points in d dimensional space, where is d is very large. We are interested in computing a Euclidean minimum spanning tree (EMST) of S. Show how you can use the Johnson-Lindenstrauss theorem to compute an approximate EMST of S. What will be the running time of the algorithm for computing an approximate EMST?*

13
Approximation Algorithms Design Techniques

——— UNDER CONSTRUCTION ———

We will focus on techniques for designing approximation algorithms. They are illustrated using classic examples. We will also elaborate on some methods for developing fixed-parameter tractable algorithms for combinatorial optimization problems.

1. Greedy
   - Maximum Weight Independent Sets using Recoverable Values
   - Vertex Cover
2. Local Search
   - Maximum Cut
   - k-Median
   - Geometric Hitting Sets
3. Metric Linear Programs
   - Min Cost st-cuts
   - Multicuts
   - Multiway Cuts
4. FPT

The material of this chapter is based on numerous sources, including lecture notes of Gupta, Ravi, and Dinitz; books of [145, 51, 40]; articles by [64, 63, 32, 55, 12]; and several video lectures.

13.1 Greedy Algorithms

We illustrate the technique using the vertex cover and the maximum weight-independent set problem in graphs.

13.1.1 Vertex Cover

The vertex cover problem is formally stated as follows.
Input: A simple undirected graph $G = (V, E)$.
Output: A subset $S \subseteq V$ of smallest cardinality such that for each edge $e = (u, v) \in E$, at least one of u or v is in S.

Our approximation algorithm for vertex cover uses maximal matching in a graph. A *matching* $M \subseteq E$ in $G = (V, E)$ is a collection of edges so that no two edges in M are incident to the same vertex. Matching M is *maximal*, if every other edge in $E \setminus M$ shares an endpoint with some edge in M. The algorithm is as follows.

Vertex Cover Using Maximal Matching

1. Compute a maximal matching M of G by greedily adding edges to M so that no two edges in M are incident to the same vertex.
2. Let $S \subseteq V$ be the set of vertices incident on the edges in M.
3. Return S as an approximation to the vertex cover of G.

We have the following observations.

Observation 13.1.1

1. *Any optimal vertex cover $S^* \subseteq V$ of G satisfies $|S^*| \geq |M|$.*
2. *Set of vertices in S forms a vertex cover of G. Moreover, the graph $G \setminus S$ is an independent set.*
3. *The set S returned by the algorithm satisfies $|S| = 2|M| \leq 2|S^*|$.*

Theorem 13.1.2 *The above greedy algorithm is a 2-approximation algorithm for the vertex cover problem. The algorithm runs in $O(|V| + |E|)$ time.*

13.1.2 Maximum Weight Independent Set

Consider the following classical problem of computing a maximum weight-independent set in a graph. The problem is formally defined as follows:

Input: An undirected graph $G = (V, E)$ where each vertex has a positive weight $w : V \to \Re^+$.
Output: A subset $S \subseteq V$ such that

(a) Independent: No two vertices in S are connected by an edge

(b) Maximality: Among all such independent sets, S has the maximum total weight, where $wt(S) = \sum_{s \in S} w(s)$.

The complexity status of this problem is as follows.

- The decision version of the MWIS problem is **NP**-Hard, both for the unweighted and weighted graphs.
- **NP**-Hard for cubic graphs, i.e., the graphs where the degree of each vertex is three.
- **NP**-Hard to approximate within a factor of $n^{1-\epsilon}$, for any $0 < \epsilon < 1$, [1].
- Can be solved in linear time for some graph classes, including trees and bounded tree-width graphs.

[1] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001

We will first see a greedy randomized algorithm.

Greedy randmized algorithm for MWIS

Input: Graph $G = (V, E)$ on n vertices with $w : V \to \Re^+$.
Output: A set S that approximates the MWIS.

Step 1: Compute an ordering of vertices in V by using a uniform at random permutation. WLOG, let the ordering be $(v_1, \ldots, v_n)$.

Step 2: $S \leftarrow \emptyset$

Step 3: For each vertex v_i in order do

If none of its neighbors are in S, $S \leftarrow S \cup \{v_i\}$

Step 4: Return S

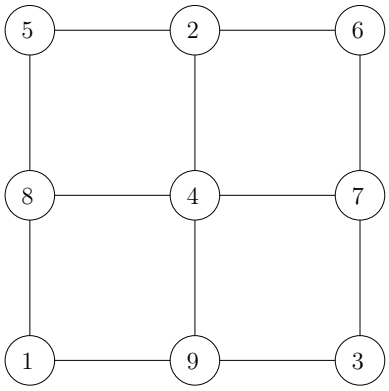


Figure 13.1: Order of the vertex is indicated in the circle. The algorithm reports $S = \{1, 2, 3\}$.

For an illustration, see Figure 13.1. We make the following observations.

Observation 13.1.3

1. *The set of vertices in S forms an independent set of G.*
2. *The algorithm is oblivious to the weights of vertices.*
3. *The algorithm runs in $O(|V| + |E|)$ time.*

Lemma 13.1.4 *Let $v \in V$ be an arbitrary vertex of G and let its degree be $deg(v)$. Then*

$$Pr(v \in S) \geq \frac{1}{deg(v)+1}$$

where probability is over the random orderings of vertices in V.

Proof. We know that a vertex v is placed in S if none of v's neighbors come before v in the ordering. This occurs with probability $\frac{1}{deg(v)+1}$.

Moreover, a neighbor w of v may come before v in the ordering, but it wasn't placed in S as one of w's neighbors (other than v) was in S. Thus, $Pr(v \in S) \geq \frac{1}{deg(v)+1}$. ∎

Lemma 13.1.5 $E\left[\sum_{v \in S} w(v)\right] \geq \sum_{v \in V} \frac{w(v)}{deg(v)+1}$

Proof. We set up an indicator random variable X_v for each vertex $v \in V$, where $X_v = \begin{cases} 1, \text{ if } v \in S \\ 0, \text{ otherwise} \end{cases}$

Note that $E[X_v] = Pr(X_v = 1) = Pr(v \in S) \geq \frac{1}{deg(v)+1}$.

Using the linearity of expectation, we have

$$\begin{aligned} E\left[\sum_{v \in S} w(v)\right] &= E\left[\sum_{v \in V} X_v w(v)\right] \\ &= \sum_{v \in V} E\left[X_v w(v)\right] \\ &= \sum_{v \in V} w(v) E\left[X_v\right] \\ &\geq \sum_{v \in V} \frac{w(v)}{deg(v)+1} \end{aligned}$$

∎

Remark 13.1.6 *If max degree of vertices in G is $\leq \Delta$,*
$E\left[\sum_{v \in S} w(v)\right] \geq \frac{1}{\Delta+1} \sum_{v \in V} w(v)$

Remark 13.1.7 *Let I be any independent set of G. Then*
$E\left[\sum_{v \in S} w(v)\right] \geq \sum_{v \in V} \frac{w(v)}{deg(v)+1} \geq \sum_{v \in I} \frac{w(v)}{deg(v)+1}$

Remark 13.1.8 *Let I^* be a maximum weight independent set of G. Then*
$E\left[\sum_{v \in S} w(v)\right] \geq \sum_{v \in I^*} \frac{w(v)}{deg(v)+1}$

The greedy randomized algorithm gives us

$$E\left[\sum_{v \in S} w(v)\right] \geq 1 \cdot \sum_{v \in I^*} \frac{w(v)}{deg(v)+1},$$

where I^* is a maximum weight independent set of G. The value 1 is called the *recoverable value*. We will see a method of Feige and Reichman [2] where 1 is replaced by 2. This results in a better estimate of $E\left[\sum_{v\in S} w(v)\right]$. First, we show that the recoverable value is < 4 unless $\mathbf{P} = \mathbf{NP}$.

[2] U. Feige and D. Reichman. Recoverable values for independent sets. *Random Structures & Algorithms*, 46(1):142–159, 2013

Lemma 13.1.9 *The maximum value of r in*

$$E\left[\sum_{v\in S} w(v)\right] \geq r \cdot \sum_{v\in I} \frac{w(v)}{deg(v)+1}$$

is strictly less than 4, *unless* $\mathbf{P} = \mathbf{NP}$.

Proof. For the cubic graphs (i.e. graphs where each vertex has degree 3), the MWIS problem is **NP**-Hard. This also holds for the unweighted cubic graphs. If $r = 4$ in $E\left[\sum_{v\in S} w(v)\right] \geq r \cdot \sum_{v\in I^*} \frac{w(v)}{deg(v)+1}$, then we have $E\left[\sum_{v\in S} w(v)\right] \geq r \cdot \sum_{v\in I^*} \frac{w(v)}{4} = \sum_{v\in I^*} w(v)$. Thus we may obtain an optimal MWIS in polynomial time for cubic graphs. This is only feasible if $\mathbf{P} = \mathbf{NP}$. ∎

Here is the modified randomized algorithm of [56].

FR13 Algorithm for MWIS

Input: Graph $G = (V, E)$ on n vertices with $w : V \to \Re^+$.
Output: A set S that approximates the MWIS.

Step 1: Compute an ordering of vertices in V by using a uniform at random permutation. WLOG, let the ordering be $(v_1, \ldots, v_n)$.

Step 2: $F \leftarrow \emptyset$

Step 3: For each vertex v_i in order do

If at most one of the neighbors of v_i has been seen so far,

$F \leftarrow F \cup \{v_i\}$

Step 4: Compute a MWIS S of the induced graph on F.

Step 5: Return S

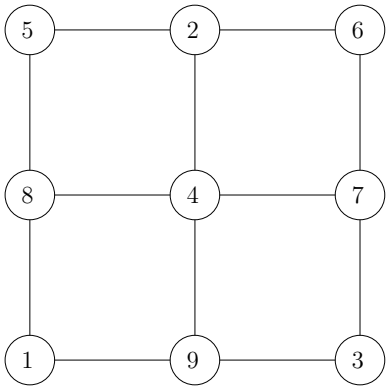


Figure 13.2: $F = \{1, 2, 3, 4, 5, 6\}$ and $S = \{1, 3, 4, 5, 6\}$

As an illustration, consider Figure 13.2.

Lemma 13.1.10 *The induced graph on F obtained at the end of Step 3 in the FR13-Algorithm is a forest.*

Proof. Consider any cycle C in G. Let v be the last vertex in C in the ordering in Step 1. Note that $v \notin F$ as both neighbors of v have been seen before v. Thus, the induced graph of F is acyclic. ∎

Lemma 13.1.11 *MWIS of the induced graph on F obtained in Step 3 in the FR13-Algorithm can be computed in linear time.*

Proof. We apply dynamic programming on a rooted tree. Consider a vertex v and let $I(v)$ represent the weight of the MWIS of the subtree rooted at v. MWIS for the subtree rooted at v is one of the following two types:

Case 1: $v \in$ MWIS: $I(v) = wt(v) + \sum_{x \in \{\text{grandchild of } v\}} I(x)$
Case 2: $v \notin$ MWIS: $I(v) = \sum_{x \in \{\text{child of } v\}} I(x)$

We can process each of the trees of F in postorder to evalaute $I(v)$ for each node v. When we encounter a node v in postorder, we know the $I(u)$'s for each of its child u . Hence, $I(v)$ can be computed in time proportional to the degree of v, and overall it takes linear time to process F. ∎

Lemma 13.1.12 *The weight of the independent S returned by the FR13-Algorithm satisfies $E\left[\sum_{v \in S} w(v)\right] \geq 2 \cdot \sum_{v \in I^*} \frac{w(v)}{deg(v)+1}$, where I^* is a maximum weight independent set of G.*

Proof. Let I be an independent set of G. Observe that $I \cap F$ is an independent set of the induced graph of F. Since S is a MWIS of the induced graph of F (see Step 4), we have

$$E\left[\sum_{v \in S} w(v)\right] \geq E\left[\sum_{v \in I \cap F} w(v)\right]$$

Consider a vertex $v \in I$. When does v makes contribution to the sum $E\left[\sum_{v \in I \cap F} w(v)\right]$?

Only if, it is included in F.

Thus, $Pr(v \in F) = \frac{2}{deg(v)+1}$, as it has to be either the 1st or the 2nd vertex among its neighbors in the permutation ordering to be included in F.

We have $E\left[\sum_{v \in I \cap F} w(v)\right] = E\left[\sum_{v \in I} w(v) X_v\right]$, where X_v is indicator r.v. stating whether $v \in F$ or $v \notin F$. Thus,

$$\begin{aligned} E\left[\sum_{v \in S} w(v)\right] &\geq E\left[\sum_{v \in I} w(v) X_v\right] \\ &= \sum_{v \in I} w(v) E\left[X_v\right] \\ &= \sum_{v \in I} w(v) \frac{2}{deg(v)+1} \end{aligned}$$

Observe that we can replace the independent set I by the MWIS I^* of G, and we have $E\left[\sum_{v\in S} w(v)\right] \geq 2 \cdot \sum_{v\in I^*} \frac{w(v)}{deg(v)+1}$. ∎

13.2 Local Search

The local search approach broadly is as follows.

1. Find a feasible solution.
2. Keep swapping a constant number of objects from the current (local) solution to improve the objective function while maintaining feasibility.
3. Stop when no more local improvements can be made.
4. Output the local solution.

For the analysis, we check whether the algorithm terminates and what is the quality of the solution produced on termination. We will illustrate the technique with the help of the following problems.

1. Single Swaps:
   - 2-approximation algorithm for max cuts in graphs.
   - 5-approximation algorithm for the metric k-median problem in graphs.
2. Multiple Swaps:
   - $(1+\epsilon)$-approximation algorithm for the geometric hitting set problem.

13.2.1 Max Cuts in Graphs

The max-cut problem is defined as follows:
Max-Cut Problem:
Input: An undirected graph $G = (V, E)$.
Output: Among all possible subsets of V, find the subset $S \subset V$ such that the number of edges between S and $\bar{S} = V \setminus S$ is maximized.

The subset S maximizing the number of edges between S and $\bar{S}$ is called the *Max-Cut* of G.

Weighted Max-Cut Problem:
Input: An undirected graph $G = (V, E)$, where each edge has a positive integer weight.
Output: Find a subset $S \subset V$ such that the sum total of the weights on the edges between S and $\bar{S} = V \setminus S$ is maximized. The subset S maximizing the total weight of edges between S and $\bar{S}$ is called the *weighted Max-Cut* of G.

We first present a local improvement algorithm for the unweighted max-cut and show that suitable modifications will help us compute the weighted max-cut.

Local Search Algorithm for Max-Cut

1. Pick any vertex $v \in V$ and set $S \leftarrow \{v\}$ and $\bar{S} = V \setminus S$.
2. If $\exists v \in \bar{S}$ such that
   $cut(S \cup \{v\}, \bar{S} \setminus \{v\}) > cut(S, \bar{S})$, set $S \leftarrow S \cup \{v\}$ and $\bar{S} \leftarrow \bar{S} \setminus \{v\}$.
3. If $\exists v \in S$ such that
   $cut(S \setminus \{v\}, \bar{S} \cup \{v\}) > cut(S, \bar{S})$, set $S \leftarrow S \setminus \{v\}$ and $\bar{S} \leftarrow \bar{S} \cup \{v\}$.
4. Repeat Steps 2 and 3 until the size of the cut doesn't increase.
5. Report $(S, \bar{S}, cut(S, \bar{S}))$.

Let us analyse the local improvement algorithm.

Lemma 13.2.1 *The algorithm terminates in $O(|E|)$ steps.*

Proof. In each iteration of Steps 2 or 3, the size of the cut increases by at least 1. Since the max-cut size is at most $|E|$, the algorithm terminates in $O(|E|)$ iterations. ∎

Lemma 13.2.2 *The cut computed by the local improvement algorithm has $\geq \frac{|E|}{2}$ edges.*

Proof. Let $(S, \bar{S})$ be the cut computed by the algorithm. Consider any vertex $v \in S$. Let v has d_v neighbors. Using the local-optimality condition, at least $\frac{d_v}{2}$ neighbors of v are in $\bar{S}$ (otherwise, we can improve the solution). The same argument applies to any vertex $v \in \bar{S}$. Thus,

$$\begin{aligned} cut(S, \bar{S}) &= \frac{1}{2} \sum_{v \in V} v's \text{ edges crossing the cut} \\ &\geq \frac{1}{2} \sum_{v \in V} \frac{d_v}{2} \\ &= \frac{1}{2} \frac{2|E|}{2} \\ &= \frac{|E|}{2} \end{aligned}$$

∎

Theorem 13.2.3 *The local improvement algorithm is a 2-approximation algorithm for the Max-Cut problem. The algorithm runs in polynomial time.*

Next, we design a local improvement algorithm for the weighted max-cut problem. For each edge $e \in E$, let w_e be its positive integer weight. For a subset $S \subset V$, define the weight of $cut(S, \bar{S})$ as $w(S, \bar{S}) = \sum_{e=uv \in E, u \in S, v \in \bar{S}} w_e$.

Local Improvement Algorithm for Weighted Max-Cut

1. Pick any vertex $v \in V$ and set $S \leftarrow \{v\}$ and $\bar{S} = V \setminus S$.
2. If $\exists v \in \bar{S}$ such that
   $w(S \cup \{v\}, \bar{S} \setminus \{v\}) > w(S, \bar{S})$,
   set $S \leftarrow S \cup \{v\}$ and $\bar{S} \leftarrow \bar{S} \setminus \{v\}$.
3. If $\exists v \in S$ such that
   $w(S \setminus \{v\}, \bar{S} \cup \{v\}) > w(S, \bar{S})$,
   set $S \leftarrow S \setminus \{v\}$ and $\bar{S} \leftarrow \bar{S} \cup \{v\}$.
4. Repeat Steps 2 and 3 until the weight of the cut stops increasing.
5. Report $(S, \bar{S}, w(S, \bar{S}))$.

Let us analyze the above algorithm. Let $W = \sum_{e \in E} w_e$ be the sum total of the weights of all edges in G. In each iteration, $w(S, \bar{S})$ increases by at least one unit, as all weights are integers. Thus the algorithm terminates in at most $O(W)$ steps. The analysis of the approximation factor is based on the observation that for each vertex $v \in S$, $\sum_{e=uv \in E, u \in \bar{S}} w_e \geq \frac{1}{2} \sum_{e=vw \in E} w_e$. What can we say about the running time of the algorithm? Is it polynomial time in input parameters? What if we double the weight of an edge?

The algorithm as stated above performs potentially $O(W)$ iterations in the worst case. This will result in a running time that is not polynomial with respect to the input parameters. So we present a modified algorithm. Let $\epsilon > 0$ be a parameter, and let $n = |V|$.

1. Pick the vertex $v \in V$ that has the maximum sum total of the weights of edges incident to it. Set $S \leftarrow \{v\}$ and $\bar{S} = V \setminus S$.
2. If $\exists v \in \bar{S}$ such that $w(S \cup \{v\}, \bar{S} \setminus \{v\}) \geq (1 + \frac{\epsilon}{n})w(S, \bar{S})$, set $S \leftarrow S \cup \{v\}$ and $\bar{S} \leftarrow \bar{S} \setminus \{v\}$.
3. If $\exists v \in S$ such that $w(S \setminus \{v\}, \bar{S} \cup \{v\}) \geq (1 + \frac{\epsilon}{n})w(S, \bar{S})$, set $S \leftarrow S \setminus \{v\}$ and $\bar{S} \leftarrow \bar{S} \cup \{v\}$.
4. Repeat Steps 2 and 3 until the weight of the cut stops increasing.
5. Report $(S, \bar{S}$ and $w(S, \bar{S}))$.

Let $(S, \bar{S})$ be the cut returned by the algorithm. We make the following observation.

Observation 13.2.4

1. *For each vertex $v \in S$, by local optimality, we have $w(S, \bar{S}) \geq w(S \setminus \{v\}, \bar{S} \cup \{v\}) - \frac{\epsilon}{n}w(S, \bar{S})$. Thus,*

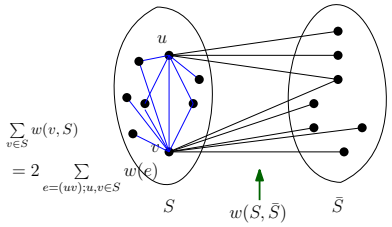
$$w(v, \bar{S}) \geq w(v, S) - \frac{\epsilon}{n}w(S, \bar{S}) \tag{13.1}$$

2. *Similarly, for each vertex $v \in \bar{S}$, we have $w(S, \bar{S}) \geq w(S \cup \{v\}, \bar{S} \setminus \{v\}) - \frac{\epsilon}{n}w(S, \bar{S})$. Thus,*

$$w(v, S) \geq w(v, \bar{S}) - \frac{\epsilon}{n}w(S, \bar{S}) \tag{13.2}$$

By computing the sum total of the inequality 13.1 for all the vertices in S, we have

$$\begin{aligned} w(S, \bar{S}) &\geq \sum_{v \in S} w(v, S) - |S|\frac{\epsilon}{n}w(S, \bar{S}) \\ &= 2 \sum_{e=uv; u,v \in S} w(e) - |S|\frac{\epsilon}{n}w(S, \bar{S}) \end{aligned}$$



Similarly, the sum total of 13.2 for all the vertices in $\bar{S}$, we have

$$w(S, \bar{S}) \geq 2 \sum_{e=uv; u,v \in \bar{S}} w(e) - |\bar{S}|\frac{\epsilon}{n}w(S, \bar{S})$$

Adding the last two inequalities we obtain

$$2w(S, \bar{S}) \geq 2 \sum_{e=uv; u,v \in S} w(e) + 2 \sum_{e=uv, u,v \in \bar{S}} w(e) - |S|\frac{\epsilon}{n}w(S, \bar{S}) - |\bar{S}|\frac{\epsilon}{n}w(S, \bar{S})$$

Simplifying,

$$\begin{aligned} w(S,\bar{S}) &\geq \sum_{e=uv;u,v\in S} w(e) + \sum_{e=uv;u,v\in \bar{S}} w(e) - \frac{\epsilon}{2} w(S,\bar{S}) \\ &= (W - w(S,\bar{S})) - \frac{\epsilon}{2} w(S,\bar{S}) \end{aligned}$$

Thus, $w(S,\bar{S}) \geq \frac{W}{2+\frac{\epsilon}{2}}$. Note that the weight of any cut is upper bounded by W, including the weight of an optimal cut. Thus, we have

Lemma 13.2.5 *The modified local improvement algorithm is $\frac{1}{2+\epsilon}$ approximation algorithm for the weighted max-cut problem.*

Next we analyze the running time. Assume that the algorithm runs for k iterations and the sets computed by the algorithm are $S_0, S_1, S_2, \ldots, S_k$. Observe that $w(S_i,\bar{S}_i) \geq (1+\frac{\epsilon}{n}) w(S_{i-1},\bar{S}_{i-1})$, for $i = 1,\ldots,k$. This results in $w(S_k,\bar{S_k}) \geq (1+\frac{\epsilon}{n})^k w(S_0,\bar{S_0})$.

We know that $w(S_0,\bar{S_0}) \geq \frac{W}{n}$ and $W(S_k,\bar{S_k}) \leq W$. Thus,

$$\begin{aligned} W &\geq W(S_k,\bar{S_k}) \\ &\geq (1+\frac{\epsilon}{n})^k w(S_0,\bar{S_0}) \\ &\geq (1+\frac{\epsilon}{n})^k \frac{W}{n} \end{aligned}$$

Therefore, we have that $k \leq \frac{\log n}{\log(1+\frac{\epsilon}{n})}$.

If $\frac{\epsilon}{n} < 1$, $\log(1+\frac{\epsilon}{n}) \geq \frac{\epsilon}{2n}$ (i.e., $\log(1+x) > x/2$ for small values of x). Thus, $k \leq \frac{\log n}{\log(1+\frac{\epsilon}{n})} \leq 2\frac{n}{\epsilon}\log n$. We summarize the result in the following theorem.

Theorem 13.2.6 *A local improvement algorithm approximates the maximum weight cut in a graph in polynomial time. The algorithms approximation factor is $\frac{1}{2+\epsilon}$ and the running time depends on $\frac{1}{\epsilon}$, $|V|$, and $|E|$.*

13.2.2 k-Median

Let $G = (V,E)$ be a complete graph on n vertices, where the costs on edges ($d : V \times V \to \Re^+$) satisfy the metric properties:

- $\forall u \in V : d(u,u) = 0$
- $\forall u,v \in V : d(u,v) = d(v,u)$
- $\forall u,v,w \in V : d(u,v) \leq d(u,w) + d(w,v)$

We first define a few quantities before formally stating the problem definition.

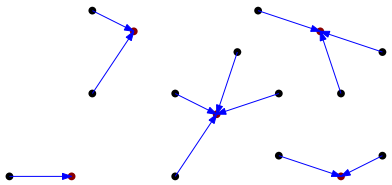
1. Facilities: Let $F \subseteq V$ such that $|F| = k$.

2. Distance to nearest facility: $d(v, F) = \min_{f \in F} d(v, f)$, where $v \in V$.

3. $cost(F) = \sum_{v \in V} d(v, F)$

k-median problem

Given the metric complete graph $G = (V, E)$, find $F \subseteq V$, $|F| = k$, such that $cost(F)$ is minimum.



For an illustration, consider points in plane with Euclidean metric and $k = 5$.

Our local search algorithm for the computation of the set of k facilities is as follows:

Local Search Algorithm for k-median
Input: A metric graph $G = (V, E)$ and an integer $k > 0$
Output: $F \subset V$ such that $|F| = k$.

Step 1 (Initialize) $F \leftarrow \emptyset$.

Select any k vertices from V.

Add them to F as the initial set of k facilities.

Setp 2 (Local improvement step)

While there exists a pair of vertices (u, v), where $u \in V \setminus F$ and $v \in F$, such that $cost(F \setminus \{v\} \cup \{u\}) < cost(F)$,

$F \leftarrow F \setminus \{v\} \cup \{u\}$.

Step 3 Report F.

We will show that the above algorithm is a 5-approximation algorithm. Let F^* be an optimal set of k-facilities for the k-median problem on the metric graph G. The set F returned by the local search algorithm satisfies $cost(F) \leq 5cost(F^*)$.

First, let us understand swap pairs and their utility in analyzing the local search algorithm.

- In Step 2 of the algorithm, if we make a swap (u, v), then $cost(F)$ improves, i.e., $cost(F \setminus \{v\} \cup \{u\}) < cost(F)$.
- After the algorithm terminates, there don't exist any more improving swap pairs. I.e., for any pair of vertices (u, v), where $u \in V \setminus F$ and $v \in F$, $cost(F \setminus \{v\} \cup \{u\}) \geq cost(F)$.
- To show $cost(F) \leq 5cost(F^*)$, we will select a set of specific non-improving swap pairs using the vertices in an optimal solution F^* and the solution F returned by the algorithm.

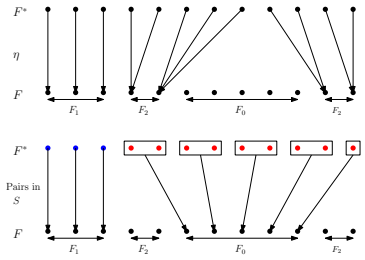
Next, we find a select set of non-improving swap pairs as follows. Let $F^* = (f_1^*, \ldots, f_k^*) \subset V$ be an optimal solution. Let $F = (f_1, \ldots, f_k) \subset V$ be the solution the local search algorithm returns. Define a mapping $\eta : F^* \to F$, that maps each facility (vertex) in F^* to the nearest facility in F. We partition $F = F_0 \cup F_1 \cup F_{\geq 2}$ based on the in-degree of function η, where

$F_0 = \{f \in F |$ no facilities in F^* maps to $f\}$

$F_1 = \{f \in F |$ exactly one facility in F^* maps to $f\}$

$F_{\geq 2} = \{f \in F |$ at least two facilities in F^* maps to $f\}$

Define the set $S \subset F^* \times F$ consisting of the following non-improving pairs of facilities:



1. All pairs corresponding to F_1 are in S. I.e. for each pair (f^*, r), where $r \in F$ and $f^* \in F^*$ and $\eta^{-1}(r) = f^*$, $(f^*, r) \in S$.

2. For the remaining facilities in F^*, pair them up and assign each pair to a unique facility in F_0.

For an illustration of set S, see the figure. A natural question is whether there are enough facilities in F_0 so that the paired-up remaining facilities in F^* can be assigned to unique facilities in F_0.

Lemma 13.2.7 $|F_0| \geq \frac{|F|-|F_1|}{2}$

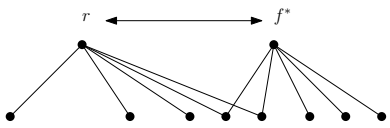
Proof. We observe that $k = |F| = |F^*|$. By definiition, $|F| = |F_0| + |F_1| + |F_{\geq 2}|$. Moreover, the number of remaining facilities in F^* are $k - |F_1| = |F| - |F_1|$. The nearest neighbors of the remaining facilities in F^* are among $F_{\geq 2}$. Observe that each facility in $F_{\geq 2}$ is near neighbor of at least two facilities of F^*. Thus, $|F_0| \geq \frac{|F|-|F_1|}{2}$. ∎

We establish some notations before we show how to bound $cost(F)$. Define functions $\phi : V \to F$ and $\phi^* : V \to F^*$ that map vertices to the nearest facilities in F and F^*, respectively. If $\phi(v) = r$, than the nearest vertex of v in F is r. For any vertex $v \in V$, we define the cost to the nearest facility in F^* by $O_v = d(v, F^*) = d(v, \phi^*(v))$. Similarly, we define $A_v = d(v, F) = d(v, \phi(v))$. Observe that $cost(F^*) = \sum_{v \in V} O_v$ and $cost(F) = \sum_{v \in V} A_v$. We define the neighborhoods of facilities as the vertices that they serve. For each facility $f^* \in F^*$, we have $N^*(f^*) = \{v \in V | \phi^*(v) = f^*\}$. Similarly, for $r \in F$, $N(r) = \{v \in V | \phi(r) = f\}$. If $F^* = (f_1^*, \ldots, f_k^*)$, then $N^*(f_1^*), \ldots, N^*(f_k^*)$ is a partition of V. Similarly, $N(r_1), \ldots, N(r_k)$ is partition of V with respect to facilities in $F = \{r_1, \ldots, r_k\}$.

Lemma 13.2.8 *Consider a (non-improving) swap pair $(f^*, r) \in S$. Suppose we bring in the facility $f^* \in F^*$ and remove r from F, i.e., $F = F \cup \{f^*\} \setminus$*

$\{r\}$. *The cost of the resulting k-median solution satisfies*

$$\sum_{v \in N^*(f^*)} (O_v - A_v) + \sum_{v \in N(r)} 2O_v \geq cost(F \cup \{f^*\} \setminus \{r\}) - cost(F) \quad (13.3)$$



Before providing proof of the lemma, we show that by summing Inequalities 13.3 over all the swap pairs in S, we have $cost(F) \leq 5cost(F^*)$.

Theorem 13.2.9 *Suppose for each swap pair* $(f^*, r) \in S$ *we have* $\sum_{v \in N^*(f^*)} (O_v - A_v) + \sum_{v \in N(r)} 2O_v \geq cost(F \cup \{f^*\} \setminus \{r\}) - cost(F)$, *than* $cost(F) \leq 5cost(F^*)$.

Proof. Each $f^* \in F^*$ appears exactly once in S, and $\bigcup_{f^* \in F^*} N^*(f^*)$ partitions V, we have

$$\sum_{(f^*,r) \in S} \sum_{v \in N^*(f^*)} (O_v - A_v) \leq cost(F^*) - cost(F) \quad (13.4)$$

Each $r \in F$ appears at most twice in S. Thus,

$$\sum_{(f^*,r) \in S} \sum_{v \in N(r)} O_v \leq 2cost(F^*) \quad (13.5)$$

As each swap pair in S is non-improving, we have

$$cost(F \cup \{f^*\} \setminus \{r\}) - cost(F) \geq 0 \quad (13.6)$$

Summing for all pairs $(f^*, r) \in S$ the inequality $\sum_{v \in N^*(f^*)} (O_v - A_v) + \sum_{v \in N(r)} 2O_v \geq cost(F \cup \{f^*\} \setminus \{r\}) - cost(F)$, and applying inequalities 13.4, 13.5, and 13.6, we obtain $cost(F^*) - cost(F) + 2 * 2cost(F^*) \geq 0$.
Thus, $cost(F) \leq 5cost(F^*)$ ∎

Now we provide proof of Lemma 13.2.8.

Proof. Note that we are swapping r by f^* in F. We are interested to upper bound $cost(F \cup \{f^*\} \setminus \{r\}) - cost(F)$. As a result of this swap, we need to reassign facilities to some of the vertices. For example, all vertices in $N(r)$ need to find a facility in $F \cup \{f^*\} \setminus \{r\}$.

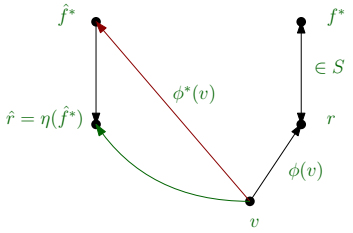
We will assign each vertex in $N^*(f^*)$ to f^* in $F \cup \{f^*\} \setminus \{r\}$. We will assign each vertex $v \in N(r) \setminus N^*(f^*)$ to $\eta(\phi^*(v))$. For all the remaining vertices, the assignment remains the same. Note that this reassignment may not map each vertex to its nearest facility. This is not a problem as we are interested in upper bound $cost(F \cup \{f^*\} \setminus \{r\}) - cost(F)$.

For facilities in $F \cup \{f^*\} \setminus \{r\}$, we assign each vertex in $N^*(f^*)$ to f^*. The expression $\sum_{v \in N^*(f^*)} (O_v - A_v)$ accounts for the difference in the costs, as we save A_v from their costs, but they cost us O_v.

There may be a vertex $v \in N^*(f^*)$ that ideally isn't served by f^* in $F \cup \{f^*\} \setminus \{r\}$. The reason is that $r' \in F \setminus \{r\}$ may be closer to v than f^*. Nevertheless, as mentioned before, we assign v to f^*, as we are trying to find an upper bound ($O(v) \geq d(v,r') \implies O_v - A_v \geq d(v,r') - A_v$).

All the vertices in $N(r) \cap N^*(f^*)$ are assigned to f^* in $F \cup \{f^*\} \setminus \{r\}$. Even if for a vertex $v \in N(r) \cap N^*(f^*)$ its nearest neighbor in $F \cup \{f^*\} \setminus \{r\}$ may not be f^*, the same upper bound argument holds.

Next, we account for the costs of members in $N(r) \setminus N^*(f^*)$. Let $v \in N(r) \setminus N^*(f^*)$. Since v isn't served by f^* in optimal $\implies$ v is served by a facility $\hat{f}^* \in F^*$, i.e., $\phi^*(v) = \hat{f}^*$.



Either $\hat{f}^* \in F$ or $\hat{f}^* \notin F$. If $\hat{f}^* \in F$: then we assign v to $\hat{f}^*$. If $\hat{f}^* \notin F$, consider $\hat{r} = \eta(\hat{f}^*)$, i.e. nearest neighbor of $\hat{f}^*$ in F. Note: $\hat{r} \neq r$. If it is, than $r \in F_1 \cup F_{\geq 2}$, we wouldn't have assigned f^* to r. We assign v to $\hat{r}$.

By triangle inequality we have $d(v,\hat{r}) \leq d(v,\hat{f}^*) + d(\hat{f}^*,\hat{r})$. Subtracting $d(v,r)$ from both the sides, we obtain $d(v,\hat{r}) - d(v,r) \leq d(v,\hat{f}^*) + d(\hat{f}^*,\hat{r}) - d(v,r)$. We know that $d(\hat{f}^*,\hat{r}) \leq d(\hat{f}^*,r)$ because of the nearest neighbor function η. Thus, $d(v,\hat{r}) - d(v,r) \leq d(v,\hat{f}^*) + d(\hat{f}^*,r) - d(v,r)$. By triangle inequality, $d(\hat{f}^*,r) - d(v,r) \leq d(v,\hat{f}^*)$. Thus,

$$\begin{aligned} d(v,\hat{r}) - d(v,r) &\leq d(v,\hat{f}^*) + d(\hat{f}^*,r) - d(v,r) \\ &\leq 2d(v,\hat{f}^*) \\ &= 2O_v \end{aligned}$$

■

Now we analyze the running time of the local search algorithm. The algorithm terminates as in each execution of Step 2, the cost improves, and it can't improve forever. Let us see how many times Step 2 is executed. Assume all $d(u,v)$ values are positive integers and let $\Delta = \sum_{u,v} d(u,v)$. The number of times Step 2 is executed is at most Δ. We can modify Step 2 to have a swap if the cost improves by at least a factor of $(1 - \frac{\epsilon}{poly(n)})$. This, in spirit, is similar to the max weight cut.

Theorem 13.2.10 *Let F^* be an optimal set of k-facilities for the k-median problem on the metric graph G. The set F returned by the local search algorithm satisfies $cost(F) \leq (5+\epsilon)cost(F^*)$. Moreover, the algorithm runs in polynomial time. Run time depends on $|V|$ and $\frac{1}{\epsilon}$.*

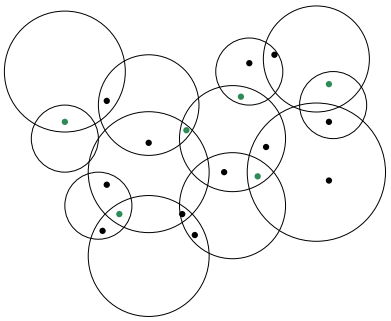
Remark 13.2.11 *In place of performing a single swap in Step 2, perform $t \geq 1$ multi-swaps. A refined analysis shows that $cost(F) \leq (3 + \frac{2}{t})cost(F^*)$.*

13.2.3 *Geometric Hitting Set*

The geometric hitting set problem is defined as follows.

Input: A set $\mathcal{D}$ of disks and a set P of points in plane.

Output: Find a subset $S \subseteq P$ of smallest cardinality that hits all disks in D.



We say a point $p \in P$ *hits* the disk $D \in \mathcal{D}$ if $p \in D$.

Next, we present a local search algorithm.

k-level Local Search algorithm for hitting set of disks
Input: A set $\mathcal{D}$ of disks and a set P of points in plane. A (large) integer $k > 0$.
Output: A subset $S \subseteq P$ that hits all disks in $\mathcal{D}$.

1. Initialization: $S \leftarrow P$. Check if S hits all disks. If not, report infeasibility and stop.
2. Local Improvement Step: Keep replacing any set of k points in S by at most $k-1$ points of P so that points in S hits all disks in $\mathcal{D}$.
3. Return S.

We will prove the following

Theorem 13.2.12 *Let $S^* \subseteq P$ be an optimal hitting set for $\mathcal{D}$. The set S returned by the algorithm satisfies $|S| \leq (1 + \frac{c}{\sqrt{k}})|S^*|$, for some constant c.*

First, let us state the locality condition. Let $B, R \subset P$ be subset of points of P, and let $G = (V = B \cup R, E)$ be a bipartite graph such that the following *locality condition* holds:

For any disk $D \in \mathcal{D}$, where $B \cap D \neq \emptyset$ and $R \cap D \neq \emptyset$, there exist points $b \in B \cap D$ and $r \in R \cap D$ such that $(b, r) \in E$. We show that the Delaunay triangulation of $B \cup R$ satisfies the locality condition.

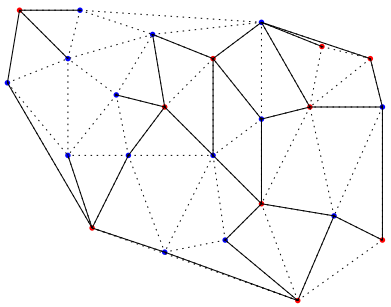


Figure 13.3: Illustration of Delaunay Graph $G = (B \cup R, E)$

Lemma 13.2.13 *Let G be the planar graph corresponding to the Delaunay triangulation of $B \cup R$, where we only keep the edges between a pair of red and blue points. The graph G satisfies the locality condition.*

Proof.
By construction, G is bipartite. If a disk $D \in \mathcal{D}$ contains points from B and R, then there is a point $b \in B$ and $r \in R$ such that the Delaunay edge br entirely lies inside D. This uses the property that the points within an arbitrary disk form a connected subgraph in a Delaunay triangulation. ∎

Let us introduce the concept of neighborhoods. For each vertex $v \in G = (V, E)$, let $N(v)$ be all the vertices adjacent to v. For a subset of vertices $W \subset V$, define $N(W) = \bigcup_{v \in W} N(v)$.

Let $B = S$ be the set returned by the local search algorithm, and let $R = S^*$ be an optimal solution for the hitting set problem. To prove Theorem 13.2.12, it suffices to show that $|B| \leq (1 + \frac{c}{\sqrt{k}})|R|$. Assume that $B \cap R = \emptyset$. This assumption is justified, as we can remove the common points and disks they hit. Note that B hits all disks in $\mathcal{D}$, and similarly, R hits all disks in $\mathcal{D}$. Consider the planar bipartite graph $G = (B \cup R, E)$ formed using the Delaunay triangulation of $B \cup R$ and retain only the red-blue edges.

If $|B \cap R| = \kappa$, $\frac{|B|}{|R|} \leq \frac{|B|-\kappa}{|R|-\kappa}$.

Lemma 13.2.14 *For any subset $B' \subset B$, $B \cup N(B') \setminus B'$ is a hitting set for $\mathcal{D}$.*

Proof. Consider any disk $D \in \mathcal{D}$. Since points in B hit all disks, some point in B hits D. If any of the points in $B \setminus B'$ hits D, points in $B \cup N(B') \setminus B'$ also hits D.

Assume only the points in B' hit the disk D. Points in R also hit all disks in $\mathcal{D}$. Let $r \in R$ hits D and let $b \in B'$ hits D. We have both the points $b, r \in D$. By the Delaunay property, there is a bichromatic edge in the Delaunay triangulation that completely lies in D. Therefore, the neighborhood set of B' also includes a red point in R in the disk D. Hence, $B \cup N(B') \setminus B'$ is a hitting set for $\mathcal{D}$. ∎

Next, we state the expansion property.

