Dynamic Algorithms

Adli Abu Shaban

April 9, 1997

Contents

1 Introduction **2**
1.1 Notations and definitions 2
1.2 Random update sequence 2

2 Trapezoidal decomposition **4**
2.1 Point location 6

3 Voronoi diagrams **7**
3.1 Conflict search 9

4 History and configuration spaces **10**
4.1 Expected performance 10

5 Rebuilding history **11**

6 Deletions in history **13**
6.1 Convex polytopes 15
6.2 Trapezoidal decompositions 15

7 Dynamic shuffling **16**

1 Introduction

If we are interested in keeping track of the update that happens to a Geometric complex like trapezoidal decomposition, semidynamic algorithms are algorithms that keep track of addition of segments into a set of segments M. Dynamic algorithms will keep track of addition as well as deletion of segments. In dynamic algorithms, our goal is to maintain a geometric complex $H(M)$. So whenever the user adds or deletes a segment from M, we want to update $H(M)$ quickly.

To evaluate dynamic algorithm, the best one can hope for is that it performs an update in time proportional to the structural change in the underlying geometric complex $H(N)$.

1.1 Notations and definitions

Consider when deletion is allowed in an update sequence $\bar{u}$. The signature of the update sequence is $\delta = \delta(\bar{u})$. Its signature $\delta = \delta(u)$ is the string of + and - symbols such that the ith symbol in $\delta(u)$ is + iff the ith update in u is addition. Let N be the set of segments or objects involved in $\bar{u}$. The size of N is referred to by n.

Not every sequence δ of + and - symbols can occur as a signature of an update sequence. Consider $r(\delta, i)$, the rank of δ at time i, to be the number of +'s minus the number of -'s in an update sequence. If δ is the signature of an update sequence $\bar{u}$, then $r(\delta, i)$ can only denote the number of current objects at time i. Then δ is a valid signature for all i iff $r(\delta, i) \geq 0$. So $r(\delta, i)$ can be denoted by just $r(\delta)$. When the update sequence involves addition only, $\delta = +^{\mathrm{n}}$ and $r(+^{\mathrm{n}}, i) = i$.**Weakly monotonic** signature, δ, happens if $r(i) = \Omega(i)$. This is equivalent to saying that there exists a constant $b \geq 1$ such that $r(i) \geq \frac{i}{b}$. In other words, there exists a few deletions in the update sequence. The more deletions in the update sequence, the more the signature gets away from weakly monotonic to nonmontonic.

1.2 Random update sequence

An update sequence is random if it is chosen from a uniform distribution on all valid sequences. Let $|\bar{u}|$ denote the length of $\bar{u}$. For each $i \leq |\bar{u}|$ let S^{i} denote the object in the ith update. Let N^{i} denote the set of undeleted objects existing at time i during the execution of $|\bar{u}|$. If the update sequence, (N, δ)-sequence, is random then each object in N is equally likely to be S^{i} and N^{i} is a random subset of N of size $r(i)$.

