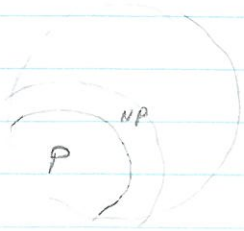
# Approximation Algorithms

1. Quick Review of NP-Hardness




2. Approximation algorithms

Problems :
- Vertex Cover
- Weighted Vertex Cover
- LP-rounding technique
- TSP for n-points in plane (TSP with triangle inequality)
- Set-Cover
- Subset-Sum
- Make-span problem from Kleinberg-Tardos.
- LP

P1: Vertex Cover in a graph.

Input: $G = (V, E)$
undirected, simple graph

Output: A subset $V' \subseteq V$ of smallest size such that $\forall\, e=(u,v) \in E$, either $u \in V'$ or $v \in V'$.
(i.e. each edge has at least one end in $V'$)

Note: The decision problem $\langle G, k \rangle$, i.e does there exists a vertex cover of size $\leq k$ in $G$ is NP-Hard

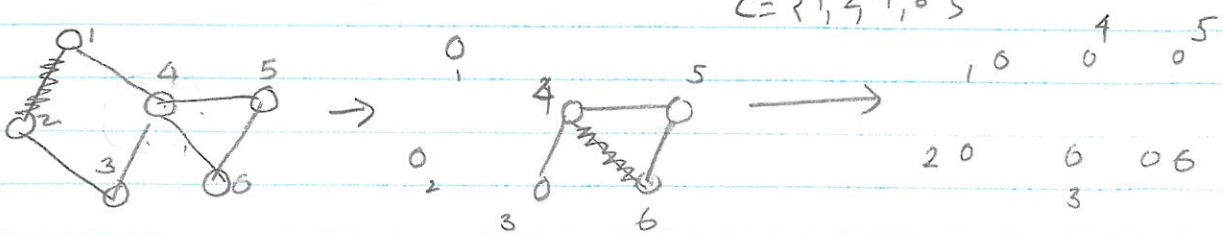
(Therefore, it is unlikely that the optimization problem can be solved in polynomial time (unless P = NP)).

Here is an approximation algorithm

Approx-Vertex-Cover

1. $C := \phi$
2. $E' := E$
3. While $E' \neq \phi$ do
   a. let $e = (u,v) \in E'$.
   b. $C := C \cup \{u\} \cup \{v\}$
   c. remove from $E'$ every edge incident on either $u$ or $v$.

Example:




OPTIMAL COVER = {4, 2, 5}.

Observe that $C$ is a vertex cover.

Claim: Approx-Vertex Cover runs in polynomial time and size of $C$ is at most 2 times the size of an optimal cover.

Pf: - Algorithm runs in polynomial time

- Edges picked in Step 3a forms a matching (independent edges)

- $|C| = 2 * \#$ Edges picked in Step 3a.

- In optimal cover, at least one vertex from each edge picked in Step 3a must be there (as these edges forms matching)

$\Rightarrow |Opt| \geq \#$ Edges picked in Step 3a.

↑ # vertices in an optimal cover.

$\Rightarrow \quad |C| \leq 2|Opt|.$

Clearly $|Opt| \leq |C|$ as $C$ is one of the covers.

Thus $|Opt| \leq |C| \leq 2|Opt|$

and thus Approx-Vertex Cover is a polynomial-time 2-approx algorithm ■.

→ Note that we do not know $|Opt| = ?$ but still we showed that $|C| \leq 2|Opt|$.

Problem 2: TSP with triangle inequality.

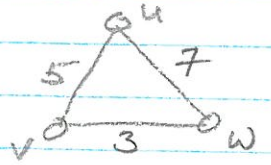
Input: A complete graph $G = (V, E)$ on n-vertices where each edge $e \in E$ has a positive cost $c(e)$.

$$c : E \to \mathbb{R}^+$$

The function $c$ satisfies triangle inequality,

i.e for any three vertices $u, v, w \in V$, the cost of the edges $uv, vw, uw$ satisfy

$$c(uv) \leq c(u,w) + c(w,v)$$




Output: Find a tour in $G$ that visits all vertices and returns back to the start vertex and has minimum total cost.

We define cost of a tour to be the sum total of cost of all edges on the tour.

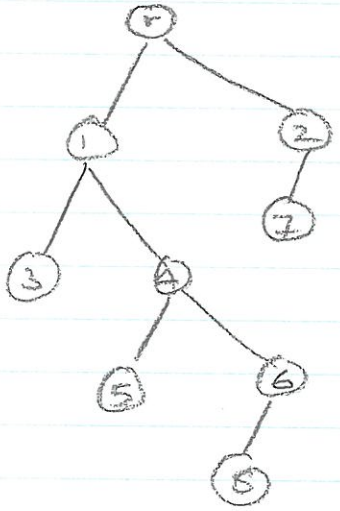
↗ Decision
↙ Problem is NP-Hard.!

