

Fractional Cascading

In this chapter we generalize the bridging mechanism of the previous section. Here is the problem we wish to solve:

Let $G = (V, E)$ be an undirected ^{→ connected} graph with k vertices. Let d be the maximal degree of any vertex. We assume that d is a constant. Each vertex v contains a sorted list $C(v)$ of real numbers.

We want to solve the following type of queries: Given a real number y and a connected subgraph (V', E') of G , locate y in all lists $C(v)$ for $v \in V'$.

Let n be the sum of the lengths of all lists $C(v)$. Of course, we can solve a query by performing a binary search in $C(v)$ for each $v \in V'$. This gives a query time of $O(|V'| \log n)$. The amount of space used is $O(n+k)$.

Fractional cascading improves the query time to $O(|V'| + \log n)$, still using $O(n+k)$ space. The technique is due to Chazelle and Guibas. The algorithms are deterministic and quite complicated. We shall give a randomized construction, which is in fact quite easy.

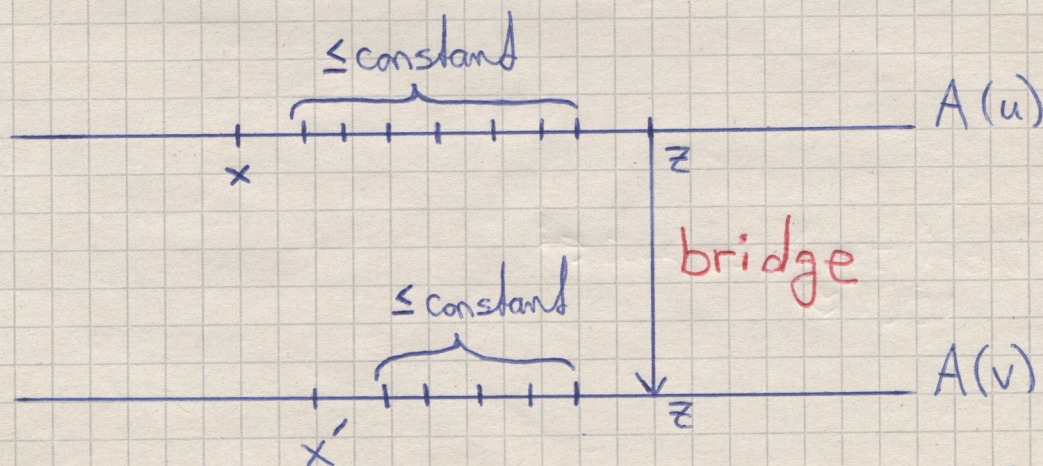
↳ due to Kurt Mehlhorn,
≈ 1991

We assume that the lists $C(v)$, $v \in V$, are pairwise disjoint. We can achieve this by replacing each $x \in C(v)$ by the pair (x, v) and storing these pairs in lexicographical order.

The basic idea is the same as in the previous section:

- * We expand each list $C(v)$ to an augmented list $A(v)$. The list $A(v)$ contains $C(v)$, the elements $-\infty$ and ∞ , and some elements that are copied from the lists of the neighbors of v in G .
- * The lists $A(u)$ and $A(v)$ for vertices u and v that are neighbors in G are connected by bridges.

The problem, of course, is which elements to copy from one list into another. Our goal is the following: Let u and v be any two neighboring vertices in G and let y be any real number. Let $x = \min \{a \in A(u) : a \geq y\}$ and $x' = \min \{a \in A(v) : a \geq y\}$. Let z be the smallest element in $A(u)$ that is at least equal to x and that contains a bridge to $A(v)$. Then, $A(u)$ contains at most a constant number of elements between x and z . Similarly, $A(v)$ contains at most a constant number of elements between x' and z .



If this condition holds, then it is clear that we can locate y in $A(v)$ in constant time once we know its position in $A(u)$.

We now show how to reach the goal in an expected sense: (35)
The expected number of elements between x and z , and between x' and z , will be bounded by a constant.

For each vertex $u \in V$, $A(u)$ will be a sorted sequence of real numbers. With each element x of $A(u)$ we store the following additional information:

- (1) pred : a pointer to the predecessor of x in $A(u)$,
- (2) next : a pointer to the successor of x in $A(u)$,
- (3) proper : a pointer to the smallest element in $C(u)$ that is at least equal to x ,
- (4) for each vertex v that is a neighbor of u there is at most one pointer to the list $A(v)$. This pointer is called a bridge from x in $A(u)$ to $A(v)$.