Lemma 13.2.15 *For every subset $B' \subseteq B$ of size $\leq k$ in the graph $G = (B \cup R, E)$, $|N(B')| \geq |B'|$, i.e. the size of the neighborhood of B' is at least $|B'|$.*

Proof. Recall that in the local search algorithm, the set B is obtained by executing the local search algorithm with parameter k. Once the algorithm terminates, there doesn't exist any improving swaps, i.e., no set of k points (vertices) in B can be replaced by $\leq k - 1$ points from P to hit all the disks in $\mathcal{D}$.

By Lemma 13.2.14, the set $B \cup N(B') \setminus B'$ is a hitting set for $\mathcal{D}$. Moreover, $|N(B')| \geq |B'|$ must hold; otherwise, the local optimality condition is violated. ∎

Now we introduce the balanced partitioning of planar graphs. In particular, let us look at Fredrickson's r-partitioning of planar graphs, Section 7.3. Let $G = (V, E)$ be a planar graph on n vertices, and let r be a number. Using the recursive application of Lipton and Tarjan's planar separator theorem, Fredrickson shows a planar graph division in regions consisting of interior and boundary vertices. Each interior vertex is contained within a region and is adjacent to vertices within that region. Boundary vertices are shared between at least two regions. We state the r-partitioning result of Frederickson without proof

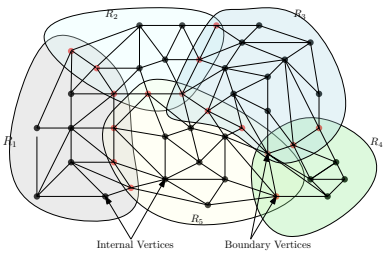


Figure 13.4: Illustration of r-partitioning

Lemma 13.2.16 *Let G be a planar graph on n vertices. A r-division divides G in $\Theta(n/r)$ regions, where each region consists of $O(r)$ vertices and $O(\sqrt{r})$ boundary vertices. A r-division of a planar graph G can be computed in $O(n \log n)$ time.*

Now let us get back to estimating the size of the local solution, i.e., the size of the set B.

Lemma 13.2.17 *Let $S \subset P$ be the set of points returned by the local search algorithm with parameter k and let $S^* \subset P$ be an optimal solution for the hitting set problem for the disks in $\mathcal{D}$ by points in P. We define the Delaunay triangulation on red-blue points where $B = S$ and $R = S^*$ and construct the bipartite graph $G = (B \cup R, E)$ by retaining only the edges between red and blue points. The following holds: $|B| \leq (1 + \frac{c}{\sqrt{k}})|R|$ for some constant c.*

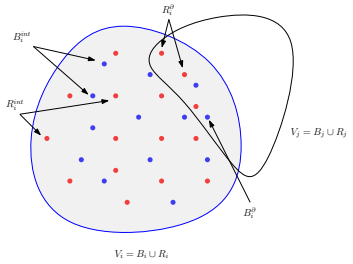


Figure 13.5: Illustration of Notation

Proof.
Assume that $n = |B| + |R|$. We apply Fredrickson's r-partitioning to the graph G, where $r = k$. This partitions G into $\Theta(n/k)$ regions, each region consisting of $\leq k$ vertices and $O(\sqrt{k})$ boundary vertices. The total number of boundary vertices is $O(n/\sqrt{k})$. Let $V_i = B_i \cup R_i$ be the set of vertices in the i-th region in the partitioning. Let B_i^{int} and B_i^{∂} be the interior and boundary blue vertices in V_i, respectively. Similarly, let R_i^{int} and R_i^{∂} be the interior and boundary red vertices in V_i, respectively.

Observe that the sum total of the boundary vertices among all the regions is $\gamma n/\sqrt{k}$, where γ is a constant from Fredrickson's r-partitioning. I.e.,

$$\sum_i (|B_i^{\partial}| + |R_i^{\partial}|) \leq \gamma n/\sqrt{k}$$

The number of interior blue vertices, $|B_i^{int}|$, in any region is at most k. By the expansion Property, Lemma 13.2.15, we know that $|B_i^{int}| \leq |N(B_i^{int})|$. Let us look at the vertices that make up $N(B_i^{int})$. Note that $N(B_i^{int}) \subseteq R_i^{int} \cup R_i^{\partial}$. Thus we have

$$|B_i^{int}| \leq |R_i^{int}| + |R_i^{\partial}|$$

Add $|B_i^{\partial}|$ on both sides, and we obtain

$$|B_i^{\partial}| + |B_i^{int}| \leq |R_i^{int}| + |R_i^{\partial}| + |B_i^{\partial}|$$

Summing over all the regions, we have

$$\sum_i \left(|B_i^{\partial}| + |B_i^{int}|\right) \leq \sum_i |R_i^{int}| + \sum_i \left(|R_i^{\partial}| + |B_i^{\partial}|\right) \tag{13.7}$$

Note that $\sum_i \left(|B_i^{\partial}| + |B_i^{int}|\right) \geq |B|$, $|R| \geq \sum_i |R_i^{int}|$, and

$\sum_i \left(|R_i^\partial| + |B_i^\partial|\right) = \gamma n/\sqrt{k} = \gamma(|B| + |R|)/\sqrt{k}$. We have

$$\begin{aligned}
|B| &\leq \sum_i \left(|B_i^\partial| + |B_i^{int}|\right) \\
&\leq \sum_i |R_i^{int}| + \sum_i \left(|R_i^\partial| + |B_i^\partial|\right) \\
&\leq |R| + \frac{\gamma}{\sqrt{k}}(|B| + |R|) \\
&\leq \left(\frac{1+\gamma/\sqrt{k}}{1-\gamma/\sqrt{k}}\right)|R| \\
&= (1+\gamma/\sqrt{k})(1+\gamma/\sqrt{k}+(\gamma/\sqrt{k})^2+(\gamma/\sqrt{k})^3+\cdots)|R| \\
&\qquad \left(\frac{1}{1-x} = 1+x+x^2+\cdots\right) \\
&\leq (1+\gamma/\sqrt{k})(1+2\gamma/\sqrt{k})|R| \text{ (Set } k \geq 4\gamma^2 \text{ and } c = 4\gamma \implies \frac{\gamma}{\sqrt{k}} \leq \frac{1}{2}) \\
&= (1+3\gamma/\sqrt{k}+2(\gamma/\sqrt{k})^2)|R| \\
&= (1+4\gamma/\sqrt{k})|R| \\
&= (1+c/\sqrt{k})|R|
\end{aligned}$$

■

We summarize the main steps in designing a local search solution.

- Design a local search algorithm with parameter k.
- Consider the solution B returned by the algorithm and an optimal solution R.
- Set up a bipartite planar graph G with bipartition B and R.
- Find a k-partitioning of G into $\Theta(n/k)$ regions, each region consisting of at most k vertices, and the boundary composed of $O(\sqrt{k})$ vertices.
- Bound the size of B in terms of the size of R using the neighborhood relationships of internal blue vertices in each region.

Extensions: Maximization problems (see Aschner et al.), Max Coverage Problems with Cardinality Constraints (see Chaplick et al.).

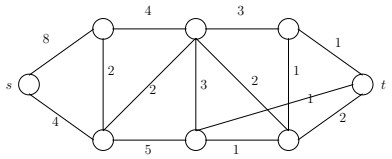
13.3 Approximation using Metric LPs

13.3.1 Min Cost st-cut

The min-cost *st*-cut problem in graphs is defined as follows
Input: An undirected graph $G = (V, E)$ on n vertices and each edge has a positive weight $w : E \to \Re^+$, and two specific vertices s and t. It

will be easier to think of G as a complete graph K_n, as all the edges in $K_n \setminus G$ are assigned a weight of 0.

Output: Find a set of edges $C \subseteq E$ of minimum total weight so that the graph $G' = (V, E \setminus C)$ has no path that between s and t. I.e., C forms a cut of minimum weight that separates s and t.



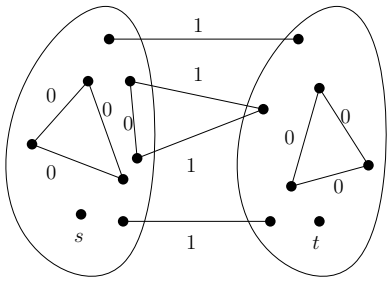
Let C be a st-cut in G. We define an indicator variables x_e for each edge e as follows.

$$x_e = \begin{cases} 1, \text{ if } e \in C, \\ 0, \text{ otherwise} \end{cases}$$

We make the following observation.

Observation 13.3.1

1. *The cost of cut equals* $\sum_{e \in E} x_e w_e$.
2. *The length of any path* $\pi(s,t)$ *joining* s *and* t *is* ≥ 1. *The length of* π *is defined as the sum total of* x_e*'s values of the edges of* π.
3. *The variables* x_e*'s assigned to the edges of* G *satisfy the metric property.*



Now we formulate the cut problem as a linear program.

Integer Metric LP Formulation

$$\min \sum_{e \in E} w_e x_e$$

Subject to:

1. Membership in the Cut: For each edge $e \in E$, $x_e \in \{0,1\}$.
2. Cut Constraint: $x_{st} \geq 1$.
3. Triangle Inequality: For every set of three distinct vertices $u, v, w \in V$: $x_{uw} + x_{wv} \geq x_{uv}$.

Integer LPs are **NP**-Hard in general, whereas the linear programs can be solved in polynomial time. Thus we relax the ILP by replacing the constraint $x_e \in \{0,1\}$ by $0 \leq x_e \leq 1$ and obtain the following LP.

Relaxed Metric LP

$$\min \sum_{e \in E} w_e x_e$$

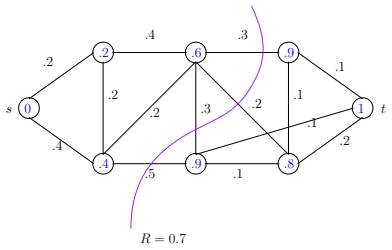
Subject to:

1. For each edge $e \in E$, $0 \leq x_e \leq 1$.
2. $x_{st} \geq 1$.
3. For every set of three distinct vertices $u, v, w \in V$: $x_{uw} + x_{wv} \geq x_{uv}$.

Let $x_e \in [0,1]$ be the value assigned to edge $e \in E$ in the solution of the relaxed LP. Let $z^* = \sum_{e \in E} w_e x_e$ be the value of the objective function. Note that x_e values satisfy

1. Triangle inequality: For any three distinct vertices $u, v, w \in V$, $x_{uv} \leq x_{vw} + x_{uw}$.
2. For any path in G between s and t, the length of the path is ≥ 1.
3. The cost of an optimal min cut in G is at least z^*.

To obtain a cut from the relaxed LP, we perform the following steps.



Step 1: Solve the relaxed metric LP to obtain x_e values for each edge $e \in E$.

Step 2: For each vertex $v \in V$, find the shortest distance $\delta(s,v)$ from s with respect to the x_e values on edges from s.

Step 3: Choose an arbitrary value $R \in (0,1)$.

Step 4: For each edge $e = (uv) \in E$ (assume $\delta(s,u) \leq \delta(s,v)$), place e in the cut if $\delta(s,u) < R < \delta(s,v)$.

Step 5: Return the edges in the cut.

The estimate on the expected weight of the cut is as follows.

Lemma 13.3.2 *The expected sum total of the weights of the edges in the cut is at most z^*.*

Proof. Let C be the collection of edges in the cut with respect to $R \in (0,1)$. Consider an arbitrary edge $e = (uv) \in E$. We first estimate the probability that $e \in C$. The edge $e \in C$ if $\delta(s,u) < R < \delta(s,v)$, i.e. $R \in (\delta(s,u), \delta(s,v))$. Therefore,

$$\begin{aligned} Pr(e \in C) &= \frac{\delta(s,v) - \delta(s,u)}{1} \\ &= \delta(s,v) - \delta(s,u) \\ &\leq x_e \end{aligned}$$

The last inequality follows from the triangle inequality as $\delta(s,v) - \delta(s,u) \leq x_e$. Thus, $Pr(e \in C) \leq x_e$. Therefore,

$$E[cost(C)] = \sum_{e \in E} w_e Pr(e \in C) \leq \sum_{e \in E} w_e x_e = z^*.$$

∎

Let us see how we can find an optimal cut. We will understand the nature of cuts when R ranges from 0 to 1. When $R = 0$, the component containing s contains only a singleton vertex s, and all the remaining vertices $V \setminus \{s\}$ are in the component of t. Analogously, when $R = 1$, the component containing s contains all the vertices except t, and all the component of t is a singleton vertex. Observe that when R ranges from 0 to 1, one by one, the vertices move to the component containing s from the component containing t. In all, there are $n = |V|$ such events. We can find all the events and return the cut that minimizes the total weight.

Interestingly,

1. If for some R the cost of the cut is $> z^*$, than there must be a cut for which the cost $< z^*$, since the average (i.e. the expected) value is z^*.

2. The cost of any st-cut is $\geq z^*$, as z^* is the objective value of relaxed LP. Hence, the cut returned by the method has the optimal cost for any $R \in (0, 1)$.

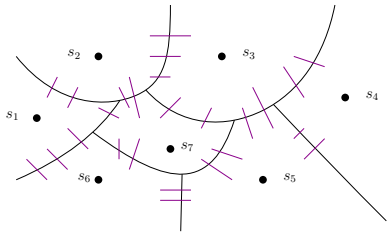
We summarize the result in the following theorem.

Theorem 13.3.3 *Using the Metric LP relaxation, we can find an optimal cut in polynomial time.*

Note that this method's time complexity depends on the algorithm for solving a linear program. There are alternate methods for computing the minimum st-cut. For example, we can calculate the maximum flow from s to t in G and apply the max flow min cut theorem (see the chapter on max flow).

13.3.2 Multiway Min Cut

The multiway cut problem on graphs is defined as follows.
Input: An undirected (complete) graph $G = (V, E)$ on n vertices and each edge has a positive weight $w : E \to \Re^+$. A set $T = \{s_1, \ldots, s_k\} \subset V$ of k vertices called terminals.
Output: Find a set of edges $C \subseteq E$ of minimum total weight so that the graph $G' = (V, E \setminus C)$ has no path between any pair of terminals in T.



Let C be a multiway cut that separates every pair of terminals. Define an indicator variable x_e for each edge e as follows:

$$x_e = \begin{cases} 1, \text{ if } e \in C, \\ 0, \text{ otherwise} \end{cases}$$

The x_e values assigned to each edge satisfies

1. Cost of the multiway cut equals $\sum_{e \in E} x_e w_e$.

2. For any pair of distinct terminals $s_i, s_j \in T$, the length of any path $\pi(s_i, s_j)$ joining s_i and s_j is ≥ 1.

3. x_e values satisfy the triangle inequality. I.e., for any three distinct vertices $u, v, w \in V$, $x_{uw} + x_{wv} \geq x_{uv}$.

As before, we will formulate an integer linear program and its relaxation.

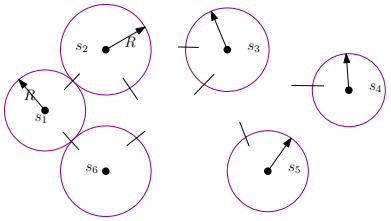
Integer Metric LP Formulation for Multiway Cuts

$$\min \sum_{e \in E} w_e x_e$$

Subject to:

1. Membership in the Cut: For each edge $e \in E$, $x_e \in \{0, 1\}$.
2. Cut Constraint: For every distinct pair $s_i, s_j \in T$, $x_{s_i s_j} \geq 1$.
3. Triangle Inequality: For every set of three distinct vertices $u, v, w \in V$: $x_{uw} + x_{wv} \geq x_{uv}$.

We replace the constraint $x_e \in \{0, 1\}$ for the relaxed linear program by $0 \leq x_e \leq 1$. Next, we outline the steps in finding the edges in the cut using the solution of the relaxed LP.



Step 1: $C \leftarrow \emptyset$.

Step 2: Solve the relaxed metric linear program to obtain x_e values for each edge $e \in E$.

Step 3: Choose an arbitrary value $R \in (0, 1/2)$.

Step 4: For each vertex $s_i \in T$, find the shortest distances $\delta(s_i, v)$ from s_i with respect to x_e values on edges. For each edge $e = (uv) \in E$ (assume $\delta(s_i, u) \leq \delta(s_i, v)$), place e in the cut C if $\delta(s_i, u) < R < \delta(s_i, v)$.

Step 5: Return the set of edges in C.

Let $s_i \in T$ be a terminal and $R \in (0, 1/2)$. Define a *ball* centered at s_i as $B(s_i, R) = \{v \in V | \delta(s_i, v) < R\}$. The ball $B(s_i, R)$ consists of all the vertices that are within the distance R of s_i. We make the following observations.

Observation 13.3.4

1. *Let $s_i, s_j \in T$ be two distinct terminals and let $B(s_i, R)$ and $B(s_j, R)$ be the set of vertices within distance of $R \in (0, 1/2)$ of s_i and s_j, respectively. Then, $B(s_i, R) \cap B(s_j, R) = \emptyset$, i.e., the balls $B(s_i, R)$ and $B(s_j, R)$ are disjoint and do not share any vertex.*

2. *Consider any edge $e = (uv) \in E$, where $u \in B(s_i, R)$. The edge $e \in C$ if $v \notin B(s_i, R)$.*

Lemma 13.3.5 *The cut C returned by the above method is a feasible multiway cut.*

Proof. We need to show that there is no path between any pair of distinct terminals $s_i, s_j \in T$ in the graph $G - C$. By the 2nd constraint of the LP, $x_{s_i s_j} \geq 1$. From the triangle inequality, any path $\pi(s_i, s_j)$ between s_i and s_j in G will have length ≥ 1. Alternatively, for any vertex $w \in V$, $\delta(s_i, w) + \delta(s_j, w) \geq 1$. Distance between any two vertices within a ball $B(s_i, R)$ is < 1. Thus, for any ball $B(s_i, R)$, only terminal that is in $B(s_i, R)$ is s_i, i.e., $B(s_i, R) \cap T = s_i$. Hence each connected component of $G \setminus C$ contains at most one terminal. ∎

Now we bound the probability of an edge to be in a cut.

Lemma 13.3.6 *Let $e = (u, v)$ be an edge in G. $Pr(e \in C) \leq 2x_e$.*

Proof. Define sets $X_1, \ldots, X_k$, where $X_i = \{v \in V | \delta(s_i, v) < 1/2\}$. Note that for any pair of distinct sets X_i and X_j, $X_i \cap X_j = \emptyset$. Moreover, $B(s_i, R) \subseteq X_i$. One of the following cases arises for the edge $e = (u, v)$.

Case 1: None of the endpoints u, v of e are in any set. This implies that $e \notin C$ and $Pr(e \in C) = 0 \leq 2x_e$.

Case 2: Both *uandv* are in the same set, say X_i.

Case 3: $u \in X_i$ and $v \in V \setminus X_i$.

Consider Case 2, where $u, v \in X_i$. Without loss of generality, assume $\delta(s_i, u) \leq \delta(s_i, v)$. We know that $R \in (0, 1/2)$. The edge $e = (u, v)$ is in the cut C if $\delta(s_i, u) < R$ and $\delta(s_i, v) > R$. By triangle inequality, we know that $\delta(s_i, v) - \delta(s_i, u) \leq x_e$. Since we are choosing R uniformly at random in $(0, 1/2)$,

$$\begin{aligned} Pr(e \in C) &= \frac{\delta(s_i, v) - \delta(s_i, u)}{\frac{1}{2}} \\ &\leq 2x_e \end{aligned}$$

Consider Case 3, where $u \in X_i$ and $v \in V \setminus X_i$. We know that $\delta(s_i, u) < 1/2$ and $\delta(s_i, v) \geq 1/2$. By the triangle inequality, we know that $\delta(s_i, v) - \delta(s_i, u) \leq x_e$. The edge $e = (u, v) \in C$ if $\delta(s_i, u) < R$.

Thus,

$$\begin{aligned} Pr(e \in C) &= Pr(R \in (\delta(s_i,u),1/2)) \\ &\leq \frac{\frac{1}{2} - \delta(s_i,u)}{\frac{1}{2}} \\ &\leq 2(\frac{1}{2} - \delta(s_i,u)) \\ &\leq 2(\delta(s_i,v) - \delta(s_i,u)) \\ &\leq 2x_e \end{aligned}$$

■

Remark 13.3.7 *If $u \in X_i$ and $v \in X_j$, then part of e lies in $B(s_i,1/2)$ and part in $B(s_j,1/2)$. Observe that $(1/2 - \delta(s_i,u)) + (1/2 - \delta(s_j,v)) \leq x_e$. Therefore, $Pr(e \in C) \leq 2((1/2 - \delta(s_i,u)) + (1/2 - \delta(s_i,u))) \leq 2x_e$.*

Lemma 13.3.8 *The expected weight of the edges in the multiway cut is at most $2z^*$, where z^* is the value of the objective function returned by the LP relaxation, i.e., $z^* = \sum_{e \in E} w_e x_e$.*

Proof. Let C be the collection of edges in the cut with respect to some $R \in (0,1/2)$. We have already seen that for an arbitrary edge $e \in E$, $Pr(e \in C) \leq 2x_e$.

$$\begin{aligned} E[cost(C)] &= \sum_{e \in E} w_e Pr(e \in C) \\ &\leq \sum_{e \in E} w_e \times 2x_e \\ &= 2\sum_{e \in E} w_e x_e \\ &= 2z^* \end{aligned}$$

■

We summarize the result on multiway cuts in the following theorem.

Theorem 13.3.9 *Let $G = (V,E)$ be a simple (complete) graph where each edge has a non-negative real weight. Let $T \subset V$ be a set of terminals. We can find a set $C \subseteq E$ with the following properties:*

1. *$G - C$ has no path connecting any pair of terminals.*
2. *The total weight of the edges in C is at most 2 times the weight of an optimal multiway cut.*
3. *We can determine C in polynomial time using the solution of the relaxed LP.*

Next, we establish an integrality gap. Consider an unweighted star graph with $k+1$ vertices. It consists of k-leaves, and all of them are connected to a central node. Let the k leaves constitute the set T of terminals. Cost of an optimal solution $= k-1$, as it is achieved by removing any set of $k-1$ edges.

Note that the cost of the relaxed linear program is $k/2$. This is achieved by setting the cost of each edge to $1/2$.

Thus, the approximation factor $\frac{k-1}{\frac{k}{2}} = 2(1-\frac{1}{k})$. Hence, using this approach, we can't do better in the worst case. This is referred to as the *integrality gap*. A different LP relaxation yields a $\frac{3}{2}$-approximation, but we won't be discussing that here.

13.3.3 Multicuts in Graphs

The multicuts problem is defined as follows.
Input: A complete graph $G = (V, E)$ with non-negative weights on edges and a set of k-vertex pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$.
Output: A set of edges $C \subseteq E$ of minimum total weight so that $G \setminus C$ has no path between s_i and t_i for $i = 1, \ldots, k$.

Let C be a multiway cut that separates every pair $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$. Define an indicator variable x_e for each edge e as follows:

$$x_e = \begin{cases} 1, \text{ if } e \in C, \\ 0, \text{ otherwise} \end{cases}$$

The assignment of x_e values to each edge satisfies

1. Cost of the multicut equals $\sum_{e \in E} x_e w_e$.
2. For any pair (s_i, t_i), length of any path between them is ≥ 1, where the length of an edge e is its x_e value.
3. For any three distinct vertices u, v, w, $x_{uw} + x_{wv} \geq x_{uv}$.

The integer and relaxed linear programming formulations are as follows:

Integer Metric LP Formulation for Multicuts

$$\min \sum_{e \in E} w_e x_e$$

Subject to:

1. Membership in the cut: For each edge $e \in E$, $x_e \in \{0, 1\}$.
2. Cut Constraint: For every pair (s_i, t_i), $x_{s_i t_i} \geq 1$.
3. Triangle inequality: For any three distinct vertices u, v, w, $x_{uw} + x_{wv} \geq x_{uv}$.

We replace the constraint $x_e \in \{0,1\}$ for the relaxed linear program by $0 \leq x_e \leq 1$. The main steps of the algorithm for finding the edges in the multicut are as follows.

Initialize:

1. Choose an $R \in (0, 1/2)$, uniformly at random.
2. Initialize the cut $C \leftarrow \emptyset$.
3. Define k blocks $X_1 = \ldots = X_k = \emptyset$.
4. Unmark all the vertices of G.

Main Steps:

Step 1: Compute a random permutation of vertices $s_1, s_2, \ldots, s_k$. WLOG, assume the ordering is $s_1, s_2, \ldots, s_k$.

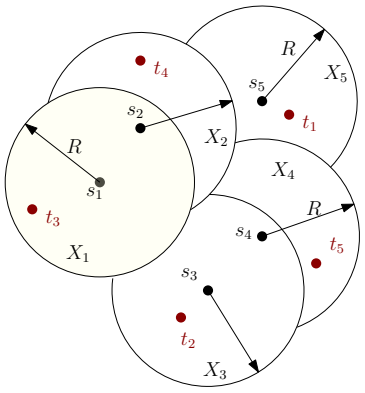
Step 2: Let $B_i(s_i, R)$ be the ball consisting of all the vertices within distance R of s_i.

For each s_i in the order of permutation, do:

For each unmarked vertex $v \in B(s_i, R)$, mark v and place it in the block X_i.

Step 3: For each edge $e = (u, v) \in E$, place it in the cut C if $u \in X_\alpha$ and $v \notin X_\alpha$.

Step 4: Return C.



We make the following observation.

Observation 13.3.10 $X_i = B(s_i, R) \setminus \bigcup_{j=1}^{i-1} B(s_j, R)$.

Lemma 13.3.11 *For each pair s_i, t_i, $i = 1, \ldots, k$, the following holds*

1. $t_i \notin X_i$.
2. *if $s_i \in X_j$ then $t_i \notin X_j$.*

Proof. Since $R < 1/2$ and from the linear program we know that $x_{s_i t_i} \geq 1$, $t_i \notin B(s_i, R)$. Since, $X_i \subseteq B(s_i, R) \implies t_i \notin X_i$.

If $s_i \in X_j$. The set X_j is defined by $s_j \implies \delta(s_j, s_i) < R < \frac{1}{2}$. All the vertices in X_j are within distance $< R$ of s_j. By the triangle inequality, any vertex in X_j is within distance $< 2R < 1$ from s_i. Since $\delta(s_i, t_i) \geq 1$, we have $t_i \notin X_j$. ∎

Now we estimate the probability of an edge to be present in the cut C.

Lemma 13.3.12 *$Pr(e \in C) \leq 2H_k x_e$, where $H_k = \sum_{i=1}^{k} \frac{1}{i} \approx \ln k$ is the k-th Harmonic number.*

Assuming that the above lemma holds, we establish that the expected cost of the cut C will be within a factor of $O(\log k)$ of an optimal cut that separates k terminal pairs.

$$\begin{aligned} E[cost(C)] &= E\left[\sum_{e \in C} w(e)\right] \\ &= \sum_{e \in E} w(e) Pr(e \in C) \\ &\leq \sum_{e \in E} 2H_k w_e x_e \\ &= 2H_k z^* \end{aligned}$$

The result is summarized in the following theorem.

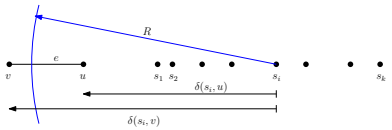
Theorem 13.3.13 *Multicuts in a graph can be approximated within a factor of $O(\log k)$ in polynomial time that separates k- terminal pairs.*

Now we present the proof of Lemma 13.3.12.

Proof. Let $e = (u, v)$. We will consider the distance from $s_1, \ldots, s_k$ to e. We define the distance from s_i to $e = (u, v)$ as $d(s_i, e) = \min(\delta(s_i, u), \delta(s_i, v))$. Without loss of generality, assume that the order of vertices according to increasing distance from e be $s_1, s_2, \ldots, s_k$. In the random ordering of the vertices in $s_1, \ldots, s_k$, consider when an endpoint u or v of e gets marked for the first time. Say it is u, and it gets marked by s_i. Without loss of generality, assume that $\delta(s_i, u) \leq \delta(s_i, v)$. This implies that $u \in X_i$ and we have two cases: (a) $v \in X_i$, and (b) $v \notin X_i$.

Consider Case (a), where $v \in X_i$. That is v is also marked by s_i. Since both the ends of the edge $e = (uv)$ are in $X_i \implies e \notin C$.

Now consider Case (b), where $v \notin X_i$. In this case $e \in C$, and we say s_i *cuts* e. We want to estimate $Pr(s_i \text{ cuts } e)$. Observe that s_i cuts e because of the following



1. s_i marked u but not v.

2. $d(s_1, e) \leq d(s_2, e) \leq \cdots \leq d(s_k, e)$.

3. Among all the vertices $\{s_1, \ldots, s_k\}$, s_i is the first vertex that marks any of the endpoints of e.

4. In the random order, none of the vertices with a smaller distance to e than s_i appeared before s_i. Otherwise, s_i won't be the first vertex marking an end of e.

The probability that s_i comes before $s_1, \ldots, s_{i-1}$ in a random permutation is $1/i$. For s_i to cut e, given that s_i comes before $s_1, \ldots, s_{i-1}$, the radius $R \in (0, 1/2)$ should fall in the range $\delta(s_i, u) < R < \delta(s_i, v)$. This happens with probability $\leq \frac{\delta(s_i,v)-\delta(s_i,u)}{1/2} \leq \frac{x_e}{1/2} = 2x_e$. Thus the probability that s_i cuts e is $2x_e/i$.

Now the probability that e is cut by any of $s_1, \ldots, s_k$ is

$$\begin{aligned} &\leq \sum_{i=1}^{k} Pr(s_i \text{ cuts } e) \\ &= \sum_{i=1}^{k} \frac{1}{i} 2x_e \\ &= 2x_e \sum_{i=1}^{k} \frac{1}{i} \\ &= 2H_k x_e \end{aligned}$$

∎

13.4 *Fixed-Parameter Tractability*

We will discuss the techniques used in designing fixed-parameter tractable algorithms, including branch-and-bound, kernelization, Nemhauser-Trotter theorem using the LP, iterative compression, and color coding. For a detailed exposition, see [3]. We will use the vertex cover problem to illustrate some of these techniques. Recall that the vertex cover problem is formally stated as follows.
Input: A simple undirected graph $G = (V, E)$.
Output: A subset $S \subseteq V$ of smallest cardinality such that for each edge $e = (u, v) \in E$, at least one of u or v is in S.

[3] Rodney G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer Publishing Company, Incorporated, 2012; and Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015

Definition 13.4.1 Fixed-Parameter Tractable (FPT) Problems.
A problem is said to be fixed parameter tractable *with respect to a parameter k if there is an algorithm with running time $f(k)n^{O(1)}$, where n is the input size of the problem and f is independent of n.*

Some of the complexity results of the vertex cover problem are as follows.

- The decision problem is **NP**-Hard.
- A greedy 2-factor polynomial time approximation algorithm.
- Exact algorithms with the following run times:
  1. A naive algorithm running in $O(|V|^{k+1})$ time.
  2. An FPT algorithm running in $O(|V|2^k)$ time.

3. A faster FPT algorithm running in $O(|V| + |E| + k^2 2^k)$ time.

Note that the time complexity of the FPT algorithm for the vertex cover problem is polynomial in graph parameters $|V|$ and $|E|$ but exponential in k - the vertex cover size. If k is small, these algorithms are efficient.

13.4.1 An FPT Algorithm

First, we outline a naive algorithm for finding a vertex cover of size k in the graph $G = (V, E)$.

- Consider all subsets $S \subseteq V$ of size k.
- Check whether $G \setminus S$ is an independent set.

Observe that the above steps can be implemented in $\binom{n}{k} O(n + m) = O(n^k(n + m))$ time, where $n = |V|$ and $m = |E|$. Next, we try to find algorithms that are polynomial in the size of the graph but are exponential in the size of the vertex cover k. The decision version of the vertex cover problem is to determine if the given graph has a vertex cover of sie $\leq k$.

Decision problem: Whether $G = (V, E)$ has a vertex cover of size $\leq k$?

1. Find a maximal matching M of G.
2. If $|M| > k$, return G has no vertex cover of size $\leq k$.
3. Let S be the set of vertices constituting the edges in M.

   Note: S forms a vertex cover of G, and $I = V \setminus S$ is an independent set.
4. Consider all possible subsets A of S of size $\leq k$ and check whether $A \cup (N(S \setminus A) \cap I)$ is a vertex cover of G of size at most k. If true, output $A \cup (N(S \setminus A) \cap I)$ as the vertex cover. ($N(X)$ represents neighbors of vertices in X in G.)

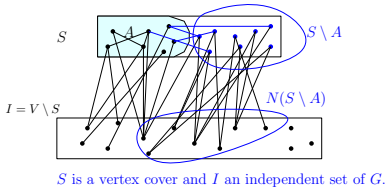


Figure 13.6: Sets I and S.

Observation 13.4.2 *Let S be a vertex cover of $G = (V, E)$. For a subset $A \subseteq S$, $A \cup (N(S \setminus A) \cap I)$ is a vertex cover of G if and only if there are no edges in E such that both of its end points are in $S \setminus A$.*

Let us analyze the running time of the above algorithm.

1. Finding maximal matching M in G requires $O(n + m)$ time.
2. Number of all possible subsets of size at most k of S is $2^{2k} = 4^k$.

3. Checking whether the set $A \cup (N(S \setminus A) \cap I)$ forms the vertex cover of size at most k requires $O(n+m)$ time.

4. Thus, the overall complexity is $4^k n^{O(1)}$.

5. The time complexity is of type $f(k)n^{O(1)}$ - a function in k (possibly exponential in k) and a polynomial function in the size of G.

Lemma 13.4.3 *The vertex cover problem is fixed-parameter tractable. Vertex cover of a graph $G = (V, E)$ on n vertices can be computed in $4^k n^{O(1)}$ time.*

13.4.2 A Branch-and-Bound Algorithm

The branch-and-bound algorithm is based on the following observation.

Observation 13.4.4 *For each edge $e = (uv)$ of G, any vertex-cover of G contains at least one of u or v.*

Algorithm VertexCoverFPT(G, k)

1. if G has no edges then return TRUE
2. if $k = 0$ then return FALSE
3. Let $e = (uv) \in E$ be an edge of G
4. if VertexCoverFPT$(G - u, k-1)$ then return TRUE
5. if VertexCoverFPT$(G - v, k-1)$ then return TRUE
6. return FALSE

The above algorithm is a decision algorithm - that answers whether G has a vertex cover of size $\leq k$. Remembering which subtree(s) of a node resulted in a positive answer, we can also find the vertex cover $S \subseteq V$ of size $\leq k$. The correctness of the above algorithm is based on induction on the vertex cover size k. If $k = 0 \implies$ G has no edges and Step 1 returns TRUE. Let $G = (V, E)$ be a graph with vertex cover S of size $k > 0$. To cover the edge $e = (uv)$, S must contain at least one of u or v. If $u \in S$, the graph $G - u$ (i.e., remove u and all its incident edges) has a vertex cover of size at most $k-1$. Step 4 returns TRUE. If $u \notin S$, then $v \in S$, $G - v$ has vertex cover of size at most $k-1$. Step 5 returns TRUE. And if both returns FALSE, then clearly G doesn't have a vertex cover of size $\leq k$.

To analyze the algorithm's time complexity, observe that the recursion 'tree' is a complete binary tree of height k. It consists

of 2^k leaves and 2^{k-1} internal nodes. Each internal node requires computation time of $O(|V|)$ (e.g., using adjacency list representation of graphs). For each leaf node, we need to check whether there are no edges in the remaining graph. Therefore, the overall running time is $(2^k + 2^{k-1})O(|V|) = O(|V|2^k)$.

Theorem 13.4.5 *Let $G = (V, E)$ be a simple undirected graph that has a vertex cover of size at most k. We can find a minimum vertex cover of G in $O(|V| \times 2^k)$ time.*

13.4.3 Kernelization

Next, we introduce the concept of *kernelization* and obtain a faster algorithm for the vertex cover problem. The main idea is as follows.

Given a problem instance Q with parameter k, we will execute an algorithm $\mathcal{A}$, running in polynomial time, to obtain an equivalent instance Q' with parameter k' such that Q has a solution if and only if Q' has a solution. We say $\mathcal{A}$ is a *kernelization algorithm* if the size of Q' and k' can be bounded by some function of k that is independent of the size of problem Q. It will be ideal to bound the size of Q' and k' by a polynomial function in k (preferably linear or quadratic functions). The kernelization algorithm $\mathcal{A}$ is usually broken down as a set of rules. For example, for the vertex cover problem, a simple rule is to remove all vertices of degree 0, and the resulting graph has a vertex of size $\leq k$ if and only if the original graph has a vertex cover of size $\leq k$.

Lemma 13.4.6 *If G has a vertex u of degree $> k$. Let $S \subseteq V$ be a vertex-cover of G with $|S| \leq k$. Then $u \in S$.*

Proof. If $u \notin S$, all its neighbors must be in S. But u has $> k$ neighbors and $|S| \leq k$. ∎

Thus, if the degree of $u > k$, we place u in the vertex cover. We remove u and all its incident edges from G, and seek for a vertex cover of size at most $k - 1$ in the resulting graph.

Corollary 13.4.7 *Let S' be the set of all vertices in G whose degree is $> k$. Let G' be the graph obtained from G by removing all vertices in S' (and their incident edges). G has a vertex cover of size $\leq k$ if and only if G' has a vertex cover of size $\leq k' = k - |S'|$.*

Lemma 13.4.8 *Let S' be set of all vertices in G whose degree is $> k$. Let G' be the graph obtained from G by removing all vertices in S' (and their incident edges). The degree of each vertex in G' is $\leq k$.*

Lemma 13.4.9 *If the graph G' has more than kk' edges, then G' doesn't have a vertex cover of size $\leq k'$.*

Proof. Each vertex can cover at most k edges in G'. Thus, k' vertices can cover at most kk' edges. ∎

Algorithm Kernelization-FPT(G, k)

1. Let S' be the vertices of G of degree $> k$. If $|S'| > k$, return FALSE.
2. Let $G' = G - S'$ and let $k' = k - |S'|$.
3. If G' has more than kk' edges, return FALSE.
4. Let G'' be the graph obtained after removing isolated vertices from G'.
5. Return VertexCoverFPT$(G'', k' = k - |S'|)$

We first establish the correctness of the algorithm. From Lemma 13.4.6, we know that all vertices in G of degree $> k$ need to be in the vertex cover. From Lemma 13.4.8, we know that if the graph G' has more than kk' edges, G cannot have a vertex cover of size $\leq k$. We use the algorithm VertexCoverFPT(G'', k'), which correctly returns the outcome of whether G'' has a vertex cover of size $\leq k'$.

Now we analyze the time complexity of the Algorithm Kernelization. Step 1 takes $O(|V| + |E|)$ time. Step 2 takes $O(|V| + |E|)$ time. Step 3 takes $O(|V| + |E|)$ time, and Step 4 takes $O(|V| + |E|)$ time. Consider the graph G'' obtained in Step 4. G'' has at most $kk' \leq k^2$ edges. Since G'' has no isolated vertices, it has $\leq 2k^2$ vertices. Graph G'' is the 'small' kernel for the vertex cover problem. We can execute an exponential time algorithm on G''.

By Theorem 13.4.5, in Step 5, execution of VertexCoverFPT(G'', k') takes $O(k^2 \times 2^k)$ time.

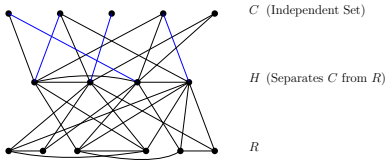
Theorem 13.4.10 *Let $G = (V, E)$ be a simple undirected graph with a vertex cover of size at most k. The vertex cover problem admits a kernel consisting of $O(k^2)$ vertices and $O(k^2)$ edges. We can find the minimum vertex cover of G in $O(|V| + |E| + k^2 2^k)$ time.*

13.4.4 Crown Decomposition

This is a general kernelization technique. The *crown decomposition* of a graph is defined as follows.

Definition 13.4.11 *Crown decomposition of a graph $G = (V, E)$ is a partitioning of the set of vertices V in three disjoint sets $V = C \cup H \cup R$ such that*

1. *There is no edge between vertices in C and R. H separates C from R.*
2. *C is a non-empty independent set.*
3. *There is a matching of size $|H|$ in the bipartite graph induced between the vertices in C and H. I.e., the matching saturates the vertices in H.*



We will show that the following holds.

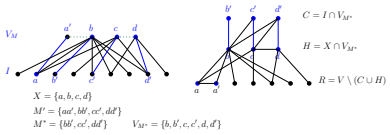
Lemma 13.4.12 *Let $G = (V, E)$ be a graph with at least $3k + 1$ vertices; none are isolated. In polynomial time we can determine either G has a matching of size at least $k + 1$ or find its crown decomposition.*

We can use any of the matching algorithms to determine whether G has a matching of size $\geq k + 1$ in polynomial time. Assume that all possible matchings have fewer than $k + 1$ edges.

Let M be a maximal matching of G. Let V_M be the set of vertices corresponding to edges in M. The vertices $I = V \setminus V_M$ form an independent set. Consider the bipartite graph $B(V_M, I)$ consisting only of edges between V_M and I in G. Let M' be a maximum matching in B and let X be a minimum vertex cover of B. Now $|X| = |M'| \leq k$, as B is bipartite graph and maximum matching in G has $< k + 1$ edges (by assumption).

We claim that $X \cap V_M \neq \emptyset$, and prove this using contradiction. Suppose not, i.e., $X \cap V_M = \emptyset$. This implies that $X \subseteq I$. We claim that $X = I$. If so, $|V_M| + |I| \leq 2k + k = 3k$, and that contradicts the fact that G has at least $3k + 1$ vertices, and thus it can't be that $V_M \cap X = \emptyset$. Now suppose $X \neq I$. Let $v \in I \setminus X$. Since no vertex of G is isolated, there is an edge uv incident on v where $u \in V_M$. But to cover the edge uv, we need to have $u \in X$. But we assumed that $V_M \cap X = \emptyset$.

Now we have that $X \cap V_M \neq \emptyset$. Since $|X| = |M'|$, exactly one end of each edge of M' is in X. Let $M^* \subseteq M'$ such that every edge in M^* has one endpoint in $X \cap V_M$. Let V_{M^*} be the union of all vertices defining the edges in M^*.



Define the sets C, H, and R for the crown decomposition as follows: $H = X \cap V_{M^*}$; $C = I \cap V_{M^*}$; $R = V \setminus (H \cup C)$. We establish some of the properties of these sets.