Approx-TSP-Tour (G, c)

1. Select a vertex $r \in V$ as the root vertex
2. Compute MST(G) rooted at r.
3. Do a preorder traversal of MST(G) and form a list of vertices when they are "first" visited in the traversal
4. Return the list of vertices in the traversal as the tour. Let H be this list

Example:

Let MST(G, r) =




Preorder traversal:

"ET" = r → 1 → 3 → 1 → 4 → 5 → 4 → 6 → 8 → 6 → 4 → 1 → r → 2 → 7 → 2 → r

"first" H = r → 1 → 3 → 4 → 5 → 6 → 8 → 2 → 7 → r

**Claim** Approx TSP-tour (G, c) is a polynomial-time 2-approx to a TSP with triangle inequality.

Polynomial time : Obvious since MST(G) can be computed in $O(|V|\lg|V| + |E|) = O(|V|^2)$ time using Prim's algo.

Preorder traversal only requires $O(|V|)$ time.

2-approximability :

Let $H^*$ be an optimal tour.

If we remove an edge from $H^*$, we obtain a path (i.e a tree) connecting all the vertices.

Thus $\text{Cost}(MST(G)) \leq \text{cost}(H^*)$ — (I)

~~Clearly~~

Consider Step 4 of the algorithm where the final tour is obtained from the preorder traversal tour "ET" (for "Euler Tour")

Note that $\text{Cost}(ET) = 2\,\text{Cost}(MST(G))$

and $\text{Cost}(H) \leq \text{Cost}(ET) = 2\,\text{cost}(MST(G))$
$\leq 2\,\text{cos}(H^*)$

triangle inequality for "shortcuts"

(I)

Thus

$$\text{cost}(MST) \leq \underline{\text{cost}(H^*)} \leq \underline{\text{Cost}(H)} \leq 2\,\text{cost}(MST(G)) \leq \underline{2\text{Cost}(H^*)}$$

∎

Problem 3: A load balancing problem.

Input: A set $m$-parallel/identical machines $M_1, M_2, \ldots, M_m$.
A set of $n$ jobs $J = \{1, 2, 3, \ldots n\}$ where each job $i$ has a execution time of $t_i$ on any given machine.

"Any machine can serve any job".

"A job can't be split among two or more machines".

Problem

Assign jobs to machine, and

let $A(i) \subseteq \{1..n\}$ be the set of jobs assigned to machine $i$.

Define

$$T[i] = \sum_{j \in A(i)} t_j$$

= Total time of completion of jobs in $A(i)$ on machine $i$.

Our objective is to

$$\text{Min}\left\{\text{Max}_{1 \leq i \leq m} T[i]\right\}.$$

↳ The Decision problem

- does there exists an assignment of jobs to machines so that $\min\{\max_{1\le i\le m} T[i]\} \le t$

is NP-Hard.

Greedy Algorithm I

1. Initialize each machine to consists of no jobs
   For i := 1 to m do [A[i] := ϕ. T[i] := 0;
2. For jobs j := 1 to n do

   [ Let i be the machine with $\min_{1\le k\le m} T[k]$.
   Assign job j to machine i
   [ A[i] = A[i] ∪ {j}
   T[i] := T[i] + $t_j$

Claim: Makespan of Greedy Algorithm I is at most 2 times the makespan of an optimal assignment

(i.e, Greedy Algo 1 is polynomial-time 2-approximate)

Proof

Let $T^*$ be the makespan of an optimal assignment and

Let $T$ = makespan of the greedy algo

$$= \max_{1 \le k \le m} T[k]$$

Note that $T^* \geq \max_{1 \le j \le n} t_j$ (max bound)

and $T^* \geq \frac{1}{m} \sum_{j=1}^{n} t_j$ (average bound)

WLOG, Let the machine $M_i$ attains the maximum load in our greedy algo

i.e $T = \left( \sum_{k \in A(i)} t_k = T[i] \right)$

Let $j$ be the last job placed on $M_i$ by the algo.

Note that just before job $j$ is placed on $M_i$, $M_i$ has the least load (greedy choice)

Thus every machine had a load of at least $T[i] - t_j$.

Therefore,

$$\underbrace{\sum_{1 \le k \le m} T[k]}_{\text{total load of all machines}} \ge m\left[T[i] - t_j\right]$$

$$\Leftrightarrow \quad T[i] - t_j \le \frac{1}{m} \sum T[k]$$

But we know that $\frac{1}{m} \sum T[k]$ is the average load and $T^* \ge \frac{1}{m} \sum T[k]$ by the average bound.

$$\Leftrightarrow \quad T[i] - t_j \le \frac{1}{m} \sum T[k] \le T^*$$

Since $t_j$ is time for job $j$,

clearly $t_j \le \max_{1 \le k \le m} \{t_k\} \le T^*$ by max-bound.