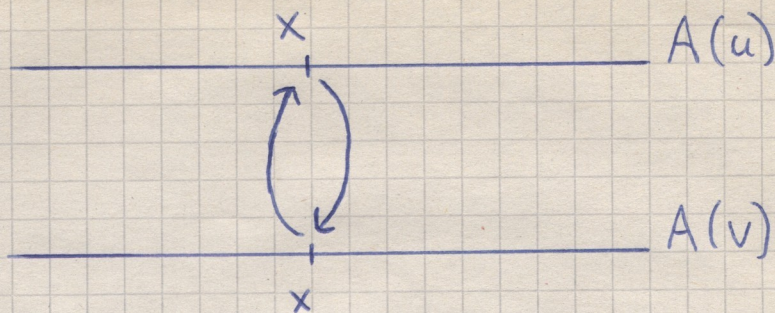
Note that each vertex has degree at most d which is assumed to be a constant. Therefore, element $x \in A(u)$ contains at most ~~d~~ d bridges. The bridges will satisfy the following

Invariant: For any two neighbouring vertices u and v , and for any $x \in A(u)$, if there is a bridge from x in $A(u)$ to $A(v)$, then

- (a) x is contained in $A(v)$ and this bridge points to the occurrence of x in $A(v)$,
- (b) x in $A(v)$ contains a bridge to $A(u)$ which points to the occurrence of x in $A(u)$.

See the picture below.

(36)



Here is the algorithm for constructing the augmented lists:

Initialization: For each vertex u , let $A(u) = \langle -\infty, \infty \rangle$.

① ~~For each~~ For each neighboring pair u and v of vertices, connect the occurrences of $-\infty$ and ∞ in $A(u)$ and $A(v)$ by bridges.

Repetition: For each vertex $u \in V$ and each element $x \in C(u)$, do the following: Let $x_0, x_1 \in A(u)$ such that $x_0 < x < x_1$. Call the following procedure:

Insert (x, x_0, x_1, u) :

- (1) insert x into $A(u)$, between x_0 and x_1 , and update the appropriate pred- and next-pointers. Store with x a bit indicating whether x belongs to $C(u)$ or $A(u) \setminus C(u)$.
- (2) forall neighbors v of u
do determine y_0 and y_1 in $A(v)$ such that $y_0 \leq x \leq y_1$
if $x = y_1$
then ($\ast$ x already occurs in $A(v)$ $\ast$)
connect the occurrences of x in $A(u)$ and $A(v)$ by bridges

else with probability $1/2d$, call $\text{Insert}(x, y_0, y_1, v)$ (37)

and connect the occurrences of x in $A(u)$ and $A(v)$ by bridges

od. f_i

Postprocessing step: For each vertex $u \in V$, walk along the list $A(u)$, starting at the end, and give each element its proper pointer. Build a binary search tree on $A(u)$, such that we can do binary search in $A(u)$.

It is clear that during the repetition the invariant is maintained. The size of the final data structure is bounded by $O(|V| + |E| + \sum_{u \in V} |A(u)|)$. Hence, we need an upper bound on the expected value of the summation.

Let u be a vertex and let $x \in C(u)$. Let K_x be the number of times element x is inserted into an augmented list.

Lemma: The expected value of K_x is at most 2.

Proof: I do not know if this is a correct proof:

$$E(K_x) \leq 1 + d \cdot \frac{1}{2d} E(K_x), \text{ which implies } E(K_x) \leq 2.$$

The following proof is correct:

(38)

Let G_{dir} be the directed version of G , i.e., each edge $\{v, w\}$ of G occurs as (v, w) and (w, v) in G_{dir} .

Let $\{X_{vw} : (v, w) \text{ an edge of } G_{\text{dir}}\}$ be a set of independent random variables with $\text{Prob}(X_{vw} = 1) = 1/2d$ and $\text{Prob}(X_{vw} = 0) = 1 - 1/2d$.

Recall that $x \in C(u)$. Let S be the set of vertices that can be reached from u by following zero or more edges (v, w) of G_{dir} with $X_{vw} = 1$. Note that S is a random variable.

It is clear that $E(K_x) = E(|S|)$.