Lemma 13.4.13 *The set $C = I \cap V_{M^*}$ is a non-empty independent set.*

Proof. The set C is independent as I is independent. Moreover, $C \neq \emptyset$ as $X \cap V_M \neq \emptyset$, and each edge in the matching M' contributes exactly one endpoint to the vertex cover X of $B(V_M, I)$. ∎

Lemma 13.4.14 *The set $H = X \cap V_{M^*}$ separates C from R. Moreover, the induced bipartite graph on $C \cup H$ has a matching of size $|H|$.*

Proof. For any vertex $v \in C = I \cap V_{M^*}$ there exists $u \in H = X \cap V_{M^*}$ such that $uv \in M^* \subseteq M'$ (and $u \in X$). This implies that $v \notin X$ as for any edge $uv \in M'$ exactly one of its ends is in X. Thus, $C \cup H$ has a matching of size $|H|$. Since $v \in I$ and $v \notin X$, all neighbors of v in $B(V_M, I)$ are in $X \cap V_{M^*} = H$. ∎

Now we analyze the time complexity of the computation of crown decomposition. The main computational steps are (a) finding a maximum matching in G, and (b) finding the sets C, H, and R. It is easy to see each step can be implemented in polynomial time. From Lemmas 13.4.13 and 13.4.14 and the above time complexity analysis, we have established the proof of Lemma **??**. Next, we construct a small kernel for vertex cover using Lemma **??**.

Algorithm VC-Kernel$\langle G, k\rangle$

1. Remove isolated vertices from G.
2. If G has $\leq 3k$ vertices, output G as the kernel and terminate.
3. Apply Lemma **??** on G. Either it reports that matching in G has $\geq k+1$ edges ($\implies$ G has a vertex cover of size $> k$) or a partitioning $V = C \cup H \cup R$.
4. Place all vertices in H in the vertex cover, and execute VC-Kernel$\langle G - H, k - |H|\rangle$.

Lemma 13.4.15 *Algorithm VC-Kernel$\langle G, k\rangle$ reports whether $G = (V, E)$ has a vertex cover of size $> k$ or outputs a kernel of size $\leq 3k$.*

Proof. If G has a matching of size $\geq k+1$, the vertex cover of G requires $\geq k+1$ vertices. Otherwise, consider the crown decomposition $V = C \cup H \cup R$. Recall C is an independent set and $H \neq \emptyset$. The set H separates C from R. A matching of size $|H|$ exists in the bipartite graph formed by C and H. This implies that H is a vertex cover of the induced graph of $C \cup H$. The subgraph $G - H$ consists of isolated vertices in C and possibly some isolated vertices in R. In the next call to VC-Kernel$\langle G - H, k - |H|\rangle$, they will be removed.

Observe that the crown decomposition reduces the problem of finding a vertex cover of size $\leq k - |H|$ in graph $G - H$. As $H \neq \emptyset$, $G - H$ is a smaller graph. Recursion terminates when G has fewer than $3k+1$ vertices. ∎

13.4.5 *Kernel from Linear Program*

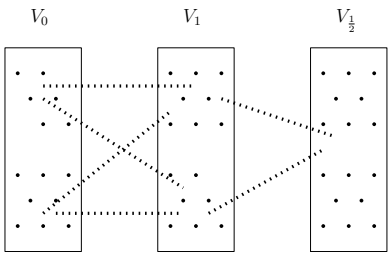
We first formulate an integer linear program for the vertex cover problem. Let $G = (V, E)$ be the given graph. Associate an indicator $0-1$ variable x_v for each vertex $v \in V$ that indicates whether v is in the cover. The LP is given by

$$\begin{aligned} \text{Objective Function:} \quad & \text{minimize } \sum_{v \in V} x_v \\ \text{Subject to:} \quad & \forall e = (uv) \in E : x_u + x_v \geq 1 \\ & x_v \in \{0, 1\} \end{aligned}$$

The above ILP results in a vertex cover. Each edge is covered because of the constraint $x_u + x_v \geq 1$, and at least one of u or v has to be 1, indicating that the corresponding vertex is in the cover. The relaxed LP is given by

$$\begin{aligned} \text{Objective Function:} \quad & \text{minimize } \sum_{v \in V} x_v \\ \text{Subject to:} \quad & \forall e = (uv) \in E : x_u + x_v \geq 1 \\ & \mathbf{0 \leq x_v \leq 1} \end{aligned}$$

Now the variables x_v's can take fractional values. The value of the objective function of the relaxed LP is a lower bound on the size of the vertex cover. We will construct a solution for the vertex cover problem using the fractional values.



Define three sets of vertices based on the LP values of variables x_v's:

$V_0 = \{v \in V | x_v < \frac{1}{2}\}$, $V_1 = \{v \in V | x_v > \frac{1}{2}\}$, and $V_{\frac{1}{2}} = \{v \in V | x_v = \frac{1}{2}\}$. We have the following observation.

Observation 13.4.16

1. *V_0, V_1, and $V_{\frac{1}{2}}$ is a partition of V, i.e. $V = V_0 \cup V_1 \cup V_{\frac{1}{2}}$.*
2. *The set V_0 is an independent set.*
3. *There are no edges between vertices in V_0 and $V_{\frac{1}{2}}$.*

The Nemhauser-Trotter theorem states there is a minimum vertex cover S such that $V_1 \subseteq S \subseteq V_1 \cup V_{\frac{1}{2}}$.

Theorem 13.4.17 *There is a minimum vertex cover $S \subseteq V$ of G such that $V_1 \subseteq S \subseteq V_1 \cup V_{\frac{1}{2}}$.*

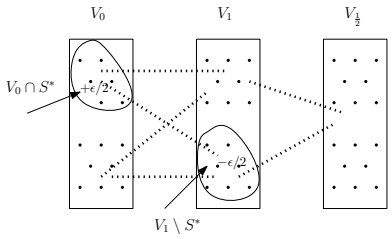
Proof. Let S^* be a minimum vertex cover of G. Let $S = (S^* \setminus V_0) \cup V_1$. Observe that S is a vertex cover of G as any vertex in V_0 is only

adjacent to vertices in V_1. Using contradiction, we show that S forms a minimum vertex cover.

Assume $|S| > |S^*|$. Observe that $|S| = |S^*| - |S^* \cap V_0| + |V_1 \setminus S^*|$. This implies that $|V_1 \setminus S^*| > |S^* \cap V_0|$ as we assumed $|S| > |S^*|$. We will construct another feasible solution of the relaxed LP with a smaller optimum value contradicting the optimality of LP.

Define $\epsilon = \min\{|x_v - \frac{1}{2}|, v \in V_0 \cup V_1\}$.

Modify x_v values as follows.



- For all vertices $v \in V_1 \setminus S^*$, set $y_v = x_v - \frac{\epsilon}{2}$.
- For all vertices $v \in V_0 \cap S^*$, set $y_v = x_v + \frac{\epsilon}{2}$.
- For all remaining vertices, set $y_v = x_v$.

Note that $\sum x_v > \sum y_v$, as we had $|V_1 \setminus S^*| > |S^* \cap V_0|$. We will show that y_v values satisfy the constraints of relaxed LP, and this will imply that x_v's are not optimal, and it will lead to a contradiction to the optimality of LP.

Consider any edge $e = (uv) \in G$. We need to show that $y_u + y_v \geq 1$. Consider the cases where one of the end vertices of any edge is in $V_0 \cap S^*$ or $V_1 \setminus S^*$, as for all other edges $y_u + y_v = x_u + x_v \geq 1$.

First, suppose $u \in V_0 \cap S^*$. In this case, v can only be in V_1. If $v \in V_1 \setminus S^*$, $x_u + x_v = y_u + \epsilon/2 + y_v - \epsilon/2 = y_u + y_v \geq 1$. If $v \in V_1 \cap S^*$, $y_u + y_v = x_u + \epsilon/2 + x_v \geq x_u + x_v \geq 1$.

Now suppose $u \in V_1 \setminus S^*$. If $v \in V_0$, a similar argument applies. If $v \in V_{\frac{1}{2}}$, $y_u + y_v = x_u - \epsilon/2 + x_v \geq 1$ as $x_v = \frac{1}{2}$ and $x_u > \frac{1}{2} + \epsilon/2$. ∎

As a result of the above lemma, we know that an optimal vertex cover S satisfies $V_1 \subseteq S \subseteq V_{\frac{1}{2}} \cup V_1$. We perform the following steps to determine if G has a vertex cover of size $\leq k$.

Step 1: If the value returned by relaxed LP is $> k$. Report G has vertex cover of size $> k$ and Stop.

Step 2: Include V_1 in the vertex cover and determine if $G \setminus (V_0 \cup V_1)$ has a vertex cover of size $\leq k - |V_1|$.

Lemma 13.4.18 *G has a vertex of size $\leq k$ if and only if $G \setminus (V_0 \cup V_1)$ has a vertex cover of size $\leq k - |V_1|$.*

Proof. We know that there is a minimum vertex cover S of G such that $V_1 \subseteq S \subseteq V_{\frac{1}{2}} \cup V_1$. If S is a vertex cover of size $\leq k$ for G, $S \setminus V_1$ is a vertex cover of size $\leq k - |V_1|$ for the graph induced by $V \setminus (V_0 \cup V_1) = V_{\frac{1}{2}}$.

For the other direction, observe that the graph induced by V_0 is isolated and only has edges to the vertices in the set V_1. If S' is a

vertex cover of the graph induced by $V_{\frac{1}{2}}$, $S' \cup V_1$ is a vertex cover of G. □ ∎

Lemma 13.4.19 $|V_{\frac{1}{2}}| \leq 2k$.

Proof. By definition, the linear program has assigned each variable $x_v \in V_{\frac{1}{2}}$ the value of $\frac{1}{2}$. Thus,

$$\begin{aligned} |V_1| &= \sum_{v \in V_{\frac{1}{2}}} 2x_v \\ &\leq 2 \sum_{v \in V} x_v \\ &\leq 2k \qquad \square \end{aligned}$$

∎

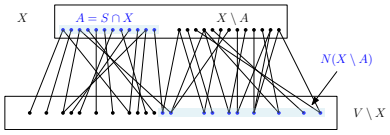
Lemma 13.4.20 *The induced graph on the vertices in $V_{\frac{1}{2}}$ forms a kernel for the vertex cover problem consisting of at most $2k$ vertices. Moreover, we can determine the kernel in polynomial time.*

Proof. Linear programs can be solved in polynomial time. Using the x_v values, we can form the sets V_0, V_1, and $V_{\frac{1}{2}}$ in $O(|V|)$ time. The computation of the induced graph on $V_{\frac{1}{2}}$ takes $O(|V| + |E|)$ time. Thus, we can determine the kernel of size $\leq 2k$ of G in polynomial time, provided it has a vertex cover of size $\leq k$. ∎

13.4.6 Iterative Compression

We start with the following property of vertex covers.

Lemma 13.4.21 *Let $X, S \subseteq V$ be two vertex covers of $G = (V, E)$. Let $A = S \cap X$, and let $N(X \setminus A)$ represent the neighbors of vertices in $X \setminus A$ in the set $V \setminus X$. The set $Y = A \cup N(X \setminus A)$ is a vertex cover of G if the graph induced by the vertices in $X \setminus A$ is an independent set.*



Proof. Since X is a vertex cover, $V \setminus X$ is an independent set. As $A \subseteq Y$, all edges incident to A are covered. Since $N(X \setminus A) \subseteq Y$, all the edges incident to $N(X \setminus A)$ are covered. If $X \setminus A$ is independent, Y is a vertex cover of G. ∎

Now we present the main idea of the iterative compression technique, where we construct a smaller vertex cover from a given larger vertex cover. Assume that we have the following.
Input: $X \subseteq V$, $G = (V, E)$, $|X| = k + 1$, and X is a vertex cover of G.
Output: Does G contain a vertex cover of size $\leq k$?

Our method is to select an arbitrary subset $A \subset X$ of $\leq k$ vertices and check whether there exists a vertex cover $S \supseteq A$ consisting of k vertices using Lemma 13.4.21.

We make the following observation.

Observation 13.4.22 *Let $N(X \setminus A)$ represents the neighbors of $X \setminus A$ in $V \setminus X$. Set $S = A \cup N(X \setminus A)$. The set S is the required vertex cover of G if*

1. $|S| \leq k$.
2. *There are no edges in the graph induced by $X \setminus A$.*

The compression algorithm for testing whether G has a vertex cover of size $\leq k$ is as follows.

Compression algorithm for testing if G has a vertex cover of size $\leq k$

Step 1: Consider an arbitrary permutation of vertices of G. Let it be $v_1, \ldots, v_n$.

Step 2: Let G_k be the graph induced by vertices $V_k = \{v_1, \ldots, v_k\}$. Note that $X = V_k$ is a vertex cover of G_k of size k.

Step 3: For $i := k+1$ to n do

1. Compute G_i by adding the vertex v_i and all of its incident edges to G_{i-1}. Note that $V_i = \{v_1, \ldots, v_i\}$.
2. Set $X \leftarrow \{v_i\} \cup X$. Note that X is a vertex cover of G_i.
3. If $|X| = k+1$, check whether there exists a vertex cover $S \subset V_i$ of size $\leq k$ for G_i. If so, set $X \leftarrow S$; otherwise, report G doesn't have a vertex cover of size $\leq k$.

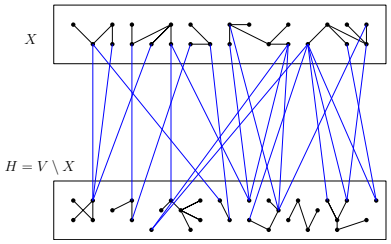
Lemma 13.4.23 *The above algorithm correctly determines whether G has a vertex cover of size $\leq k$ in $O(2^k|V|(|V|+|E|))$ time.*

Proof. Note that $G = G_n$. At the start of the iteration $i \in \{k+1, n\}$, we know that X is a vertex cover of size $\leq k$ for the graph G_{i-1}. If $|X \cup v_i| \leq k$, we already have a vertex cover of size $\leq k$ for G_i. Otherwise, we apply the observation as X is a vertex cover of G_i consisting of $k+1$ vertices, and we are seeking a vertex cover S of size at most k. We consider all possible subsets A of size $\leq k$ of X and determine whether there exists $S \supset A$ consisting of $\leq k$ vertices that cover G_i. The outcome is either we find a set S or we fail. If we find S, we set $X \leftarrow S$ and proceed to the next iteration. If we fail, G can't have a vertex cover of size $\leq k$ as its subgraph G_i doesn't admit a vertex cover of size $\leq k$.

Running time for each iteration is $O(2^k(|V| + |E|)$. ∎

Let us look at another example of the iterative compression technique. Let $G = (V, E)$ be a simple undirected graph. A subset $S \subset V$ of vertices is called a *feedback vertex set* (FVS) if the graph induced on the vertices $V \setminus S$ (denoted by $G(V \setminus S)$) is acyclic. The decision problem is whether G contains an FVS of size at most k.

We will first look into a specific version of the FVS problem, called the *disjoint feedback vertex cover* problem, and show how an iterative compression technique can be applied to answer the decision problem. For the disjoint feedback vertex set problem, the input consists of $G = (V, E)$, a parameter k, and an FVS $X \subset V$ of size $k + 1$. We need to decide whether G has FVS $S \subseteq V \setminus X$ of size $\leq k$. We denote this problem as D-FVS(G, X, k).



First, we make the following observation.

Observation 13.4.24

- *If X is FVS of $G = (V, E)$, the graph $G(H = V \setminus X)$ induced on the vertices $H = V \setminus X$ is a forest.*
- *Some $S \subset H$ can be FVS of G provided that the graph $G(X)$ induced on the vertices of X is acyclic.*

We exhaustively apply the following reduction rules to simplify the graph to find a disjoint FVS.

R1: Delete all vertices of degree ≤ 1 from G as they can't be in any FVS of G.

R2: If $\exists v \in H$ that has two or more edges incident to the same component in X (i.e., $G(X \cup \{v\})$ has cycle(s)), v has to be in FVS. Thus, remove v from G and solve D-FVS$(G \setminus \{v\}, X, k-1)$. If $k < 0$, report G doesn't have a D-FVS of size $\leq k$.

R3: Let $v \in H$ be a vertex of degree two in G and let u and w be its neighbors. If u or $w \in H$, remove v and add an edge uw (this may create a multi-edge between u and w).

Note that

1. In R3, if both $u, w \in X$, no shortcut is added. The reason is that none of the vertices in X can be in D-FVS.
2. After rules R1-R3 are applied exhaustively, each vertex in H has degree at least 2. For all non-isolated vertices in $G(H)$, their degree is ≥ 3.

Let us see the graph's structure obtained after exhaustively applying the above rules.

Observation 13.4.25 *Let G be the graph obtained after exhaustively applying the reduction rules R1-R3.*

1. *The number of connected components in $G(X)$ is $\leq k+1$.*
2. *Consider the forest $G(H)$ induced by vertices in H. For any isolated vertex in $G(H)$, its degree is ≥ 2 in G. For any non-isolated vertex in $G(H)$, its degree is ≥ 3 in G.*

Perform the following branching steps for any degree ≤ 1 vertex v of $G(H)$. During each branching, we also apply the reduction rules.

$v \in$ D-FVS: Execute D-FVS$(G \setminus \{v\}, X, k-1)$.

$v \notin$ D-FVS: Move v to the set X, merge the components in X that are adjacent to v, and execute D-FVS$(G, X \cup v, k)$.

We make some observations about the branching process.

Observation 13.4.26

1. *In each call to branching, we either reduce k by 1, or reduce the number of connected components in X by at least 1. Therefore, the branching process terminates in at most $2k+1$ steps, as $\leq k+1$ components are in $G(X)$.*
2. *Moving a degree ≤ 1 vertex $v \in G(H)$ to X is safe as $G(X \cup \{v\})$ is acyclic. Otherwise, we would have applied the reduction rule R2.*
3. *If ever $k < 0$, we terminate and report that D-FVS(G, X, k) has no solution.*
4. *We may reach a situation during branching where we have a single component in $G(X)$. Remember that we are still applying the reduction rules R1-R3, and that ensures what vertices will be added to D-FVS.*

Now we have

Lemma 13.4.27 *Discrete feedback vertex set problem D-FVS(G, X, k) can be solved in $O(4^k n^{O(1)})$ time, where n is the number of vertices in G.*

Proof. Rules R1-R3 can be implemented in polynomial time with respect to the size of G. The branching process terminates in at most $2k+1$ steps, where in each step, either we include a vertex v of degree ≤ 1 of $G(H)$ in D-FVS or exclude it. Thus, the branching tree has $2^{2k+1} = O(4^k)$ nodes. ∎

Our iterative compression algorithm for testing whether G has FVS of size $\leq k$ is as follows.

Iterative compression algorithm for FVS

Step 1: Consider an arbitrary permutation of vertices of G. Let it be $v_1, \ldots, v_n$.

Step 2: Let G_k be the graph induced by vertices $V_k = \{v_1, \ldots, v_k\}$. Note that $X = V_k$ is an FVS of G_k of size k.

Step 3: For $i := k+1$ to n do

1. Compute G_i by adding vertex v_i and all of its incident edges to G_{i-1}. Now $V_i = \{v_1, \ldots, v_i\}$.
2. Set $X \leftarrow \{v_i\} \cup X$. Note that X is a FVS of G_i.
3. If $|X| = k+1$, check whether there exists a FVS $S \subset V_i$ of size $\leq k$ for G_i. If so, set $X \leftarrow S$; otherwise, report G doesn't have an FVS of size $\leq k$.

   To find S, we try all subsets $A \subset X$ and solve for D-FVS$(G(V_i) \setminus A, X \setminus A, k - |A|)$.

Theorem 13.4.28 *For a given graph G and a parameter k, we can check in $5^k n^{O(1)}$ time whether G has a feedback vertex set of size at most k, where n is the number of vertices in G.*

Proof. Recall that in the D-FVS(G, X, k) problem, the input consists of a graph G, a parameter k, a FVS $X \subset V$ of size $k+1$, and the problem is to decide whether G has FVS $S \subseteq V \setminus X$ of size $\leq k$.

Consider any iteration $i \geq k+1$ of the algorithm. We have the FVS $X \cup \{v_i\}$ of size $\leq k+1$ for G_i. Thus, $G_i(V_i \setminus X)$ is a forest. Our task is to decide if a subset $S \subset V_i$ of size $\leq k$ exists such that S is FVS of G_i.

For this, guess which vertices of S are from X. Assume $A = S \cap X$ and consider the graph $G_i(V_i \setminus A)$. We want an FVS of size $\leq k - |A|$ in $G_i(V_i \setminus A)$ where all its vertices are from the set $V_i \setminus X$. This is precisely the D-FVS$(G(V_i) \setminus A, X \setminus A, k - |A|)$ problem.

Next, we analyze the running time. In iteration $i \geq k+1$, we try all possible subsets A of X, where $|X| = k+1$, and for each subset A, we make a call to an appropriate D-FVS$(G(V_i) \setminus A, X \setminus A, k - |A|)$ problem. We know that the running time for the D-FVS problem is $4^{k-|A|} n^{O(1)}$. Therefore, the total running time for the i-th iteration is

$$\sum_{j=0}^{k} \binom{k+1}{j} 4^{k-j} n^{O(1)} = 5^k n^{O(1)}$$

Note that $(1+4)^k = \sum_{j=0}^{k} \binom{k+1}{j} 1^j \cdot 4^{k-j} = 5^k$. Since i ranges from $k+1$ to n, the total running time for the FVS decision problem is $5^k n^{O(1)}$. ∎

13.4.7 Color Coding

Consider the *longest path* problem in an undirected graph.

Input: An undirected simple graph $G = (V, E)$ on n vertices and a positive integer $k \leq n$.

Output: Does there exists a simple path consisting of at least k vertices in G.

The color coding scheme due to Alon, Yuster and Zwick [4] answers the decision problem as follows.

[4] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995

Color coding for finding long paths

Step 1 Repeat Steps 2 and 3 for $O(k^k \ln n)$ times.

Step 2 Color the vertices of G uniformly at random independently from $\{1, \ldots, k\}$.

Step 3 Check if there exists a path $\Pi = (v_1, \ldots, v_k)$ such that each vertex v_i is colored i. If such a path exists, output TRUE and terminate.

Step 4 Output G contains no simple path of length k.

Lemma 13.4.29 *If there exists a simple path $\Pi = (v_1, \ldots, v_k)$ in G, the probability that for all $i \in \{1, \ldots, k\}$, v_i is assigned color i is $1/k^k$.*

Lemma 13.4.30 *If a simple path $\Pi = (v_1, \ldots, v_k)$ exists in G, the probability that the above algorithm fails to report such a path is at most $1/n$.*

Proof. The probability that any particular trial finds the path Π is $p = 1/k^k$. This is the probability of success in a trial. Therefore, the probability of failure is $(1 - p)$. Let X be the number of trials till we see the first success. X is a geometric random variable. Now, $Pr(X > \frac{1}{p} \ln n) \leq (1 - p)^{\frac{1}{p} \ln n} \leq e^{-p \frac{1}{p} \ln n} = \frac{1}{n}$. Therefore, the probability that the algorithm cannot find Π after $k^k \ln n$ trials is at most $1/n$. ∎

Lemma 13.4.31 *If G contains a simple path of length k, the above algorithm finds a path in $O(k^k poly(n))$ time with a probability of at least $1 - 1/n$. If G contains no simple path of length k, the above algorithm always correctly reports the non-existence of such paths.*

Proof. If G contains a simple path Π of length k, we have seen that the probability of the success of the algorithm is at least $1 - 1/n$. For the running time, once we color the vertices of G, we need to determine if there is a simple path such that the first vertex is colored

1, the second one is colored 2, and so on. This can be checked in time proportional to the size of the graph. Observe that if G doesn't have a simple path of length k, no coloring can produce an affirmative answer in Step 3 of the algorithm. Hence the algorithm will always reach Step 4 and correctly answer the non-existence of such paths. ∎

In [7], it is shown that a slightly modified algorithm can reach a similar conclusion in $O((2e)^k poly(n))$ time. The above algorithm is altered as follows.

Faster color coding for finding long paths

Step 1 Repeat Steps 2 and 3 for $O(e^k \ln n)$ times.

Step 2 Color the vertices of G uniformly at random independently from $\{1, \ldots, k\}$.

Step 3 Check if there exists a *colorful path* $\Pi = (v_1, \ldots, v_k)$ in G. A path is colorful if each vertex on the path has a distinct color. If a colorful path exists, output TRUE and terminate.

Step 4 Output G contains no simple path of length k.

Observe that if there exists a path Π of length k in G, the probability that it is colorful in a random coloring is $\frac{k!}{k^k} \geq \frac{1}{e^k}$. (This uses the standard approximation, $k! \geq \frac{k}{e}^k$.) Observe that the probability of success is $1/e^k$ as compared to $1/k^k$ previously. Once we color the vertices of G, we need to find whether G has a colorful path of length k. We will employ dynamic programming that runs in $2^k poly(n)$ time for this. The details are left as an exercise. We summarize the result.

Theorem 13.4.32 *If G contains a simple path of length k, we can report one such path in $O((2e)^k poly(n))$ time with probability at least $1 - 1/n$. If G contains no simple path of length k, the algorithm always correctly reports the non-existence of such paths.*

13.5 Tree Metrics - In Progress

13.6 Exercises

13.1 *Show that a maximum independent set of a tree can be computed in linear time. Show that the maximum weight-independent set of a weighted tree can also be computed in linear time.*

13.2 *Consider the following problem on Trees.*

Input: *A rooted tree* $T = G = (V, E)$ *on* n *vertices and each edge has a positive weight* $w : E \to \Re^+$. *A set of* k*-vertex pairs* $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$, *where* $s_i \neq t_i$ *for all* $i = 1, \ldots, k$.
Output: *Find a set of edges* $C \subseteq E$ *of minimum total weight so that the graph* $G' = (V, E \setminus C)$ *has no path between* s_i *and* t_i *for* $i = 1, \ldots, k$.

Formulate this problem as a Linear Program. Each edge $e \in E$ *has a* $0-1$ *indicator variable* x_e *indicating the membership of* e *in* C.
Consider the following algorithm:

Step 1: $C \leftarrow \emptyset$. *Choose an* $R \in (0, 1/2)$ *uniformly at random.*

Step 2: Solve the Relaxed LP to obtain x_e *values for each edge* $e \in E$.

Step 3: Let r *be root of* T. *For each vertex* v, *compute the shortest path distance* $\delta(r, v)$ *from* r *to* v, *where the weight of an edge* e *is its* x_e *value.*

Step 4: For each edge $e = (uv) \in E$ *(assume* $\delta(r, u) \leq \delta(r, v)$*), place* e *in the cut* C *if* $\delta(r, u) \in (R + \frac{1}{2}i, R + \frac{1}{2}(i+1))$ *and* $\delta(r, v) > R + \frac{1}{2}(i+1)$, *for some integer* $i \geq 0$.

Step 5: Return the set of edges C.

Answer the following:

1. *Show that in the graph* $G - C$, *there is no path between* s_i *and* t_i *for any* $i \in \{1, \ldots, k\}$.

2. *For any edge* $e \in E$, *show that* $Pr(e \in C) \leq 2x_e$.

3. *Show that* $E[w(C)] \leq 2z^*$, *where* z^* *is the value of the objective function returned by the LP relaxation* ($z^* = \sum_{e \in E} w_e x_e$).

4. *Show that in polynomial time, we can find a cut whose total weight is within a factor of* 2 *of an optimal cut for a tree that separates each* s_i *from* t_i.

13.3 *Recall that in the k-median problem for a graph* $G = (V, E)$, *we need to find a set* F *of* k *vertices in* G *such that the* $cost(F) = \sum_{v \in V} d(v, F)$ *is minimized. Note that* $d(v, F)$ *is the shortest path distance from* v *to the nearest vertex in* F. *Using local search, we found a solution that is* 5*-optimal (or* $(3 + \frac{2}{t})$*-optimal if you allow* $t \geq 1$ *swaps) for general graphs. Suppose the graph* G *is a tree* T *on* n *nodes with non-negative edge lengths. Can we solve the k-median problem optimally on trees in polynomial time?*
Hint: Dynamic programming. Think of what the table will look like - what the subproblems are - how the solutions of subproblems can help find the solution to the problem. What approximately is the time complexity?

13.4 *The multiway cut problem is stated as follows:*

Multiway Cut Problem:

Input: *An undirected (complete) graph $G = (V, E)$ on n vertices and each edge has a positive weight $w : E \to \Re^+$. A set $T = \{s_1, \ldots, s_k\} \subset V$ of k vertices called terminals.*
Output: *Find a set of edges $C \subseteq E$ of minimum total weight so that the graph $G' = (V, E \setminus C)$ has no path between any pair of terminals in T.*

We will present an algorithm for this problem using the min-cost $s - t$ cut problem. Recall that in the min-cost $s - t$ cut problem:

Min-Cost $s - t$ cut Problem:

Input: *Same graph as above, but $T = \{s, t\}$.*
Output: *Find a set of edges $C \subseteq E$ of minimum total weight so that the graph $G' = (V, E \setminus C)$ has no path between s and t. I.e., C forms a cut of minimum weight that separates s from t.*

To solve the multiway cut problem we perform the following steps for each vertex $s_i \in T$:

1. *Add a new vertex t in V.*
2. *Add the $k - 1$ edges to E, s_1t, s_2t, …, $s_{i-1}t$, $s_{i+1}t$, …, s_kt, each of infinite weight.*
3. *Let the resulting graph be G_i. Compute a min cost $s_i - t$ cut in G_i and let it be C_i.*
4. *Restore the graph G by removing the added edges and the vertex t.*

We report $C = \bigcup_{i=1}^{k} C_i$ as the solution for the multiway cut problem.
Answer the following:

1. *Prove that C is a valid solution for the multiway cut problem, i.e., the graph $(V, E \setminus C)$ has no path that connects any pairs of distinct terminals in T.*
2. *Let C^* be an optimal multiway cut in G. For each $i = 1, \ldots, k$, let $C_i^* \subset C^*$ be the set of edges in C^* that separates s_i from all other terminals in $T \setminus \{s_i\}$. Show that $wt(C_i) \leq wt(C_i^*)$ for all $i = 1, \ldots, k$. (Note that $wt(S)$ represents the sum total of weights of edges in S, i.e. $\sum_{e \in S} w(e)$.*
3. *Show that any edge $e = (u, v) \in E$ appears in at most two sets in $\{C_1^*, C_2^*, \ldots, C_k^*\}$.*
4. *Show that $w(C) \leq 2w(C^*)$.*

5. *For any set $C_j \in \{C_1, \ldots, C_k\}$ show that $C \setminus C_j$ is a valid solution for the multiway cut problem.*

6. *Show that there is a set $C_j \in \{C_1, \ldots, C_k\}$ such that $w(C \setminus C_j) \leq (2 - \frac{2}{k})w(C^*)$.*

7. *Conclude that one can find an approximate multiway cut in G in polynomial time that is within a factor of $2(1 - \frac{1}{k})$ of an optimal cut.*

13.5 *Show that a maximum independent set of a maximal outerplanar graph can be computed in linear time.*

13.6 *In this exercise, we will design an approximation algorithm for computing an approximation to the maximum independent set in a planar graph using Baker's shifting technique* [5]. *Let H be a plane graph (i.e., an embedded planar graph). We assign levels to each vertex of H. All the vertices on the boundary of H are level 1 vertices. Remove all level 1 vertices. The boundary vertices in the reduced graph are level 2 vertices. Similarly, define vertices on other levels. We say H is k-*outerplanar *if no vertex in H of level $> k$ exists. It is known that a maximum independent set of a k-outerplanar graph can be computed in $O(8^k n)$ time. Let $G = (V, E)$ be an embedded planar graph on n vertices. Let $l(v)$ denote the level of each vertex v in G. Answer the following questions.*

[5] Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994

1. *Let $S_i \subset V$, where $S_i = \{v \in V : l(v) \mod k = i\}$. Consider the graph obtained by removing the vertices S_i (and their incident edges) from G, i.e., the graph induced by vertices in $V - S_i$. Let G_i be the resulting graph. Show that G_i is a collection of disjoint k-outerplanar graphs.*

2. *Show that the maximum independent set of G_i can be computed in $O(8^k n)$ time.*

3. *Let $G_0, \ldots, G_{k-1}$ be the induced graphs of $V - S_0, \ldots, V - S_{k-1}$, respectively. Let the maximum independent sets of $G_0, \ldots, G_{k-1}$ be $I_0, \ldots, I_{k-1}$, respectively. Let I^* be a maximum independent set of G. Show that $\max_{j \in \{0, k-1\}} |I_j| \geq (1 - \frac{1}{k})|I^*|$.*

4. *Conclude that for a constant k, a factor $(1 - 1/k)$ approximation to the maximum independent set of a planar graph can be computed in polynomial time.*

13.7 *Let G be a simple graph on n vertices where each vertex is colored by one of the colors from the set $\{1, \ldots, k\}$. Design a dynamic programming algorithm that reports whether G contains a colorful path of length k in $O(2^k poly(n))$ time.*

13.8 *Let $G = (V, E)$ be a simple undirected graph on n vertices. Design a (randomized) decision algorithm, running in $f(k)n^{O(1)}$ time that reports*

whether G has a set of k disjoint triangles. A triangle in G is formed by three distinct vertices $u, v, w \in V$, such that $uv, vw, uw \in E$. Two triangles are disjoint if they do not share any vertex.

13.9 *You are standing in the middle of a very long horizontal street at the location M, where houses are on the North side of the road, and the street is oriented East-West. Houses are equally spaced, say every 100 meters. Each home has a unique number, but the numbering is chaotic and doesn't follow a sequential order. In fact, we can't make any assumptions about the distribution of the house numbers. We want to visit house #3801, but we don't have any way of knowing whether it is on the East or the West side of M. We employ the following strategy to find the house #3801.*

1. *Initialize $x := 100$ meters.*

2. *Repeat the following steps till success:*

   (a) *From M, travel distance of x towards East. If we find #3801 on the way, we stop the search and terminate.*

   (b) *Return to M. Set $x := 2x$.*

   (c) *From M, travel distance of x towards West. We stop and terminate the search if we find #3801 on the way.*

   (d) *Return to M. Set $x := 2x$.*

Let D be the distance from M to the house # 3801. Assume that $D \geq 100$. Show that the total distance that we travel using the above method is $\leq 9D$. Note that the algorithm doesn't know the value of D.

The following exercises are about the **maximum k-coverage** problem. The input to the maximum k-covergae problem consists of a universe U of n elements, and a collection of m sets $\mathcal{S} = \{S_1, \ldots, S_m\}$ of U, where each $S_i \subset U$, and an integer parameter $k > 0$. The max k-coverage problem is to find k sets from $\mathcal{S}$ such that their union has the largest possible cardinality.

13.10 *Consider the following greedy algorithm for the maximum k-coverage problem.*

Greedy Algorithm for max k-coverage problem

Step 1: Set $\mathcal{S}' := \emptyset$.

Step 2: For $i := 1$ *to* k *do*

- *Pick the set from* $\mathcal{S}$ *that covers the maximum number of uncovered elements of* U.
- *Without loss of generality, let that set be* X.
- $\mathcal{S}' := \mathcal{S}' \cup X$.

Step 3: Output $\mathcal{S}'$

Answer the following questions:

1. *Construct an example for* $k = 2$, *where the competitive ratio of the greedy algorithm is at most* $3/4$.
   Hint: Consider three sets, S_1, S_2, *and* S_3, *where* S_2 *and* S_3 *are disjoint and both contain half of the elements of the universe. Choose set* S_1 *so that the greedy algorithm selects* S_1 *and one of* S_2 *or* S_3 *and has a competitive ratio of at most* $3/4$.
2. *Construct an example for* $k = 3$, *where the competitive ratio of the greedy algorithm is at most* $19/27$.
3. *Construct an example where the competitive ratio of the above greedy algorithm is at most* $1 - (1 - \frac{1}{k})^k$, *for large values of* k.
4. *Show that for* $k = 2$, *the competitive ratio of the greedy algorithm is always* $\geq 3/4$.

13.11 *In this exercise, we will show that the approximation ratio of the greedy algorithm for the max k-coverage problem is* $1 - \left(1 - \frac{1}{k}\right)^k \approx 1 - \frac{1}{e}$ *for large values of* k.

1. *Suppose that the greedy algorithm has selected* $i - 1$ *sets, and in this step, we want to select the i-th set. We assume* $i < k$. *Without loss of generality, let the selected* $i - 1$ *sets be* $S_1, \ldots, S_{i-1}$, *and* S_i *will be the set selected in the i-th step. Furthermore, assume that* $L_{i-1} = |\bigcup_{j=1}^{i-1} S_j|$ *is the total number of elements of* U *that are covered by the union of* $S_1, \ldots, S_{i-1}$. *Let* L^* *be the maximum number of elements in the union of an optimal choice of* k *sets from* $\mathcal{S}$. *Show that* S_i *covers at least* $\frac{1}{k}(L^* - L_{i-1})$ *new elements, i.e.,* $L_i - L_{i-1} \geq \frac{1}{k}(L^* - L_{i-1})$.
2. *Show that* $L_k \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) L^*$.
3. *Conclude that the competitive ratio of the greedy algorithm is* $1 - \left(1 - \frac{1}{k}\right)^k$ *and* $\lim_{k\to\infty} 1 - \left(1 - \frac{1}{k}\right)^k = 1 - \frac{1}{e}$.

13.12 *In the* minimum set cover *problem, the input consists of a universe U of n elements, and a collection of m sets $\mathcal{S} = \{S_1, \ldots, S_m\}$ of U, where each $S_i \subset U$, where $U = \bigcup_{X \in \mathcal{S}} X$. The problem is to find the minimum number of sets from $\mathcal{S}$ such that their union covers U. We design an approximation algorithm for the set cover problem by adapting the greedy algorithm for the maximum k- coverage problem for the given U and $\mathcal{S}$. Let L^* be the optimal number of elements in U that can be covered by k sets from $\mathcal{S}$. We know that the max coverage greedy algorithm returns k sets from $\mathcal{S}$ that covers at least $\left(1 - (1 - \frac{1}{k})^k\right) L^*$ elements of U. Since we want to cover all the elements of U, we keep running the greedy step (pick the set that covers the maximum number of uncovered elements) till all the elements of U are covered. After executing the greedy step l times, assume we cover all the elements.*

Show the following.

1. *Let L_i be the number of covered elements at the end of greedy step i. Let L^* be the maximum number of elements that can be covered by choosing k sets. Show that $L_i \geq \frac{L^*}{k} \sum_{j=0}^{i-1} \left(1 - \frac{1}{k}\right)^j$, for all $i = 1, \ldots, l$.*

2. *Assume that the size of an optimal set cover is k. This implies that the max-coverage by selecting optimal k sets is $n = |U|$. Consider the $i = (l-1)$-st iteration. Show that $n\left(1 - e^{-\frac{l-1}{k}}\right) \leq L_{l-1} \leq n - 1$.*

3. *Show that $l \leq (1 + \ln n)k$ and conclude that the competitive ratio of finding an approximate set cover using the greedy algorithm is $1 + \ln n$.*

13.13 *Let B be a Boolean matrix of dimensions $n \times n$. We say that a column j* hits *a row i if $B[i, j] = 1$. We say a set of columns $C \subset \{1, \ldots, n\}$ hits a set $R \subseteq \{1, \ldots, n\}$ of rows if for every $i \in R$, there exists a $j \in C$ such that $B[i, j] = 1$.*

For example, let $B = \begin{bmatrix} \mathbf{1} & 0 & \mathbf{1} & 1 \\ 0 & 0 & 0 & 1 \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & 1 & \mathbf{1} & 0 \end{bmatrix}$

For $C = \{1, 3\}$, the first and third columns hit the rows $R = \{1, 3, 4\}$ as $B[1, 1] = B[3, 1] = B[1, 3] = B[4, 3] = 1$.

Given a Boolean matrix B and a number c^*, $1 \leq c^* < n$, let r^* be the optimal number of rows that can be hit by c^* columns. Devise an algorithm, running in polynomial time in n, that finds c^* columns of B so that the number of rows hit by them is as close to r^* as possible. State the steps of your algorithm. Establish what is the fraction of rows your solution hits as compared to r^* in the worst case. Justify your claim.

The following exercises are about finding *max-weight independent set* of intervals on a line. Given a collection of n line segments $I =$

$\{I_1, \ldots, I_n\}$ on a horizontal line, where each line segment $I_i = [a_i, b_i]$ and has a non-negative weight w_i. Note that $a_i \leq b_i$. Further, we assume that these line segments are given with respect to the sorted order of the right endpoints, i.e., $b_1 \leq b_2 \leq \cdots \leq b_n$. Our problem is to find an independent set of $I' \subseteq I$, such that no two intervals in I' intersect, and among all possible independent sets, it has the largest weight. We will devise a 4 approximation algorithm by using a randomized rounding linear programming algorithm.

These exercises are based on notes of Chandra Chekuri on Approximation Algorithms.

13.14 *Show that if the weights of all intervals are the same, the maximum independent set can be computed in polynomial time.*

13.15 *Design an integer linear program (ILP) for the maximum weight-independent set of the intervals. For each interval $I_i \in I$, associate a $0-1$ variable x_i, where $x_i = 1$ if and only if the interval I_i is in the independent set. The objective function $OPT_{ILP} = \max \sum_{i=1}^{n} w_i x_i$. Let $S = \{I_j | x_j = 1, \text{ for } j = 1, \ldots, n\}$. Show that S is an independent set and $w(S) = \sum_{I_j \in S} w(I_j)$ equals the weight of a maximum weight independent set.*

13.16 *Consider the relaxed ILP, where the constraint $x_i \in \{0, 1\}$ is replaced by $0 \leq x_i \leq 1$. Show that the objective value of the relaxed LP, $OPT_{LP} \geq OPT_{ILP}$.*

13.17 *We solve the relaxed LP and obtain values for the variables $x_1, \ldots, x_n$. Since, these values may be fractions, we round them to integeral values as follows. For each x_i, independently with probability $x_i/2$, we round it to 1, else it is set to 0. Let the rounded value of x_i be x'_i. Note that $x'_i \in \{0, 1\}$. Define the set $R = \{I_j | x'_j = 1, \text{ for } j = 1, 1 \ldots, n\}$. Show that $E[w(R)] = OPT_{LP}/2$.*

13.18 *Define indicator random variables $Y_1, \ldots, Y_n$, where*

$$Y_j = \begin{cases} 1, \text{ if } I_j \in R \\ 0, \text{ otherwise} \end{cases}$$

Show that $Y_1, \ldots, Y_n$ are independent random variables. Show that $E[Y_j] = x_j/2$.