Thus

$$T[i] \le T^* + t_j \le 2T^*$$

Moreover $T[i] \ge T^*$, as $T[i]$ is the value returned by a greedy algo

Thus $T^* \le T[i] \le 2T^*$. □

Construct an example where greedy-algo achieves a ratio of almost 2.

(e.g think of $n = m(m-1)+1$ jobs. Let us say that first $n-1$ jobs require 1 time unit each and the last one requires "m" time units. Then the makespan of greedy algo is $2m-1$, whereas optimal one has a makespan of $m$.)

Exercise: Show that the greedy algo is a $2 - 1/m$ - approximation algo.

An alternate algorithm with improved approximation ratio.

Idea: Take care of large jobs first.

New Algorithm:

Sort jobs in non-increasing order of processing time and then the greedy algo allocates them one by one to machines which have so far least completion times

Claim: The modified greedy algo is a

$\frac{3}{2}$ - approximation algorithm.

Pf: First note that if $n < m$, then each job goes to a distinct machine and the algo is optimal

WLOG assume that jobs in sorted order are such that $t_1 \geq t_2 \geq t_3 \geq \cdots \geq t_n$

Also if $n > m$, then

$$T^* \geq 2t_{m+1} \quad \text{— (C)}$$

Since the first $m$ jobs with times $t_1, t_2, \ldots, t_m$ are placed on distinct machines and each of them takes at least $t_{m+1}$ as their completion times
$\Rightarrow \exists$ a machine that is assigned at least two jobs among first $m+1$ jobs, thus requires $\geq 2t_{m+1}$ time.

As before, let $M_i$ be the machine with maximum load as determined by the greedy algo.

$M_i$ may have one or more jobs.

If $|A[i]| = 1$, then Greedy algo is optimal.

Assume that $|A[i]| \geq 2$.

Let $j$ be the last job assigned to $M_i$ with time $t_j$.

$j \geq m+1$ (as first $m$ jobs are assigned to distinct machines)

Thus $t_j \leq t_{m+1} \leq \frac{1}{2} T^*$

↑ Follows from (C).

As before $T[i] - t_j \leq T^*$

$\Leftrightarrow \quad T[i] \leq T^* + t_j \leq \frac{3}{2} T^*$

$\Rightarrow \quad T^* \leq T[i] \leq \frac{3}{2} T^*$ ∎

Problem 4: k-center problem.

Input: An integer $k>0$, a set $S$ of $n$ sites and a distance function $d: S \times S \to \mathbb{R}^+$ satisfying triangle inequality.

e.g: $n$-points in the plane and distance is the usual Euclidean distance.

Problem: Find $k$-centers so that the maximum distance of all sites to their nearest center is minimized.

Let $C$ be the set of $k$-center.

For each site $s \in S$, define $d(s, C)$ is the distance between $s$ and nearest site in $C$,

i.e, $d(s,C) = \min_{c \in C} d(s,c)$.

Definition: $C$ is called an $r$-cover of $S$ if for each site $s \in S$, $\exists$ a $c \in C$ such that $d(s,c) \le r$.

The minimum $r$, for which $C$ is a $r$-cover of $S$ is called the covering radius of $S$.

Let $r(C)$ denote the radius of min-cover.

**Problem:** Find a collection $C$ of $k$ centers so that $r(C)$ (the covering radius) for a given set $S$ of $n$-sites is minimized.

Problem is NP-Hard.

- Greedy algo is not a good choice.
  (i.e, find the first "best" center, then the second, ...)

Let us assume that $\delta > 0$ is some real number
Suppose "magically" we know the optimum radius $\delta$ and we want to find a set of $k$-sites $C$ such that $r(C) \leq 2\delta$.
Let us execute the following algorithm.

**Algorithm 1**

1. $S' = S$ ($S'$ is the set of sites that need to be covered)
2. $C = \emptyset$
3. While $S' \neq \phi$ do

   Select a site $s \in S'$ and
   $C := C \cup \{s\}$.

   Remove from $S'$ all sites that are within a distance $\leq 2\delta$ from $s$ (including $s$).
4. If $|C| \leq k$ then return $C$ as a set of selected sites
   else "report that covering radius for $S > \delta$".

Claim 1: Suppose algo selects centers in $C$ such that $|C| > k$. Then for any set $C^*$ of $k$-centers, $r(C^*) > \delta$.

Proof: By Contradiction.

Assume $\exists$ a set of at most $k$-centers $C^*$ such that $r(C^*) \leq \delta$.

Since ~~Note that~~ each center $c \in C$ selected by the greedy algo is also an element of $S$, ~~Thus~~ $\exists$ a center $c^* \in C^*$ such that $d(c, c^*) \leq \delta$. Call $c^*$ to be close to $c$ in this case

Claim: No center $c^* \in C^*$ can be close to two different centers of the greedy algo.