For any vertex v define the random variable X_v by

$$X_v = \begin{cases} 1 & \text{if } v \in S, \\ 0 & \text{if } v \notin S. \end{cases}$$

Then,

$$E(|S|) = E\left(\sum_v X_v\right) = \sum_v E(X_v) = \sum_v \text{Prob}(X_v = 1).$$

How do we estimate $\text{Prob}(X_v = 1)$? Let P_v be the set of all paths in G_{dir} from u to v . Then,

$$\text{Prob}(X_v = 1) = \text{Prob}(\exists p \in P_v : X_e = 1 \text{ for all edges } e \text{ on } p)$$

$$\leq \sum_{p \in P_v} \text{Prob}(X_e = 1 \text{ for all edges } e \text{ on } p)$$

$$= \sum_{p \in P_v} \left(\frac{1}{2d}\right)^{\text{length of } p}$$

It follows that

(39)

$$\begin{aligned}
 E(K_x) &\leq \sum_v \sum_{p \in P_v} \left(\frac{1}{2d} \right)^{\text{length of } p} \\
 &= \sum_{p \in \bigcup_v P_v} \left(\frac{1}{2d} \right)^{\text{length of } p} \\
 &= \sum_{i=0}^{\infty} \left(\frac{1}{2d} \right)^i \cdot \# \text{ paths of length } i \text{ starting } \textcircled{\text{in}} u \text{ at} \\
 &\leq \sum_{i=0}^{\infty} \left(\frac{1}{2d} \right)^i \cdot d^i \\
 &= \sum_{i=0}^{\infty} \left(\frac{1}{2} \right)^i \\
 &= 2.
 \end{aligned}$$

→ in G_{dir}

□
 This lemma shows that any element x of any list $C(u)$ occurs, expected, at most twice in $\bigcup_v A(v)$. Therefore, the expected size of $\bigcup_v A(v)$ is at most $2k + 2 \sum_v |C(v)| = O(k+n)$. (Recall that each list $A(v)$ contains $-\infty$ and ∞ . This causes the term $2k$.) Since each list element contains a constant amount of additional information, it follows that the expected size of the entire data structure is bounded by

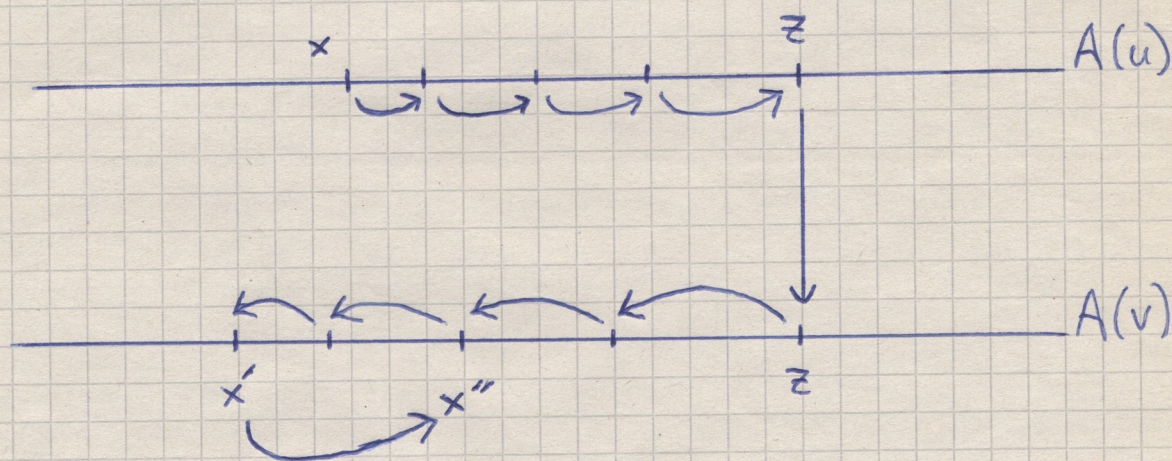
$$O(|V| + |E| + \cancel{E} E\left(\sum_v |A(v)|\right)) = O(n+k).$$

Lemma: The data structure has expected size $O(n+k)$.

(40)

Let u and v be two neighboring vertices and let $y \in \mathbb{R}$. Assume we have located y in $A(u)$. Then we locate y in $A(v)$ and $C(v)$ as follows:

- (1) let x be the smallest element in $A(u)$ that is at least equal to y ;
- (2) starting in x , follow next-pointers, until we reach the first element z that contains a bridge to $A(v)$;
- (3) traverse the bridge to $A(v)$;
- (4) starting in z in $A(v)$, follow pred-pointers until we are at the smallest element x' that is at least equal to y ;
- (5) follow the proper-pointer of x' to the element x'' of $C(v)$.