13.19 *Show that the intervals in R may not be independent. We extract a S of R by executing the following algorithm:*

1. $S := \emptyset$.
2. *For* $i := n$ *down to* 1 *do (note that the intervals are sorted with respect to their right endpoints)*
   If $I_i \in R$ *and* $S \cup \{I_i\}$ *is an independent set*
   $S := S \cup \{I_i\}$.

Show that S returned by the above algorithm is an independent set.

13.20 *Define indicator random variables* $Z_1, \ldots, Z_n$, *where*

$$Z_j = \begin{cases} 1, \text{ if } I_j \in S \\ 0, \text{ otherwise} \end{cases}$$

Show that the random variables $Z_1, \ldots, Z_n$ *may not be independent. For each interval* $I_i = (a_i, b_i)$, *define the set* A_i *to be all the intervals* I_j, *where* $j > i$ *and* $b_i \in I_j$. *Answer the following.*

1. *Show that if* $A_i \cap S \neq \emptyset$, $Z_i = 0$.
2. *Show that* $Pr(A_i \cap S \neq \emptyset) \leq Pr(A_i \cap R \neq \emptyset)$.
3. *Show that* $Pr(A_i \cap R \neq \emptyset) \leq 1/2$.
4. *Show that* $Pr(A_i \cap S = \emptyset) \geq 1/2$.

13.21 *Given the random variables* Y_i*'s and* Z_i*'s, show the following*

1. $Pr(Z_i = 0/Y_i = 1) \leq 1/2$.
2. $Pr(Z_i = 1/Y_i = 1) \geq 1/2$.

13.22 *Consider* $w(S) = \sum_{I_j \in S} w(I_j)$. *Answer the following*

1. *Show that* $Pr(Z_i = 1) = Pr(Y_i = 1) \cdot Pr(Z_i = 1/Y_i = 1)$.
2. *Show that* $E[w(S)] = \sum_{i \in \{1,\ldots,n\}} w_i \cdot Pr(Z_i = 1)$.
3. *Show that* $E[w(S)] \geq \frac{1}{4} OPT_{LP}$.
4. *Conclude that the above method returns an independent set* $S \subseteq I$, *whose weight is at least* $1/4$*-th of the weight of a maximum weight independent set of I.*

The following exercises are about finding a *max-weight independent set* among a collection of n weighted disks in plane. Let $D = \{D_1, \ldots, D_n\}$ be a collection of n disks. Each disk D_i has a positive weight w_i. Let c_i denotes the center of disk D_i, for $i = 1, \ldots, n$.

A subset $I \subseteq D$ is independent if no two disks in I have a point in common. We will design a constant factor approximation algorithm running in polynomial time. The decision problem, whether there is an independent set of size at least k, is NP-Hard.

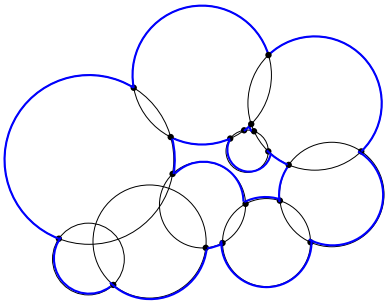


Figure 13.7: Boundary of union of discs $U(D)$ shown in blue.

13.23 *Consider the union of disks in D, i.e., $U(D) = \bigcup_{i=1}^{n} D_i$. See Figure 13.7. The boundary $U(D)$ is defined by the arcs of the disks and the vertices introduced by the intersection of a pair of disks. Define the graph $G = (V, E)$ on disks as follows. Each disk is a vertex in V, and two vertices u and v are connected if (a) disks corresponding to u and v have a non-empty intersection, and (b) at least one of the intersections points of these disks is on the boundary of $U(D)$. For simplicity, we take the centers of the disks as the vertices in the graph G. Show that the number of vertices (and arcs) on the boundary of $U(D)$ is $O(|D|)$ by answering the following questions.*

1. *Show that even if two disks D_u and D_v have a non-empty intersection, there may not be an edge between u and v in G. See Figure 13.8, and consider the disks centered at c_1 and c_3.*
2. *Let c_pc_q and c_rc_s be two arbitrary distinct edges in G. Note that c_pc_q and c_rc_s also correspond to segments in the plane. Since c_pc_q is an edge in G, at least one of the intersection points of disks D_p and D_q, say x_{pq}, is on the boundary of $U(D)$. Similarly, since c_rc_s is an edge in G, at least one of the intersection points of disks D_r and D_s, say x_{rs}, is on the boundary of $U(D)$. Let l be the perpendicular bisector of x_{pq} and x_{rs}. Argue that c_pc_q and c_rc_s do not intersect l and lie on opposite sides of l. See Figure 13.9.*
3. *Conclude that graph G is planar.*
4. *Show that the number of vertices in $U(D)$ is $O(|D|)$ by arguing that each edge of G can introduce at most two vertices on $U(D)$.*

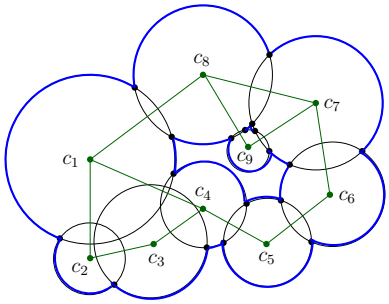


Figure 13.8: The graph $G = (V, E)$.

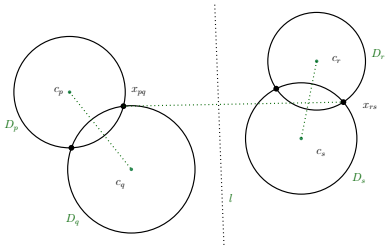


Figure 13.9: Edges c_pc_q and c_rc_s of G do not intersect.

13.24 *Given the notation of the previous exercise, define k-*level *in the arrangement of disks as the collection of all points $p \in \Re^2$ that are in exactly k disks of D, i.e., they are depth k points. See Figure 13.10. Define $\leq k$-*level *as the collection of all points whose depth is $\leq k$, i.e. it is the union of points in 0-level, 1-level, ..., k-level.*

Let $V_k(D)$ denote the set of vertices in $\leq k$-level. In this exercise, we will use the probabilistic method to show that $\leq k$-level has $O(nk)$ vertices. Pick a random sample R of disks from D, where each disk in D is chosen uniformly at random with probability $1/k$. Answer the following.

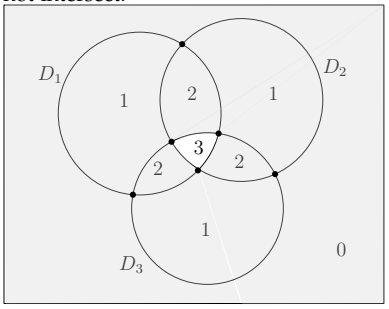


Figure 13.10: 0, 1, 2, and 3-levels in an arrangement of disks. $\leq$ 2-level is shown in shaded.

1. *Show that the expected number of disks in R is $E[|R|] = n/k$.*
2. *Let $U(R)$ denote the union of disks in R and let $V(R)$ be the set of vertices on the boundary of $U(R)$. Show that $E[|V(R)|] \leq c\frac{n}{k}$, for some constant c.*

3. *Define an indicator random variable X_v for each vertex $v \in V_k(D)$ as follows:*

$$X_v = \begin{cases} 1, v \text{ is a vertex on the boundary of } U(R), \text{ i.e., } v \in V(R) \\ 0, \text{ otherwise} \end{cases}$$

For a vertex $v \in V_k(D)$, $v \in V(R)$ if both the disks that define v in $U(R)$ are selected, and no disks that contain v are selected. Show that $Pr(X_v = 1) = (\frac{1}{k})^2(1 - \frac{1}{k})^\alpha$, where α is the number of disks that contain v in their interior for some $k \geq \alpha \geq 0$.

4. *Given that $1 - x \geq e^{-2x}$ for $0 \leq x \leq 1/2$. Show that $Pr(X_v = 1) \geq \frac{1}{e^2k^2}$.*

5. *Show that $E[|V(R)|] \geq \frac{|V_k(D)|}{e^2k^2}$.*

6. *Conclude that $|V_k(D)| = O(kn)$.*

13.25 *The* intersection graph *of a collection of disks $D = \{D_1, \ldots, D_n\}$ has a vertex corresponding to each disk, and there is an edge between two vertices if and only if the corresponding disks intersect. Show that for a collection of disks D with maximum depth k, the size of their intersection graph is $O(nk)$, i.e., it has n vertices and $O(nk)$ edges.*

13.26 *Answer the following.*

1. *For a graph $G = (V, E)$ on n vertices, if every subgraph has a vertex with degree $\leq \Delta$, show that it has an independent set of size $\geq \frac{n}{\Delta+1}$.*

2. *For a graph $G = (V, E)$ on n positively weighted vertices, if every subgraph has a vertex with degree $\leq \Delta$, show that it has an independent set of weight $\geq \frac{W}{\Delta+1}$, where W is the sum total of weights of all vertices. Hint: Color the graph with $\Delta + 1$ colors.*

13.27 *Show that for a collection of n positively weighted disks $D = \{D_1, \ldots, D_n\}$ in the plane with a total weight W and maximum depth k, we can find an independent set of weight $\Omega(W/k)$.*

13.28 *Construct an example where the set of n disks in the plane have $\Omega(n)$ depth and an independent set of size $\Omega(n)$.*

13.29 *Design an integer linear program ILP for determining the maximum weight independent set of disks for a given collection of n disks D. For each disk $D_i \in D$, define a $0 - 1$ indicator variable x_i stating the membership of D_i in the independent set. Note that for each region r in the arrangement of disks in D, $\sum_{r \in D_i} x_i \leq 1$. Let OPT_{ILP} be the weight of a maximum weight independent set, i.e., $OPT_{ILP} = \sum_{i=1}^{n} w_i x_i$, where $w_i \geq 0$ is the weight of the disk D_i.*

13.30 *For the integer linear program of the previous exercise, relax the constraint that $x_i \in \{0,1\}$ to $0 \leq x_i \leq 1$ to get a linear program LP. Show that the optimal value of the relaxed linear program OPT_{LP} is at least OPT_{ILP}.*

13.31 *Using the x_i values of LP of the previous exercise, we create a multiset of disks $\hat{D}$ as follows. Choose a large positive integer k. For each disk $D_i \in D$, create $\lceil kx_i \rceil$ copies of D_i, each of weight w_i. Answer the following.*

1. *Show that the total weight of disks in $\hat{D} \geq kOPT_{ILP}$.*

2. *Show that for all points $p \in \Re^2$, depth of p in $\hat{D} = \sum_{p \in D_i} \lceil kx_i \rceil \leq k + n$.*

13.32 *Show that for the set of n weighted disks in D, using the relaxed LP with parameter k and $\hat{D}$, we can obtain an independent set of weight $\Omega\left(\frac{k \cdot OPT}{k+n}\right)$. By choosing $k >> n$, show that we obtain an $O(1)$ factor approximation for the maximum weight independent set of n weighted disks D in the plane.*

13.33 *In the colored set cover problem, the input consists of a universe U of n elements and a collection of m sets $\{S_1, \ldots, S_m\}$, where each set $S_i \subset U$, and it is colored by one of the possible k colors. All sets having color i are said to belong to color class i. We want to report k sets, one from each color class so that their union has the largest cardinality. We execute the following procedure.*

> *Initialize : Pick k sets, one from each color class. Without loss of generality, let the picked sets be $\mathcal{S} = \{S_1, \ldots, S_k\}$, where S_i belongs to the color class i.*
>
> *Improvement Step: Repeat the following until no further improvement takes place:*
>
> *For each color class i, check if there is a set $S_i' \neq S_i$ such that by replacing S_i by S_i' in $\mathcal{S}$, the number of elements covered increases. If so, replace S_i by S_i' in $\mathcal{S}$.*

Answer the following

1. *Show that the above procedure terminates.*

2. *Let $\mathcal{S} = \{S_1, \ldots, S_k\}$ be the k sets returned by the procedure. Let $A = \cup_{i=1}^{k} S_i$. Let $\mathcal{O} = \{O_1, \ldots, O_k\}$ be the k sets in an optimal cover and let $O = \cup_{i=1}^{k} O_i$. Let $X = O \setminus A$, i.e., the elements of the universe U that are covered in optimal solution $\mathcal{O}$ but not in the sets returned by the algorithm. Define $d_i = |X \cap O_i|$. Show that $X \leq \sum_{i=1}^{k} d_i$.*

3. *Let n_i be the number of elements that are uniquely covered by S_i, i.e., these are the elements that are not in other sets of $\mathcal{S}$ except S_i. Show that $|A| \geq \sum_{i=1}^{k} n_i$.*

4. *Show that for any i, if we replace S_i by O_i in $\mathcal{S}$, we do not increase the total number of elements covered. Conclude that $n_i \geq d_i$.*

5. *Conclude that $|A| \geq \frac{1}{2}|O|$. (Note that $|X| = |O \setminus A|$.)*

6. *Conclude that the above procedure returns an $1/2$-optimal solution in polynomial time.*

13.34 *Let $G = (V, E)$ be a simple undirected graph, and we want to test if it has a vertex cover of size $\leq k$. We execute the following algorithm.*

Step 1: Consider an arbitrary permutation of vertices of G. Let it be $v_1, \ldots, v_n$.

Step 2: Let G_k be the graph induced by vertices $V_k = \{v_1, \ldots, v_k\}$. Note that $X = V_k$ is a vertex cover of G_k of size k.

Step 3: For $i := k + 1$ to n do

1. *Compute G_i by adding the vertex v_i and all of its incident edges to G_{i-1}. Note that $V_i = \{v_1, \ldots, v_i\}$.*
2. *Set $X \leftarrow \{v_i\} \cup X$. Note that X is a vertex cover of G_i.*
3. *If $|X| = k + 1$, check whether there exists a vertex cover $S \subset V_i$ of size $\leq k$ for G_i. If so, set $X \leftarrow S$; otherwise report G doesn't have a vertex cover of size $\leq k$.*

Show that the algorithm correctly answers in $O(2^k|V|(|V| + |E|))$ time whether G has a vertex cover of size $\leq k$.

14
Probabilistic Methods

We will focus on

1. Second moment method method.
2. Cliques in random graphs $G(n,p)$.
3. Thresholds for random geometric graphs.
4. Lovász Local Lemma

The material for Cliques in $G(n,1/2)$ is adapted from Nikhil Bansal's class notes and [1]. Thresholds for geometric graphs are from [2].

[1] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005

[2] Ahmad Biniaz, Evangelos Kranakis, Anil Maheshwari, and Michiel Smid. Plane and planarity thresholds for random geometric graphs. In *Proc. ALGOSENSORS 2015 (Patras, Greece)*, Lecture Notes in Computer Science, Berlin, Germany, 2015. Springer

14.1 Preliminaries

Recall basic probability definitions from Chapter 2. For a random variable X, its k-th moment is the expected value of X^k, $E[X^k]$, for $k \geq 1$. For example, the first moment is its expected value $E[X]$, second moment is related to the variance. The *variance* of a random variable X is defined to be $V[X] = E[(X-\mu)^2]$, where $\mu = E[X]$. Now consider

$$\begin{aligned} V[X] &= E[(X-\mu)^2] \\ &= E[X^2 - 2\mu X + \mu^2] \\ &= E[X^2] - 2\mu E[X] + E[\mu^2] \\ &= E[X^2] - E[X]^2 \end{aligned}$$

The (positive) square-root of variance is the standard deviation $\sigma(X) = \sqrt{V[X]}$. Moreover, for two independent random variables, X and Y,

$$Var[X+Y] = Var[X] + Var[Y].$$

Markov's inequality states that for a non-negative discrete random variable X and $t > 0$, a constant,

$$P(X \geq t) \leq \frac{E[X]}{t}.$$

This is one of the most basic inequality using the first moment for a non-negative random variable X. Note that if $X \geq 0$, $P(X > 0) = P(X \geq 1) \leq E[X]$. Observe that if $E[X] \to 0$, then $X = 0$ almost always. What can we say when $E[X] \to \infty$? Is it true that $X > 0$ almost always? In general this is not true, see Exercise 14.2.

Chebyshev's inequality states that the probability decreases at least quadratically in the number of standard deviations, i.e.,

$$Pr(|X - \mu| \geq t\sqrt{Var[X]}) \leq \frac{1}{t^2}.$$

To illustrate the two inequalities, consider the following experiment. Toss a fair coin n times and estimate what is the probability of getting $\geq \frac{3}{4}n$ heads? Let X_i be a 0-1 random variable indicating the outcome of the i-th toss, where $X_i = 1$ (respectively, $X_i = 0$) indicates that the outcome is a head (respectively, tail). Let $X = \sum_{i=1}^{n} X_i$ and X represents the total number of heads in n-tosses. Observe that $E[X_i] = \frac{1}{2}$ and $V[X_i] = E[{X_i}^2] - E[X_i]^2 = \frac{1}{2} * 1^2 + \frac{1}{2} * 0^2 - (\frac{1}{2})^2 = 1/4$. Thus $E[X] = \frac{n}{2}$ and since X_i's are independent, $V[X] = \frac{n}{4}$. To estimate $Pr[X \geq \frac{3}{4}n]$ using Markov's inequality, set $t = \frac{3}{2}$ and it results in $Pr[X \geq \frac{3}{2} * \frac{n}{2}] \leq \frac{2}{3}$. To estimate it using Chebyshev's inequality, $Pr(|X - \mu| \geq t\sqrt{Var[X]}) \leq \frac{1}{t^2}$, we need to set $t = \sqrt{\frac{n}{4}}$ to obtain $Pr(|X - \frac{n}{2}| \geq \sqrt{\frac{n}{4}}\sqrt{\frac{n}{4}}) \leq \frac{4}{n}$.

We will use the following result for the second moment method.

Theorem 14.1.1 *If $X \geq 0$ is a random variable taking integer values, $Pr(X = 0) \leq \frac{Var[X]}{E[X]^2}$. Moreover, if $Var[X] = o(E[X]^2)$ for large values of n, $Pr(X = 0) = o(1)$, i.e., $X > 0$ almost always.*

Proof. We will use Chebyshev's inequality.

Observe that $Pr(X = 0) \leq Pr(|X - E[X]| \geq E[X])$.

By substituting $t = \frac{E[X]}{\sqrt{Var[X]}}$, we have

$$Pr(|X - E[X]| \geq t\sqrt{Var[X]}) \leq \frac{1}{t^2},$$

and we obtain

$$Pr(X = 0) \leq \frac{Var[X]}{E[X]^2}.$$

If $Var[X] = o(E[X]^2)$, for large values of n we have ,

$$Pr(X = 0) = \lim_{n \to \infty} \frac{Var[X]}{E[X]^2} \to 0,$$

or in words, $Pr(X = 0) = o(1)$. ■

The *covariance* of two random variables Y and Z is defined as

$$Cov[Y, Z] = E[YZ] - E[Y]E[Z].$$

Note that if Y and Z are independent, $Cov[Y, Z] = 0$.

Let $X = \sum_{i=1}^{n} X_i$ be the sum of n random variables $X_1, \ldots, X_n$. Then, $V[X] = \sum_{i=1}^{n} V[X_i] + \sum_{i \neq j} Cov[X_i, X_j]$. See Exercise 14.1.

If each X_i is an indicator random variable with $P(X_i = 1) = p_i$, then $V[X_i] = p_i(1 - p_i) \leq p_i = E[X_i]$. Observe that in this case, $V[X] = \sum_{i=1}^{n} V[X_i] + \sum_{i \neq j} Cov[X_i, X_j] \leq E[X] + \sum_{i \neq j} Cov[X_i, X_j]$.

Let $G(n, p)$ be a *random graph* on n vertices where each edge between a pair of vertices occurs (independently of other edges) with probability p. A graph property $\mathcal{P}$ is *increasing* if a graph G satisfies $\mathcal{P}$, then by adding edges to G, the property $\mathcal{P}$ remains valid in G. Similarly, $\mathcal{P}$ is *decreasing* if a graph G satisfies $\mathcal{P}$, then by removing edges from G, the property $\mathcal{P}$ remains valid in G. $\mathcal{P}$ is called a *monotone* property if $\mathcal{P}$ is either increasing or decreasing. Connectivity and "having a clique of size k" are increasing monotone properties, while planarity and "being plane" are decreasing monotone properties in $G(n, r)$, where the value of r increases.

Define the term w.h.p. (with high probability) to mean that the probability tends to 1 as $n \to \infty$. For an increasing property $\mathcal{P}$, the *threshold* is a function $t(n)$ such that if $p = o(t(n))$ then w.h.p. $\mathcal{P}$ does not hold in $G(n, p)$, and if $p = \omega(t(n))$ then w.h.p. $\mathcal{P}$ holds in $G(n, p)$. Symmetrically, for a decreasing property $\mathcal{P}$, the threshold is a function $t(n)$ such that if $p = o(t(n))$ then w.h.p. $\mathcal{P}$ holds in $G(n, p)$, and if $p = \omega(t(n))$ then w.h.p. $\mathcal{P}$ does not hold in $G(n, p)$.

14.2 *Cliques in a random graph*

Consider a random graph $G(n, p = 1/2)$. Note that $G(n, p)$ is a graph on n vertices where each edge between a pair of vertices occurs (independently of other edges) with probability p. We will provide an estimate on the size of the largest clique in $G(n, 1/2)$ using the second moment method. In particular we will show

Theorem 14.2.1 *Let $\omega(G)$ denote the size of the largest clique in a graph G. Given an $\epsilon > 0$,*

(a) $Pr[\omega(G(n, 1/2)) > (2 + \epsilon) \log n] = o(1)$ and

(b) $Pr[(2 - \epsilon) \log n \leq \omega(G(n, 1/2)) \leq (2 + \epsilon) \log n] = 1 - o(1)$.

Proof. To prove the first part of this theorem, we will estimate how

many subsets of k vertices in $G(n,1/2)$ will form a clique. Fix a subset S of k vertices and let X_S be a 0-1 indicator r.v. indicating whether S is a clique or not. If S is a clique, $X_S = 1$, otherwise $X_S = 0$. Since each edge in S is chosen independent of other edges, $Pr[X_S = 1] = (\frac{1}{2})^{\binom{k}{2}}$ and $E[X_S] = Pr[X_S = 1]$.

Let X denote the total number of cliques of size k in $G(n,1/2)$ and let V be the set of vertices in $G(n,1/2)$. Observe that

$$E[X] = \sum_{S \subset V : |S| = k} E[X_S] = \binom{n}{k} \left(\frac{1}{2}\right)^{\binom{k}{2}} \leq n^k \left(\frac{1}{2}\right)^{\binom{k}{2}} = \left(\frac{\sqrt{2}n}{2^{k/2}}\right)^k.$$

By setting $k = (2+\epsilon)\log n$ as in Theorem, we observe that $\frac{\sqrt{2}n}{2^{k/2}} \leq \sqrt{2}n^{-\epsilon/2}$. For large values of n, $n^{-\epsilon/2} < 1$. Also $(n^{-\epsilon/2})^k$ is $o(1)$ (as n increases, $k = (2+\epsilon)\log n$ also increases). Thus by Markov's inequality $Pr[X \geq 1] \leq E[X] = o(1)$, i.e. the probability that there is a clique on k vertices in $G(n,1/2)$ is very small for large values of n.

Next we show the second part of Theorem 14.2.1 using the Second Moment Method. Let X_S and X be as before. Note that $\binom{n}{k} \geq (\frac{n}{k})^k$. Using the value of $k = (2-\epsilon)\log n$, observe that for large values of n, $E[X] = \binom{n}{k}\left(\frac{1}{2}\right)^{\binom{k}{2}} \to \infty$. Therefore, we will show that $Var[X] = o(E[X]^2)$ and then use Theorem 14.1.1 to conclude that the probability that there is no clique is $o(1)$ and hence the probability that there is a clique is $1 - o(1)$.

Let us first compute $Var[X] = E[X^2] - E[X]^2$. Note that

$$\begin{aligned} E[X^2] &= E\left[\sum_{S \subset V : |S| = k} \sum_{T \subset V : |T| = k} X_S X_T\right] \\ &= \sum_{S \subset V : |S| = k} \sum_{T \subset V : |T| = k} E[X_S X_T]. \end{aligned}$$

The last equality follows from the linearity of expectation. Moreover,

$$\begin{aligned} E[X]^2 &= E[X]E[X] \\ &= \sum_{S \subset V : |S| = k} E[X_S] \sum_{T \subset V : |T| = k} E[X_T] \\ &= \sum_{S \subset V : |S| = k} \sum_{T \subset V : |T| = k} E[X_S]E[X_T] \end{aligned}$$

Now consider two sets of k vertices S and T. If they have fewer than two vertices in common, whether S is a clique or not has no influence on whether T is a clique or not. Therefore, for $|S \cap T| < 2$, $E[X_S X_T] = E[X_S]E[X_T]$. Thus we need to consider

$$Var[X] = \sum_{S \subset V : |S| = k} \sum_{T \subset V : |T| = k} E[X_S X_T] - E[X_S]E[X_T] \qquad (14.1)$$

only for those pairs of S and T such that $|S \cap T| \geq 2$, since for the other values $E[X_S X_T]$ cancels $E[X_S]E[X_T]$. Thus, Equation 14.1 can be rewritten as

$$Var[X] = \sum_{S \subset V : |S|=k} \sum_{l=2}^{k} \sum_{T \subset V : |T|=k, |S \cap T|=l} E[X_S X_T] - E[X_S]E[X_T]. \quad (14.2)$$

Since, $E[X_S]E[X_T]$ is non-negative, we can obtain the following inequality from Equation 14.2.

$$Var[X] \leq \sum_{S \subset V : |S|=k} \sum_{l=2}^{k} \sum_{T \subset V : |T|=k, |S \cap T|=l} E[X_S X_T]. \quad (14.3)$$

Now observe that there are $\binom{n}{k}$ possibilities for choosing vertices for forming the set S of size k, there are $\binom{k}{l}$ possibilities for selecting l vertices that are common between S and T and there are $\binom{n-k}{k-l}$ possibilities for choosing the remaining vertices in $T \setminus S$. Hence,

$$Var[X] \leq \sum_{l=2}^{k} \binom{n}{k} \binom{k}{l} \binom{n-k}{k-l} E[X_S X_T]. \quad (14.4)$$

Let us compute $E[X_S X_T]$. Since X_S and X_T are 0-1 r.v., $E[X_S X_T]$ is same as the probability that S is a clique on k vertices and T is a clique on k vertices, where they have l common vertices. Let us estimate the number of edges in $S \cup T$. The two cliques of size k have a total of $2\binom{k}{2}$ edges, but we should remove $\binom{l}{2}$ edges from this count as they are counted twice. The total number of edges in $S \cup T$ is $2\binom{k}{2} - \binom{l}{2}$. Thus,

$$E[X_S X_T] = \left(\frac{1}{2}\right)^{2\binom{k}{2} - \binom{l}{2}} \quad (14.5)$$

Substituting expression for $E[X_S X_T]$ in Equation 14.4, we obtain

$$Var[X] \leq \sum_{l=2}^{k} \binom{n}{k} \binom{k}{l} \binom{n-k}{k-l} 2^{\binom{l}{2} - 2\binom{k}{2}}. \quad (14.6)$$

Next we show that $Var[X]/E[X]^2$ is $o(1)$.

Note that $E[X] = \binom{n}{k} 2^{-\binom{k}{2}}$.

Define

$$f(l) = \frac{\binom{n}{k}\binom{k}{l}\binom{n-k}{k-l} 2^{\binom{l}{2} - 2\binom{k}{2}}}{(\binom{n}{k} 2^{-\binom{k}{2}})^2}.$$

This can be rewritten as

$$f(l) = \frac{\binom{k}{l}\binom{n-k}{k-l} 2^{\binom{l}{2}}}{\binom{n}{k}}.$$

Thus,

$$\frac{Var[X]}{E[X]^2} \leq \sum_{l=2}^{k} f(l). \quad (14.7)$$

Consider

$$f(2) = \frac{\binom{k}{2}\binom{n-k}{k-2}2^{\binom{2}{2}}}{\binom{n}{k}}$$
$$= \frac{k(k-1)\binom{n-k}{k-2}}{\binom{n}{k}}$$

Using some loose bounds, e.g. $k = (2-\epsilon)\log n << n$, $\binom{n-k}{k-l} \leq \frac{(n-k)^{k-l}}{(k-l)!}$, and $\binom{n}{k} \geq \frac{(n-k)^k}{k!}$, we can show that $f(2) = o(1)$ by simplifying the expression of $f(2)$. It seems that (but I don't know how to show this in a simple way) that $f(l) = o(1)$ for $l = 2, \ldots, k$. This concludes the proof of Theorem 14.2.1. ■

From the above proof we see that if we want that a random graph $G(n,p)$ to have a clique of size $O(\log n)$ almost surely, we need to choose $p > 1/2$.

14.3 *Thresholds for Random Geometric Graphs*

Given a set P of points in the plane and a positive parameter r, the *disk graph* is the geometric graph with vertex set P that has a straight-line edge between two points $p, q \in P$ if and only if $|pq| \leq r$, where $|pq|$ denotes the Euclidean distance between p and q. If $r = 1$, then the disk graph is referred to as the *unit disk graph*. A *random geometric graph*, denoted by $G(n,r)$, is a geometric graph formed by choosing n points independently and uniformly at random in a unit square; two points are connected by a straight-line edge if and only if they are at Euclidean distance at most r, where $r = r(n)$ is a function of n and $r \to 0$ as $n \to \infty$.

We say that two line segments in the plane *cross* each other if they have a point in common that is interior to both segments. Two line segments are *non-crossing* if they do not cross. Note that two non-crossing line segments may share an endpoint. A geometric graph is said to be *plane* if its edges do not cross, and *non-plane*, otherwise. A graph is *planar* if and only if it does not contain K_5 or $K_{3,3}$ as a minor.

A graph property $\mathcal{P}$ is *increasing* if a graph G satisfies $\mathcal{P}$, then by adding edges to G, the property $\mathcal{P}$ remains valid in G. Similarly, $\mathcal{P}$ is *decreasing* if a graph G satisfies $\mathcal{P}$, then by removing edges from G, the property $\mathcal{P}$ remains valid in G. $\mathcal{P}$ is called a *monotone* property if $\mathcal{P}$ is either increasing or decreasing. Connectivity and "having a clique of size k" are increasing monotone properties, while planarity and "being plane" are decreasing monotone properties in $G(n,r)$, where the value of r increases.

Define the term w.h.p. (with high probability) to mean that the probability tends to 1 as $n \to \infty$. For an increasing property $\mathcal{P}$, the

threshold is a function $t(n)$ such that if $r = o(t(n))$ then w.h.p. $\mathcal{P}$ does not hold in $G(n,r)$, and if $r = \omega(t(n))$ then w.h.p. $\mathcal{P}$ holds in $G(n,r)$. Symmetrically, for a decreasing property $\mathcal{P}$, the threshold is a function $t(n)$ such that if $r = o(t(n))$ then w.h.p. $\mathcal{P}$ holds in $G(n,r)$, and if $r = \omega(t(n))$ then w.h.p. $\mathcal{P}$ does not hold in $G(n,r)$.

We will investigate thresholds in random geometric graphs for having a connected subgraph of constant size, being plane, and being planar. In Section 14.3.1 we show that for a constant k, the distance threshold for having a connected subgraph on k points is $n^{\frac{-k}{2k-2}}$. We show that the same threshold is valid for the existence of a clique of size k. In Section 14.3.2, we prove that $n^{-2/3}$ is a distance threshold for a random geometric graph to be plane. In Section 14.3.3, we prove that $n^{-5/8}$ is a distance threshold for a random geometric graph to be planar.

14.3.1 *The threshold for having a connected subgraph on k points*

In this section, we look for the distance threshold for the existence of connected subgraphs of constant size; this is an increasing property. For a given constant k, we show that $n^{\frac{-k}{2k-2}}$ is the threshold function for the existence of a connected subgraph on k points in $G(n,r)$. Specifically, we show that if $r = o(n^{\frac{-k}{2k-2}})$, then w.h.p. $G(n,r)$ has no connected subgraph on k points, and if $r = \omega(n^{\frac{-k}{2k-2}})$, then w.h.p. $G(n,r)$ has a connected subgraph on k points. We also show that the same threshold function holds for the existence of a clique of size k.

Theorem 14.3.1 *Let $k \geq 2$ be an integer constant. Then, $n^{\frac{-k}{2k-2}}$ is a distance threshold function for $G(n,r)$ to have a connected subgraph on k points.*

Proof. Let $P_1, \ldots, P_{\binom{n}{k}}$ be an enumeration of all subsets of k points in $G(n,r)$. Let $DG[P_i]$ be the subgraph of $G(n,r)$ that is induced by P_i. Let X_i be the random variable such that

$$X_i = \begin{cases} 1 & \text{if } DG[P_i] \text{ is connected,} \\ 0 & \text{otherwise.} \end{cases}$$

Let the random variable X count the number of sets P_i for which $DG[P_i]$ is connected. It is clear that

$$X = \sum_{i=1}^{\binom{n}{k}} X_i. \tag{14.8}$$

Observe that $\mathrm{E}[X_i] = \Pr[X_i = 1]$. Since the random variables X_i have

identical distributions, we have

$$\mathrm{E}[X] = \binom{n}{k}\mathrm{E}[X_1]. \tag{14.9}$$

We obtain an upper bound and a lower bound for $\Pr[X_i = 1]$. First, partition the unit square into squares of side equal to r. Let $\{s_1, \ldots, s_{1/r^2}\}$ be the resulting set of squares. For a square s_t, let S_t be the $kr \times kr$ square which has s_t on its left bottom corner; see Figure 14.1(a). S_t contains at most k^2 squares each of side length r (each S_t on the boundary of the unit square contains less than k^2 squares). Let $A_{i,t}$ be the event that all points in P_i are contained in S_t. Observe that if $DG[P_i]$ is connected then P_i lies in S_t for some $t \in \{1, \ldots, 1/r^2\}$. Therefore,

$$\text{if } DG[P_i] \text{ is connected, then } (A_{i,1} \vee A_{i,2} \vee \cdots \vee A_{i,1/r^2}),$$

and hence, using the argument that all the k points must lie within the specified area of $kr \times kr$, we have

$$\Pr[X_i = 1] \leq \sum_{t=1}^{1/r^2} \Pr[A_{i,t}] \leq \sum_{t=1}^{1/r^2} (k^2 r^2)^k = k^{2k} r^{2k-2}. \tag{14.10}$$

Now, partition the unit square into squares with diagonal length equal to r. Each such square has side length equal to $r/\sqrt{2}$. Let $\{s_1, \ldots, s_{2/r^2}\}$ be the resulting set of squares. Let $B_{i,t}$ be the event that all points of P_i are in s_t. Observe that if all points of P_i are in the same square, then $DG[P_i]$ is a complete graph and hence connected. Therefore,

$$\text{if } (B_{i,1} \vee B_{i,2} \vee \cdots \vee B_{i,2/r^2}), \text{ then } DG[P_i] \text{ is connected,}$$

and hence we have

$$\Pr[X_i = 1] \geq \sum_{t=1}^{2/r^2} \Pr[B_{i,t}] = \sum_{t=1}^{2/r^2} \left(\frac{r^2}{2}\right)^k = \frac{1}{2^{k-1}} r^{2k-2}. \tag{14.11}$$

Since $k \geq 2$ is a constant, Inequalities (14.10) and (14.11) and Equation (14.9) imply that

$$\mathrm{E}[X_i] = \Theta(r^{2k-2}), \tag{14.12}$$

$$\mathrm{E}[X] = \Theta(n^k r^{2k-2}). \tag{14.13}$$

If $n \to \infty$ and $r = o(n^{\frac{-k}{2k-2}})$ we conclude that the following inequalities are valid

$$\begin{aligned} \Pr[X \geq 1] &\leq \mathrm{E}[X] \text{ (by Markov's Inequality)} \\ &= \Theta(n^k r^{2k-2}) \text{ (by (14.13))} \\ &= o(1). \end{aligned} \tag{14.14}$$

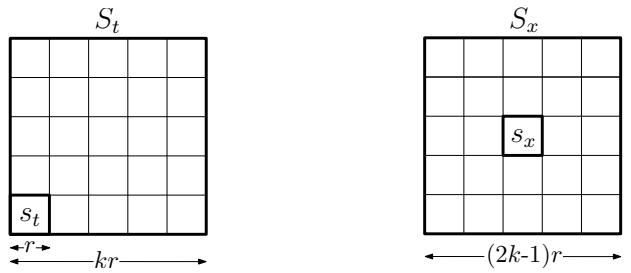


Figure 14.1: (a) The square S_t has s_t on its left bottom corner. (b) The square S_x which is centered at s_x.

Therefore, w.h.p. $G(n,r)$ has no connected subgraph on k points.

In the rest of the proof, we assume that $r = \omega(n^{\frac{-k}{2k-2}})$. In order to show that w.h.p. $G(n,r)$ has at least one connected subgraph on k vertices, we show, using the second moment method, that $\Pr[X=0] \to 0$ as $n \to \infty$. Recall from Chebyshev's inequality that

$$\Pr[X=0] \leq \frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2}. \tag{14.15}$$

Therefore, in order to show that $\Pr[X=0] \to 0$, it suffices to show that

$$\frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2} \to 0. \tag{14.16}$$

In view of Identity (14.8) we have

$$\mathrm{Var}(X) = \sum_{1 \leq i,j \leq \binom{n}{k}} \mathrm{Cov}(X_i, X_j), \tag{14.17}$$

where $\mathrm{Cov}(X_i,X_j) = \mathrm{E}[X_iX_j] - \mathrm{E}[X_i]\mathrm{E}[X_j] \leq \mathrm{E}[X_iX_j]$. If $|P_i \cap P_j| = 0$ then $DG[P_i]$ and $DG[P_j]$ are disjoint. Thus, the random variables X_i and X_j are independent, and hence $\mathrm{Cov}(X_i,X_j) = 0$. It is enough to consider the cases when P_i and P_j are not disjoint. Assume $|P_i \cap P_j| = w$, where $w \in \{1,\ldots,k\}$. Thus, in view of Equation (14.17), we have

$$\begin{aligned}\mathrm{Var}(X) &= \sum_{w=1}^{k} \sum_{|P_i \cap P_j| = w} \mathrm{Cov}(X_i, X_j) \\ &\leq \sum_{w=1}^{k} \sum_{|P_i \cap P_j| = w} \mathrm{E}[X_iX_j]. \end{aligned} \tag{14.18}$$

The computation of $\mathrm{E}[X_iX_j]$ involves some geometric considerations which are being discussed in detail below. Since X_i and X_j are 0-1 random variables, X_iX_j is a 0-1 random variable and

$$X_iX_j = \begin{cases} 1 & \text{if both } DG[P_i] \text{ and } DG[P_j] \text{ are connected,} \\ 0 & \text{otherwise.} \end{cases}$$

By the definition of the expected value we have

$$\begin{aligned}\mathrm{E}[X_iX_j] &= \Pr[X_j = 1|X_i = 1]\Pr[X_i = 1] \\ &= \Pr[X_j = 1|X_i = 1]\mathrm{E}[X_i]. \end{aligned} \tag{14.19}$$

By (14.12), $\mathrm{E}[X_i] = \Theta(r^{2k-2})$. It remains to compute $\Pr[X_j = 1|X_i = 1]$, i.e., the probability that $DG[P_j]$ is connected given that $DG[P_i]$ is connected. Consider the k-tuples P_i and P_j under the condition that $DG[P_i]$ is connected. Let x be a point in $P_i \cap P_j$. Partition the unit square into squares of side length equal to r. Let s_x be the square containing x. Let S_x be the $(2k-1)r \times (2k-1)r$ square centered at s_x. S_x contains at most $(2k-1)^2$ squares each of side length r (if S_x is on the boundary of the unit square then it contains less than $(2k-1)^2$ squares); see Figure 14.1(b). The area of S_x is at most $(2kr)^2$, and hence the probability that a specific point of P_j is in S_t is at most $4k^2r^2$. Since P_i and P_j share w points, in order for $DG[P_j]$ to be connected, the remaining $k-w$ points of P_j must lie in S_x. Thus, the probability that $DG[P_j]$ is connected given that $DG[P_i]$ is connected is at most $(4k^2r^2)^{k-w} \leq c_w r^{2k-2w}$, for some constant $c_w > 0$. Thus, $\Pr[X_j = 1|X_i = 1] \leq c_w r^{2k-2w}$. In view of Equation (14.19), we have

$$\mathrm{E}[X_iX_j] \leq c'_w \cdot r^{2k-2w} \cdot r^{2k-2} = c'_w r^{4k-2w-2}, \tag{14.20}$$

for some constant $c'_w > 0$.

Since P_i and P_j are k-tuples that share w points, $|P_i \cup P_j| = 2k-w$. There are $\binom{n}{2k-w}$ ways to choose $2k-w$ points for $P_i \cup P_j$. Since we choose w points for $P_i \cap P_j$, $k-w$ points for P_i alone, and $k-w$ points for P_j alone, there are $\binom{2k-w}{w,k-w,k-w}$ ways to split the $2k-w$ chosen points into P_i and P_j. Based on this and Inequality (14.20), Inequality (14.18) turns out to

$$\begin{aligned}\mathrm{Var}(X) &\leq \sum_{w=1}^{k} \sum_{|P_i \cap P_j| = w} \mathrm{E}[X_iX_j] \\ &\leq \sum_{w=1}^{k} \binom{n}{2k-w}\binom{2k-w}{w,k-w,k-w} c'_w r^{4k-2w-2} \\ &\leq \sum_{w=1}^{k} c''_w n^{2k-w} r^{4k-2w-2}. \end{aligned}$$

for some constants $c''_w > 0$. Consider (14.16) and note that by (14.13), $\mathrm{E}[X]^2 \geq c'' n^{2k} r^{4k-4}$, for some constant $c'' > 0$. Thus,

$$\begin{aligned}\frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2} &\leq \sum_{w=1}^{k} \frac{c''_w n^{2k-w} r^{4k-2w-2}}{c'' n^{2k} r^{4k-4}} = \sum_{w=1}^{k} \frac{c''_w}{c''} \cdot \frac{1}{n^w r^{2w-2}} \\ &= \frac{c''_1}{c''} \cdot \frac{1}{n^1 r^0} + \frac{c''_2}{c''} \cdot \frac{1}{n^2 r^2} + \cdots + \frac{c''_k}{c''} \cdot \frac{1}{n^k r^{2k-2}} \end{aligned} \tag{14.21}$$

Since $r = \omega(n^{\frac{-k}{2k-2}})$, all terms in (14.21) tend to zero. This proves the convergence in (14.16). Thus, $\Pr[X = 0] \to 0$ as $n \to \infty$. This implies that if $r = \omega(n^{\frac{-k}{2k-2}})$, then $G(n,r)$ has a connected subgraph on k vertices with high probability. ■

In the following theorem we show that if $k = O(1)$, then $n^{\frac{-k}{2k-2}}$ is also a threshold for $G(n,r)$ to have a clique of size k; this is an increasing property.