Pf: By the greedy choice, any pair of centers chosen by greedy algo are at least $2\delta$ distance apart. Thus $\nexists$ a center $c^* \in C$ which is close to two greedy centers.

Hence, for each center $c^* \in C^*$, $\exists$ a unique center $c \in C$ such that $d(c^*, c) \leq \delta$.

$\Rightarrow |C^*| \geq |C|$ and hence $C^*$ has $> k$ centers, a contradiction $\lightning$.

Above algo, if succesfully finds a set $C$, such that $|C| \leq k$, then we know that $r(C^*) \leq 2\delta$.

If $|C| > k$, then $r(C^*) > \delta$.

How can we use the above algo!

One can try to do "binary search" for the value of $r$ by choosing $\delta$ between 0 and diameter of $S$.

Alternatively, we run the following algo (doesn't require $\delta$).

Algorithm 2.

1. If $|S| \leq k$, then $C = S$ and return $(C)$.
2. Let $s \in S$. Set $C := \{s\}$.
3. While $|C| < k$ do
   - Select a site $s \in S$ which is furthest away from $C$, i.e it maximize $\text{dist}(s, C)$ $s \in S$.
   - $C := C \cup \{s\}$.
4. Return $(C)$.

Claim: The above algo computes a set $C$ of $k$-sites in polynomial time and its a 2-approximation, i.e $r(C^*) \leq r(C) \leq 2r(C^*)$

Pf: Running time: Easy to see.

Proof of 2-approximability (by contradiction)

Suppose $r(C) > 2r(C^*)$.

$\Rightarrow \exists$ a site $s \in S$ that is at a distance greater than $2r(C^*)$ from every site in $C$.

Consider the execution of greedy algo at some intermediate stage where so far $C' \subset C$ has been identified and let $c'$ be the next site that will be added to $C$.

Observe that $c'$ will be at a distance $> 2r(C^*)$ from each site in $C'$ (by the greedy choice as it maximizes the distance).

Thus we have

$$dist(c', C') \geq dist(s, C') \geq dist(s, C) > 2r(C^*).$$

Consider Algorithm 1 (page 15) and the above algo. Since in this algo, we are adding a center which is at least at a distance more than $2r(C^*)$ = 2* optimal radius, we are actually executing Algorithm 1 for the first $k$-iterations.

But in Algorithm 1, $S' \neq \phi$ as $s \in S'$!

$\Rightarrow$ Algorithm 1 will select more than $k$ centers for $\delta = r(C^*)$.

$\Rightarrow$ By Claim 1 (p. 16) $r(C^*)$ is not an optimal radius.

(This contradicts our assumption on $r(C^*)$)

Thus $r(C) \leq 2r(C^*)$. ∎

## Problem 5: Set-Covering

Input:

A Set $X = \{x_1, x_2, \ldots, x_n\}$

$\mathcal{F}$ = A family of Subsets of $X$

$\mathcal{F} = \{S_1, S_2, \ldots, S_\alpha\}$, where $S_i \subseteq X$

$X = \bigcup_{1 \le i \le \alpha} S_i$

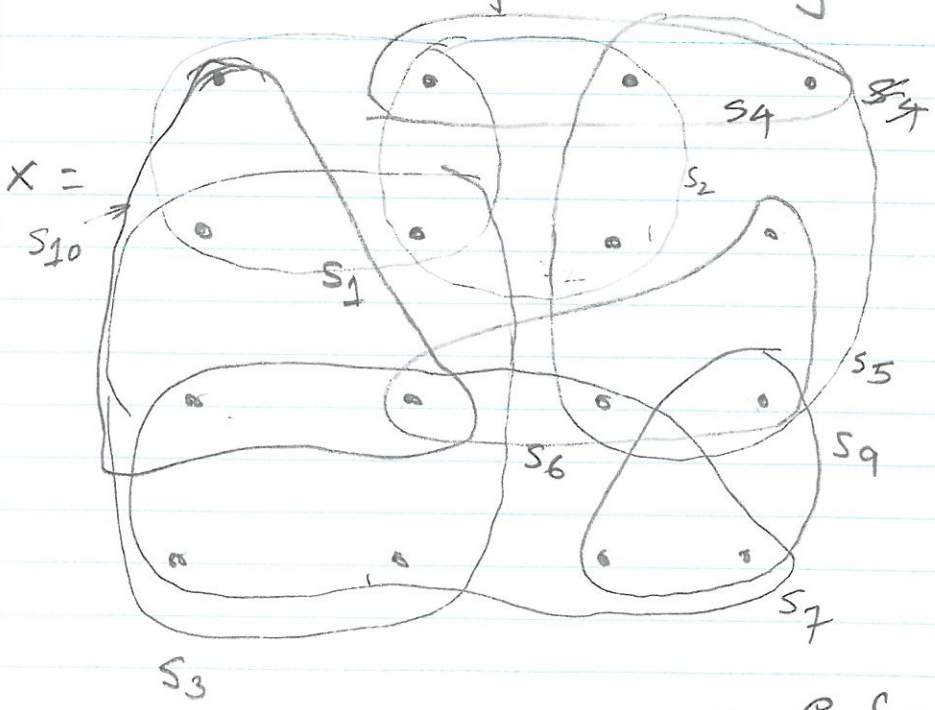
Problem: Find the minimum size subset $\mathcal{C} \subseteq \mathcal{F}$

such that $X = \bigcup_{S \in \mathcal{C}} S$

(smallest number of subsets in $\mathcal{F}$ that together cover all elements of $X$).

example.