Convince yourself that x'' is the smallest element of $C(v)$ that is at least equal to y .

Lemma: The expected number of pointers that are followed in Steps (1)-(5) is at most $2d+2$. (41)

Proof: It suffices ~~that~~ to prove that the expected number of pointers that are followed in Steps (2) and (4) is at most $2d$.

Let a be the number of elements in $A(u)$ that are at least equal to x and that are less than z . Let b be the number of elements in $A(v)$ that are at least equal to x' and that are less than z .

Exercise: Prove that none of these $a+b$ elements contains a bridge between $A(u)$ and $A(v)$.

Let $L = a+b$. Then, in Steps (2) and (4), we follow L pointers. We estimate the expected value of L .

~~Since an element in $A(u)$ resp. $A(v)$~~ Since the probability that an element in $A(u)$ resp. $A(v)$ does not contain a bridge to $A(v)$ resp. $A(u)$ is at most $1 - 1/2d$, we have for any integer l , $\text{Prob}(L \geq l) \leq (1 - 1/2d)^l$.

Exercise: Prove that the expected value of L , i.e.,

$$\sum_{l=0}^{\infty} l \cdot \text{Prob}(L=l), \text{ is equal to } \sum_{l=1}^{\infty} \text{Prob}(L \geq l).$$

Hence,
$$E(L) = \sum_{l=1}^{\infty} \text{Prob}(L \geq l) \leq \sum_{l=1}^{\infty} (1 - 1/2d)^l$$

$$= 2d(1 - 1/2d) \leq 2d. \quad \square$$

Lemma: If the lists $C(v)$, $v \in V$, are sorted already, then (42)
the data structure can be built in $O(n+k)$
expected time.

Proof: The initialization takes $O(|V| + |E|) = O(k + dk) = O(k)$ time. The postprocessing step takes time $O(\sum_v |A(v)|)$. We saw already that the expected value of this summation is bounded by $O(n+k)$.

The lemmas on pages 37 and 41 imply that for each $u \in V$ and each $x \in C(u)$, the procedure $\text{Insert}(x, x_0, x_1, u)$ takes $O(1)$ expected time. Therefore, the entire repetition takes $O(\sum_u |C(u)|) = O(n)$ expected time. $\blacksquare$

It remains to give the query algorithm:

Let $y \in R$ and let (V', E') be a connected subgraph of G . We want to locate y in all lists $C(v)$, $v \in V'$.

(1) Take a vertex $u_0 \in V'$ and locate y in $A(u_0)$ by binary search. Follow the proper-pointer to locate y in $C(u_0)$.

(2) Traverse the subgraph (V', E') starting in u_0 . If going from vertex u to vertex v , use the procedure on page 40 to locate y in $A(v)$ and $C(v)$.

Exercise: Prove that the expected query time is bounded by $O(|V'| + \log n)$.

We summarize the final result :

(43)

Theorem : Let G be an undirected graph with k vertices. Assume each vertex has degree at most d , which is a constant. Each vertex v of G stores a sorted list $C(v)$ of real numbers. Let $n = \sum_v |C(v)|$.

In $O(n+k)$ expected time, we can build a data structure of expected size $O(n+k)$, such that for any $y \in \mathbb{R}$ and any connected subgraph (V', E') of G , we can locate y in all lists $C(v)$, $v \in V'$, in total expected time $O(|V'| + \log n)$.

Fractional cascading can often be applied in cases where we have (44) a binary search tree that stores geometric objects w.r.t. some ordering, and where each node contains an associated structure storing a subset of the objects w.r.t. another ordering. In such cases, a query is solved by querying certain associated structures, with the same query element. We have seen several examples: Range trees, interval trees, segment trees and the data structure for point location.

We consider one more example:

Let $C_1, C_2, \dots, C_k$ be a collection of k non-empty lists, each being a sorted sequence of real numbers. Let $n = \sum_{i=1}^k |C_i|$.

We want to store these lists in a data structure such that the following type of query can be solved:

Given a real number y and a subset Q of $\{1, 2, \dots, k\}$, locate y in the list C_i for each $i \in Q$.

The naive way to solve a query, i.e., do a binary search in C_i for each $i \in Q$, takes $O(|Q| \log n)$ time. We show how to improve this to $O(|Q| \log^k k / |Q| + \log n)$. Note that this is really an improvement: We assume that each list C_i is non-empty. Therefore, $n \geq k$.