Theorem 14.3.2 *Let $k \geq 2$ be an integer constant. Then, $n^{\frac{-k}{2k-2}}$ is a distance threshold function for $G(n,r)$ to have a clique of size k.*

Proof. By Theorem 14.3.1, if $r = o(n^{\frac{-k}{2k-2}})$, then w.h.p. $G(n,r)$ has no connected subgraph on k vertices, and hence it has no clique of size k. This proves the first statement. We prove the second statement by adjusting the proof of Theorem 14.3.1, which is based on the second moment method. Assume $r = \omega(n^{\frac{-k}{2k-2}})$. Let $P_1, \ldots, P_{\binom{n}{k}}$ be an enumeration of all subsets of k points. Let X_i be equal to 1 if $DG[P_i]$ is a clique, and 0 otherwise. Let $X = \sum X_i$.

Partition the unit square into a set $\{s_1, \ldots, s_{1/r^2}\}$ of squares of side length r. Let S_t be the $2r \times 2r$ square which has s_t on its left bottom corner. If $DG[P_i]$ is a clique then P_i lies in S_t for some $t \in \{1, \ldots, 1/r^2\}$. Therefore,

$$\Pr[X_i = 1] \leq 4^k r^{2k-2}.$$

Now, partition the unit square into a set $\{s_1, \ldots, s_{2/r^2}\}$ of squares with diagonal length r. If all points of P_i fall in the square s_t, then $DG[P_i]$ is a clique. Thus,

$$\Pr[X_i = 1] \geq \frac{1}{2^{k-1}} r^{2k-2}.$$

Since $k \geq 2$ is a constant, we have

$$\begin{aligned} \mathrm{E}[X_i] &= \Theta(r^{2k-2}), \\ \mathrm{E}[X] &= \Theta(n^k r^{2k-2}). \end{aligned}$$

In view of Chebyshev's inequality we need to show that $\frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2}$ tends to 0 as $n \to \infty$. We bound $\mathrm{Var}(X)$ from above by Inequality (14.18). Consider the k-tuples P_i and P_j under the condition that $DG[P_i]$ is a clique. Let $|P_i \cap P_j| = w$, and let x be a point in $P_i \cap P_j$. Partition the unit square into squares of side length r. Let s_x be the square containing x. Let S_x be the $3r \times 3r$ square centered at s_x. In order for $DG[P_j]$ to be a clique, the remaining $k - w$ points of P_j must lie in S_x. Thus,

$$\mathrm{E}[X_i X_j] \leq c'_w r^{4k-2w-2},$$

for some constant $c'_w > 0$. By a similar argument as in the proof of Theorem 14.3.1, we can show that for some constants $c'', c''_w > 0$ the followings inequalities are valid:

$$\mathrm{Var}(X) \leq \sum_{w=1}^{k} c''_w n^{2k-w} r^{4k-2w-2},$$
$$\frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2} \leq \sum_{w=1}^{k} \frac{c''_w}{c''} \cdot \frac{1}{n^w r^{2w-2}}.$$

Since $r = \omega(n^{\frac{-k}{2k-2}})$, the last inequality tends to 0 as n goes to infinity. This completes the proof for the second statement. ■

As a direct consequence of Theorem 14.3.2, we have the following corollary.

Corollary 14.3.3 *n^{-1} is a threshold for $G(n,r)$ to have an edge, and $n^{-\frac{3}{4}}$ is a threshold for $G(n,r)$ to have a triangle.*

14.3.2 The threshold for $G(n,r)$ to be plane

In this section we investigate the threshold for a random geometric graph to be plane; this is a decreasing property. Recall that $G(n,r)$ is plane if no two of its edges cross. As a warm-up exercise we first prove a simple result which is based on the connectivity threshold for random geometric graphs, which is known to be $\sqrt{\ln n / n}$.

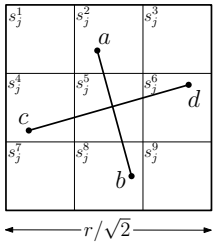


Figure 14.2: A square of diameter r which is partitioned into nine sub-squares.

Theorem 14.3.4 *If $r \geq \sqrt{\frac{c \ln n}{n}}$, with $c \geq 36$, then w.h.p. $G(n,r)$ is not plane.*

Proof. In order to prove that w.h.p. $G(n,r)$ is not plane, we show that w.h.p. it has a pair of crossing edges. Partition the unit square into squares each with diagonal length r. Then subdivide each such square into nine sub-squares as depicted in Figure 14.2. There are $\frac{18}{r^2}$ sub-squares, each of side length $\frac{r}{3\sqrt{2}}$. The probability that no point lies in a specific sub-square is $(1 - \frac{r^2}{18})^n$. Thus, the probability that there exists an empty sub-square is at most

$$\frac{18}{r^2}\left(1 - \frac{r^2}{18}\right)^n \leq n\left(1 - \frac{c \ln n}{18n}\right)^n \leq n^{1-c/18} \leq \frac{1}{n},$$

when $c \geq 36$. Therefore, with probability at least $1 - \frac{1}{n}$ all sub-squares contain points. By choosing four points a, b, c, and d as depicted in Figure 14.2, it is easy to see that the edges (a, b) and (c, d) cross. Thus, w.h.p. $G(n, r)$ has a pair of crossing edges, and hence w.h.p. it is not plane. ■

In fact, Theorem 14.3.4 ensures that w.h.p. there exists a pair of crossing edges in each of the squares. This implies that there are $\Omega\left(\frac{n}{\ln n}\right)$ disjoint pair of crossing edges, while for $G(n, r)$ to be not plane we need to show the existence of at least one pair of crossing edges. Thus, the value of r provided by the connectivity threshold seems rather weak. By a different approach, in the rest of this section we show that $n^{-\frac{2}{3}}$ is the correct threshold.

Lemma 14.3.5 *Let (a, b) and (c, d) be two crossing edges in $G(n, r)$, and let Q be the convex quadrilateral formed by a, b, c, and d. Then, two adjacent sides of Q are edges of $G(n, r)$.*

Proof. Refer to Figure 14.3. At least one of the angles of Q, say $\angle cad$, is bigger than or equal to $\pi/2$. It follows that in the triangle $\triangle cad$ the side cd is the longest, i.e., $|cd| \geq \max\{|ac|, |ad|\}$. Since $|cd| \leq r$, both $|ac|$ and $|ad|$ are at most r. Thus, ac and ad—which are adjacent—are edges of $G(n, r)$. ■ ■

Figure 14.3: (a) Illustration of Lemma 14.3.5. (b) Crossing edges (a, b) and (c, d) form an anchor.

In the proof of Lemma 14.3.5, a is connected to b, c, and d. So the distance between a to each of b, c, and d is at most r. Thus, we have the following corollary.

Corollary 14.3.6 *The endpoints of every two crossing edges in $G(n, r)$ are at distance at most $2r$ from each other. Moreover, there exists an endpoint which is within distance r from other endpoints.*

Based on the proof of Lemma 14.3.5, we define an *anchor* as a set $\{a, b, c, d\}$ of four points in $G(n, r)$ such that three of them form a triangle, say $\triangle cad$, and the fourth vertex, b, is connected to a by an edge which crosses cd; see Figure 14.3(b). We call a as the *crown* of the anchor. The crown is within distance r from the other three points. Note that bc and bd may or may not be edges of $G(n, r)$. In view of Lemma 14.3.5, two crossing edges in $G(n, r)$ form an anchor. Conversely, every anchor in $G(n, r)$ introduces a pair of crossing edges.

Observation 14.3.7 *$G(n,r)$ is plane if and only if it has no anchor.*

Theorem 14.3.8 *$n^{-\frac{2}{3}}$ is a threshold for $G(n,r)$ to be plane.*

Proof. In order to show that $G(n,r)$ is plane, by Observation 14.3.7, it is enough to show that it has no anchors. Every anchor has four points and it is connected. By Theorem 14.3.1, if $r = o(n^{-\frac{2}{3}})$, then w.h.p. $G(n,r)$ has no connected subgraph on 4 points, and hence it has no anchors. This proves the first statement.

We prove the second statement by adjusting the proof of Theorem 14.3.1 for $k = 4$. Assume $r = \omega(n^{-\frac{2}{3}})$. Let $P_1, \ldots, P_{\binom{n}{4}}$ be an enumeration of all subsets of 4 points. Let X_i be equal to 1 if $DG[P_i]$ contains an anchor, and 0 otherwise. Let $X = \sum X_i$. In view of Chebyshev's inequality we need to show that $\frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2}$ tends to 0 as $n \to \infty$.

Partition the unit square into a set $\{s_1, \ldots, s_{2/r^2}\}$ of squares with diagonal length r. Then, subdivide each square s_j, into nine subsquares $s_j^1, \ldots, s_j^9$ as depicted in Figure 14.2. If each of $s_j^1, s_j^3, s_j^7, s_j^9$ or each of $s_j^2, s_j^4, s_j^6, s_j^8$ contains a point of P_i, then $DG[P_i]$ is a convex clique of size four and hence it contains an anchor. Thus,

$$\Pr[X_i = 1] \geq \frac{r^6}{2^3} \cdot \frac{2}{9^4}.$$

This implies that $\mathrm{E}[X_i] = \Omega(r^6)$, and hence $\mathrm{E}[X] = \Omega(n^4 r^6)$. Therefore,

$$\mathrm{E}[X]^2 \geq c'' n^8 r^{12},$$

for some constant $c'' > 0$. By a similar argument as in the proof of Theorem 14.3.1 we bound the variance of X from above by

$$\mathrm{Var}(X) \leq c_1'' n^7 r^{12} + c_2'' n^6 r^{10} + c_3'' n^5 r^8 + c_4'' n^4 r^6.$$

Since $r = \omega(n^{-\frac{2}{3}})$, $\frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2}$ tends to 0 as $n \to \infty$. That is, w.h.p. $G(n,r)$ has an anchor. By Observation 14.3.7, w.h.p. $G(n,r)$ is not plane. ∎

As a direct consequence of the proof of Theorem 14.3.8, we have the following:

Corollary 14.3.9 *With high probability if a random geometric graph is not plane, then it has a clique of size four.*

Note that every anchor introduces a crossing and each crossing introduces an anchor. Since, every anchor is a connected graph and has four points, by (14.13) we have the following corollary.

Corollary 14.3.10 *The expected number of crossings in $G(n,r)$ is $\Theta(n^4 r^6)$.*

14.3.3 *The threshold for $G(n,r)$ to be planar*

In this section we investigate the threshold for the planarity of a random geometric graph; this is a decreasing property. By Kuratowski's theorem, a finite graph is planar if and only if it does not contain a subgraph that is a subdivision of K_5 or of $K_{3,3}$. Note that any plane random geometric graph is planar too; observe that the reverse statement may not be true. Thus, the threshold for planarity seems to be larger than the threshold of being plane. By a similar argument as in the proof of Theorem 14.3.4 we can show that if $r \geq \sqrt{c \ln n/n}$, then w.h.p. each square with diagonal length r contains K_5, and hence $G(n,r)$ is not planar.

Theorem 14.3.11 *$n^{-\frac{5}{8}}$ is a threshold for $G(n,r)$ to be planar.*

Proof. By Theorem 14.3.2, if $r = \omega(n^{-\frac{5}{8}})$, then w.h.p. $G(n,r)$ has a clique of size 5. Thus, w.h.p. $G(n,r)$ contains K_5 and hence it is not planar. This proves the second statement of the theorem.

If $r = o(n^{-\frac{5}{8}})$, then by Theorem 14.3.1, w.h.p. $G(n,r)$ has no connected subgraph on 5 points, and hence it has no K_5. Similarly, if $r = o(n^{-\frac{3}{5}})$, then w.h.p. $G(n,r)$ has no connected subgraph on 6 points, and hence it has no $K_{3,3}$. Since $n^{-\frac{5}{8}} < n^{-\frac{3}{5}}$, it follows that if $r = o(n^{-\frac{5}{8}})$, then w.h.p. $G(n,r)$ has neither K_5 nor $K_{3,3}$ as a subgraph.

Note that, in order to prove that $G(n,r)$ is planar, we have to show that it does not contain any subdivision of either K_5 or $K_{3,3}$. Any subdivision of either K_5 or $K_{3,3}$ contains a connected subgraph on $k \geq 5$ vertices. Since $n^{-5/8} < n^{-k/(2k-2)}$ for all $k \geq 5$, in view of Theorem 14.3.1, we conclude that if $r = o(n^{-\frac{5}{8}})$, then w.h.p. $G(n,r)$ has no subdivision of K_5 and $K_{3,3}$, and hence $G(n,r)$ is planar. This proves the first statement of the theorem. ■

As a direct consequence of the proof of Theorem 14.3.11, we have the following:

Corollary 14.3.12 *With high probability if a random geometric graph does not contain a clique of size five, then it is planar.*

14.4 *Lovász Local Lemma - In Progress*

We will outline an algorithmic proof of the following.

Let $E_1, \ldots, E_n$ be a set of n events where the following holds.

1. $Pr(E_i) \leq p$ for $1 \leq i \leq n$.
2. Every event is mutually independent of all but at most d other events.
3. $4dp \leq 1$.

Lovász local lemma states that $Pr(\cap_{i=1}^{n} \bar{E}_i) > 0$.

14.4.1 *Bibliographic Notes*

Random graphs were first defined and formally studied by Gilbert in [65] and Erdös and Rényi [54]. It seems that the concept of a random geometric graph was first formally suggested by Gilbert in [66] and for that reason is also known as Gilbert's disk model. These classes of graphs are known to have numerous applications as a model for studying communication primitives (broadcasting, routing, etc.) and topology control (connectivity, coverage, etc.) in idealized wireless sensor networks as well as extensive utility in theoretical computer science and many fields of the mathematical sciences.

An instance of Erdös-Rényi graph [54] is obtained by taking n vertices and connecting any two with probability p, independently of all other pairs; the graph derived by this scheme is denoted by $G_{n,p}$. In $G_{n,p}$ the threshold is expressed by the edge existence probability p, while in $G(n,r)$ the threshold is expressed in terms of r. In both random graphs and random geometric graphs, property thresholds are of great interest [18, 27, 62, 69, 114]. Note that edge crossing configurations in $G(n,r)$ have a geometric nature, and as such, have no analogues in the context of the Erdös-Rényi model for random graphs. However, planarity, and having a clique of specific size are of interest in both $G_{n,p}$ and $G(n,r)$.

Bollobás and Thomason [19] showed that any monotone property in random graphs has a threshold function. See also a result of Friedgut and Kalai [62], and a result of Bourgain and Kalai [26]. In the Erdös-Rényi random graph $G_{n,p}$, the connectivity threshold is $p = \log n/n$ and the threshold for having a giant component is $p = 1/n$; see [5]. The planarity threshold for $G_{n,p}$ is $p = 1/n$; see [18, 133].

A general reference on random geometric graphs is [126]. There is extensive literature on various aspects of random geometric graphs of which we mention the related work on coverage by [76, 84] and a review on percolation, connectivity, coverage and colouring by [13]. As in random graphs, any monotone property in geometric random graphs has a threshold function [27, 69, 103, 114]. Exercise **??** is derived from the notes of Chi-Lau.

Random geometric graphs have a connectivity threshold of $\sqrt{\ln n/n}$; see [73, 123, 124]. Gupta and Kumar [73] provided a connectivity threshold for points that are uniformly distributed in a disk. By a result of Penrose [125], in $G(n,r)$, any threshold function for having no isolated vertex (a vertex of degree zero) is also a connectivity threshold function. Panchapakesan and Manjunath [123] showed

that $\sqrt{\ln n/n}$ is a threshold for being an isolated vertex in $G(n,r)$. This implies that $\sqrt{\ln n/n}$ is a connectivity threshold for $G(n,r)$. For $k \geq 2$, the details on the k-connectivity threshold in random geometric graphs can be found in [125, 126]. Connectivity of random geometric graphs for points on a line is studied by Godehardt and Jaworski [68].

The book by Alon and Spencer is an excellent resource on the second moment method [5].

14.5 Exercises

14.1 *Let $X = \sum_{i=1}^{n} X_i$ be the sum of n random variables $X_1, \ldots, X_n$. Show that $V[X] = \sum_{i=1}^{n} V[X_i] + \sum_{i \neq j} Cov[X_i, X_j]$. Note that each pair (X_i, X_j), $i \neq j$, shows up twice in the sum, as they are ordered pairs.*

14.2 *For a non-negative random variable X, if $E[X] \to \infty$, then is it true that $X > 0$ almost always?*

Hint: Consider the following random variable.

$$X = \begin{cases} n^2, \text{ with probability } 1/n \\ 0, \text{ otherwise} \end{cases}$$

14.3 *As a corollary to Theorem 14.1.1, show that if $V[X] = o(E[X]^2)$, then $X \sim E[X]$.*

14.4 *Consider the random graph $G(n,p)$ where each edge is chosen independently with probability p. Show that $n^{-2/3}$ is a threshold function for containing K_4 in $G(n,p)$.*

14.5 *2MM:ex-dia2 In this exercise we will establish a sharp threshold of $p = \sqrt{\frac{2\ln n}{n}}$ for a random graph $G(n,p)$ to have a diameter ≤ 2. Observe that the property that G has a diameter ≤ 2 is an increasing property. A pair of vertices (i,j) are said to be a* bad pair *if the shortest path distance between i and j in G is > 2. Obviously, G has a diameter ≤ 2, if it has no bad pair. Define X_{ij} to be an indicator variable that states if (i,j) is a bad pair. Let $X = \sum_{i,j} X_{ij}$ be the random variable indicating the number of bad pairs. Show the following.*

1. $E[X_{ij}] = (1-p)(1-p^2)^{n-2}$.
2. $E[X] \approx \frac{n^2}{2} e^{-p^2 n}$.
3. *For $p = \sqrt{\frac{c\ln n}{n}}$, $E[X] \approx \frac{1}{2} n^{2-c}$.*

4. *If $c > 2$, $E[X] \to 0$, i.e. there are no bad pairs w.h.p and diameter is at most 2.*

5. *If $c < 2$, $E[X] \to \infty$.*

6. *For $c < 2$, we will employ second moment method to show that $X \geq 1$ almost surely, and thus G will have diameter ≥ 3. Consider*

$$E[X^2] = E[(\sum_{i<j} X_{ij})^2] = E[\sum_{i<j,k<l} X_{ij}X_{kl}].$$

We split this sum into three parts based on how many vertices in i, j, k, l are distinct. Answer the following.

   (a) *Show that if i, j, k, and l are all distinct, the variables X_{ij} and X_{kl} are independent. Moreover, $E[\sum_{i<j,k<l} X_{ij}X_{kl}] = \sum_{i<j,k<l} E[X_{ij}]E[X_{kl}] \leq \sum_{i<j} E[X_{ij}] \sum_{k<l} E[X_{kl}] \leq E[X]^2$.*

   (b) *Show that if there are only two distinct vertices in i, j, k, l, $E[\sum_{i<j,k<l} X_{ij}X_{kl}] = E[\sum_{i<j} X_{ij}^2] = E[X]$.*

   (c) *Suppose there are three distinct vertices in i, j, k, l. Let (i, j) and (i, k) be the two bad pairs. For any other vertex u, either it is not adjacent to i or it is not adjacent to both j and k. Show that this event happens with probability $(1-p) + p(1-p)^2 \approx 1 - 2p^2$. In this case, show that $E[X_{ij}X_{kl}] \approx (1-p)^2(1-2p^2)^{n-3} \approx e^{-2p^2 n}$. Conclude that $E[\sum_{i<j,k<l} X_{ij}X_{kl}] \leq \frac{n^3}{2} e^{-2p^2 n}$, by showing that there are at most $3\binom{n}{3}$ such triples. Show that for $p = \sqrt{\frac{c \ln n}{n}}$, this sum is $\frac{1}{2} n^{3-2c} = o(n^{4-2c}) = o(E[X]^2)$.*

   (d) *Conclude that $E[X^2] \leq E[X]^2 + E[X] + o(E[X]^2) = (1 + o(1))E[X]^2$ and $V[X] = o(E[X]^2)$.*

   (e) *Using Theorem 14.1.1, show that $X > 0$ almost surely.*

   (f) *Put everything together and show that for $c < 2$ and $p = \sqrt{\frac{c \ln n}{n}}$, almost surely $G(n, p)$ has diameter ≥ 3.*

14.6 *(Research Question:) Extend Theorem 14.3.1 for connected subgraphs of k vertices where k is not necessarily a constant, and for connected subgraphs of k vertices which have diameter δ.*

14.7 *Let $E_1, \ldots, E_n$ be a set of n events where the following holds.*

1. *$Pr(E_i) \leq p$ for $1 \leq i \leq n$.*

2. *Every event is mutually independent of all but at most d other events.*

3. *$4dp \leq 1$.*

Lovász local lemma states that $Pr(\cap_{i=1}^{n} \bar{E}_i) > 0$.

Let $G = (V, E)$ *be a simple undirected graph with k pairs of vertices* $(s_1, t_1), \ldots, (s_k, t_k)$. *For* $1 \leq i \leq k$, *let* $\mathcal{P}_i$ *be a collection of L paths connecting* s_i *and* t_i. *Suppose each path in* $\mathcal{P}_i$ *does not share edges with with more than C paths in* $\mathcal{P}_j$, $j \neq i$, *and* $8kC/L \leq 1$. *Using LLL show that there is a way to choose* $P_i \in \mathcal{P}_i$ *such that* $\{P_1, \ldots, P_k\}$ *are edge disjoint. Can we find such a set* $\{P_1, \ldots, P_k\}$ *of k paths in polynomial time?*

15
Clustering

We will focus on

1. Clustering Problem
2. Algorithms for Clustering
3. k-Means
4. k-Means++ Algorithm

Keywords: Clustering, k-Means, k-Means++

15.1 Introduction

The generic clustering problem is to partition a set of objects into subsets so that objects within a subset are similar to each other according to some measure as compared to the objects from other subsets. Each subset is called a *cluster*. Object similarity is measured according to some distance function that depends on the application. In this chapter, we will discuss some of the classical setups and present some methods for clustering objects into k clusters for some positive integer k. As a warm-up, consider the following setup.

The input to a clustering problem consists of a set $X = \{x_1, \ldots, x_n\}$ of n objects and an integer $k > 0$. For every pair $x_i, x_j \in X$, we have the distance $d(x_i, x_j) \geq 0$ such that $d(x_i, x_i) = 0$ and $d(x_i, x_j) = d(x_j, x_i)$. The objective is to partition objects in X into k non-empty clusters $C_1, \ldots, C_k$, such that the *gap* between the clusters is as large as possible. Distance between two groups $C_i, C_j \subset X$ is defined as the smallest distance between a pair of points (a, b), where $a \in C_i$ and $b \in C_j$. I.e., $\text{gap}(C_i, C_j) = \min_{a \in C_i, b \in C_j} d(a, b)$. For a given clustering $C_1, \ldots, C_k$ of X, $\text{gap}(X) = \min_{1 \leq i < j \leq k} \text{gap}(C_i, C_j)$. Here is a simple method based on minimum spanning trees to compute the clustering

of X into k non-empty clusters that maximizes the gap. See Figure 15.1 for an illustration.

Clustering using MST

1. Define a complete graph $G = (V = X, E)$, where each edge $e = (x_i, x_j)$ has a weight $d(x_i, x_j)$.
2. Construct a minimum spanning tree T of G.
3. Delete $k-1$ most expensive edges from T.
4. Output the resulting k-connected components $C_1, \ldots, C_k$.

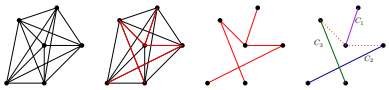


Figure 15.1: Partitioning in three clusters. Dotted edges are the two most expensive edges of MST.

Lemma 15.1.1 *The components $C_1, \ldots, C_k$ returned by the above algorithm constitutes a k-clustering of X that maximizes the gap.*

Proof. Left as an exercise. ■

15.2 *k-Means Clustering*

The k-Means clustering problem is as follows. The input consists of a set $X = \{x_1, \ldots, x_n\}$ of n-points in Euclidean d-dimensional space $\mathbb{R}^d$ and an integer $0 < k \leq n$. The objective is to partition X into k non-empty clusters $C_1, \ldots, C_k$. Points within a cluster should be *close* to each other compared to points outside the cluster. The closeness is defined with respect to a potential function and cluster centers as follows.

Let $C_1, \ldots, C_k$ be a k-clustering of X with centers $\mathcal{C} = \{c_1, \ldots, c_k\}$, where $c_i \in \mathbb{R}^d$. Define the *potential function*

$$\Phi_{\mathcal{C}}(X) = \sum_{x \in X} \min_{c \in \mathcal{C}} d(x, c)^2 = \sum_{x \in X} \min_{c \in \mathcal{C}} ||x - c||^2$$

Note that $\Phi_{\mathcal{C}}(X)$ is the sum of the squared distance between each point x in X to its nearest center in $\mathcal{C}$.

Given X and k, in the k-Means clustering problem, we need to find k-centers $\mathcal{C}$ such that the corresponding clustering $C_1, \ldots, C_k$ minimizes $\Phi_{\mathcal{C}}(X)$ among all possible clusterings. As such the decision problem where we want to find 2 clusters that has the potential smaller than certain value is known to be **NP**-Hard [43]. A popular clustering method that is widely used is Llyod's heuristic [106]. The main steps are as follows.

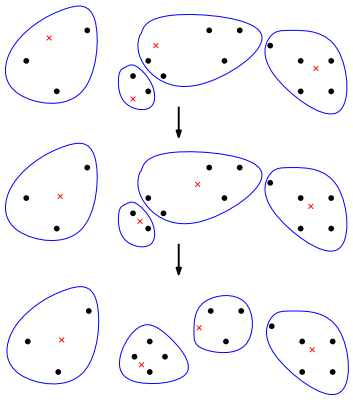


Figure 15.2: An illustration of a Phase of Llyod's Heuristic

k-Means Llyod's Heuristic [106]:

1. Select Initial *Centers*: Arbitrary choose k-centers $\{c_1, \ldots, c_k\}$, where $c_i \in \Re^d$, and initialize $\mathcal{C} = \{c_1, \ldots, c_k\}$.
2. Partition X: Compute sets $C_1, \ldots, C_k$ with respect to centers in $\mathcal{C}$. Point $x \in X$ is assigned to the cluster C_i if x's nearest center in $\mathcal{C}$ is c_i.
3. Recompute Centers: For each $i \in \{1, \ldots, k\}$, set c_i (the new cluster center) to be the center of the mass of points in C_i
4. Repeat Steps 2 & 3 till $\mathcal{C}$ no longer changes.

Together, Steps 2 and 3 form a *phase* of Llyod's heuristic. Let us first see why the above heuristic terminates. For this we will show in each phase (i.e., the execution of Steps 2 & 3) the value of the potential function decreases. Since the potential cannot decrease forever, the heuristic eventually terminates. First, we show that for a given set S of points, its centre of mass minimizes the potential function. The *centre of mass* of a point set S is the point whose each coordinate is the average of the respective coordinates of points in S.

Lemma 15.2.1 *Consider a set of points S. Let m^* denote the center of mass of S. Let z be an arbitrary point. Define $\Delta(S, z) = \sum_{x \in S} d(x, z)^2$. Then $\Delta(S, z) = \Delta(S, m^*) + |S| d(m^*, z)^2$.*

Proof. Assume we are in 2-dimensions. Let $z = (z_x, z_y)$ and $S = \{p_1, \ldots, p_n\}$, where $p_i = (x_i, y_i)$. Note that $m^* = \left(\frac{\sum_i x_i}{n}, \frac{\sum_i y_i}{n} \right)$.

Now,

$$\Delta(S, z) = \sum_{p \in S} d(p, z)^2 = \sum_{p=(x_i, y_i) \in S} \left((x_i - z_x)^2 + (y_i - z_y)^2 \right),$$

and

$$\Delta(S, m^*) = \sum_{p \in S} d(p, m^*)^2 = \sum_i \left(x_i - \frac{\sum x_i}{n} \right)^2 + \sum_i \left(y_i - \frac{\sum y_i}{n} \right)^2.$$

Thus,

$$\begin{aligned}
&\Delta(S, z) - \Delta(S, m^*) \\
&\quad = n z_x^2 + n z_y^2 - 2 z_x \textstyle\sum x_i - 2 z_y \sum y_i + n \left(\frac{\sum x_i}{n} \right)^2 + n \left(\frac{\sum y_i}{n} \right)^2 \\
&\quad = n \left[z_x^2 + z_y^2 - 2 z_x \tfrac{\sum x_i}{n} - 2 z_y \tfrac{\sum y_i}{n} + \left(\tfrac{\sum x_i}{n} \right)^2 + \left(\tfrac{\sum y_i}{n} \right)^2 \right] \\
&\quad = |S| d(m^*, z)^2.
\end{aligned}$$

■

Corollary 15.2.2 *If S is a single cluster with initial center z, then moving the cluster center to m^* reduces the potential as $\Delta(S,z) - \Delta(S,m^*) = |S|d(m^*,z)^2 \geq 0$.*

Thus, for each cluster, it is best to choose the cluster center to be its center of mass (see Step 3). Hence, each phase of Llyod's heuristic reduces the potential. Though, it seems that this heuristic produces a good clustering, but there are instances where the worst-case competitive ratio is unbounded.

Lemma 15.2.3 *Competitive ratio of Llyod's heuristic is unbounded, i.e., $\frac{\Phi_{\mathcal{C}}(X)}{\Phi_{\mathcal{C}^*}(X)} \to \infty$, where $\mathcal{C}^*$ is an optimal clustering of X.*

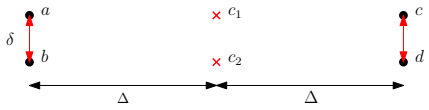


Figure 15.3: Llyod's heuristic has unbounded competitive ratio

Proof. Consider the example in Figure 15.3.

The input consists of four points $X = \{a, b, c, d\}$ that need to be partitioned into two clusters. These points are located at the intersections of two horizontal and two vertical lines. The horizontal lines are separated by a distance $\delta \to 0$, whereas the vertical lines are separated by a large distance 2Δ. The optimal clustering $\mathcal{C}^*$ partitions X in two clusters (a,b) and (c,d). It is clear that the potential $\Phi_{\mathcal{C}^*}(X) \to 0$.

Suppose that the initial cluster centers that are chosen in Llyod's heuristic (in Step 1) are c_1 and c_2, where c_1 (respectively, c_2) is the midpoint of a and c (respectively, b and d). With this choice of $\mathcal{C} = \{c_1, c_2\}$, the clustering obtained after the execution of Step 2 is $\{(a,c),(b,d)\}$. Moreover, Step 3 will not change the choice of c_1 and c_2. Thus, the heuristic will terminate and return $\mathcal{C}$ as the resulting clustering. Observe that $\Phi_{\mathcal{C}}(X)$ is large. Thus, $\frac{\Phi_{\mathcal{C}}(X)}{\Phi_{\mathcal{C}^*}(X)} \to \infty$. ■

In the next section, we will show that initial cluster centres can be chosen to ensure that the worst-case competitive ratio is bounded.

15.3 k-Means++ Clustering

Now we present the randomized algorithm of Arthur and Vassilvitskii [10] for choosing the cluster centers so that the competitive ratio is bounded in expectation. Let $D(x)$ be the shortest distance from x to the nearest center among the current set of centers.

k-Means++ Algorithm [10]:

Step 1: Choose an initial cluster center c_1 uniformly at random from X. Set $\mathcal{C} := \{c_1\}$.

Step 2: (**Randomization Step**) Choose the next center c_t by selecting a point $x \in X$ with probability $\frac{D(x)^2}{\sum_{y \in X} D(y)^2}$. Set $\mathcal{C} := \{c_t\} \cup \mathcal{C}$

Step 3: Repeat Step 2 till k centers are chosen.

Step 4: Execute Steps 2-4 of Llyod's Heuristic, where $\mathcal{C} = \{c_1, \ldots, c_k\}$ are the initial centers.

Step 5: Return $\mathcal{C}$.

In this section, we will show that the randomized k-Means++ algorithm is $8(\ln k + 2)$-competitive. Let the k-centers returned by k-Means++ algorithm be $\mathcal{C}$, and let $\mathcal{C}^*$ be an optimal clustering. We will show that $E[\Phi_{\mathcal{C}}(X)] \leq 8(\ln k + 2))\Phi_{\mathcal{C}^*}(X)$. We first make a few observations.

1. In the Randomization Step, the points of X that are farther from the currently chosen centers have a higher chance of being selected.

2. Note that the claim about the competitive ratio holds for the clustering obtained after Step 3. Step 4 may further improve, as we know that in each phase of Llyod's heuristic the potential decreases.

3. Proof is non-trivial. Consider clusters of an optimal solution $\mathcal{C}^*$. Note that in Step 1, we choose the first cluster center uniformly at random. In Step 2, we choose clusters with respect to probabilities with respect to distances. The proof proceeds along the following lines.
   - The algorithm is 2-competitive with respect to the points in the optimal cluster, say A, from where the first center c_1 is chosen in Step 1.
   - The algorithm is 8-competitive in all those clusters of optimal from which the algorithm chooses a center in Step 2.
   - The algorithm is $8(\ln k + 2)$-competitive in all those clusters of optimal from which the algorithm doesn't choose a center.

Before, we discuss the proof of [10], we introduce some useful notation. Let $d(x, c) = ||x - c||$ denote the Euclidean distance between points x and c and let $D(x)$ be the shortest distance from x to the nearest center in $\mathcal{C}$ (or $\mathcal{C}^*$). Recall that $\Phi_{\mathcal{C}}(X)$ refers to the potential

with respect to the point set X, where $\mathcal{C}$ are the chosen centers, i.e., $\Phi_{\mathcal{C}}(X) = \sum_{x \in X} D(x)^2$. Analogously, we define $\Phi_{\mathcal{C}^*}(X)$ as the potential of X with respect to an optimal clustering $\mathcal{C}^*$. For a subset $A \subseteq X$, define $\Phi_{\mathcal{C}}(A) = \sum_{x \in A} D(x)^2$.

Let us analyze the expected value of the potential after we choose the first center in the k-Means++ algorithm.

Lemma 15.3.1 *Let A be an arbitrary cluster in optimal clustering $\mathcal{C}^*$. Let $\mathcal{C} = \{c_1\}$ be the clustering with exactly one center c_1 that is chosen from A uniformly at random in Step 1. Then, $E[\Phi_{\mathcal{C}}(A)] = 2\Phi_{\mathcal{C}^*}(A)$.*

Proof. By definition of expected value, $E[\Phi_{\mathcal{C}}(A)] = \sum_{a_0 \in A} \frac{1}{|A|} \sum_{a \in A} ||a - a_0||^2$, as the probability of choosing an element $a_0 \in A$ uniformly at random is $\frac{1}{|A|}$, and once we choose a_0, the potential is $\sum_{a \in A} ||a - a_0||^2$.

From Corollary 15.2.2, in $\mathcal{C}^*$, cluster center of A will be its center of mass, say m^*. Using Lemma 15.2.1, we have

$$\begin{aligned} E[\Phi_{\mathcal{C}}(A)] &= \frac{1}{|A|} \sum_{a_0 \in A} \left(\sum_{a \in A} ||a - m^*||^2 + |A| ||a_0 - m^*||^2 \right) \\ &= \frac{1}{|A|} \sum_{a_0 \in A} \sum_{a \in A} ||a - m^*||^2 + \frac{1}{|A|} \sum_{a_0 \in A} |A| ||a_0 - m^*||^2 \\ &= 2 \sum_{a \in A} ||a - m^*||^2 \\ &= 2\Phi_{\mathcal{C}^*}(A) \end{aligned}$$

■

Assume that we are at some intermediate stage in the algorithm, and the current clustering is $\mathcal{C}$. Let A be a cluster in an optimal clustering $\mathcal{C}^*$. Suppose that in the next iteration (i.e., in the execution of Step 2) of k-Means++ algorithm, let us say we choose the center c_t from the cluster A, and add to $\mathcal{C}$. Then, the following lemma states the expected value of the potential of elements in A in the amalgamated clustering $\mathcal{C} = \mathcal{C} \cup \{c_t\}$.

Lemma 15.3.2 $E[\Phi_{\mathcal{C}}(A)] \leq 8\Phi_{\mathcal{C}^*}(A)$.

Proof. By triangle inequality we have for all a and a_0,
$D(a_0) \leq D(a) + ||a - a_0||$.
Note that for reals x and y, $\frac{1}{2}(x + y)^2 \leq x^2 + y^2$.
Thus, we have
$\frac{1}{2}(D(a_0))^2 \leq \frac{1}{2}(D(a_0) + ||a - a_0||)^2 \leq D(a)^2 + ||a - a_0||^2$.
Equivalently, $D(a_0)^2 \leq 2D(a)^2 + 2||a - a_0||^2$.
Summing over all elements of A, we have
$\sum_{a \in A} D(a_0)^2 \leq \sum_{a \in A} \left(2D(a)^2 + 2||a - a_0||^2\right)$.

Or, $D(a_0)^2 \leq \frac{2}{|A|} \sum_{a \in A} D(a)^2 + \frac{2}{|A|} \sum_{a \in A} (a - a_0)^2$.

Probability of choosing $a_0 \in A$ as a center is $\frac{D(a_0)^2}{\sum_{a \in A} D(a)^2}$.

Then, $E[\Phi_{\mathcal{C}}(A)] = \sum_{a_0 \in A} \frac{D(a_0)^2}{\sum_{a \in A} D(a)^2} \sum_{a \in A} \min(D(a), (a - a_0))^2$.

Substituting the expression for $D(a_0)^2$ in $E[\Phi_{\mathcal{C}}(A)]$ we obtain,

$$E[\Phi_{\mathcal{C}}(A)] \leq \frac{2}{|A|} \sum_{a_0 \in A} \frac{\sum_{a \in A} D(a)^2}{\sum_{a \in A} D(a)^2} \sum_{a \in A} \min(D(a), (a - a_0))^2 + \frac{2}{|A|} \sum_{a_0 \in A} \frac{\sum_{a \in A} (a - a_0)^2}{\sum_{a \in A} D(a)^2} \sum_{a \in A} \min(D(a), (a - a_0))^2.$$

Substitute for $\min(D(a), (a - a_0))^2 \leq (a - a_0)^2$ in 1st part and $\min(D(a), (a - a_0))^2 \leq D(a)^2$ in 2nd part and we obtain

$E[\Phi_{\mathcal{C}}(A)] \leq \frac{4}{|A|} \sum_{a_0 \in A} \sum_{a \in A} (a - a_0)^2 = 8\Phi_{\mathcal{C}^*}(A)$ (By Lemma 15.3.1). ■

So far, we have analyzed the cases where the algorithm chooses centers from optimal clusters and have shown that the competitive ratio is within a factor of 8. Thus, if the selection of centers by the algorithm hits all the clusters of an optimal solution, our algorithm's competitive ratio will be bounded by a constant. But, the algorithm may fail to pick a center from an optimal cluster. Next, we finish the analysis of the general case. We follow the notation used in [10] and the analysis of [42].

Without loss of generality, let $\mathcal{C} = \{c_1, c_2, \ldots, c_t, \ldots, c_k\}$ be the cluster centers returned by the k-Means++ algorithm. It incurs a total cost (potential) of $\Phi_{\mathcal{C}}(X)$. We will spread this cost over the k iterations. We will say that at the end of iteration t, $1 \leq t \leq k$, $\mathcal{C}_t$ be the cluster centers chosen by the algorithm. Note that $\mathcal{C} = \mathcal{C}_k$. Let us fix an optimal clustering $\mathcal{C}^*$. Let H_t be the set of clusters of $\mathcal{C}^*$ that are *hit* by centers in $\mathcal{C}_t$, and let $U_t = [k] \setminus H_t$ be the set of clusters of $\mathcal{C}^*$ that aren't hit (or covered) at the end of iteration $t \in [k]$. We define $W_t = t - |H_t|$ as the number of *wasted* iterations, i.e., the iterations where the algorithm fails to hit a new cluster of $\mathcal{C}^*$. We will show that the cost incurred by the algorithm at the end of iteration $t \in [k]$ is

$$\Phi_{\mathcal{C}_t}(H_t) + \frac{W_t \Phi_{\mathcal{C}_t}(U_t)}{|U_t|}. \tag{15.1}$$

At the start, no clusters of optimal are hit and hence $H_0 = \emptyset$ and $W_0 = 0$ as no iterations are wasted. In that case, Equation 15.1

evaluates to 0, which signifies that the cost is 0, to begin with. When $t = k$, the Equation 15.1 evaluates to $\Phi_{\mathcal{C}}(H_t) + \Phi_{\mathcal{C}}(U_t) = \Phi_{\mathcal{C}}(X)$, as $W_k = |U_k|$. Thus, for the two extreme values of t, Equation 15.1 captures the correct value of the potential. Furthermore, the first term $\Phi_{\mathcal{C}_t}(H_t)$ captures the cost of clusters that are hit by the centers chosen in the algorithm for any $t \in [k]$, and by Lemma 15.3.2, the expected cost $E[\Phi_{\mathcal{C}_t}(H_t)] \leq 8\Phi_{\mathcal{C}^*}(X)$. Thus, our task is to evaluate the second term in Equation 15.1, which we will denote by $\Psi_t = \frac{W_t \Phi_{\mathcal{C}_t}(U_t)}{|U_t|}$, for any $t \in [k]$.