Let $X$ be represented by "."




$\mathcal{F}$ consists of 10 sets.

Clearly we can find a $\mathcal{C} \subseteq \mathcal{F}$ such that it covers all elements of $X$ and has size $\le |\mathcal{F}|$.

e.g $\mathcal{C} = \{S_{10}, S_1, S_5, S_2\}$

Does there exists a cover $\mathcal{C}$, s.t $|\mathcal{C}| \leq k$ is NP-Hard !

A Greedy approximation algorithm.

Greedy Set-Cover $(X, \mathcal{F})$

1. $U := X$ ; ($U$ stands for uncovered elements)
2. $\mathcal{C} := \phi$ ;
3. While $U \neq \phi$ do

   Select $S \in \mathcal{F}$ such that $|S \cap U|$ is maximized.

   $U := U - S$

   $\mathcal{C} := \mathcal{C} \cup S$.
4. Return $(\mathcal{C})$

for our example, the sets will be selected as follows (in order)

$S_7$ ; $S_5$ ; $S_1$

Thus $\mathcal{C} = \{S_7, S_5, S_1\}$

and $|\mathcal{C}| = 1$.

Claim: Greedy Set-Cover is a polynomial-time $\rho(n)$-approximation algorithm, where $\rho(n) = H\{\max\{|S| : S \in \mathcal{F}\}\}$.

Note that $H(n) = \sum_{i=1}^{n} \frac{1}{i} \approx \ln n$

Proof:

Polynomial Time: Obvious.

$\rho(n)$-approximability:

Idea: We will assign a "cost" of 1 each set selected by greedy algo and then will distribute this cost over the elements of that set which are covered for first time.

Let $\mathcal{C}^*$ be an optimal cover and let $\mathcal{C}$ be the cover returned by the algorithm.

WLOG, assume that Greedy-algo first selects set $S_1$, then $S_2$, then $S_3$, - - - - -

Let $c_x$ be the cost allocated to element $x \in X$.

Note that $x$ will be assigned this cost when it is covered for the first time. It is only assigned cost once.

Let us assume that $x$ is covered for the first time by the set $S_i$ when selected by the greedy algo.

Then $c_x = \dfrac{1}{|S_i - (S_1 \cup S_2 \cdots S_{i-1})|}$

$|S_i - (S_1 \cup S_2 \cdots S_{i-1})|$ : total # of new elements covered by $S_i$.

Also total cost assigned for each set $S_i$ is "1". ~~by the~~

Thus

$$|\mathcal{C}| = \sum_{x \in X} c_x \qquad \text{— (1)}$$

Also each element $x \in X$ is at least in one set of an optimal cover $C^*$.

Thus $$\sum_{S \in C^*} \sum_{x \in S} c_x \geq \sum_{x \in X} c_x \qquad \text{— (2)}$$

An element $x \in X$ may be present in two sets in $C^*$ and ~~&~~ thus its cost is counted twice whereas on the right it is only counted once.

We claim that

for any set $S \in \mathcal{F}$, $\sum_{x \in S} c_x \leq H(|S|)$. — (3)

Let us first assume this is true.

Then (1), using (2) and (3) becomes

$$|\mathcal{C}| \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x \leq \sum_{S \in \mathcal{C}^*} H(|S|)$$

$$\Leftrightarrow |\mathcal{C}| \leq |\mathcal{C}^*| H(\max\{|S| : S \in \mathcal{F}\}).$$

↳ Let us prove (3)

Consider any set $S \in \mathcal{F}$ and define

$u_i = |S - (S_1 \cup S_2 \cup \cdots \cup S_i)|$

(These are the number of elements left uncovered in set $S$ after greedy algo has chosen first $i$-sets)

Define $u_0 = |S|$ and let $k$ be the last index such that $u_k = 0$ and $u_i > 0$ for $0 \leq i \leq k-1$.

~~(k is +~~

Note that $u_{i-1} \geq u_i$

and $u_{i-1} - u_i$ are the # elements of $S$ covered for the first time by $S_i$.

Thus

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} (u_{i-1} - u_i) \frac{1}{|S_i - (S_1 \cup S_2 \cdots \cup S_{i-1})|}$$

$u_{i-1} - u_i$: # elements of $S$ covered by $S_i$ for the first time.