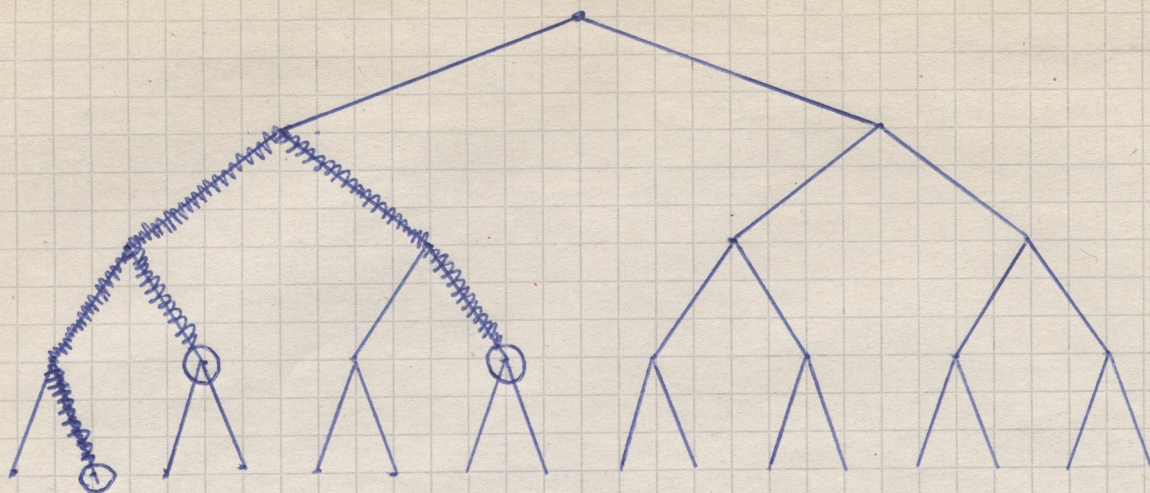
We want to apply fractional cascading. Hence, we need a graph G of bounded degree. Take for G a balanced binary search tree storing the values $1, 2, \dots, k$ in its nodes. For $1 \leq i \leq k$, node i stores the list C_i . Run the fractional cascading algorithm that

constructs the augmented lists A_i , $1 \leq i \leq k$. This will give (45)
our data structure. Its expected size is $O(n)$.

How do we solve a query? Let $y \in \mathbb{R}$ and let $Q \subseteq \{1, 2, \dots, k\}$. In order to use the fractional cascading information, we need a connected subgraph of G that contains all nodes of Q . Of course this subgraph should not be too large. We take for this subgraph the minimum spanning tree T of Q , which is constructed as follows:

Each node of G will have a flag for marking purposes. Initially, all nodes are unmarked. For each node $i \in Q$, traverse the path from i to the root, marking each node along the way and stopping as soon as a node is encountered that is marked already. At the end of this process, the set of marked nodes forms a spanning tree of Q . It is not necessarily minimum since it always contains the root. Therefore, we remove the branch joining the root to the lowest common ancestor of all nodes of Q . This is done as follows: Start at the root i of G . If $i \in Q$ and i has exactly one son j that is marked, then unmark i and iterate with j . Stop iterating if a node is encountered that belongs to Q or that has 2 sons that are marked. See the picture on the next page. Circled nodes belong to Q .

At this moment, we have the minimum spanning tree T of Q . Now we can solve the query: Using the fractional cascading information, locate y in the list C_i for all $i \in T$. Moreover,



- For each $i \in Q$, output the location of y in C_i .
Finally, at the end, we unmark all marked nodes.

What is the query time? It takes $O(|T| + \log k)$ time to compute the minimum spanning tree T . Also, unmarking the nodes of T at the end of the procedure can be done in $O(|T|)$ time. Locating y in the lists C_i for all $i \in T$, takes $O(|T| + \log n)$ expected time. Hence, the total expected query time is bounded by $O(|T| + \log n)$.

It can be shown that $|T| = O(|Q| + |Q| \log^{k/|Q|})$. This proves that the expected query time is bounded by

$$O(|Q| \log^{k/|Q|} + \log n).$$

For other applications of fractional cascading, see

B. Chazelle and L.J. Guibas. Fractional Cascading: II. Applications. Algorithmica 1 (1986), pp. 163-191.