Claim 15.3.3 *For any $t \in [k-1]$, $E[\Psi_{t+1} - \Psi_t] \leq \frac{\Phi_{\mathcal{C}_t}(H_t)}{k-t}$.*

First we show that using this claim, we can establish the competitive ratio of the k-Means++ algorithm.

Theorem 15.3.4 *Let the k-centers returned by k-Means++ algorithm for a point set X be $\mathcal{C}$. Let $\mathcal{C}^*$ be an optimal clustering of X. Then, $E[\Phi_{\mathcal{C}}(X)] \leq 8(2 + \ln k)\Phi_{\mathcal{C}^*}(X)$, i.e., the algorithm is $8(2 + \ln k)$-competitive.*

Proof. We know that $\Phi_{\mathcal{C}}(X) = \Phi_{\mathcal{C}}(H_k) + \Phi_{\mathcal{C}}(U_k)$ as some of the clusters of $\mathcal{C}^*$ are hit and some aren't hit by the centers $\mathcal{C}$ chosen in the k-Means++ algorithm. From Lemma 15.3.2, we know that $E[\Phi_{\mathcal{C}}(H_k)] \leq 8\Phi_{\mathcal{C}^*}(X)$.

Observe that $\Phi_{\mathcal{C}}(U_k) = \Psi_k = \sum_{t=0}^{k-1} (\Psi_{t+1} - \Psi_t)$. Therefore, by linearity of expectation and Claim 15.3.3, we have

$$\begin{aligned}
E[\Phi_{\mathcal{C}}(X)] &= E[\Phi_{\mathcal{C}}(H_k) + \Phi_{\mathcal{C}}(U_k)] \\
&= E[\Phi_{\mathcal{C}}(H_k)] + E[\Phi_{\mathcal{C}}(U_k)] \\
&\leq 8\Phi_{\mathcal{C}^*}(X) + \sum_{t=0}^{k-1} E[\Psi_{t+1} - \Psi_t] \\
&\leq 8\Phi_{\mathcal{C}^*}(X) + \sum_{t=0}^{k-1} \frac{\Phi_{\mathcal{C}_t}(H_t)}{k-t} \\
&\leq 8\Phi_{\mathcal{C}^*}(X) \left(1 + 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k}\right) \\
&\leq 8(2 + \ln k)\Phi_{\mathcal{C}^*}(X)
\end{aligned}$$

■

Finally, let us prove Claim 15.3.3.

Proof. Assume that the $(t+1)$-st center c_{t+1} in the k-Means++ algorithm is chosen from the cluster C^*_α of optimal, i.e. $c_{t+1} \in C^*_\alpha$. There are two cases:

Case 1: c_{t+1} hits a previously chosen cluster, i.e., $\alpha \in H_t$.

In this case, $H_{t+1} = H_t$ and $U_{t+1} = U_t$ as no new clusters are covered. Moreover, this iteration increases the number of wasted iterations by 1, i.e. $W_{t+1} = W_t + 1$. Thus,

$$\begin{aligned}
\Psi_{t+1} - \Psi_t &= \frac{W_{t+1}\Phi_{\mathcal{C}_{t+1}}(U_{t+1})}{|U_{t+1}|} - \frac{W_t\Phi_{\mathcal{C}_t}(U_t)}{|U_t|} && (15.2)\\
&\leq \frac{(W_t+1)\Phi_{\mathcal{C}_t}(U_t)}{|U_t|} - \frac{W_t\Phi_{\mathcal{C}_t}(U_t)}{|U_t|} && (15.3)\\
&= \frac{\Phi_{\mathcal{C}_t}(U_t)}{|U_t|} && (15.4)
\end{aligned}$$

The reason that the 2nd inequality holds is that by adding more centers, potential cannot increase. Thus, $\Phi_{\mathcal{C}_{t+1}}(U_t) \leq \Phi_{\mathcal{C}_t}(U_t)$.

Case 2: c_{t+1} covers a new cluster, i.e., $\alpha \in U_t$. In this case, $H_{t+1} = H_t \cup \{\alpha\}$ and $U_{t+1} = U_t \setminus \{\alpha\}$ as a new cluster C^*_α of optimal is covered. Moreover, this iteration doesn't increase the number of wasted iterations, i.e. $W_{t+1} = W_t$. We have

$$\begin{aligned}
\Psi_{t+1} &= \frac{W_{t+1}\Phi_{\mathcal{C}_{t+1}}(U_{t+1})}{|U_{t+1}|}\\
&= \frac{W_t\Phi_{\mathcal{C}_{t+1}}(U_t \setminus C^*_\alpha)}{|U_t| - 1}\\
&\leq \frac{W_t\left(\Phi_{\mathcal{C}_t}(U_t) - \Phi_{\mathcal{C}_t}(C^*_\alpha)\right)}{|U_t| - 1}
\end{aligned}$$

We need to bound the cost for $\Phi_{\mathcal{C}_t}(C^*_\alpha)$, where C^*_α is a randomly chosen cluster from U_t in $\mathcal{C}^*$. Its expected cost is given by

$$\begin{aligned}
E[\Phi_{\mathcal{C}_t}(C^*_\alpha)] &= \sum_{i \in U_t} \frac{\Phi_{\mathcal{C}_t}(C^*_i)}{\Phi_{\mathcal{C}_t}(U_t)}\Phi_{\mathcal{C}_t}(C^*_i)\\
&\geq \frac{\Phi_{\mathcal{C}_t}(U_t)}{|U_t|}
\end{aligned}$$

The above derivation uses Cauchy-Schwarz inequality and $\sum_{i \in U_t} \Phi_{\mathcal{C}_t}(C^*_i) = \Phi_{\mathcal{C}_t}(U_t)$.

Using the expressions $\Psi_{t+1} \leq \frac{W_t\left(\Phi_{\mathcal{C}_t}(U_t) - \Phi_{\mathcal{C}_t}(C^*_\alpha)\right)}{|U_t|-1}$ and $E[\Phi_{\mathcal{C}_t}(C^*_\alpha)] \geq \frac{\Phi_{\mathcal{C}_t}(U_t)}{|U_t|}$, we derive an expression for $E[\Psi_{t+1}]$ for Case 2.

Cauchy-Schwarz inequality states that for any two vectors a and b in $\Re^d$, $|a \cdot b| \leq |a||b|$, where "$\cdot$" represents the dot product. See Exercise 15.2.

$$\begin{aligned} E[\Psi_{t+1}] &\leq E\left[\frac{W_t\left(\Phi_{\mathcal{C}_t}(U_t) - \Phi_{\mathcal{C}_t}(C^*_\alpha)\right)}{|U_t| - 1}\right] \\ &\leq \frac{W_t}{|U_t| - 1}\left(E[\Phi_{\mathcal{C}_t}(U_t)] - E[\Phi_{\mathcal{C}_t}(C^*_\alpha)]\right) \\ &\leq \frac{W_t}{|U_t| - 1}\left(\Phi_{\mathcal{C}_t}(U_t) - \frac{\Phi_{\mathcal{C}_t}(U_t)}{|U_t|}\right) \\ &= \frac{W_t}{|U_t|}\Phi_{\mathcal{C}_t}(U_t) \\ &= \Psi_t \end{aligned}$$

Therefore, as expected, in this case $E[\Psi_{t+1} - \Psi_t] \leq 0$.

Now, we put Cases 1 and 2 together to derive an expression for $E[\Psi_{t+1} - \Psi_t]$. For this, we need to calculate the probability that we are in Case 1 times the difference in the potential in Case 1 and the probability that we are in Case 2 times the difference in potential in Case 2. We can ignore Case 2 as $E[\Psi_{t+1} - \Psi_t] \leq 0$. The probability that we are in Case 1 is $\frac{\Phi_{\mathcal{C}_t}(H_t)}{\Phi_{\mathcal{C}_t}(X)}$ and by Equation 15.4, $E[\Psi_{t+1} - \Psi_t] \leq \frac{\Phi_{\mathcal{C}_t}(U_t)}{|U_t|}$. Thus, we have

$$\begin{aligned} E[\Psi_{t+1} - \Psi_t] &\leq \frac{\Phi_{\mathcal{C}_t}(H_t)}{\Phi_{\mathcal{C}_t}(X)}\frac{\Phi_{\mathcal{C}_t}(U_t)}{|U_t|} \text{ (Case 1) } + 0 \text{ (Case 2)} \\ &\leq \frac{\Phi_{\mathcal{C}_t}(H_t)}{|U_t|} \\ &= \frac{\Phi_{\mathcal{C}_t}(H_t)}{k - t} \end{aligned}$$

■

What we have seen in the analysis is that if the centers in the k-Means++ algorithm are chosen from each cluster of $\mathcal{C}^*$, the algorithm is 8-competitive. Whereas, if the algorithm doesn't choose centers from some of the $\mathcal{C}^*$ clusters, then the $8(\ln k + 2)$-factor analysis applies.

Now, we discuss the computational complexity of the k-Means++ algorithm. In Step 2, for each point $x \in X$, we need to determine the closest center in the current set $\mathcal{C}$. Since the size of $|\mathcal{C}| \leq k$, for each point $x \in X$, it takes $O(kd)$ time to determine the closest point in X. Note that X consists of n points in $\Re^d$. Thus, the overall execution of Step 2 takes $O(nd)$ time. We execute Step 2 k times; thus the overall complexity of Steps 1-3 of k-Means++ algorithm is $O(nk^2d)$-time. The computation of each phase of Llyod's algorithm is similar, as it requires the computation of centroids and then determining its nearest centroid for each point in X. It is difficult to determine how

many phases are required in Llyod's heuristic until it terminates. Experimental studies indicate that the heuristic converges very quickly. Therefore, we can conclude that the execution time of k-Means++ algorithm is $O(nk^2d \times \tau)$, where τ is the number of phases Llyod's method needs for convergence.

15.4 Bibliographic Notes

k-Means heuristic is sketched in [106]. The k-Means++ randomized algorithm is discussed in [10]. A somewhat simplified analysis is presented in [42], and our notes are derived from this analysis. The **NP**-Hardness of clustering is discussed in [43]. There are several resources on clustering, including a new section in the 4th edition of [37] in the Machine-Learning Algorithms chapter.

In [36], it is shown that after finding the k centers using the k-Means++ algorithm, additionally, if we execute ϵk local search steps, for some $\epsilon > 0$, then the competitive ratio is within a factor of $O(\frac{1}{\epsilon^3})$ with a constant probability. In each step of local search, one chooses a point of $x \in X$ with probability $\frac{D(x)^2}{\sum_{y \in X} D(y)^2}$ and checks if it will be better to exchange x with some other point into the set $\mathcal{C}$, i.e., whether the exchange leads to a decrease in the potential.

In [11], rather than sampling a single point in each of the Randomization Step (Step 2) of the k-Means++ algorithm, they suggest sampling $O(k)$ points and repeating Step 2 for $O(\log n)$ rounds. In all, $O(k \log n)$ centers are chosen. For each point, determine how many points of the set X are closest to it; that is the weight of this point. These weighted points are clustered into k clusters using the k-Means++ clustering. Now, these centers are used as the initial cluster centers for Llyod's algorithm. The authors show that this algorithm performs better and faster experimentally.

Suppose in the k-Means++ algorithm, instead of picking one center in each execution of Step 2, we pick $l \geq 2$ candidate centres randomly using the sampling based on the squared distance to the nearest centres. Among these centers, greedily select the one that minimizes the potential as the chosen center for this iteration. In [72], it is shown that this variant has a competitive ratio of $O(l^3 \log^3 k)$. This seems counterintuitive as k -Means++ is $O(\log k)$-competitive, but the authors provide a lower bound that almost matches the upper bound.

15.5 Exercises

15.1 *Prove Lemma 15.1.1. First prove the lemma for $k = 2$.*

15.2 *Let $a_1, \ldots, a_n$ be n real numbers. Using Cauchy-Schwarz inequality, show that* $\sum_{i=1}^{n} a_i^2 \geq \frac{1}{n}\left(\sum_{i=1}^{n} a_i\right)^2$.

15.3 *Consider the k-Means clustering using Lloyd's heuristic for a particular set P of 5000 points in the plane. The points are partitioned into three groups: A, B, and C. Group B consists of 3000 points uniformly distributed in a circle of radius 1 centered around the origin. Group A consists of 1000 points, uniformly distributed in a circle of radius 1 centered at $(-10, 0)$. Group C consists of 1000 points uniformly distributed in a circle of radius 1 centered at $(10, 0)$. Suppose we want to compute a clustering of this point set into three clusters using Lloyd's algorithm. In this algorithm, we choose three initial centers, x, y, and z, and cluster all the points according to which of x, y, or z they are closest to. The result will be three clusters, which may or may not coincide with the groups A, B, and C. A cluster reported by the algorithm is correct if it consists of all and only the points from a particular group. Assume the initial centers x, y, and z are chosen independently and uniformly at random (with replacement) from the set P. What is the probability that A is correct? What is the probability that C is correct? What is the probability that both are correct?*

15.4 *Generalize the proof of Lemma 15.2.1 for dimensions $d > 2$.*

15.5 *Let $C_1, \ldots, C_k$ be a k-clustering of X with centers $\mathcal{C} = \{c_1, \ldots, c_k\}$, where $c_i \in \mathbb{R}^d$. We defined the potential function as*

$$\Phi_{\mathcal{C}}(X) = \sum_{x \in X} \min_{c \in \mathcal{C}} d(x, c)^2 = \sum_{x \in X} \min_{c \in \mathcal{C}} ||x - c||^2 = \sum_{x \in X} D(x)^2.$$

Now suppose, we change the metric. For $l \geq 1$, define

$$\Phi^l_{\mathcal{C}}(X) = \sum_{x \in X} \min_{c \in \mathcal{C}} d(x, c)^l = \sum_{x \in X} \min_{c \in \mathcal{C}} ||x - c||^l = \sum_{x \in X} D(x)^l.$$

In the k-Means++ algorithm, in Step 2, now choose the next center with probability $\frac{D(x)^l}{\sum_{y \in X} D(y)^l}$. Answer the following:

1. *Show that the following analog of Lemma 15.3.1 holds. Let A be an arbitrary cluster in optimal clustering $\mathcal{C}^*$. Let $\mathcal{C}$ be the clustering with exactly one center that is chosen from A uniformly at random. Then, $E[\Phi^l_{\mathcal{C}}(A)] = 2^l \Phi^l_{\mathcal{C}^*}(A)$.*
   Hint: See [10].

2. *Show the following equivalent of Lemma 15.3.2. Let A be an arbitrary cluster in optimal $\mathcal{C}^*$. Let $\mathcal{C}$ be an arbitrary clustering. Suppose the next center to $\mathcal{C}$ in the k-Means++ algorithm is added from A, $E[\Phi^l_{\mathcal{C}}(A)] \leq 2^{2l} \Phi^l_{\mathcal{C}^*}(A)$.*

15.6 *Consider the following variant of the clustering problem, called the* k-center *problem. Let* $G = (V, E)$ *be a complete graph and each edge has a positive distance. The distances satisfy the metric property (symmetric, triangle inequality, etc.). We are asked to find a subset* $S \subset V$ *of size* k*, such that the maximum distance from any vertex* $v \in V$ *to its nearest neighbor in* S *is minimized. Consider the following greedy algorithm.*

> *Step 1: Select arbitrarily any vertex* $v \in V$*, and set* $S := \{v\}$*.*
>
> *Step 2: While* $|S| < k$ *do:*
>
> *Find the vertex* $v \in V$ *that is furthest from* S*, and set* $S := S \cup \{v\}$*.*
>
> *Step 3: Return* S*.*

Show that the above greedy algorithm provides a 2*-approximation to the* k*-center problem. Formally, let* S^* *be an optimal solution, where the largest distance of any vertex* $v \in V$ *to* S^* *is* $\leq r^*$*. Show that for every vertex* $v \in V$*, the distance to* S *is* $\leq 2r^*$*.*

Hint: Consider the optimal clustering with respect to S^* *and let* $V_1^*, \ldots, V_k^*$ *be the partition of* V *with with respect to nearest neighbors in* S^**. Argue what happens when* S *has vertices from each set and when* S *misses vertices from some set.*

15.7 *Let* $G = (V, E)$ *be a simple graph. A subset* $S \subset V$ *is called a* dominating set *in* G *if for every vertex* $v \in V$*, either* $v \in S$ *or at least one of the neighbors of* v *is in* S*. Show that one can construct a complete graph* G' *on the vertex set* V *for the* k*-center problem, where each edge has a weight from the set* $\{1, 2\}$ *such that* G' *has* k*-centers* S*, where distance of each vertex to its nearest center in* S *is* 1 *if and only if* G *has a dominating set of size* k*.*

15.8 *Given that deciding whether a graph* $G = (V, E)$ *has a dominating set of size at most* k *is* **NP***-Hard, show that it isn't possible to approximate the* k*-center problem by a factor that is strictly less than* 2 *in polynomial time unless* $\mathbf{P} = \mathbf{NP}$*.*

16
Network Flow

Caution: This is a pre-preliminary draft. This chapter was last edited in early 2000.

We will focus on

1. Flow network
2. Maximum flow problem
3. Ford and Fulkerson's max flow algorithm
4. Flow augmentation
5. Edmonds-Karp flow algorithm

16.1 What is a Flow Network

A *flow network* consists of the following:

1. A simple finite directed graph $G = (V, E)$.
2. Two specified vertices, namely source s and target t.
3. For each edge $e \in E$, a non-negative number $c(e)$ called the capacity. If a pair of vertices u and v are not joined by an edge, then $c(u, v) = 0$.

Flow: A flow function f in G is a real-valued function

$$f : V \times V \to \Re$$

that satisfies the following three properties.

1. **Capacity Constraint**: For all $u, v \in V$, $f(u, v) \leq c(u, v)$.

2. **Skew Symmetry** (A tough constraint to see!): For all $u, v \in V$, $f(u,v) = -f(v,u)$. This is for notational purposes, and basically says that flow from a vertex u to vertex v is the negative of the flow in the reverse direction.

3. **Flow conservation**: For all $u \in V - \{s,t\}$, $\sum_{v \in V} f(u,v) = 0$. This uses the skew symmetry property, otherwise we have to sum up the flow values coming into a vertex and that should be equal to the sum of the outgoing flow values from that vertex. This is same as the Kirchoff law for current in an electrical circuit, i.e. no node can hold the current, or no node can hold the flow, or whatever comes in goes out. There is no reservoir at a node.

The value of the flow is defined to be the flow out of the source s or the flow into the target t, i.e.

$$|f| = \sum_{v \in V} f(s,v) = \sum_{v \in V} f(v,t).$$

The **Maximum Flow Problem** to find the flow of maximum value in a given flow network.

See Figure 16.1 for an example of a network flow.

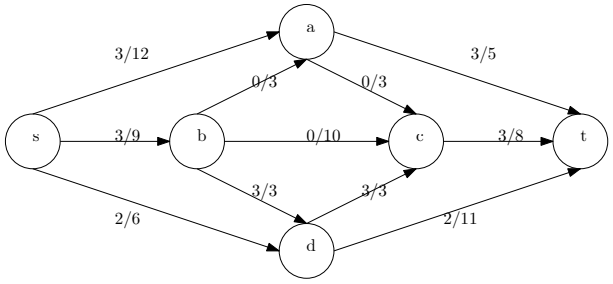


Figure 16.1: An example of a network flow with a flow of 8 from s. Each edge shows the amount of flow on that edge (numerator term) and the total capacity (denominator term).

16.2 *Ford and Fulkerson's Algorithm*

This is an iterative method for computing the flow.

Ford-Fulkerson-Method (G, s, t)

1. Initialize the flow f to 0.
2. While there exists an augmenting path p, augment the flow f along p.
3. Return f.

An augmenting path is a path from s to t along which additional flow can be sent. This path is found using the concept of residual

networks. The residual network consists of those edges which can admit more flow. The residual capacity $c_f(u,v)$ of an edge (u,v) in a flow network is given by

$$c_f(u,v) = c(u,v) - f(u,v).$$

In our example $c_f(s,a) = 12-3 = 9$, $c_f(a,s) = 0-(-3) = 3$, $c_f(b,a) = c(b,a) - f(b,a) = 3-0 = 3$. Given a flow network G and the flow function f, the residual network $G_f = (V, E_f)$ consists of the same vertex set and the edges E_f are defined as follows:

$$E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}.$$

The residual network of our example is given in Figure 16.2.

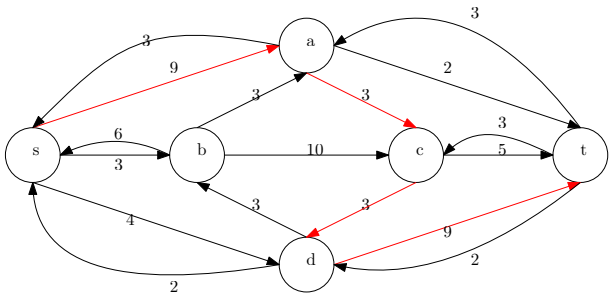


Figure 16.2: Residual network corresponding to the flow in Figure 16.1. The red-path from s to t is an augmenting path, with a residual capacity of 3.

As we can see that there is an augmenting path in this network (the red path), and the flow can be augmented along this path, by a value of 3. Hence we get a new flow network with the total flow value equals to $8+3 = 11$ given in Figure 16.3. Note that edge dc has a flow of 0 after the augmentation, whereas it had a flow of 3 units before the augmentation.

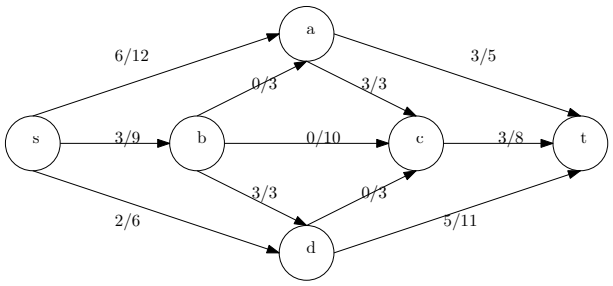


Figure 16.3: Flow network after augmenting the flow from Figures 16.1 and 16.2.

The new residual network that we obtain for the flow corresponding to the flow network in Figure 16.3 is given in Figure 16.4. Note that there exists an augmenting path that can further increases the flow value by 4.

The new flow graph is shown in Figure 16.5 and the corresponding residual network is shown in Figure 16.6.

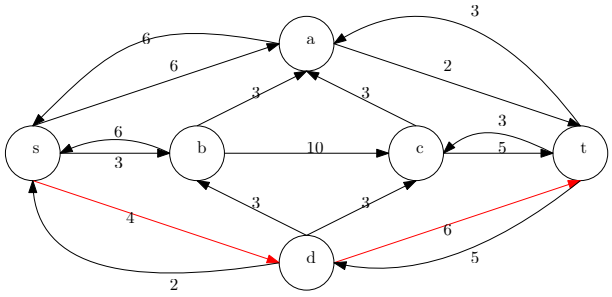


Figure 16.4: Residual network corresponding to the flow in Figure 16.3.

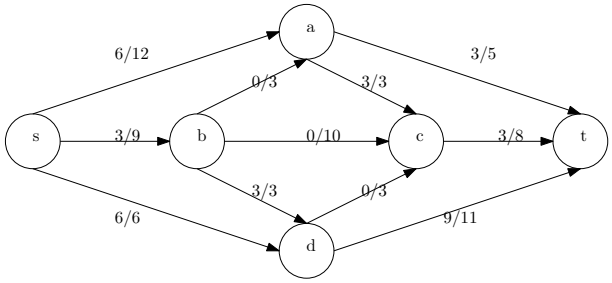


Figure 16.5: Flow network after augmenting the flow from Figures 16.3 and 16.4.

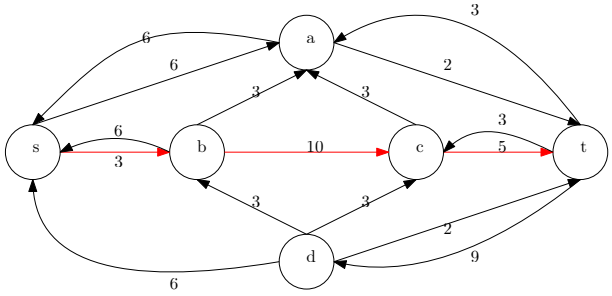


Figure 16.6: Residual network corresponding to the flow in Figure 16.5.

This will continue for a few more iterations and after that there is no path between s and t in the residual network. The corresponding figures are Figure 16.7 and Figure 16.8.

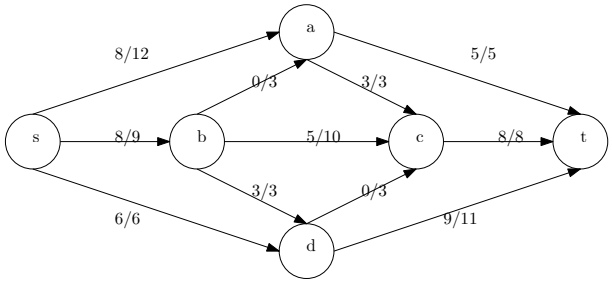


Figure 16.7: Resulting Flow network.

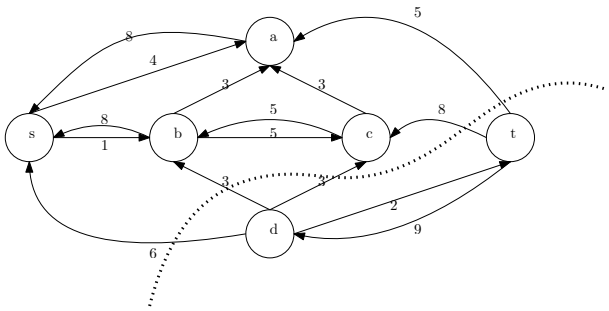


Figure 16.8: Residual network corresponding to the flow in Figure 16.7.

Now consider the residual network in Figure 16.8, where there are no paths joining s and t. As can be seen from the figure, there is a path from s to every vertex in the set $\{s, a, b, c\}$, and there are paths from vertices $\{d, t\}$ to t. This automatically partitions the set of vertices into two, call it a $s - t$ cut $\{S, T\}$, where $s \in S$ and $t \in T$ (in our example, $S = \{s, a, b, c\}$ and $T = \{d, t\}$). See Figure 16.9. Define the capacity of a cut as follows

$$c(S, T) = \sum_{u \in S, v \in T} c(u, v).$$

In other words consider the edges crossing the cut, and sum up the capacities of the edges which go from a vertex in the set S to a vertex in the set T. Define the net flow across the cut to be

$$f(S, T) = \sum_{u \in S, v \in T} f(u, v).$$

In other words the net flow is the sum of the positive flow on edges going from S to T minus the sum of the positive flows on edges going from T to S (recall the skew symmetry property). Amazingly in our example $f(S, T) = c(S, T)$. Is it always true or just a luck! Before we get to this, a few observations.

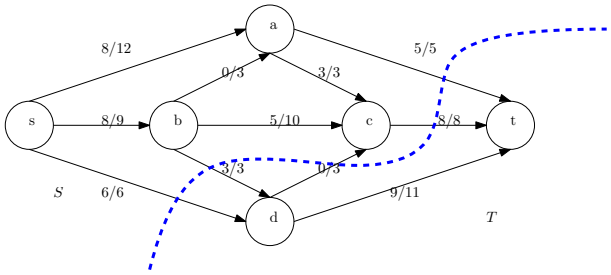


Figure 16.9: An illustration of a $s-t$ cut. Note that $f(S,T) = \sum_{u\in S, v\in T} f(u,v) = f(s,d) + f(b,d) - f(d,c) + f(c,t) + f(a,t) = 22$ and $c(S,T) = \sum_{u\in S, v\in T} c(u,v) = c(s,d) + c(b,d) + c(c,t) + c(a,t) = 22$.

Observation 16.2.1 *For any $s-t$ cut S, T, and flow f*

$$|f| \leq c(S,T).$$

This follows from the definition of the flow across the cut. The flow $f(S,T)$ is defined to be the sum of the positive flows along the edges in the forward direction, i.e. the ones going from vertices in S to vertices in T minus the sum of the positive flows along the edges in the reverse direction. If we ignore the reverse direction, then clearly the flow along each edge in the forward direction is bounded by the capacity of the edge. Sum of these capacities is the capacity of the cut and hence the observation.

The following observation explains why the flow f' found using the augmenting paths in the residual graph G_f, can be augmented with the flow f in G, to obtain a new flow in G of a higher value $|f + f'| \geq |f|$.

Observation 16.2.2 *Let G be the flow network with flow f and G_f be the corresponding residual network and let f' be the flow in G_f. Then the flow sum $f + f'$ is a flow in G and its value is $|f + f'| = |f| + |f'|$.*

Proof. To prove that $f + f'$ is a flow in G, we need to prove that the three conditions are satisfied. We show the capacity constraint, and others are left as an exercise. The capacity constraint follows from

$$(f + f')(u,v) = f(u,v) + f'(u,v) \leq f(u,v) + (c(u,v) - f(u,v)) = c(u,v).$$

Observe that

$$|f + f'| = \sum_{v\in V} (f + f')(s,v) = \sum_{v\in V} f(s,v) + \sum_{v\in V} f'(s,v) = |f| + |f'|.$$

■

Theorem 16.2.3 *Let f be a valid flow in the flow network $G = (V,E)$ from the source s to the target t, then the following statements are equivalent.*

1. *Flow f is a maximum flow.*

2. *Residual network G_f does not contain an augmenting path.*

3. *There exists some cut $c(S,T)$ such that $|f| = c(S,T)$.*

This is the famous max-flow min-cut theorem.

Proof. Recall that to prove that the three statements are equivalent we need to show that $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 1$.
$1 \Rightarrow 2$: Let f be a maximum flow and, for contradiction, assume that there exists an augmenting path in G_f. Then we can increase the flow along the path using Observation 16.2.2 and contradicting that f is a maximum flow.
$2 \Rightarrow 3$: Define the set

$$S = \{v \in V | \text{there is a path from } s \text{ to } v \text{ in } G_f\}$$

and

$$T = V \setminus S.$$

Also observe that $s \in S$ and $t \in T$, so it is a valid $s - t$ cut. Moreover for all edges (u,v) crossing the cut, where $u \in S$ and $v \in T$, $f(u,v) = c(u,v)$, otherwise $(u,v) \in E_f$ and $v \in S$, which is not possible. Net flow across the cut (S,T) is $|f|$! Why? (Think about this yourself!) So we have shown a cut where 3 holds.
$3 \Rightarrow 1$: We know that the capacity of any cut is an upper bound to the value of the flow. If for a cut we obtain the equality, then we have attained the max-flow. (In other words the capacity of minimum cut is the value of the maximum flow!). ■

This proves the correctness of the Ford-Fulkerson algorithm. The algorithm iteratively increases the value of the flow using augmenting paths and returns the value of the flow, the maximum flow, when it is not able to find an augmenting path in the residual graph. How do we analyze the complexity of this algorithm?

First a special case where all capacities are integers. Observe that value of all flows computed during the algorithm are integers. In each iteration of the algorithm, the value of flow increases by at least 1. If f^* is a maximum flow, then the number of iterations in the algorithm are bounded by $|f^*|$. It is easy to see that each iteration requires $O(|E|)$ time; this involves computing residual graph (i.e. capacities on at most $2|E|$ edges), and computing a path between s and t (directed dfs or bfs). Hence the algorithm runs in $O(|f^*||E|)$ time - this is a strange complexity since the running time depends upon the value of the output! Is there a better way to analyze this algorithm!

16.3 Edmonds-Karp Algorithm

In this algorithm, in the Ford-Fulkerson method, a particular path is chosen to be an augmenting path in the residual graph. A BFS tree rooted at s is computed in the residual graph and an unweighted shortest path from s to t is chosen to be an augmenting path. It turns out that this variation leads to an algorithm that runs in $O(|V||E|^2)$ time. Here is the main lemma - let $\delta_f(s,v)$ denote the shortest path distance between s and v in the unweighted residual graph G_f, corresponding to the flow network G with flow function f.

Lemma 16.3.1 *Shortest path distance for each vertex $v \in V - \{s,t\}$ in G_f is non-decreasing with each flow augmentation.*

Proof.

Caution: This is a little bit strange proof, and the proof in generic terms goes as follows. To prove the statement P, the contradictory proof assumes that $\neg P$ is true. Inside the proof, we need to establish a claim C, which is proved using contradiction. Assume $\neg C$ is true. The contradiction is arrived at by showing that $\neg C$ is valid only if P is true. Since $\neg P$ is assumed to be true, thus implying that C is true. Once we show that C is true, the contradiction to the original assumption is arrived at.

Assume that for a vertex $v \in V - \{s,t\}$, the shortest path decreases after a flow augmentation. Let f be the flow before the augmentation, and f' be the flow after the augmentation. Let v be the vertex with minimum $\delta_{f'}(s,v)$ whose distance was decreased by the augmentation (i.e. $\delta_{f'}(s,v) < \delta_f(s,v)$). Let u be the vertex just before v in the shortest path from s to v in $G_{f'}$, i.e. $(u,v) \in E_{f'}$. Then $\delta_{f'}(s,u) = \delta_{f'}(s,v) - 1$. Moreover, $\delta_{f'}(s,u) \geq \delta_f(s,u)$ (by choice of v).

Now we will show that $(u,v) \notin E_f$, and as a consequence of that, we will arrive at contradiction (somehow!).

First why $(u,v) \notin E_f$? Assume not, i.e., suppose $(u,v) \in E_f$. Then, by triangle inequality, $\delta_f(s,v) \leq \delta_f(s,u) + 1$. But, $\delta_f(s,u) + 1 \leq \delta_{f'}(s,u) + 1 = \delta_{f'}(s,v)$. This implies that $\delta_f(s,v) \leq \delta_{f'}(s,v)$, contradicts our assumption!

Now consider the scenario that $(u,v) \notin E_f$ and $(u,v) \in E_{f'}$. The flow from v to u must have been increased in Edmonds - Karp algorithm and this edge must be on a shortest path. This implies that $\delta_f(s,v) = \delta_f(s,u) - 1 \leq \delta_{f'}(s,u) - 1 = \delta_{f'}(s,v) - 2$, and this contradicts the assumption that $\delta_{f'}(s,v) < \delta_f(s,v)$. ■

In each iteration of the augmenting path algorithm, at least one edge becomes critical, i.e. flow value becomes equal to its capacity.

The critical edge disappears from the residual network. Of course the flow along this edge may be decreased in the future, and this edge may reappear again in the residual network, but this cannot happen more than $|V|/2$ times. Why?

Say (u, v) became critical, then $\delta_f(s, v) = \delta_f(s, u) + 1$. Flow along (u, v) is decreased only if (v, u) appears on an augmenting path. Let this happens when f' is the flow and note that $\delta_{f'}(u) = \delta_{f'}(v) + 1$. Since the shortest path distances are monotone, this implies that

$$\delta_{f'}(s, u) = \delta_{f'}(v) + 1 \geq \delta_f(s, v) + 1 = \delta_f(s, u) + 2.$$

Therefore the distance to u from the source has increased by at least 2 between two consecutive times that (u, v) became critical. The maximum distance is at most $|V|$ and hence an edge can become critical at most $|V|/2$ times. There are $O(|E|)$ edges in all in the residual graph, and hence the number of augmentations (or iterations) are bounded by $O(|V||E|)$ times. Each augmentation can be implemented in $O(|E|)$ time, and hence flow between s and t in the graph $G = (V, E)$ can be computed in $O(|V||E|^2)$ time.

16.4 Applications of Network Flow

We can use the flow networks to compute Maximum Matching in a Bipartite Graphs. Recall that a Graph $G = (V = A \cup B, E)$ is bipartite, if the set of vertex V is partitioned into two sets A and B, such that all the edges in the graph are between vertices of A to vertices in B. A matching in a graph is a collection of edges such that no two edges in the matching are incident to the same vertex. A matching in G is called a *maximum matching* if the cardinality of the number of edges in it is maximum among all matchings in G. Note that there can be a number of maximum matching in a graph. Using flow networks we can compute easily maximum matching in G. Here is the simple method. We add two vertices, namely s and t, to the set of vertices in G. Vertex s is connected to all the vertices in the set A by directed edges from s. The capacity of all these edges is set to 1. The capacity of all the edges in the set E, i.e., the edges joining vertices in the set A to vertices in the set B, is set to 1 and they are directed from vertices in A to vertices in B. Lastly vertices in B are joined to t by directed edges with capacity 1. Let G' be the resulting flow network. Compute the maximum $s - t$ flow in G'. Observe that the value of the flow is the size of the maximum matching. Why ?

Note that value of flow in each of the edge will be an integral value, since all the capacities are integers (this is one of the exercises in [1]). Since the capacity of all the edges between vertices in A and B is 1, the value of the flow on these edges is either 0 or 1. This implies

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 4th edition, 2022

that no two edges in E are incident on the same vertex will ever have nonzero flow. In other words the edges in E which have nonzero flow are the edges in a matching. Also maximum matching corresponds to a largest set of independent edges in G and each of these edges can admit a flow of value 1 and at the same time satisfy all the three conditions required for a flow network. Hence maximum matching in G corresponds to a valid flow in G'.

16.5 Exercises

16.1 *Construct a network flow example with 7 vertices and 11 directed edges, where each edge has a positive capacity and compute the maximum flow and minimum cut in this graph. You should show some of the steps in the algorithm. (Follow Edmonds-Karp shortest path heuristic).*

16.2 *Assume that we have a network flow graph $G = (V, E)$ with positive capacities on each of the edges and two specified vertices s and t. Suggest an efficient algorithm to find an edge in E, such that setting its capacity to zero (i.e. deleting this edge) will result in the largest decrease in the maximum flow in the resulting graph.*

16.3 *Suppose we are given a flow network G, where edges have positive integer capacities, and $\mathcal{C} = \sum_{u,v \in V} c(u, v)$, where $c(u, v)$ is the capacity of the edge $e = (u, v) \in E$. Show the following*

1. *The value of the max flow is an integer.*
2. *There is an assignment of non-negative integer flow values on each edge of G, satisfying all the flow conservation conditions, so that G achieves max flow.*
3. *Show that the number of iterations required in the Ford-Fulkerson's algorithm (Residual network, find an augmenting path, augment the flow, repeat) is $O(\mathcal{C})$.*
4. *Show that in the worst case, Ford-Fulkerson's algorithm, as stated in Part 3 runs in exponential time.*
5. *Construct an example, where one can realize the worst case as stated in Part 4.*

16.4 *Let $G = (V, E)$ be the flow network. Let $C = \max_{(u,v) \in E} c(u, v)$ be the maximum capacity. Show the following:*

1. *Minimum cut of G has a capacity of at most $C|E|$.*
2. *For a given number $K > 0$, show how to find an augmenting path of capacity at least K in $O(|E|)$ time, provided that such a path exists.*

3. *Execute the following algorithm:*

   (a) *Initialize the flow* $f = 0$*;*

   (b) $K = 2^{\lfloor \log C \rfloor}$*;*

   (c) *While* $K \geq 1$ *do*

      i. *While there exists an augmenting path* p *of capacity at least* K *then augment flow* f *along* p*.*

      ii. $K := K/2$*;*

   (d) *Return* f*.*

   Show that the above algorithm computes Max-Flow.

4. *Show that the loop in Step 3c(i) is executed at most* $O(|E|)$ *times for each value of* K*.*

5. *Show that the algorithm runs in* $O(|E|^2 \log C)$ *time.*

16.5 *Let* $G = (V, E)$ *be a flow network. Recall that* G *is a complete graph, where some of the edges may have a capacity of zero. Suppose your task in the max flow problem is to increase the flow of a network as much as possible, but you are only allowed to increase the capacity of only one edge, whose capacity is strictly larger than zero. First show that there are networks where such an edge may not exist, i.e. increasing the capacity of a single edge (*> 0 *capacity) will not alter the value of the max-flow. Show that there are networks, where such an edge may exist. Try to design an algorithm which can detect whether flow can be increased.*

16.6 *A simple undirected graph* $G = (V, E)$ *is called k-edge connected if removal of any set of k-edges keeps G still connected. (e.g. cycles are 1-edge connected.) Show how to compute edge connectivity of G by invoking at most* $|V|$ *network flow computations.*

17
Additional Exercises

This is mainly a dump of questions that have been asked in the assignments over years in COMP 5703, COMP 5112, COMP 4804, COMP 3804 and COMP 3801. The relevant subject material may not be in the notes!

17.1 Problems

1. Let $G = (V, E)$ be a simple undirected graph. Provide an algorithm running in $O(|V| + |E|)$ time that outputs whether G contains a cycle or not. If it contains a cycle - then it needs to output at least one cycle. What graph representation you have used for your algorithm? Justify and and see how it influences the complexity analysis.

2. Design an algorithm that determines whether a directed graph $G = (V, E)$ is an acyclic graph (i.e., it doesn't contain a directed cycle). Your algorithm must run in $O(|V| + |E|)$ time.

3. Typically departments in universities (like Carleton) offer many courses, but to register in a course, one needs to have completed all the required prerequisite courses. We can easily model this relationship as a directed graph, where each course is a vertex, and a directed edge from course u to v if and only if u is a prerequisite course for taking w. It should be clear that this graph should not contain any directed cycles (otherwise we won't graduate!). (For example, if COMP 1405 and COMP 1805 are required for taking COMP 2402, and COMP 2402 is required for taking COMP 3804, we will have directed edges from vertices corresponding to COMP 1805 and COMP 1405 to COMP 2402, and a directed edge from COMP 2402 to COMP 3804.) Given a directed graph $G = (V, E)$ in adjacency list representation, representing the courses and their prerequisites, your task is to compute minimum number of terms one needs to spend in the department to complete the degree,

where you can assume that you can do any number of courses in any term, provided that the prerequisite conditions are met. What if you limit the number of courses within a term that the student can take to 6?

4. Given a directed graph $G = (V, E)$, where each vertex has a distinct integer label. For each vertex v, define $R(v)$ to be the set of all vertices $w \in V$ for which there is a directed path from v to w in G. Furthermore, for each vertex $v \in V$, define $\mathsf{MinLabel}(v)$ to be the vertex with the minimum label in the set $R(v)$. Provide an algorithm, running in $O(|V| + |E|)$ time, that computes $\mathsf{MinLabel}(v)$ for all vertices $v \in V$.