$|S_i - (S_1 \cup S_2 \cdots \cup S_{i-1})|$: # elements of $x$ covered by $S_i$ for the first time.

Note that

$$|S_i - (S_1 \cup S_2 \cdots \cup S_{i-1})| \geq |S - (S_1 \cup S_2 \cdots \cup S_{i-1})|$$

$= \max$

because $S_i \in \mathcal{F}$ was the set chosen by greedy algo maximizing # elements that will be covered by a set in $\mathcal{F}$ for the first time.

Thus

$$|S_i - (S, US_2 \cdots US_{i-1})| \geq |S - (S, US_2 \cdots US_{i-1})|$$

$$= u_{i-1}$$

↑ by definition.

Hence,

$$\sum_{x \in S} c_x \leq \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{u_{i-1}}$$

$$= \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{u_{i-1}}$$

as $= u_{i-1} - u_i + 1 - 1$
$= u_{i-1} - u_i$

$$\leq \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j} \quad \text{as } j \leq u_{i-1}$$

$$= \sum_{i=1}^{k} \left[ \sum_{j=1}^{u_{i-1}} \frac{1}{j} - \sum_{j=1}^{u_i} \frac{1}{j} \right]$$

$$= \sum_{i=1}^{k} \left[ H(u_{i-1}) - H(u_i) \right]$$

$$= H(u_0) - H(u_k) = H(u_0) = H(|S|)$$

↑ $u_k = 0$.

□

## Problem 7: Subset Sum

Input: $S = \{x_1, x_2, \ldots, x_n\}$ a set of positive integers

$t > 0$ an integer.

Output: A subset $S' \subseteq S$ such that

sum of elements in $S'$ is as large as possible but $\leq t$.

↙ NP-Hard : Does there exists a $S' \subseteq S$

s.t $\sum_{x \in S'} x \stackrel{?}{=} t$.

First: An exact algorithm running in exponential time.

Exact-Subset-Sum $(S, t)$

1. $n := |S|$
2. $L_0 = \langle 0 \rangle$
3. For $i := 1$ to $n$ do

   $L_i = \text{Merge-lists}(L_{i-1}, L_{i-1} + x_i)$

   remove from $L_i$ every element greater than $t$.
4. Return the largest element of $L_n$.

What is $L_i$?

The list of sums of all subsets of $\{x_1, x_2, \ldots, x_i\}$ that do not exceed $t$.

e.g. Let $S = \{2, 4, 7, 8\}$ and ~~$t = 9$~~ $t = 10$

then $L_0 = \langle 0 \rangle$

$L_1 = \langle 0, 2 \rangle$

$L_2 = \langle 0, 2, 4, 6 \rangle$

$L_3 = \langle 0, 2, 4, 6, 7, 9 \rangle$

$L_4 = \langle 0, 2, 4, 6, 7, 8, 9, 10 \rangle$

In this case we will return 10.

- Note that each $L_i$ is sorted and

$$|L_i| \leq 2|L_{i-1}|.$$

- $L_i$ contains sum of all possible subsets of $\{x_1, x_2, \cdots, x_i\}$ which are at most $t$. <Prove it by induction on $i$>

Claim: Exact_Subset_Sum $(S, t)$ finds the largest sum $\leq t$ in $O(2^n)$ time.

Pf: Follows from above observations.

Also we can find the subset $S' \subseteq S$ which forms the largest sum by keeping track of elements that contribute to the sum. □

## An approximation

Main idea: Maintain smaller lists $L_i'$.

If two entries in $L_i$ are very "close" to each other, maintain only one of them.

Suppose $0 < \delta < 1$.

Let $L$ be a list. ~~generated by executing~~

We trim $L$ by a parameter $\delta$ to obtain the list $L'$ with the following properties

(a) $L' \subseteq L$

(b) $\forall y \in L \setminus L'$, $\exists$ a $z \in L'$ such that $\frac{y}{1+\delta} \leq z \leq y$

Example

Let $L = \langle 10, 11, 12, 14, 15, 16, 18, 19, 23, 25 \rangle$

Let $\delta = 0.1$

then $L' = \langle 10, 12, 14, 16, 18, 23, \cancel{25} \rangle$

11 is represented by 10
15 is represented by 14
19 is represented by 18
25 is represented by 23.

Following procedure trims the lists L w.r.t $\delta$ $0<\delta<1$

Trim $(L, \delta)$

1. Let $L = \langle y_1, y_2, \cdots, y_m \rangle$, where $y_1 \leq y_2 \leq y_3 \cdots \leq y_m$.
2. ~~$L' := \langle y_1 \rangle$~~ $last := y_1$.
3. $L' := \langle y_1 \rangle$
4. for $i := 2$ to $m$ do

```
[ if y_i > last.(1+δ)
      [ append y_i to end of L'
      [ last := y_i
```

5. Return $L'$.

Now suppose for the subset-sum problem we are interested to find the largest sum $\leq t$. Since we can't find the largest in polynomial time we are looking for some value of sum which is "close" to the optimum.

Let us try to find a value which is within a factor of $1+\epsilon$ of the optimum sum for some $0<\epsilon<1$.

Consider the following method:

Approx-Sum $(S, t, \epsilon)$

1. $n = |S|$
2. $L_0 := \langle 0 \rangle$
3. for $i := 1$ to $n$ do
   - $L_i = \text{Merge-List}(L_i, L_{i-1} + x_i)$
   - $L_i = \text{Trim-List}(L_i, \delta = \epsilon/2n)$
   - remove from $L_i$ every element $> t$
4. Let $\beta$ be the largest element in $L_n$
5. Report $\beta$

Let $y^*$ be optimum sum for the Subset-Sum problem.

Note that $\beta \leq y^* \leq t$

$\beta \leq y^*$ because $\beta$ is still sum of elements of some subset of $S$.

$y^* \leq t$ property of optimum sum

First we want to show that

$$\frac{y^*}{\beta} \leq 1 + \epsilon.$$

First consider some subset $S' \subseteq \{x_1, x_2, \ldots, x_i\}$

and let $y = \sum_{x \in S'} x$.

We claim that in the list $L_i$ $\exists$ an element $z$ $(z \in L_i)$

such that $\frac{y}{\left(1+\frac{\epsilon}{2n}\right)^i} \leq z \leq y$. — (A)

(i.e $z$ is "close" to $y$)

(This is proved using induction on $i$.
Left as an exercise
Assume this is true.

$\Rightarrow$ For $y^*$, (A) must hold w.r.t. $L_n$,

i.e.

$$\frac{y^*}{\left(1+\frac{\epsilon}{2n}\right)^n} \leq z \leq y^* \text{ for some } z \in L_n.$$

$$\Rightarrow \frac{y^*}{z} \leq \left(1+\frac{\epsilon}{2n}\right)^n$$

$\Rightarrow$ Inequality shown hold for $\beta$ as this is largest element in $L_n \leq t$.

Thus

$$\frac{y^*}{\beta} \leq \left(1+\frac{\epsilon}{2n}\right)^n \overset{?}{\leq} e^{\epsilon/2}$$

(? — calculus)

$$\leq 1+\epsilon/2+(\epsilon/2)^2$$

$$\leq 1+\epsilon .$$

What about running time of approx-sum?

It is polynomial in parameters $\{n, \log t, 1/\epsilon\}$

What is $|L_i|$ ?

Note that two successive elements $z \neq z'$ of $L_i$ satisfy $\frac{z'}{z} > 1+\frac{\epsilon}{2n}$

Thus each list contains a maximum of

$$2+\left\lfloor \log_{1+\frac{\epsilon}{2n}} t \right\rfloor \text{ values.}$$

↑ may contain 0, and 1 ~~and of~~

Note that

$$2+\left\lfloor \log_{1+\frac{\epsilon}{2n}} t \right\rfloor \leq 2+\log_{1+\frac{\epsilon}{2n}} t$$

$$= 2+\frac{\ln t}{\ln 1+\frac{\epsilon}{2n}}$$

$$\leq 2 + \frac{2n\left(1 + \epsilon/2n\right) \ln t}{\epsilon}$$

$$\left[\frac{x}{1+x} \leq \ln(1+x) \leq x \quad \forall \; x > -1.\right.$$

$$< 2 + \frac{3n \ln t}{\epsilon}$$

Since $|L_i| < 2 + \underbrace{\frac{3n \ln t}{\epsilon}}$

polynomial in $\frac{1}{\epsilon}, n, \ln t$

the total running time is polynomial in these parameters. ∎

Note that for any value of $0 < \epsilon < 1$, we can reach within a factor of $(1+\epsilon)$ of optimum and running time is polynomial in $\frac{1}{\epsilon}, n, \log t$ (input parameters)

As $\epsilon \to 0$, we reach close to optimum but $\frac{1}{\epsilon} \to \infty$.

Such kind of approximation algorithms are called polynomial time approximation schemes (PTAS).

Problem 8

Vertex Cover by Linear Programming.

Input: A graph $G=(V,E)$

$w : V \to \mathbb{R}^+$

Problem: for any subset $V' \subseteq V$, define $w(V') = \sum_{u \in V'} w(u)$.

Find a vertex cover of $G$ of minimum weight.

if $V' \subseteq V$ is a vertex cover then

$\forall\ e=(u,v) \in E$ : $u \in V'$ or $v \in V'$.

Earlier we studied the problem when $w(u) = 1\ \forall\ u \in V$.

And were interested to find the smallest set $V'$ of cardinality that is a vertex cover of $G$.

Consider the following formulation.