5. Let s and t be two specific vertices of an undirected connected simple graph $G = (V, E)$ on n vertices, where any path between s and t in G consists of at least $n/2 + 2$ vertices. Show that there is a vertex $v \in V$, $v \neq s$ and $v \neq t$, such that any path from s to t passes through v. Also, provide an algorithm, running in $O(|V| + |E|)$ time, for identifying such a vertex v for a given pair of vertices $s, t \in V$. (Note that by removing v from G, we disconnect s and t.)

6. There are two algorithms for a problem of size n. The recurrence for the running time of Algorithm I is $T(n) = 2T(n/2) + n$, and the recurrence for the running time of Algorithm II is $T(n) = T(n/3) + T(2n/3) + n$. Is the running time of Algorithm II asymptotically smaller than that of Algorithm I for large values of n? You can assume that $T(1) = 1$.

7. What does the recurrence $T(n) = T(\lceil \frac{7}{10} n \rceil) + T(\lceil \frac{1}{5} n \rceil) + n$, where $T(1) = 1$, evaluates to ?

8. Is it true that if the SATISFIABILITY problem can be solved in polynomial time, then $\mathcal{P} = \mathcal{NP}$?

9. Is it true that if a graph $G = (V, E)$ on 9 vertices has a clique of size 6, than the complement of G has a vertex cover of size 3? (Recall that a *clique* is a complete graph. A *vertex cover* $C \subseteq V$ is the set of vertices such that for each edge $e = (u, v) \in E$, $u \in C$ or $v \in C$. *Complement* $G' = (V, E')$ of the graph $G = (V, E)$ is a graph on the same vertex set V and $e' = (u, v) \in E'$ if and only if $(u, v) \notin E$.)

10. Given an undirected connected graph $G = (V, E)$, in adjacency list representation, can it be decided within $O(|V| + |E|)$ time whether there is a path between two specific vertices x and y consisting of at most 50 edges, where $x, y \in V$?

11. Is it sufficient to perform 6200 multiplications in order to multiply four matrices
$A \times B \times C \times D$, where dimensions of matrix A is 10×100, Bs is 100×20, Cs is 20×2 and Ds is 2×10? (Note that if we have two matrices X and Y, with dimensions $p \times q$ and $q \times r$, respectively, then pqr multiplications are sufficient to compute XY.)

12. Let S be a set of n-real numbers. Given a real value t, we need to find a subset $S' \subseteq S$, such that the sum total of the elements of S' equals to t or report that no such S' exists. The following two-step algorithm finds the set S' (if it exists):

    (a) Compute all possible subsets of S.

    (b) For each subset S' of S, check whether the sum of the elements of S' equals to t.

    The following complexity analysis of the above algorithm is provided:
    Since, in all, there are $O(n^2)$ subsets of S, and the sum total of the elements in any subset can be computed in $O(n)$ time, the above algorithm runs in $O(n^3)$ time.
    Is the above analysis correct?

13. What is the length of the longest increasing subsequence of the following sequence:
    $\langle 7, 3, 2, 19, 4, 11, 12, 6, 8, 9, 5, 27, 12, 16, 51, 42, 13 \rangle$

14. Suppose in the Minor Hockey League Championship play off series between Nepean Pirates and Kanata Thunders, there are three possible playoff games planned. A team that wins two games is declared the champion. Outcome of each game is either a win or a loss - there are no ties! The first game is played in the Pirates Arena, the Second game is played in the Thunder's arena, and if required the third game will be played in the Pirates arena. A team wins with a probability of 2/3rd in its home arena, and with a probability of 1/3rd in the opposition's arena. What is the probability that Nepean Pirates will be declared the Champion?

15. Suppose that 8% of all bicycle racers use steroids, that a bicyclists who uses steroids tests positive for steroids 96% of the time, and that a bicyclist who does not use steroids tests positive for steroids 9% of times. What is the probability that a randomly selected bicyclist who tests positive for steroids actually uses steroids?

16. Let $F(x)$ and $G(x)$ be two polynomials of degree d, where d is a positive integer. The polynomial F is given as a product of d monomials, and the polynomial G is given in the standard

form. For example, $F(x) = (2x+1)(x-1)(x+2)(3x-1)$ and $G(x) = 6x^4 + 7x^3 - 12x^2 - 3x + 2$. To check whether $F(x) \stackrel{?}{=} G(x)$, we can convert $F(x)$ to the standard form, and then verify whether $F(x)$ and $G(x)$ are identical. Unfortunately, converting $F(x)$ to standard form is cumbersome and an expensive operation.

A simple randomized algorithm to check whether $F(x) = G(x)$ is as follows. Choose an integer r uniformly at random from the range $[1, ..., 100d]$. Evaluate $F(r)$ and $G(r)$. If $F(r) \neq G(r)$ report $F(x) \neq G(x)$, otherwise report $F(x) = G(x)$.

Since evaluating a degree d polynomial takes time proportional to $O(d)$, the above algorithm is very fast and simple. Observe the following. If $F(r) \neq G(r)$, then clearly $F(x) \neq G(x)$ and algorithm reports the correct answer. If $F(x) = G(x)$, then no matter what values of r we choose, $F(r) = G(r)$ and the algorithm reports the correct answer. But, if $F(r) = G(r)$, we cannot conclusively say that $F(x) = G(x)$, as we may land up choosing r to be the root of the equation $F(x) - G(x) = 0$. First show that the above algorithm reports the wrong answer with probability at most $1/100$. Suggest some method(s) so that the probability of error can be further reduced, say for example to $1/10000$.

17. As a promotion, the NewAge Cereal has placed a toy car in each of its cereal boxes. You can determine the color of the toy car, only by buying and then opening the cereal box. Each toy car is of a monochromatic color among possible $n \geq 1$ colors. Once you collect cars of all possible colors, then you win a real car. The company officials have ensured that a cereal box is equally likely to contain a car of any of the possible n-colors. Let X be the random variable equal to the number of cereal boxes that need to be purchased to obtain at least one toy car of each of the colors. Let X_j be the random variable equal to the number of additional cereal boxes that must be purchased after cars of j different colors have been collected until a car of new color is obtained, for $j = 0, 1, 2, \ldots, n-1$. Answer the following questions:

    (a) Show that $X = \sum_{j=0}^{n-1} X_j$.

    (b) Show that after cars of j distinct colors have been obtained, the probability that the color of the car in the next cereal box that is purchased is new (i.e. different from any of the j colors) is $\frac{n-j}{n}$.

    (c) Show that X_j has a geometric distribution with parameter $\frac{n-j}{n}$.

    (d) Show that $E(X) = n \sum_{j=1}^{n} \frac{1}{j}$.

    (e) Suppose that $n = 100$. Use the approximation $\sum_{j=1}^{n} \frac{1}{j} \approx \ln n + 0.5772$ to determine the expected number of cereal boxes that needs to be bought to collect cars of all different colors.

18. Let S be a set of $n > 0$ distinct real numbers. Is it possible to report the k largest elements of S in sorted (increasing) order in $O(n)$ time, where $k = \lceil \frac{n}{\log n} \rceil$?

19. Let $G = (V, E)$ be a connected simple undirected graph where each edge has a positive weight. Consider the following problem. Find a connected spanning subgraph $G' = (V, E')$ of G , where $E' \subseteq E$, that minimizes the sum total of the weights of edges in G'. Is this problem $\mathcal{NP}$-Complete?

20. Recall that the 3CNF-SATISFIABILITY formula on n-boolean variables consists of AND of k clauses, where each clause consists of ORs of three literals. (For example, $(\bar{p} \vee q \vee r) \wedge (p \vee \bar{r} \vee s)$ is a 3CNF formula with two clauses on four variables.) Determining whether there exists a satisfying assignment for a 3CNF formula is an $\mathcal{NP}$-Complete problem. Suppose, all the ANDs are replaced by ORs, and all the ORs are replaced by ANDs in the 3CNF-Forumala - that is now we have OR of k clauses, where each clause consists of ANDs of three literals. (For our example, the new formula will be $(\bar{p} \wedge q \wedge r) \vee (p \wedge \bar{r} \wedge s)$.) Is the problem of determining the satisfying assignment of the new formula is still an $\mathcal{NP}$-Complete problem?

21. If a depth-first search traversal of a directed graph $G = (V, E)$ has no back edges, then is it true that the vertices of G can be assigned integer labels, so that for each directed edge $e = (uv) \in E$ (i.e., an edge e directed from vertex u to vertex v) the label assigned to u is strictly smaller than the label assigned to v?

22. Does the following algorithm computes a Minimum Spanning Tree (MST) of a weighted undirected connected graph $G = (V, E)$?

    Step 1: Sort edges in E in order of decreasing weight.

    Step 2: $T := E$

    Step 3: For each edge e taken in the order of decreasing weight do:
    if $T - \{e\}$ is connected, then discard e from T.

    Step 4: Return T as MST of G.

23. Does the following algorithm computes a Minimum Spanning Tree (MST) of a weighted undirected connected graph $G = (V, E)$?

    Step 1: $T := \emptyset$.

    Step 2: For each edge e, taken in an arbitrary order perform (**i**) and (**ii**):

    i: $T := T \cup \{e\}$.

ii: If T has a cycle c and let e' be a maximum weight edge in c, then $T := T - \{e'\}$ (i.e., remove e' from T).

Step 3: Return T as MST of G

24. Consider a simple undirected connected graph $G = (V, E)$, where weight of each edge is a positive real number. Let e be the edge with the largest weight in any minimum spanning tree of G. Consider the graph G' obtained from G by removing all edges in G whose weight is equal or larger than the weight of e. Is G' always disconnected?

25. Suppose you have an array A of n real numbers. You want to determine whether there are two indices $1 \leq i \leq j \leq n$ such that $\sum_{k=i}^{j} A[k]$ is an integer. Design an $O(n \log n)$ time algorithm to solve the decision problem. We can assume that for a real number x, $\lfloor x \rfloor$ results in returning the integer part of x in $O(1)$ time, e.g. $\lfloor \pi \rfloor = 3$.

26. This question is based on the *cut lemma* for minimum spanning trees. Let $G = (V, E)$ be a connected graph where each edge has a positive weight. If for any cut of G, there is a unique edge in the cut of minimum weight then show that minimum spanning tree of G is unique. Show that the converse may not be true using an example. I.e., construct a graph G that has a unique minimum spanning tree, but there are cut(s) in G containing multiple edges having the minimum weight.

27. Let $G = (V, E)$ be a weighted simple connected graph, and assume that all edge weights are distinct and positive. A *bottleneck spanning tree* T of G is a spanning tree of G whose largest edge weight is minimum over all spanning trees of G. Construct an example of a weighted graph G and a spanning tree T of G such that T is a bottleneck spanning tree but not a minimum spanning tree of G. Show that any minimum spanning tree of G is a bottleneck spanning tree.

28. Consider a connected graph $G = (V, E)$ where each edge has a non-zero positive weight. Furthermore, assume that all edge weights are distinct. Using the cut property, first show that that for each vertex $v \in V$, the edge incident to v with minimum weight belongs to a Minimum Spanning Tree (MST). Can you use this to devise an algorithm for MST - the above step identifies at least $|V|/2$ edges in MST - you can collapse these edges, by identifying the vertices and then recursively apply the same technique - the graph in the next step has at most half of the vertices that you started with - and so on. What is the running time of your algorithm?

Note that for an edge $e = uv$ in the graph $G = (V, E)$, *identifying* vertex u with v or *collapsing* e is the following operation: Replace the vertices u and v by a new vertex, say u'. Remove the edge between u and v. If there was an edge from u (respectively, v) to any vertex w ($w \neq u$ and $w \neq v$), then we add an edge (with the same weight as of edge uw (respectively, vw)), between the vertices u' and w. This transforms graph G to a new graph $G' = (V', E')$, where $|V'| < |V|$ and $|E'| < |E|$. Note that G' may be a multigraph (i.e., between a pair of vertices, there may be more than one edge). For example, if uv, uw, and vw are edges in G, then G' will have two edges between u' and w when we identify u with v. We can transform G' to a simple graph by keeping the edge with the lower weight among uw and vw as the representative for $u'w$ for the computation of MST.

29. Let $G = (V, E)$ be a connected simple graph, where each edge has a weight of 3. Devise an algorithm, running in $O(|V| + |E|)$ time, for computing shortest path distances from a specific vertex $s \in V$ to all other vertices of G.

30. Prove that the distance values extracted from the heap (priority queue) over the entire execution of Dijkstra's single source shortest path algorithm, in a directed connected graph with positive edge weights, is a NON-Decreasing sequence. Where is this fact used in the correctness of the algorithm?

31. In the summer vacation, you decided to travel to various communities in Northern Canada by your favorite ATV (All-Terrain Vehicle). Each of the communities you want to visit is represented as a vertex in your travel graph (a total of $|V|$ communities). Moreover, you are provided with distances between all pairs of communities. Think of your input graph as a complete graph (i.e. every pair of vertices are joined by an edge), and the weight of an edge, say $e = (uv)$ is the distance between the communities u and v. Since this is in far North, and the routes between communities are not used that often, the gas stations are only located in communities (there are absolutely no gas stations which are outside a community). Furthermore, we can assume that each community has at least one gas station. Once you completely fill up the tank of your ATV, it has an upper limit, say of Δ kilometers, which it can travel, and to travel any further it needs to fill up (which means at that point it needs to be in a community!). You need to answer the following two questions

    (a) First design a method, running in $O(|V| + |E|)$ time, which can answer whether is there some path which your ATV can

take, so that you can travel between two particular communities, say s and t. It is obvious that if the distance between s and t is at most Δ, then you can travel directly without refuelling. Otherwise, you can travel between s and t, provided there are communities where we can refuel and proceed. [For fun you may like to see whether you can travel from La Loche (in Sask.) to Mandorah (in Northen Territories), when your ATV with full tank can travel at most 100 Kms.]

(b) Design an algorithm running in $O(|E| \log |V|)$ time to determine the smallest value of Δ, which will enable you to travel from s to t.

32. Design dynamic programming algorithms for the following problems:

(a) Given a word w made of n alphabets, determine the longest palindrome in w. For example, afternoon has a palindrome of size 4 ("noon").

(b) Given a word w made of n alphabets, determine the longest subsequence of alphabets in w that make a palindrome. For example, in Alabama the longest palindromic subsequence is "Aaaa".

33. Given an unlimited supply of coins of denominations $x_1, \cdots, x_n$, we wish to make change for a certain value X. (We are not worried about finding the minimum set of coins that adds to X; we just want to make change; and also note that sometimes we may not be able to make an exact change). Design a dynamic programming algorithm running in $O(nX)$ time, that can decide whether the change for the amount X can be made or not? (Note that all the quantities $(X, x_1, \cdots, x_n)$ are integers.)

34. Consider the following variation of the above problem. Given coins of denominations $x_1, \cdots, x_n$, we wish to make change for a certain value X. But for any denomination, we can use at most one coin (absolutely no repetitions). (We are not worried about finding the minimum set of coins which adds up to X; we just want to make change; and also note that sometimes we may not be able to make an exact change). Design a dynamic programming algorithm running in $O(nX)$ time, that can decide whether the change for the amount X can be made or not?

35. Assume that you have a chocolate bar of length n inches. You have a satisfaction chart, that indicates that if you eat a piece of length i inches, for $1 \leq i \leq n$, then you get a satisfaction of s_i. We can

assume that all the quantities involved ($n, i, s_1, s_2, \ldots, s_n, ...$) are positive integers. You want to decide, using dynamic programming, what is the best way to make pieces of your chocolate bar so that you get a maximum satisfaction when you consume the whole bar. What is the time complexity of your dynamic programming algorithm?
For an example, suppose that the bar is 5-inches long, and if your satisfaction chart says:

Length of Piece in inches (i)	1	2	3	4	5
Satisfaction(s_i)	2	7	9	6	12

Then various ways to partition the 5-inch bar with its satisfaction values are as follows:
$5 = 2 + 3$, with satisfaction $7 + 9 = 16$.
$5 = 1 + 1 + 3$, with satisfaction $2 + 2 + 9 = 13$.
$5 = 2 + 2 + 1$, with satisfaction $7 + 7 + 2 = 16$.
No split, $5 = 5$, with satisfaction $12 = 12$.
.....

36. Let $T = (V, E)$ be a binary tree on n nodes. Note that T may not be a balanced tree. You want to find a subset of vertices $S \subset V$, such that for each edge $e = (uv) \in E$, atleast one of u or v is in S. Design an algorithm, running in polynomial time in n, that finds smallest such set S (i.e. $|S|$ is minimized among all such subsets $S \subset V$).

37. State in your own words what are the complexity classes $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-Hard, and $\mathcal{NP}$-Complete. Give an example of a problem in each of these classes. Are all problems in $\mathcal{P}$ in $\mathcal{NP}$?

38. Define what is a Vertex Cover and what is an Independent Set in a simple undirected graph $G = (V, E)$. Define the decision versions of the problems of computing (a) a minimum vertex cover and (b) a largest Independent Set in G. Provide a polynomial time reduction of the Vertex Cover problem to the Independent Set problem. Show that the algorithm that transforms one problem to the other problem runs in polynomial time, and it is a valid reduction (i.e. solution of one problem can be obtained from the solution of other problem).

39. Recall the SATISFIABILITY problem. Given n-Boolean variables, $x_1, \ldots, x_n$, a Boolean formula ϕ in the Conjunctive-Normal Form (CNF) is made of AND of $k > 0$ clauses $\phi = C_1 \wedge C_2 \wedge C_3 \cdots \wedge C_k$, where each clause C_i is made of OR's of one or more literals. Each literal is either a Boolean variable or its complement. For example, $C_3 = (x_1 \vee \overline{x_4} \vee x_7)$ is a clause made of three literals. The SATISFIABILITY problem is whether we can find a satisfying

assignment for ϕ, i.e. finding an assignment of Boolean values to $x_1, \ldots, x_n$ that makes ϕ true.

Given an undirected graph $G = (V, E)$, we say that G can be 3-*colored* if we can assign one of the colors from {Red, Blue, Green} to each vertex so that for any edge $e = (u, v) \in E$, u and v should get different colors. The decision version of the 3-coloring problem is to know whether there is an assignment of colors to vertices so that G is 3-colored.

Recall the definition of polynomial time reducibility and show that the 3-coloring problem is polynomial time reducible to the SATSIFIABILITY problem.
Hint: Note that for each vertex v_i, you need to assign it one of the colors from {Red, Blue, Green}. If v_i is assigned Red (Blue or Green) color, then we can say it's corresponding Boolean variable is R_i (B_i or G_i, respectively). Express the coloring constraints in terms of clauses. For example v_i needs to get at least one of the colors, therefore we can say that the corresponding clause is $(R_i \cup B_i \cup G_i)$. Come up with expressions for no vertex receiving more than one color, no two neighboring vertices receiving the same color, etc.

40. Assume that the decision version of the 3-SAT, Vertex Cover, and Clique problems are NP-complete. An independent set of an undirected graph $G = (V, E)$ is a subset I of V such that no two vertices in I are connected by an edge in E. The decision version of the independent set problem $\langle G, k \rangle$, $k \geq 0$, is to determine whether G has an independent set of size at least k. Prove that the Independent Set problem is NP-Complete.

41. Let $G = (V, E)$ be an undirected connected simple graph. A matching is a set of edges of G such that no two edges in the set are incident on the same vertex. A matching is **maximal** if it is not a proper subset of any other matching. A matching is **maximum** if the number of edges in the matching is largest. A vertex cover of $G = (V, E)$ is a set of vertices $V' \subseteq C$ such that if $(u, v) \in E$, then either $u \in V'$ or $v \in V'$ or both in V'. The size of the vertex cover is the cardinality of the set V'.

    (a) Show that the size of a maximum matching in G is a lower bound on the size of any vertex cover of G.

    (b) Consider a maximal matching M in $G = (V, E)$. Let

    $$T = \{v \in V : \text{some edge in M is incident on v}\}.$$

    What can you say about the subgraph of G induced by the

vertices of G that are not in T. Conclude that $2|M|$ is the size of a vertex cover for G.

(c) Present an $O(|E|)$ time algorithm to compute maximal matching.

(d) Conclude that the above algorithm for computing maximal matching is a 2-approximation algorithm for maximum matching.

42. We know that there are graphs that have more than one MST's. Let T be a Minimum Spanning Tree of $G = (V, E)$, and let L be the sorted list of edge weights of T (in non-decreasing order). Show that for any other MST T' of G, the list L is also the sorted list of edge weights of T'.

43. There is a road network between cities which is given to you as an undirected graph, and the vertices are the cities and there is an edge between two vertices, if and only if there is a direct road (not going through any other city) between the corresponding two cities. The weight of an edge is the distance between the two cities. There is a proposal to add one new road to this network, and there is a list E' of pairs of cities between which the new road can be built. Each such potential road has an associated length (the distance between the cities). As a politician, you need to decide which new road should be built, so that the new road leads to the maximum decrease in the distance between two specific (favorite) cities, say s and t. Give an efficient algorithm for determining which edge $e \in E'$ should be chosen so that it leads to the maximum decrease in the shortest path distance between s and t.

44. Given a binary tree with all the relevant pointers (child/parent), describe a simple algorithm, running in linear time, that can compute and store the size of the subtrees at each node in the tree. (Size of a subtree at a node v is the total number of nodes, including itself, in the subtree rooted at v.) Justify the correctness and prove that that the algorithm runs in linear time.

45. Outline a search algorithm, running in $O(\log n)$ time, to report the i-th smallest number in a set consisting of n elements. The set is represented as a red-black tree where in addition to the usual information that we store at a node (pointer to left child, right child, parent, key, colour) we also store the size of the subtree at that node. The parameter i is supplied to the search algorithm at the run-time and assume that the red-black tree with the additional information about the size of the subtrees has been precomputed.

46. Assume that we have n-integers in the range 1000 to 9999. In the radix-sort method, we sort them by first sorting them using the (stable) counting sort by the Least-Significant digit and then the next least significant digit and so on.
Why does this algorithm work? Sketch the main idea in the proof. Why does the algorithm fail when we first sort them using the most-significant bit, the second most significant bit, and so on?

47. Assume that we are given n intervals (possibly overlapping) on a line. Each interval is specified by its left and right end points. Devise an efficient algorithm that can find a point on the line which is contained in the maximum number of intervals. Your algorithm should run in in $O(n \log n)$ time. Show that this problem has $\Omega(n \log n)$ lower bound. What is the model of computation you want to use for showing the lower bound?

48. Devise an $O(n \log k)$ time algorithm to merge k-sorted lists into a single sorted list, where n is the total number of elements in all input lists.

49. Suppose all edge weights are positive integers in the range $1..|V|$ in a connected graph $G = (V, E)$. Devise an algorithm for computing Minimum Spanning Tree of G whose running time is better than that of Kruskal's or Prim's algorithm.

50. Consider a connected graph $G = (V, E)$ where each edge has a non-zero weight. Furthermore, assume that all edge weights are distinct. Show that for each vertex $v \in V$, the edge incident to v with minimum weight belongs to a Minimum Spanning Tree. Can you use this to devise an algorithm for MST - the above step identifies at least $|V|/2$ edges in MST - you can collapse these edges, by identifying the vertices and then recursively apply the same technique - the graph in the next step has at most half of the vertices that you started with - and so on. What is the running time of your algorithm?

51. Suppose you are given n-points in the plane. We can define a complete graph on these points, where the weight of an edge $e = (u, v)$ is the Euclidean distance between u and v. We need to partition these points into k non-empty clusters, for some $n > k > 0$. The property that this clustering should satisfy is that the minimum distance between any two clusters is maximized. (The distance between two clusters A and B is defined to be the minimum among the distances between pair of points, where one point is from cluster A and the other from cluster B.) Show that the connected components obtained after running Kruskal's

algorithm till it finds all but the last $k-1$ (most expensive) edges of MST produce an optimal clustering.

52. Although the 3CNF-SAT is NP-Complete, show that in polynomial time we can determine whether a boolean formula given in the disjunctive normal form is satisfiable; the formula consists of n variables and k clauses. A formula is in Disjunctive normal form if clauses are joined by ORs and literals within a clause are joined by ANDs (DNF is OR of ANDs and CNF is AND of ORs). You need to provide an algorithm and show that it is correct and runs in polynomial time with respect to n and k.

53. Given an integer $m \times n$ matrix A and an integer m-vector b, the $0-1$ integer programming problem asks whether there is an integer n-vector x with elements in the set {0,1} such that $Ax \leq b$. Prove that 0-1 integer programming problem is NP-Complete by providing a reduction from 3CNF-SAT or Subset-Sum problem.

54. Construct an instance of the subset-sum problem corresponding to the following 3CNF-SAT:

    $$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

    Provide a satisfying assignment to 3CNF-SAT and show that it provides a valid solution for the subset-sum problem.

55. For the same 3CNF-SAT, illustrate the reduction to the clique problem. Construct an equivalent graph and show that you have an appropriate size clique and vice verse for a satisfying assignment.

56. Consider the problem of CNF-Satisfiability (we are not restricting the number of literals in each clause), where each variable only occurs at most twice (positive and negative forms of a variable are not counted separately). Show that satisfiability can be determined in polynomial time in this case.

    Hint: If a variable only occurs in its positive (or negative) form, then we can satisfy those clauses and remove them from consideration. The problem starts when we have both versions of the variable (one clause containing it in positive form and the other in the negative form - but then the following resolution rule applies: if we have two clauses of the form $(x \vee w \vee y \vee z) \wedge (\neg x \vee u \vee y \vee z)$, then this is satisfiable if and only if the clause $(y \vee z \vee u \vee w)$ is satisfiable. In other words, we can remove the variable x from consideration!)

57. You have invited 500 guests for your graduation party in a huge hall in Chateau Laurier. The guests need to be seated where each

table has 20 seats. Unfortunately, the guests are not all friendly with each other; for sure, you want to avoid two of your guests sitting at the same table if they are not friendly. Suppose you know the complete friendship matrix F (a 0-1 matrix indicating whether a pair of guests (i, j) are friendly or not). Can you devise a decision algorithm to decide whether it is possible to hold this party with the restriction of 20 per table and no two enemies land on the same table? What about, in general, where n is the number of guests, and k is the number of guests per table, and we can assume k divides n? Is this problem NP-Complete?

58. Suppose you are given a sorted array $A[1...\infty]$, in ascending order, of infinitely many real numbers and a real number x. Show how you will perform the following operations:

    (a) How to find an index i, such that $A[i] \geq x$ in $O(\log i)$ time.

    (b) How to find an index i such that $A[i] \geq x$ in $O(\log \log i)$ time.

59. Consider two sets A and B, each having n integers in the range from 0 to cn, where $c > 1$ is a constant. Define the Cartesian sum of A and B as the set C given by $C = \{x + y : x \in A \text{ and } y \in B\}$. Note that the integers in C are in the range 0 to $2cn$. We want to find all the elements of C and the number of times each element of C is realized as a sum of elements in A and B. Provide an $O(n \log n)$ algorithm for this problem.

60. Given an undirected graph $G = (V, E)$, each vertex $v \in V$ has an associated positive weight $w(v)$. For any vertex cover $V' \subseteq V$, define the weight of the vertex cover $w(V') = \sum_{v \in V'} w(v)$. The goal is to find a vertex cover of minimum weight. Provide an Integer Linear Programming formulation for this problem. Then provide a relaxation of the integer program. Show that we can obtain a 2-approximation algorithm for vertex cover using the rounding technique. Provide a formal proof to show that your solution results in a 2-approximation.

61. Given a simple graph $G = (V, E)$, we define a cut to be a partition of the vertex set V into two non-empty sets A and B, where $A \cup B = V$ and $A \cap B = \emptyset$. An edge $(a, b) \in E$ is said to cross the cut if $a \in A$ and $b \in B$. The size of the cut corresponding to the partition (A, B) is defined as the number of edges crossing the cut. The maximum cut problem is to find a partition of V such that the size of the cut is maximized. Consider the following algorithm:
    Step 1: Find any partition of V.
    Step 2: For every vertex $v \in V$, if v would have more edges crossing the cut if placed in the opposite partition, then move v to the opposite partition.

Prove the following

(a) Prove that the above algorithm runs in polynomial time. What is the running time?

(b) Prove that the size of the cut produced by the above algorithm is at least half of the size of the maximum cut. (In other words, it's a 1/2-approximation algorithm.)

62. Although the 3CNF-SAT is $\mathcal{NP}$-Complete, show that in polynomial time we can determine whether a boolean formula given in disjunctive normal form is satisfiable. The formula consists of n variables and k clauses. (A formula is in Disjunctive normal form, if clauses are joined by ORs and literals within a clause are joined by ANDs). You need to provide an algorithm whose running time is polynomial in n and k.
Show what is wrong with the following argument: Given a 3CNF-SAT, we can use distributive law to construct an equivalent formula in Disjunctive Normal Form. Here is an example:
$(x_1 \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_1} \vee \bar{x_2}) = (x_1 \wedge \bar{x_1}) \vee (x_1 \wedge \bar{x_2}) \vee (x_2 \wedge \bar{x_1}) \vee (x_2 \wedge \bar{x_2}) \vee (\bar{x_3} \wedge \bar{x_1}) \vee (\bar{x_3} \wedge \bar{x_2})$.
We have just now shown that DNF is in $\mathcal{P}$. Does this imply that the 3CNF-SAT is in $\mathcal{P}$?

63. Given a sequence of n positive integers $X = (x_1, \cdots, x_n)$, and a positive integer t. Provide an algorithm that answers the following question in $O(nt)$ time: Does a subset of $X' \subseteq X$ exist, such that the sum of the elements in X' equals t?

64. Let $G = (V, E)$ be an undirected simple graph. A set of vertices $S \subseteq V$ is said to be an independent set, if there is no edge between any pair of vertices in S (i.e., if $e = (uv) \in E$ then the proposition ($u \in S$ and $v \in S$) is false). If G is a tree, design an algorithm, running in polynomial time, that outputs the size of the largest independent set in G. Next, let us assume that each vertex of the tree is associated with a positive weight. Now, design an algorithm running in polynomial time, which finds a naximum weight independent set of the tree. Weight of a set of vertices is the sum total of the weights of the vertices forming this set.

65. Prove that the solution of the following Linear Program will result in evaluating a Minimum Spanning Tree T of a connected, weighted, undirected graph $G = (V, E)$. Let w_{uv} denote the weight of an edge $e = (uv) \in E$, and an indicator variable X_{uv} denote the presence or absence of an edge $e = (uv) \in E$ in T. If $X_{uv} = 1$ then $(uv) \in T$, and if $X_{uv} = 0$ then $(uv) \notin T$. Here is the Linear Program:

$$\text{Minimize} \sum_{uv \in E} w_{uv} X_{uv}$$

$$\text{where}, \forall uv \in E, 0 \leq X_{uv} \leq 1$$

$$\sum_{uv \in E} X_{uv} \geq |V| - 1$$

$$\forall S \subseteq V, \sum_{uv \in E, u \in S, v \in S} X_{uv} \leq |S| - 1.$$

Think in terms of what is meaning of each of these constraints. What conditions force X_e to take only 0-1 values and not the fractional ones?

66. In the above formulation of the Minimum Spanning Tree, can you estimate how many constraints will be required? Should one use linear programming formulation to compute a Minimum Spanning Tree of a graph consisting of 1000 vertices?

67. Given an undirected connected unweighted graph $G = (V, E)$, with two specified vertices s and t, where $s, t \in V$. Write a linear program that computes the length of the shortest path (i.e., with respect to the number of segments in the path) between s and t. Give some reasoning why your Linear Program finds the shortest distance. Also, estimate how many constraints you need.

68. Look at the A.M. Turing Award winners list - this is like the Nobel prize in CS. Identify at least three award winners who have worked in the field of Algorithms and/or Data Structures. Give some reasoning for your choice. List two of their main contributions (i.e., publications) for each. State in your own words what is their main contribution.

69. Prove that the expected running time of finding the closest pair among a set of n-ponts in the plane using the randomized incremental construction is $O(n)$.

70. Given a well-separated pair decomposition for a set of n-points in plane,

    (a) : How to determine the closest pair of points in $O(n)$ time.

    (b) Show how you can find an approximation to the diameter of the point set in $O(n)$ time. What kind of approximation factor it will be in terms of the separation parameter s. (Diameter is the largest distance among the pairs.)

71. Let P be a set of n-points in the plane. Define the complete graph $G = (V, E)$ as the graph where $V = P$ and an edge $e = (uv)$

between each pair of points $u, v \in P$. The weight of e is the Euclidean distance between u and v. Euclidean minimum spanning tree (EMST) of P is defined as the minimum spanning tree of G. Show that EMST can be approximated in $O(n \log n)$ time using the well-separated pair decomposition. Note that G has $\Omega(n^2)$ edges; hence, we cannot directly apply any of the minimum spanning tree algorithms.

72. Show that the Jaccard Distance which is defined as $1-$ the Jaccard Similarity between the two sets is a metric.

73. (a) Prove that a matching is maximum if and only if there are no augmenting paths with respect to that matching.
(b) Prove that a bipartite graph $G = (V = A \cup B, E)$ has a perfect matching if and only if for any subset $S \subseteq A$ the number of vertices adjacent to S in B (denote it by $N(S)$) must be as large as $|S|$ (i.e. $|N(S)| \geq |S|, \forall S \in A$).

74. Present a proof, in your own words, of the Isolating Lemma that is used in the PRAM parallel algorithm for maximum matching. Where is it required in the parallel algorithm for maximum matching?

75. When applying amplification constructions to a locality-sensitive family of functions, we can apply an AND composition followed by ORs or vice-versa. Which order of the composition is 'better', and why? Explain when you would apply AND followed by ORs, and when will you like to use ORs followed by ANDs.

76. Let C be a circle and V be a set of n distinct vertices on its boundary. Form a maximal plane graph on V (i.e., we connect as many pairs of vertices as possible by straight line segments so that no two edges cross each other in their interior). Notice that we obtain a plane triangulation of V. Call this triangulation X. Show that X has a treewidth of 2, and its tree decomposition can be computed in polynomial time.

77. Let $X = \{x_1, x_2, \cdots, x_n\}$ be a set of n-elements. Each element of x_i has a positive weight $w_i > 0$. Let $\mathcal{Y} = \{Y_1, Y_2, \cdots, Y_m\}$ be a set of subsets of X (i.e. each $Y_i \subset X$). A subset $H \subseteq X$ is called *nice* if $H \cap Y_i \neq \emptyset$, for $i = 1, \cdots, m$. The decision problem of finding a nice set H of weight at most W is NP-Hard. Let W^* be the weight of the nice set with the smallest possible weight. Let $\gamma = \max\{|Y_i|, i = 1, \cdots, m\}$. Provide an approximation algorithm, running in polynomial time, that computes a nice set whose weight is at most γW^*.

78. Given a set P of points the plane. For each point $p \in P$ we denote by $b(p)$ the maximum Euclidean distance between p and any point in P, i.e., $b(p) = \max\{|pq| : q \in P\}$. We denote a point $p \in P$ with minimum $b(p)$ as the *center* of P. Let p be the center of P. Present a constant time algorithm that finds a point p' in P such that $b(p') \leq 2 \cdot b(p)$. Analyze the running time and prove the correctness of your algorithm.

79. Suppose you have a set S of n-points in the plane. The task is to construct an approximate traveling salesperson tour $TSP(S)$ of S. All distances are measured with respect to Euclidean distance. We follow the following strategy. Choose any point $s \in S$, and initialize a trivial tour $T = \langle ss \rangle$. Now we will grow this tour. Find a point $v \in S \setminus \{s\}$ closest to s, and update the tour to include v and the current tour becomes $T = \langle svs \rangle$. Suppose the tour currently consists of $k+1$ vertices $T = \langle su_1u_2...u_ks \rangle$. Now find a vertex in $v \in S$ that is closest to (but distinct from) $s, u_1, u_2, ..., u_k$. Let v be the closest vertex to $u \in \{s, u_1, u_2, ..., u_k\}$. The tour is updated by inserting v after u in T. (For example, if v was closest to u_3, than the new tour will be $T = \langle su_1u_2u_3vu_4...u_ks \rangle$.) We repeat this process till all the points in S are added to T. Show that the cost of T is at most twice the cost of an optimal tour. (Note that the cost of a tour is the sum total of the costs of all the edges in the tour.)

80. Consider a utility matrix M where the rows represent users and the columns represent items. The singular value decomposition of $M = U\Sigma V^T$ (approximated to two decimal places) is given as follows.

$$M = \begin{bmatrix} 2 & 0 & 2 \\ 1 & 1 & 0 \\ 1 & 2 & 3 \\ 2 & 4 & 4 \\ 2 & 5 & 5 \end{bmatrix} \approx \begin{bmatrix} .19 & .95 & -.06 \\ .09 & -.01 & -.87 \\ .35 & .11 & .44 \\ .57 & -.08 & -.18 \\ .70 & -.25 & .05 \end{bmatrix} \begin{bmatrix} 10.41 & 0 & 0 \\ 0 & 2.04 & 0 \\ 0 & 0 & 1.18 \end{bmatrix} \begin{bmatrix} .32 & .63 & .69 \\ .65 & -.68 & .31 \\ -.68 & -.35 & .64 \end{bmatrix}$$

Answer the following:

(a) What is the rank of matrix M?

(b) How many concepts are there among the items of M?

(c) For a query user $u = [0, 1, 0]$, what will be its representation in the concept space?

(d) What will be the best rank one approximation of M (note that the resulting U will have one column, and V^T will have one row).

(e) How much energy will be lost in the approximation of M of part (d).

81. Suppose you have three advertisers, A_1, A_2, and A_3, each with a budget of B dollars. Advertiser A_1 bids only for items of type a, A_2 bids for items of types a and b, and A_3 bids for items of types a, b, and c. Assume that the click-through rate is \$1 for each of the advertisers. Assume that the online sequence of $3B$ queries consists of a random permutation of B queries for a, B queries for b, and B queries for c. Is it true that the competitive ratio of the Balance algorithm is at least $2/3$? (Note that an optimal offline algorithm earns a revenue of $3B$). For example, if $B = 4$, a possible query sequence may be baccacbabbca.

82. Consider a web graph corresponding to a directed cycle of length 3. For example, let a, b, c be the set of three vertices, and then the directed edges are (a, b), (b, c), and (c, a). What will each page rank be when we use the teleportation probability $\beta = 0.2$?

83. Suppose you want to rent an apartment in Old Ottawa South and have hired an agent to show all the possible apartments within your budget over the following weekend. Suppose the agent wants to show you n apartments and tells you that as soon as you make an offer to any of them, it will be accepted. As a computer scientist, you compute a random permutation of the order in which you want to see these apartments and let the permuted order be $\pi_1, \pi_2, \pi_3, \ldots, \pi_n$. You tell the agent that on Saturday, you will see the first $\frac{n}{e}$ of these, i.e., the apartments labeled $\pi_1, \pi_2, \ldots, \pi_{\frac{n}{e}}$. (We assume here that e divides n.) After viewing each of these apartments, you make some mental notes, but at the end of the day Saturday, you tell the agent that you would like to see the remaining ones in the order of the permutation $\pi_{\frac{n}{e}+1}, \ldots, \pi_n$, on Sunday. The strategy you have decided to employ on Sunday is to make an offer to rent the first apartment you see, which you think is better than what you have seen so far (including what you saw on Saturday), and then terminate your visit to any remaining unvisited apartments. On your Sunday's apartment hunting venture with the agent, you make an offer for apartment π_α. (We may not find any better apartment on Sunday and hence may not make any offer - to keep the arguments simple, if you prefer, you may assume that you make an offer.) Answer the following questions:

    (a) Suppose you would have seen all the apartments, then your ranking of the apartments to rent (from highest to lowest), without loss of generality, be $1, 2, 3, \ldots, n$ (i.e., 1 is best, 2 is second best, ..., n is the worst). Note that the order you visit the apartments is a random permutation of $\{1, 2, \ldots, n\}$. Suppose the smallest apartment number, i.e., the most preferred, among

$\pi_1, \pi_2, \ldots, \pi_{\frac{n}{e}}$ be x. Show that $\pi_\alpha < x$.

(b) Show that the probability of making the optimal choice in the above strategy is given by
$\sum_{i=\frac{n}{e}+1}^{n} Pr[\text{We see the apartment } \pi_i \text{ and } \pi_i = 1]$.

(c) Show that the probability of making the optimal choice can also be expressed as
$\sum_{i=\frac{n}{e}+1}^{n} Pr[\pi_i = 1 \text{ and minimum of } \{\pi_1, \pi_2, \ldots, \pi_{i-1}\} \text{ is in } \{\pi_1, \pi_2, \ldots, \pi_{\frac{n}{e}}\}]$.

(d) Show that the probability of making an optimal choice simplifies to $\sum_{i=\frac{n}{e}+1}^{n} \frac{1}{n} \times \frac{\frac{n}{e}}{i-1}$. Conclude by showing that the probability that the above strategy selects the best apartment is approximately $1/e = 37\%$. (Recall that the n-th Harmonic number $\sum_{i=1}^{n} \frac{1}{i} \approx \ln n$.)

84. Suppose a service company dispatches people to service the equipment at various sites. The company has its base in Vancouver with two employees, but it can get service requests from Ottawa, Toronto, or Vancouver. Each of its employees is paid an allowance proportional to their travel distance, which is extra to their salary. Each evening the company receives a maximum of two service requests for the next day, and it needs to decide which employee should serve which requests. The company wants to minimize the total allowance paid to its employees. (For example, if one of the employees is already in Ottawa, and is assigned to serve a call from Ottawa for the next day, then there is no additional allowance that needs to be paid to this employee for the next day.) We can make some (unrealistic) assumptions like the travel time is zero, each employee can only handle one service call in a day, a service request can be finished within the same day by either of the employees, both employees are equally capable, and employee can be made to stay overnight in any of the locations for free.) The distance between Ottawa-Vancouver is 3540 km, Ottawa-Toronto is 350 km, and Toronto-Vancouver is 3400 km. (If you need to, you can make more assumptions, but please state them.) Design an online scheduling strategy. Compare your strategy against an optimal offline schedule and figure out what may be your worst-case competitive ratio. For example, you may try to see your competitive ratio for 10 days, 20 days,...