Associate an indicator variable $x(v)\ \forall\ v \in V$.

$$x(v) = \begin{cases} 1 & \text{if } v \text{ is in the cover} \\ 0 & \text{if } v \text{ is not in the cover} \end{cases}$$

and consider the following optimization problem:

$$\min \sum_{v \in V} x(v)\, w(v)$$

subject to $\forall e=(uv) \in E : x(u) + x(v) \geq 1$ — (A)

and $\forall v \in V,\ x(v) \in \{0,1\}$

Note that the condition $x(v) \in \{0,1\}$ says that either the vertex $v$ is in the cover or it is not in the cover.

Note that the for $e=(uv) \in E$
$x(u) + x(v) \geq 1 \Rightarrow$ at least one of $u$ or $v$ has to be in the cover.

Note that ~~Minimize~~ $\sum_{v \in V} x(v)\, w(v)$ adds up the weight of all vertices in the cover

Observe:

Let $V' = \{v \mid v \in V$ such that $x(v)=1\}$.

$V'$ is a vertex cover.

$w(V')$ is minimum.

Formulation (A) is called an

Integer Linear Program → ?

↑ Variables can only take integral values

↑ All constraints and objective function are linear

Infact in our case it is a 0-1 ILP as variables can only take binary values.

Since the vertex cover problem is NP-Hard, 0-1 ILP is also NP-Hard. Hence it is unlikely that 0-1 ILP's can be solved in polynomial time.

To approximate vertex cover following relaxation is done:

(B)

$$\min \sum_{v \in V} x(v)\, w(v)$$

subject to $\forall uv \in E \quad x(u) + x(v) \geq 1$

and $\forall v \in V, \quad 0 \leq x(v) \leq 1$

Note that now we do not require $x(v)$'s to take only integral values.

Formulation (B) is called the relaxed-ILP.

There are polynomial time algorithms to solve linear programs (Khachiyan 79, Karmarkar 84) and more recently.

But the solution of (B) may return fractional values for $x(v)$'s !

Note that any feasible solution for (A) is also a feasible solution for (B) but not vice-versa.

Now our algorithm for computing approximate weighted vertex cover is as follows.

Approx-Weighted-Vertex-Cover $(G, w)$

1. $C := \phi$
2. Formulate ILP corresponding to $G$.
3. Relax ILP and solve the corresponding linear program (LP) and obtain an optimal solution for LP.

   Let $\bar{x}(v)$ be the value assigned to each variable $x(v)$ in LP.
4. $\forall v \in V$, if $\bar{x}(v) \geq 1$ then $C := C \cup \{v\}$.
5. Return $C$.

Claim 1: $C$ is a vertex cover.

Proof: We will show that each edge $e = (uv) \in E$ is covered by at least one vertex in $C$.

~~Note that since $e = (uv) \in E$~~

Pick an arbitrary edge $e=(u,v) \in E$.

~~Since~~ Consider the corresponding constraint $x(u)+x(v) \geq 1$ in formulation (B).

Since LP is feasible, this implies that the values of the variables $x(u)$ & $x(v)$ satisfy the constraint,

i.e $\bar{x}(u)+\bar{x}(v) \geq 1$

or at least one of them is $\geq 1/2$.

~~Hence~~ If $\bar{x}(u) \geq 1/2$ then $u \in C$ or

if $\bar{x}(v) \geq 1/2$ then $v \in C$. □

Claim 2: Weight of cover C, $w(C)$, is atmost twice the weight of an optimal vertex cover.

Proof: Let $z^* = \min \sum_{v \in V} x(v) w(v)$ be the weight corresponding to formulation (A)

Note that $z^*$ is the weight of an optimal vertex cover.

Let $z = \sum_{v \in V} \bar{x}(v)\, w(v)$ be the weight corresponding to solution of relaxed Version (i.e Formulation B)

Note that $z \leq z^*$ — (Equation 1)

↑ LP (formulation B) includes ILP (formulation A)

Now what is the weight of set C returned by Approx-vertex-cover.

Let us look at the following.

$$z = \sum_{v \in V} \bar{x}(v)\, w(v)$$

$$\geq \sum \bar{x}(v)\, w(v)$$

$\forall v \in V : \bar{x}(v) \geq 1/2$

$$\geq \sum \frac{1}{2} w(v)$$

$\forall v \in V : \bar{x}(v) \geq 1/2$ → corresponds to elements in C

Thus $z \geq \frac{1}{2} \sum_{v \in C} w(v) = \frac{1}{2} w(C)$

Using Equation 1 $z^* \geq z \geq \frac{1}{2} w(C)$ (As C is some cover)

$\Leftrightarrow 2z^* \geq w(C) \geq z^*$ ∎

Putting Claim 1 & 2 together with the fact that LP can be solved in polynomial time, we have

Theorem: Approx-vertex-cover is a polynomial time 2-approximation algorithm for the weighted vertex cover problem.

Techinque:

Formulate an ILP

↓

Solve the relaxed LP

↓

Convert fractional values to integral values for each variable.

{ rounding
{ randomization
{ .....