85. Let $G = (V, E)$ be an undirected connected graph without any cycles, where V is the set of vertices and E is the set of edges. Show that, in polynomial time, you can find a set of vertices $V' \subseteq$

V of minimum cardinality, such that for each edge $e = (uv) \in E$, at least one of u or v is in V'.

86. Let $G = (V, E)$ be an undirected connected graph. Let $S \subseteq V$ be the largest subset of vertices such that there is no edge $e = (u, v) \in E$ such that $u, v \in S$. Given S, show that we can construct a minimum vertex cover of G in polynomial time. What can you say about the complexity of finding such a subset S?

87. Suppose a warehouse in Mississauga packages all the items in boxes that must be shipped to a distributor in Ottawa. For shipping, each box needs to weigh $\leq W$, where W is a positive integer. We can fit as many items as we want in a box provided that the sum total of their weights is at most W. Moreover, each item I_k, $k \in \{1, \ldots, n\}$, has weight w_k, where $0 < w_k \leq W$. The cost of shipping is proportional to the number of boxes used. The strategy the warehouse employs to package these items in boxes is as follows:

    (a) Set $k = 1$.
    (b) Open a new box.
    (c) While I_k can be placed in the open box without exceeding its weight capacity and $k \leq n$,
        i. Place I_k in the open box.
        ii. Set $k := k + 1$.
    (d) Close, Tape, and Ship the box.
    (e) If $k \leq n$, GOTO Step b.

    Show that the warehouse's number of boxes is at most two times the minimum number of boxes required to package all the n items.

88. Suppose you have a collection $|S| + |F|$ parallel machines, where S is the set of (identical) slow machines and F is the set of (identical) fast machines. Assume that the fast machines can undertake twice as much work as the slow machines per unit of time. We have a set of n independent jobs, each with its processing time requirement, that needs to be assigned to these machines. Note that if a job requires t time units for completion, it takes t time units on a slow machine and $\frac{t}{2}$ time units on a fast machine. Any job can be performed on any machine, but a job cannot be split into smaller jobs, and once a job is assigned to a machine, it cannot be moved to another machine. You need to design an algorithm, running in polynomial time, to assign these jobs to these parallel machines so that the makespan is within three times the optimal makespan. Recall that the makespan of a set of parallel machines is the maximum total processing time over all the machines of all the jobs assigned to a machine.

89. Consider the following approximation algorithm for the metric-TSP problem on $G = (V, E)$. Let $n = |V|$.

    (a) Start from any vertex $v \in V$ and initialize $H = \langle v \rangle$.

    (b) While $|H| < n$ do

        i. Find a vertex $u \in V \setminus H$ whose distance to any vertex in H is minimum. Let the nearest vertex to u in H be w.

        ii. Modify H by inserting u immediately after w in H. (For example, if before the execution of this step $H = \langle vabcwxyz \rangle$, then at the conclusion of this step $H = \langle vabcwuxyz >$.)

    (c) Without loss of generality, let $H = \langle v_1 = v, v_2, \ldots, v_n \rangle$. Construct $T = \{v_1v_2, v_2v_3, \ldots, v_{n-1}v_n, v_nv_1\}$.

    (d) Report T.

    Show that T is a TSP-tour in G. Show that its cost is at most twice the cost of an optimal TSP tour.

90. Formulate the set cover problem as an integer linear program.

91. Consider the following instance of the set cover problem. We have a universe B consisting of elements and a collection $\mathcal{S}$ of subsets of B with the property that each element of B is contained in exactly three sets in $\mathcal{S}$. Consider the following algorithm:

    (a) $\mathcal{C} := \emptyset$, $U := B$.

    (b) While $U \neq \emptyset$ do

        i. Pick any element $u \in U$ and find the three sets in $\mathcal{S}$ that contain u. Without loss of generality, let these sets be S_1, S_2, S_3.

        ii. $\mathcal{C} := \mathcal{C} \cup \{S_1\} \cup \{S_2\} \cup \{S_3\}$.

        iii. $U := U \setminus \{S_1 \cup S_2 \cup S_3\}$.

    (c) Report $\mathcal{C}$.

    Note that $\mathcal{C}$ is the set of subsets in the cover reported by the algorithm. Show that $\mathcal{C}$ is a cover and the number of subsets reported by the algorithm (i.e., $|\mathcal{C}|$) is within three times the size of an optimal cover.

92. Let us look at the following variant of the independent set of squares problem. Let S be a collection of n axis-aligned squares. We are interested in finding an independent set of S that maximizes the total area of the squares in it. (Recall that two squares are independent if they do not share a point in their interior.) Consider the following algorithm:

    (a) Initialize $I := \emptyset$.

(b) Take the largest square s from S and add it to I.

(c) Remove all squares from S that overlap s.

(d) Iterate Steps b-c until S is empty.

(e) Report I.

Show that the total areas of the squares in I is at least $\frac{1}{9}$th of the area of an optimal solution (i.e. $1 \leq \frac{area(OPT)}{area(I)} \leq 9$).

93. Let $G = (V, E)$ be a simple undirected graph where the degree of each vertex is at most 15. We say that a set $V' \subseteq V$ is an independent set in G if for any pair of vertices $u, v \in V'$, $uv \notin E$. Design an algorithm, running in polynomial time, that computes an independent set of G whose size is at least $\frac{1}{15}$th of the size of the largest independent set.

94. Suppose we have a set $X = S \cup R$ of n-points in the plane, where set S is called the set of Steiner points and the set R is called the set of terminals. Our task is to design the *least cost network* $LCN(X, R)$ (e.g., a tree), which connects all points in R, and it may or may not use any points from S. Note that points in R (and whatever points in S that are used) are the vertices of $LCN(X, R)$. The cost of a network is the sum total of the lengths of all the segments in that network. Construct an example to show that points in S may not be required to form $LCN(X, R)$. Construct an example to show that at least one or more points in S are required to form $LCN(X, R)$. In general, the decision version of this problem is NP-Hard. Show that the minimum-spanning tree of points in R is a 2-approximation to the $LCN(X, R)$ problem, i.e., the cost of $MST(R)$ is at most twice the cost of $LCN(X, R)$.

95. You are given n-processes $P_1, ..., P_n$, where n is a large integer. These processes are trying to access a single shared database. We assume that time is divided into discrete rounds. In a single round, the database can only be accessed by exactly one process. If two or more try to access the database simultaneously in a round, all the processes are locked out for that round. Here is an algorithm that is employed to access the database.
In each round t, $t = 1, ..., T$, where T is a large integer, do the following

(a) Each process P_i, $1 \leq i \leq n$, flips a coin, where probability of obtaining heads is $1/n$ and probability of obtaining tails is $1 - 1/n$.

(b) If there is exactly one process (say P_i) whose coin flip is 'heads', then P_i is allowed to access the database in this round.

Answer the following:

(a) Let A_{it} be the event that the process P_i accesses the database in round t. What is $Pr(A_{it})$?

(b) Show that the probability that the process P_i does not get access to the database in any of the rounds is at most $e^{-\frac{T}{en}}$.

(c) Let $T = 2en \ln n$ and let E be the event that each process P_i accesses the database at least once during the rounds $1, 2, ..., T$. Show that $Pr(E) \geq 1 - 1/n$.

Recall that $e^{-x} \geq 1 - x$ and for large n, $(1 - 1/n)^{n-1} \geq 1/e$.

96. This exercise will look at Gabow's Scaling Algorithm for Shortest Paths. Let $G = (V, E)$ be an undirected connected graph where each edge $e \in E$ has an integer positive weight $w(e)$, where $w : E \to [1, \ldots, W]$. Let $k = \lceil \log(W + 1) \rceil$ be the number of bits in the binary representation of W. For an edge e, for all $1 \leq i \leq k$, define $w_i(e) = \left\lfloor \frac{w(e)}{2^{k-i}} \right\rfloor$. It is the weight of the edge defined by the most-significant i bits. For example, if $w(e) = 11 = (01011)_2$, than $w_4(e) = \left\lfloor \frac{w(e)}{2^{k-i}} \right\rfloor = \left\lfloor \frac{11}{2^{5-4}} \right\rfloor = 5 = (0101)_2$. Similarly, $w_3(e) = \left\lfloor \frac{11}{2^{5-3}} \right\rfloor = 2 = (010)_2$, and $w_2(e) = \left\lfloor \frac{11}{2^{5-2}} \right\rfloor = 1 = (01)_2$. Let $\delta(s, v)$ denote the length of shortest path in G between s and v, where $s, v \in V$. Define $\delta_i(s, v)$ as the shortest path length between s and v when the weight of each edge e is restricted to be $w_i(e)$. Note that $\delta(s, v) = \delta_k(s, v)$. This exercise asks you to design an algorithm running in $O(|E| \log W)$ time that computes shortest path distance $\delta(s, v)$ from a vertex $s \in V$ to every other vertex $v \in V$ by answering the following:

(a) Show that if the keys in the priority queue are integers in the range $1, \ldots, |E|$, then after initializing the queue in $O(|E|)$ time, each decrease-key operation can be implemented in $O(1)$ time. Moreover, show that an intermix of $O(|V|)$ extract-min operations and $O(|E|)$ decrease-key operations can be implemented in a total of $O(|E|)$ time.

(b) Show that if $\delta(s, v) \leq |E|$ for each vertex $v \in V$, we can determine all the $\delta(s, v)$'s in $O(|E|)$ time.

(c) Show that for each e, $w_1(e) = \left\lfloor \frac{w(e)}{2^{k-1}} \right\rfloor \in \{0, 1\}$. Moreover, show that $\delta_1(s, v)$ for each vertex v can be computed in $O(|E|)$ time.

(d) For $i = 2, \ldots, k$, show that for any edge e, $w_i(e) \in \{2w_{i-1}(e), 2w_{i-1}(e) + 1\}$.

(e) For $i = 2, \ldots, k$, show that $\delta_i(s, v) \geq 2\delta_{i-1}(s, v)$. Furthermore, show that $\delta_i(s, v) \leq 2\delta_{i-1}(s, v) + |V| - 1$.

(f) For $i = 2, \ldots, k$, and for any edge $e = (u, v) \in E$, define the i-th scaled weight $\hat{w}_i(uv) = w_i(u, v) + 2\delta_{i-1}(s, u) - 2\delta_{i-1}(s, v)$. Show, using the triangle inequality and the fact that $w_i(u, v) \geq 2w_{i-1}(u, v)$, $\hat{w}_i(uv) \geq 0$.

(g) For $i = 2, \ldots, k$, for any vertex $v \in V$, define i-th scaled shortest path $\hat{\delta}_i(s, v)$ to be the shortest path from s to v with respect to the weights $\hat{w}_i(uv)$. Using the definition of $\hat{w}_i(uv)$, show that $\delta_i(s, v) = \hat{\delta}_i(s, v) + 2\delta_{i-1}(s, v)$.

(h) Show that $\hat{\delta}_i(s, v) \leq |E|$ for each vertex $v \in V$.

(i) For all vertices $v \in V$, show that given $\delta_{i-1}(s, v)$, $\delta_i(s, v)$ can be computed in $O(|E|)$ time.

(j) Conclude that shortest path $\delta(s, v) = \delta_k(s, v)$ for all vertices $v \in V$ can be computed in $O(|E| \log W)$ time.

97. (Lovasz [1]) Let $G = (V, E)$ be a simple undirected graph such that the maximum degree of any vertex is at most k. By a partition of vertex set V into two subsets V_1 and V_2, we mean that $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$. We refer to the induced graph on the vertices of V_1 by $G_1 = (V_1, E_1)$. Note that $E_1 = \{e = uv \in E | u, v \in V_1\}$.

[1] László Lovász. On decomposition of graphs. In *Studia Scientiarum Mathematicarum Hungarica 1*, pages 237–238, 1966

(a) Show that if $k = 1$, then V can be partitioned in two disjoint sets of vertices V_1 and V_2 such that the degree of each vertex in the induced graph on V_1 is 0 and the degree of each vertex on the induced graph on V_2 is 0.

(b) Assume $k \geq 2$. Let k_1 and k_2 be two positive integers such that $k_1 + k_2 = k - 1$. We will show that there is a partition of vertex set V into two disjoint sets V_1 and V_2 such that vertices in the induced graph $G_1 = (V_1, E_1)$ has degree at most k_1 and the vertices in the induced graph $G_2 = (V_2, E_2)$ has degree at most k_2. Consider the partition of $V = V_1 \cup V_2$ that minimizes the quantity $\Gamma = k_1(2|E_2| - |V_2|) + k_2(2|E_1| - |V_1|)$. Answer the following.

i. Consider an arbitrary vertex $v \in V_1$. Assume it has a neighbors in G_1 and b neighbors in G_2 with respect to G. Show that $a + b \leq k$.

ii. Show that if we move v from V_1 to V_2, the value of Γ changes by $k_1(2b - 1) - k_2(2a - 1)$.

iii. Show that $k_1(2b - 1) - k_2(2a - 1) \geq 0$.

iv. Using $a + b \leq k$, $k_1 + k_2 + 1 = k$, and $k_1(2b - 1) - k_2(2a - 1) \geq 0$, show that $a \leq k_1 + 1/2$.

v. Conclude that degree of each vertex $v \in V_1$ is at most k_1.

vi. Consider an arbitrary vertex $w \in V_2$ and let it has x neighbors in G_2. Using a similar argument show that $x \leq k_2 + 1/2$. Conclude that degree of each vertex $w \in V_2$ is at most k_2.

(c) Let α be a positive integer. Let $k_1, k_2, \ldots, k_\alpha$ be non-negative numbers such that $k_1 + k_2 + \cdots + k_\alpha = k - \alpha + 1$. We will show that V can be partitioned into disjoint subsets $V = V_1 \cup \cdots \cup V_\alpha$ such that the induced graphs on $V_1, \ldots, V_\alpha$ have degrees at most $k_1, \ldots, k_\alpha$, respectively. Let us denote the induced graphs on $V_1, \ldots, V_\alpha$ by $G_1, \ldots, G_\alpha$, respectively. The proof is by induction on α and can be derived by solving the following exercises:

i. (Base Case) When $\alpha = 1$, show that the result holds trivially. Assume $\alpha = 2$, and given any choice of positive integers k_1 and k_2, such that $k_1 + k_2 = k - 1$. Using the previous exercise, show that we can always partition V into two parts V_1 and V_2 such that the induced graph on V_1 has degree at most k_1 and the induced graph on V_2 has degree at most k_2.

ii. Formulate an induction hypothesis.

iii. Inductive Step. Answer the following.

- Assume $\alpha \geq 3$. Let us define $k'_1 = k_1$ and $k'_2 = k_2 + \cdots + k_\alpha + \alpha - 2$. Show that $k'_1 + k'_2 = k - 1$ and V can be partitioned in two sets V'_1 and V'_2 such that degree of each vertex in the induced graph on V'_1 is at most k'_1 and the degree of each vertex in the induced graph on V'_2 is at most k'_2.
- Show that $k_2 + \cdots + k_\alpha = k'_2 - (\alpha - 1) + 1$.
- Show that by induction hypothesis V'_2 can be partitioned into $V_2, V_3, \ldots, V_\alpha$ such that the induced graph on V_i has degree k_i, where $2 \leq i \leq \alpha$.
- Conclude that Lovasz result on partitioning the graph of degree at most k into subgraphs of desired degrees holds.

98. (Hochanbaum [2]) In this exercise we will find an approximation to the maximum weight independent set in a simple undirected graph $G = (V, E)$, where each vertex has degree at most k and it is assigned a positive weight. Answer the following questions.

(a) Show that the maximum weight independent set of an undirected graph where degree of each vertex is at most 2 can be computed in polynomial time.

(b) Let $G = (V, E)$ be a simple graph on n vertices and assume that the degree of each vertex is $\leq k$. Let $\alpha = k/3$. Show that V can be partitioned into $V = V_1 \cup V_2 \cup \cdots \cup V_\alpha$ such the vertices in the induced graph $G_i = (V_i, E_i)$ for $1 \leq i \leq \alpha$ have degree ≤ 2.

(c) For each of the induced graph $G_i = (V_i, E_i)$ in the above partitioning compute the maximum weight independent set and let it be $I_i \subseteq V_i$. Among $\{I_1, \ldots, I_\alpha\}$ let I be the set of maximum weight. Let I^* be the maximum weight independent set of G. Show that weight of I is at least $\frac{3}{k}$ times the weight of I^*.

[2] Dorit Hochbaum. *Approximation Algorithms for NP-Hard Problems*. PWS, 1997

(d) Show that V can be partitioned into $V = V_1 \cup V_2 \cup \cdots \cup V_\alpha$ such the vertices in the induced graph $G_i = (V_i, E_i)$ for $1 \leq i \leq \alpha$ have degree ≤ 2 in polynomial time. See how to implement the inductive proof of the previous exercise in a constructive manner in polynomial time.

(e) Conclude that we can obtain a $\frac{3}{k}$-approximation of maximum weight independent set in a graph whose degree is bounded by k in polynomial time.

99. (Lovasz [3]) Let $G = (V, E)$ be a simple undirected graph such that $n = |V|$ and degree of each vertex is $\leq k$. Let n_1 and n_2 be two positive numbers such that $n_1 + n_2 = n$. In this exercise we will show that there exists a partition of set V into subsets V_1 and V_2 such that $n_1 = |V_1|$ and $n_2 = |V_2|$ that satisfies the following property. Let k_1 be the maximum degree among the vertices in the induced graph $G_1 = (V_1, E_1)$ and let k_2 be the maximum degree among the vertices in $G_2 = (V_2, E_2)$. We will show that $k_1 + k_2 \leq k$ by the following exercises.

[3] László Lovász. On decomposition of graphs. In *Studia Scientiarum Mathematicarum Hungarica 1*, pages 237–238, 1966

(a) Consider that partition of $V = V_1 \cup V_2$, where $n_1 = |V_1|$ and $n_2 = |V_2|$, and it minimizes the number of edges in E_1. Let $v \in V_1$ and $w \in V_2$ be the vertices of maximum degree in V_1 and V_2, respectively. Assume that w has k_2' neighbors in V_1 in G. Show that $k \geq k_2 + k_2'$.

(b) Assume we construct $V_1' = V_1 \cup \{w\} \setminus \{v\}$ and $V_2' = V_2 \cup \{v\} \setminus \{w\}$. Show that $|V_1'| = |V_1|$ and $|V_2'| = |V_2|$. Show that the edge set E_1' of the induced graph of V_1' satisfies $|E_1'| = |E_1| + k_2' - k_1$.

(c) Show that $k_2' - k_1 \geq 0$ and conclude that $k_1 + k_2 \leq k$.

100. Let $G = (V, E)$ be a simple undirected bipartite graph. Let A be its node-edge incidence matrix of dimension $|V| \times |E|$, defined as

$$A_{ve} = \begin{cases} 1, \text{ if } v \text{ is an endpoint of an edge } e \\ 0, \text{ otherwise} \end{cases}$$

Show that every square submatrix of A has determinant 0 or ± 1.

Note that the rows (respectively, columns) of A corresponds to the vertices (resp., edges) of G. Consider the column corresponding to the edge $e = (uv) \in E$. That column consists of exactly two 1s, one corresponding to the row of u and the other to the row of v, and all the remaining entries are 0.

101.(a) State and prove Euler's relation for planar graphs that relates the number of edges, faces, and vertices.

(b) Consider any planar graph G on n vertices. Are there at least cn vertices in G that have a degree at most 20? Note that $0 < c < 1$ is a constant independent of n.

(c) Consider any planar graph G on n vertices. Let I be a largest independent set in G where each vertex $v \in I$ has degree at most 20 in G. Is it true that $|I| \geq n/1000$?

102.(a) Show that every simple undirected graph on $n \geq 2$ vertices has at least two vertices of equal degree.

(b) Show that a tree with $n \geq 2$ vertices have at least two vertices of degree 1.

103. Let $G = (V, E)$ be a simple, connected, and undirected graph on $n \geq 3$ vertices. Answer the following:

(a) Suppose we are also given that G is not a complete graph. Show that three vertices always exist u, v, and $w \in V$ such that $uv, vw \in E$ and $uw \notin E$.

(b) Show that any two longest paths in G have a vertex in common.

104. Let H be a graph, possibly disconnected. A connected component C of H is odd if C has an odd number of vertices. Let $o(H)$ denote the number of odd components in H. Tutte's theorem states that a graph $G = (V, E)$ has a perfect matching if and only if for every subset $S \subset V$, $o(G - S) \leq |S|$. Prove one of the directions of Tutte's theorem. Apply this theorem to show that every 3-regular graph without cut edges has a perfect matching.

105. Show that every 3-regular bipartite graph has a perfect matching.

106. Show that every k-chromatic graph has at least k vertices of degree at least $k - 1$.

107. Suppose we have a flow network N. It is specified as a directed graph $N = (V, E)$ with (integer) capacities on edges, and two specific vertices $s, t \in V$, called the source and the target, respectively. The flow problem is to find the maximum amount of flow that can be sent from s to t, respecting the capacities on the edges and satisfying the flow conservation properties. Suppose, in addition to the capacities on the edges of N, we also have (non-negative) integer capacities on the vertices V. For example, if a vertex $v \in V$ has a capacity of 5, the maximum amount of flow that can pass through v is 5. Given a flow network N with vertex and edge capacities, can you find a way to transform it to another flow network N' that only has edge capacities, such that the maximum flow between s and t in N is the same as the flow between the corresponding source and target vertices in N'.

108. Let $G = (V, E)$ be a simple undirected graph. A set $S \subseteq V$ is a vertex cover of G, if for each edge $e = (uv)$, $u \in S$ or $v \in S$. Deciding whether a graph has a vertex cover of size k is an NP-Complete problem. Consider the following greedy algorithm:

Step 1: $S := \emptyset$.

Step 2: If G has no edges, return S and STOP.

Step 3: Let v be the vertex with maximum degree in G.
$S := S \cup \{v\}$.
Remove v and all its incident edges from G.

Step 4: Return to Step 2.

Show that the greedy algorithm is not a constant factor approximation algorithm for the vertex cover problem.

Consider a bipartite graph $G = (A \cup B, E)$, where $|A| = k!$, $|B| = k!H_k$, where H_k is the k-th Harmonic number. Vertices in B are grouped in k groups. The 1st group consists of $k!/k$ vertices, 2nd group consists of $k!/(k-1)$ vertices,..., k-th groups consists of $k!/1!$ vertices. Form the edge set of G so that an execution of the greedy algorithm possibly lands up picking all the vertices in the set B, whereas the optimal solution consists of picking only the vertices of A. This will result in an H_k-approximation, as $|B|/|A| = H_k$.

109. Let M be a maximal matching in G. Show that the vertex cover size in G is at least the number of edges in M. Let S be the set of vertices incident on the edges corresponding to a maximal matching M in G. Show that S forms a vertex cover of G. Show that S is a 2-approximation to the minimum vertex cover in G.

110. In this exercise, we will design a $(1+\epsilon)$-approximation algorithm running in $O(2^{\frac{1}{\epsilon}}n)$ time for the partition problem, where $\epsilon > 0$ is a constant. In the partition problem, we are given a set S of n numbers, where $s_1 \geq s_2 \geq \cdots \geq s_n$, and the problem is to partition S into sets A and B, such that $\max(w(A), w(B))$ is minimized, where $w(A) = \sum_{s_i \in A} s_i$. The algorithm is as follows:

Step 1: Set $m = \lceil \frac{1}{\epsilon} \rceil - 1$.

Step 2: Partition $\{s_1, s_2, \cdots, s_m\}$ optimally by looking at 2^m subsets. Let the optimal solution be A and B.

Step 3: For $i := m+1$ to n do
if $w(A) \leq w(B)$, $A = A \cup \{s_i\}$,
else $B = B \cup \{s_i\}$.

Step 4: Return A and B as the partition of S.

Assume that when the algorithm terminates, without loss of generality, $w(A) \geq w(B)$, and let s_k be the last element assigned to A. Answer the following:

(a) Show that if $m \geq n$, we have an optimal solution.

(b) Show that if s_k was assgined to A in Step 2, the sets A and B returned by the algorithm forms an optimal solution.

(c) Assume that s_k was assigned to A in Step 3. Show that in the step when s_k was assigned to A, $w(A) - s_k \leq w(B)$.

(d) Assume that $2z = \sum_{s_i \in S} s_i$. By our assumption, we know that when the algorithm terminated, $w(A) \geq z$, and we are interested in estimating $w(A)/z$. Answer the following:

i. Given that $w(A) - s_k \leq w(B)$. Conclude that $w(A) - s_k \leq 2z - w(A)$, equivalently $2w(A) \leq 2z - s_k$.

ii. Given that $s_1 \geq \cdots \geq s_m \geq \cdots \geq s_k$, show that $s_k \leq \frac{2z}{m+1}$.

iii. Show that $\frac{w(A)}{z} \leq 1 + \frac{1}{m+1} \approx 1 + \epsilon$.

(e) Show that the algorithm's running time is $O(2^m n)$.

(f) Conclude that the above algorithm is a PTAS, i.e. polynomial time approximation scheme.
Note: For a FPTAS, fully polynomial time approximation scheme, see [4].

111. The following scheduling problem is based on a result of Tsitsiklis [5]. Assume that we have n customers $1, \ldots, n$ located at positions $x_1 < \cdots < x_n$ on a horizontal line l, respectively. Each customer i has a deadline d_i, a positive real number, by which it needs to be served. The server is initially located at the position x^* corresponding to the customer i^* on l, where $i^* \in \{1, \ldots, n\}$. We assume that the server can travel to the customers with uniform speed (= 1 unit/min), and the time it takes to serve a customer is negligible. We must determine if the server can serve all the customers respecting their deadlines. The following exercises will help us devise a dynamic programming algorithm running in $O(n^2)$ time for the decision problem.

(a) Show that a necessary condition for any customer $i \in \{1, \ldots, n\}$ to be served is $|x_i - x^*| \leq d_i$.

(b) Construct an example where $i \in \{1, \ldots, n\}$, $|x_i - x^*| \leq d_i$ holds for each customer, but the server can't serve all the customers.

(c) For the rest of the problem, we will assume that $|x_i - x^*| \leq d_i$ for all $i \in \{1, \ldots, n\}$. Let i, j be such that $1 \leq i \leq i^* \leq j \leq n$. Please assume that the server visits the i-th customer at the location x_i for the first time at time t after it has already visited all the customers $i+1, \ldots, j$ respecting their deadlines. If there is a feasible schedule, we define $V^-(i, j)$ as the smallest value of t; otherwise, $V^-(i, j) = +\infty$. Similarly, define $V^+(i, j)$ to be the smallest value of t for which there is a feasible schedule to visit x_j for the first time at time t, given that the customers $i, \ldots, j-1$ have been already visited respecting the deadlines. Show that for any $i < i^*$, $V^-(i, i^*) = x^* - x_i$. Similarly, show that for any $i > i^*$, $V^+(i^*, i) = x_i - x^*$.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 4th edition, 2022

[5] John N. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22(3):263–282, 1992

(d) For any i, j such that $1 \leq i \leq i^* \leq j \leq n$, let
$U^+(i,j) = \min(V^+(i,j-1) + x_j - x_{j-1}, V^-(i,j-1) + x_j - x_i)$.
Show that $V^+(i,j) = \begin{cases} U^+(i,j), \text{ if } U^+(i,j) \leq d_j \\ +\infty, \text{ otherwise} \end{cases}$

(e) Similarly, for any i, j such that $1 \leq i \leq i^* \leq j \leq n$, let
$U^-(i,j) = \min(V^-(i+1,j) + x_{i+1} - x_i, V^+(i+1,j) + x_j - x_i)$.
Show that $V^-(i,j) = \begin{cases} U^-(i,j), \text{ if } U^-(i,j) \leq d_i \\ +\infty, \text{ otherwise} \end{cases}$

(f) Show that if $\min(V^-(i,j), V^+(i,j)) < \infty$, we have a feasible schedule where the server can serve all the customers.

(g) Show that in $O(n^2)$ time, we can determine if there is a feasible schedule.

Bibliography

[1] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003. Special Issue on PODS 2001.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.

[3] L. Aleksandrov and H. Djidjev. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM J. Discret. Math.*, 9(1):129–150, February 1996.

[4] Lyudmil Aleksandrov, Hristo Djidjev, Hua Guo, and Anil Maheshwari. Partitioning planar graphs with costs and weights. *J. Exp. Algorithmics*, 11, February 2007.

[5] N. Alon and J. H. Spencer. *The probabilistic method*. John Wiley & Sons, 3rd edition, 2007.

[6] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[7] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[8] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS '06. 47th Annual IEEE Symposium on*, pages 459–468, 2006.

[9] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[10] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual*

ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007, pages 1027–1035. SIAM, 2007.

[11] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622–633, 2012.

[12] Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

[13] P. Balister, A. Sarkar, and B. Bollobás. Percolation, connectivity, coverage and colouring of random geometric graphs. In *Handbook of Large-Scale Random Networks*, pages 117–142. Springer, 2008.

[14] Ahmad Biniaz, Evangelos Kranakis, Anil Maheshwari, and Michiel Smid. Plane and planarity thresholds for random geometric graphs. In *Proc. ALGOSENSORS 2015 (Patras, Greece)*, Lecture Notes in Computer Science, Berlin, Germany, 2015. Springer.

[15] J.K. Blitzstein and J. Hwang. *Introduction to Probability*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press, 2014.

[16] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[17] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

[18] B. Bollobás. *Random graphs*. Cambridge University Press, 2001.

[19] Béla Bollobás and Andrew Thomason. Threshold functions. *Combinatorica*, 7(1):35–38, 1987.

[20] Béla Bollobás. *Modern Graph Theory*. Graduate texts in mathematics. Springer, Heidelberg, corrected edition, 1998.

[21] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Elsevier, New York, 1976.

[22] Otakar Borůvka. O jistém problému minimálním. *Práce mor. přírodověd. spol. v Brně III*, 3:37–58, 1926.

[23] Otakar Borůvka. Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí. *Elektrotechnickỳ obzor*, 15:153–154, 1926.

[24] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel H. M. Smid, and Yihui Tang. On the false-positive rate of bloom filters. *Inf. Process. Lett.*, 108(4):210–213, 2008.

[25] J. Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.

[26] Jean Bourgain and Gil Kalai. Threshold intervals under group symmetries. *Convex Geometric Analysis MSRI Publications Volume 34*, pages 59–63, 1998.

[27] Milan Bradonjić and Will Perkins. On sharp thresholds in random geometric graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 500–514, 2014.

[28] Sergey Brin, Rajeev Motwani, Lawrence Page, and Terry Winograd. What can you do with a web in your pocket? *IEEE Data Eng. Bull.*, 21(2):37–47, 1998.

[29] Sergey Brin and Lawrence Page. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 56(18):3825–3833, 2012.

[30] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60:327–336, 1998.

[31] A.Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29, 1997.

[32] Gruia Călinescu, Howard J. Karloff, and Yuval Rabani. Approximation algorithms for the 0-extension problem. *SIAM J. Comput.*, 34(2):358–372, 2004.

[33] James Carrel. Fundamentals of Linear Algebra. `https://www.math.ubc.ca/~carrell/NB.pdf`, 2005. [Online; accessed 2-May-2019].

[34] Timothy M. Chan. Backwards analysis of the Karger-Klein-Tarjan algorithm for minimum spanning. *Inf. Process. Lett.*, 67(6):303–304, 1998.

[35] Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

[36] Davin Choo, Christoph Grunau, Julian Portmann, and Vaclav Rozhon. k-means++: few more steps yield constant approximation. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1909–1917. PMLR, 13–18 Jul 2020.

[37] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 4th edition, 2022.

[38] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[39] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[40] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[41] Artur Czumaj and Christian Sohler. Testing euclidean minimum spanning trees in the plane. *ACM Trans. Algorithms*, 4(3):31:1–31:23, 2008.

[42] S. DasGupta. Algorithms for *k*-means clustering. https://cseweb.ucsd.edu/~dasgupta/291-geom/kmeans.pdf, 2023.

[43] Sanjoy Dasgupta. The hardness of *k*-means clustering. Technical report, Department of Computer Science and Engineering, University of California, San Diego, 2008.

[44] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Struct. Algorithms*, 22(1):60–65, 2003.

[45] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.

[46] Mayur Datar and Piotr Indyk. Locality-sensitive hashing scheme based on p-stable distributions. In *In SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM Press, 2004.

[47] L.B. de Paula, R.S. Villaca, and M.F. Magalhaes. A locality sensitive hashing approach for conceptual classification. In

Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on, pages 408–413, 2010.

[48] Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 101–107. SIAM, 2013.

[49] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[50] B. Dixon, M. Rauch, and R. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992.

[51] Rodney G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer Publishing Company, Incorporated, 2012.

[52] Petros Drineas and Michael W. Mahoney. Lectures on randomized numerical linear algebra. *CoRR*, abs/1712.08880, 2017.

[53] D.P. Dubhashi and A. Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

[54] P. Erdös and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5:17–61, 1960.

[55] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.

[56] U. Feige and D. Reichman. Recoverable values for independent sets. *Random Structures & Algorithms*, 46(1):142–159, 2013.

[57] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, January 1968.

[58] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(2):298–305, 1973.

[59] Philippe Flajolet and G. Nigel Martin. Probabilistic counting. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 76–82. IEEE Computer Society, 1983.

[60] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987.

[61] Greg N. Frederickson. Optimal algorithms for tree partitioning. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '91, pages 168–177, Philadelphia, PA, USA, 1991. Society for Industrial and Applied Mathematics.

[62] E. Friedgut and G. Kalai. Every monotone graph property has a sharp threshold. *Proceedings of the American Mathematical Society*, 124(10):2993–3002, 1996.

[63] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

[64] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Multiway cuts in node weighted graphs. *J. Algorithms*, 50(1):49–61, 2004.

[65] E. Gilbert. Random graphs. *The Annals of Mathematical Statistics*, pages 1141–1144, 1959.

[66] E. Gilbert. Random plane networks. *Journal of the Society for Industrial & Applied Mathematics*, 9(4):533–543, 1961.

[67] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. pages 518–529, 1997.

[68] Erhard Godehardt and Jerzy Jaworski. On the connectivity of a random interval graph. *Random Struct. Algorithms*, 9(1-2):137–161, 1996.

[69] A. Goel, S. Rai, and B. Krishnamachari. Sharp thresholds for monotone properties in random geometric graphs. In *Proceedings of STOC*, pages 580–586. ACM, 2004.

[70] D. Gorisse, M. Cord, and F. Precioso. Locality-sensitive hashing for chi2 distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(2):402–409, 2012.

[71] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1994.

[72] Christoph Grunau, Ahmet Alper Özüdoğru, Václav Rozhoň, and Jakub Tětek. *A Nearly Tight Analysis of Greedy k-means++*, pages 1012–1070.

[73] Piyush Gupta and P. R. Kumar. Critical power for asymptotic connectivity in wireless networks. In *Stochastic Analysis, Control, Optimization and Applications*, pages 547–566, 1998.

[74] Torben Hagerup. An even simpler linear-time algorithm for verifying minimum spanning trees. In Christophe Paul and Michel Habib, editors, *Graph-Theoretic Concepts in Computer Science*, volume 5911 of *Lecture Notes in Computer Science*, pages 178–189. Springer Berlin Heidelberg, 2010.

[75] Torben Hagerup and Christine Rüb. A guided tour of Chernoff bounds. *Inf. Process. Lett.*, 33(6):305–308, 1990.

[76] P. Hall. On the coverage of k-dimensional space by k-dimensional spheres. *The Annals of Probability*, 13(3):991–1002, 1985.

[77] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(1):321–350, 2012.

[78] Frank Harary. *Graph theory*. Addison-Wesley, 1991.

[79] Dorit Hochbaum. *Approximation Algorithms for NP-Hard Problems*. PWS, 1997.

[80] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[81] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.

[82] S. Hu. Efficient video retrieval by locality sensitive hashing. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 2, pages 449–452, 2005.

[83] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[84] S. Janson. Random coverings in several dimensions. *Acta Mathematica*, 156(1):83–118, 1986.

[85] Woojay Jeon and Yan-Ming Cheng. Efficient speaker search over large populations using kernelized locality-sensitive hashing.

In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4261–4264, 2012.

[86] Yushi Jing and S. Baluja. Visualrank: Applying pagerank to large-scale image search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(11):1877–1890, 2008.

[87] William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (in honor of Professor Shizuo Kakutani, held on June 8-11, 1982, at Yale University, New Haven, Connecticut)*, pages 189–206. 1984.

[88] K.A. Kala and K. Chitharanjan. Locality sensitive hashing based incremental clustering for creating affinity groups in hadoop; hdfs - an infrastructure extension. In *Circuits, Power and Computing Technologies (ICCPCT), 2013 International Conference on*, pages 1243–1249, 2013.

[89] Bala Kalyanasundaram and Kirk Pruhs. An optimal deterministic algorithm for online b-matching. *Theor. Comput. Sci.*, 233(1-2):319–325, 2000.

[90] Zixiang Kang, Wei Tsang Ooi, and Qibin Sun. Hierarchical, non-uniform locality sensitive hashing and its application to video identification. In *Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on*, volume 1, pages 743–746 Vol.1, 2004.

[91] David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42(2):321–328, March 1995.

[92] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 352–358. ACM, 1990.

[93] G.O.H Katona. Solution of a problem of A. Ehrenfeucht and J. Mycielski. *Journal of Combinatorial Theory, Series A*, 17(2):265–266, 1974.

[94] V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18(2):263–270, 1997.

[95] Valerie King. A simpler minimum spanning tree verification algorithm. In SelimG. Akl, Frank Dehne, Jörg-Rüdiger Sack,

and Nicola Santoro, editors, *Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 440–448. Springer Berlin Heidelberg, 1995.

[96] Philip N. Klein and Robert E. Tarjan. A randomized linear-time algorithm for finding minimum spanning trees. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, STOC '94, pages 9–15, New York, NY, USA, 1994. ACM.

[97] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[98] Robert Kleinberg. Cs8820: Analysis of algorithms, 2012.

[99] Donald E. Knuth. *The art of computer programming, volume 1-3*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[100] J. Komlos. Linear verification for spanning trees. In *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pages 201–206, 1984.

[101] J. Komlós. Linear verification for spanning trees. *Combinatorica*, 5(1):57–65, 1985.

[102] D. Kozen. *The design and analysis of algorithms*. Springer, 1992.

[103] Bhaskar Krishnamachari, Stephen B. Wicker, Rámon Béjar, and Marc Pearlman. Critical density thresholds in distributed wireless networks. In *Communications, information and network security*, 2002.

[104] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(6):1092–1104, 2012.

[105] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[106] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982.

[107] L. Lovász and M.D. Plummer. *Matching Theory*. Akadémiai Kiadó, Budapest, 1986. Also published as Vol. 121 of the North-Holland Mathematics Studies, North-Holland Publishing, Amsterdam.

[108] László Lovász. On decomposition of graphs. In *Studia Scientiarum Mathematicarum Hungarica 1*, pages 237–238, 1966.

[109] A. Maheshwari and M. Smid. *Introduction to Theory of Computation*. Free Online, 2012.

[110] Udi Manber. Finding similar files in a large file system. In *in Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, 1994.

[111] Claire Mathieu. A primal-dual analysis of the ranking algorithm, 2011.

[112] Jiri Matousek. *Lectures on Discrete Geometry*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[113] Jiří Matoušek. On the distortion required for embedding finite metric spaces into normed spaces. *Israel Journal of Mathematics*, 93(1):333–344, 1996.

[114] Gregory L. Mccolm. Threshold functions for random graphs on a line segment. *Combinatorics, Probability and Computing*, 13:373–387, 2001.

[115] Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007.

[116] P.L. Meyer. *Introductory probability and statistical applications*. Addison-Wesley, Boston, MA, USA, 1970.

[117] Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, January 1997.

[118] Jayadev Misra and David Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982.

[119] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[120] Tanmoy Mondal, Nicolas Ragot, Jean-Yves Ramel, and Umapada Pal. A fast word retrieval technique based on kernelized locality sensitive hashing. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1195–1199, 2013.

[121] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[122] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. Otakar borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1–3):3 – 36, 2001. <ce:title>Czech and Slovak 2</ce:title>.

[123] Padma Panchapakesan and D Manjunath. On the transmission range in dense ad hoc radio networks. In *Proceedings of IEEE Signal Processing Communication (SPCOM)*, 2001.

[124] Mathew D. Penrose. The longest edge of the random minimal spanning tree. *The annals of applied probability*, pages 340–361, 1997.

[125] Mathew D. Penrose. On k-connectivity for a geometric random graph. *Random Struct. Algorithms*, 15(2):145–164, 1999.

[126] Mathew D. Penrose. *Random geometric graphs*, volume 5. Oxford University Press Oxford, 2003.

[127] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.

[128] Vittal Rao. Advanced Matrix Theory and Linear Algebra for Engineers. https://nptel.ac.in/syllabus/111108066/, 2019. [Online; accessed 2-May-2019].

[129] Z. Rasheed, H. Rangwala, and D. Barbara. Lsh-div: Species diversity estimation using locality sensitive hashing. In *Bioinformatics and Biomedicine (BIBM), 2012 IEEE International Conference on*, pages 1–6, 2012.

[130] Kenneth H. Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill Higher Education, 5th edition, 2002.

[131] Tim Roughgarden. Cs261: A second course in algorithms, 2016. Stanford University.

[132] Michiel Smid. Carleton University, Ottawa, Canada, 2014.

[133] J. H. Spencer. *Ten lectures on the probabilistic method*, volume 52. SIAM, 1987.

[134] Daniel A. Spielman. *Spectral and Algebraic Graph Theory*. Yale University, USA, 2019.

[135] G. Strang. *Linear Algebra and Learning from Data*. Wellesley-Cambridge Press, 2019.

[136] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, Wellesley, MA, fifth edition, 2016.

[137] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[138] Robert Tarjan. *Data Structures and Network Algorithms*, volume 44, chapter 6. Minimum Spanning Trees, pages 71–83. SIAM, 1983.

[139] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, April 1975.

[140] Robert Endre Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715, October 1979.

[141] John N. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22(3):263–282, 1992.

[142] A. Tucker. *Linear algebra: an introduction to the theory and use of vectors and matrices*. Macmillan Publishing Company, 1993.

[143] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990.

[144] Qiang Wang, Zhiyuan Guo, Gang Liu, and Jun Guo. Entropy based locality sensitive hashing. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 1045–1048, 2012.

[145] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, 2011.