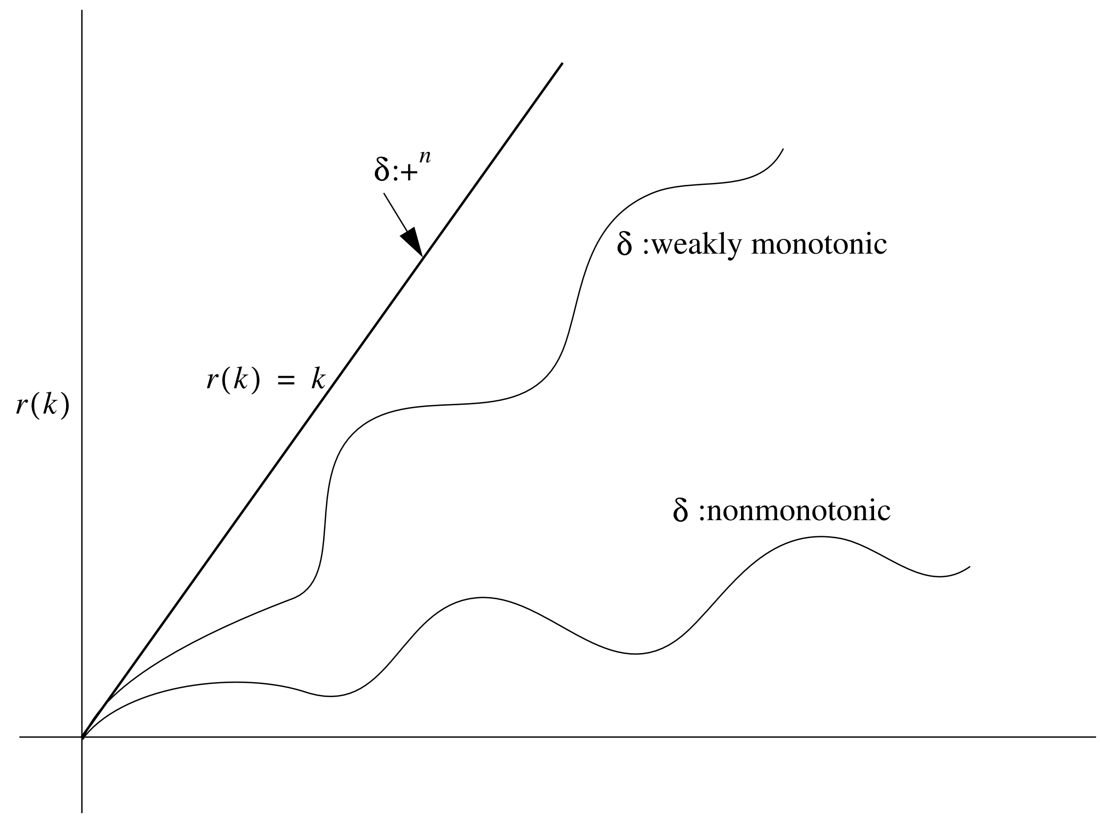

Figure 1: Figure 1:Examples of signatures

2 Trapezoidal decomposition

Consider N^{k} to be the set of undeleted segments of a trapezoidal decomposition existing at time k. Let $H(N^{\mathrm{k}})$ denote the resulting trapezoidal decomposition. Our goal is to maintain $H(N^{\mathrm{k}})$. Let $S = S_k + 1$ be the segment involved in the $(k+1)$th update. Our goal is to update $H(N^{\mathrm{k}})$ to $H(N^{\mathrm{k+1}})$ quickly.

Addition: Here, the $(k+1)$th update is an addition of S to N^{k}. If we assume that the trapezoid in $H(N^{\mathrm{k}})$ containing an endpoint of S, we can add S to $H(N^{\mathrm{k}})$ as the semidynamic algorithm. The point location issue will be discussed later.

Deletion: The $(k+1)th$ update is the deletion of segment S from N^{k}. First the segment S is made transparent to all vertical attachments adjacent to it. Those vertical attachments are made to pass through S because segment S does not exist to them any more. Here we obtain a temporary partition, $\hat{H}(N^{\mathrm{k}})$. The final partition, $N^{\mathrm{k+1}}$ can be obtained from $\hat{H}(N^{\mathrm{k}})$ by erasing the segment S. The whole update can be done in time proportional to $\sum_f$ face-length(f), where f ranges over all trapezoids in $H(N^{\mathrm{k}})$ adjacent to S. **Analysis**

First case, when the $(k+1)$th update is *addition* of S to N^{k}. The cost of endpoint location in segment S will be ignored here. The excepted cost, conditional on a fixed $N^{\mathrm{k+1}}$, is proportional to

$$\frac{|H(N^{\mathrm{k+1}})|}{|N^{\mathrm{k+1}}|} = O\left(\frac{|N^{\mathrm{k+1}}| + I_k + 1}{|N^{\mathrm{k+1}}|}\right) = O\left(1 + \frac{I_k + 1}{r(k+1)}\right),$$

where I_{k+1} is the number intersection among the segment in $N^{\mathrm{k+1}}$.

Second case, consider the case when the $(k+1)$th update involves *deletion* of segment S from N^{k}. The expected cost, conditional on fixed N^{k}, is proportional to

$$\frac{|H(N^{\mathrm{k}})|}{|N^{\mathrm{k}}|} = O\left(\frac{|N^{\mathrm{k}}| + I_k}{|N^{\mathrm{k}}|}\right) = O\left(1 + \frac{I_k + 1}{r(k+1)}\right),$$

where I_k is the number intersection among the segment in N^{k}.

Now, we are going to calculate the expected cost of the algorithm over a random (N, δ)-sequence. In the random sequence, N^{k} is a random subset of N of size $r(k)$. The expected value of I_k is $O(r(k)^2 I/n^2)$, where I is the number of intersection among the segments in N. So the expected total cost is bounded within a constant factor by,

$$\sum_{k=1}^{|\bar{u}|} 1 + \frac{r(k)I}{n^2} = n + \frac{I}{n^2} \sum_{k=1}^{|\bar{u}|} r(k) = O(n + I)$$

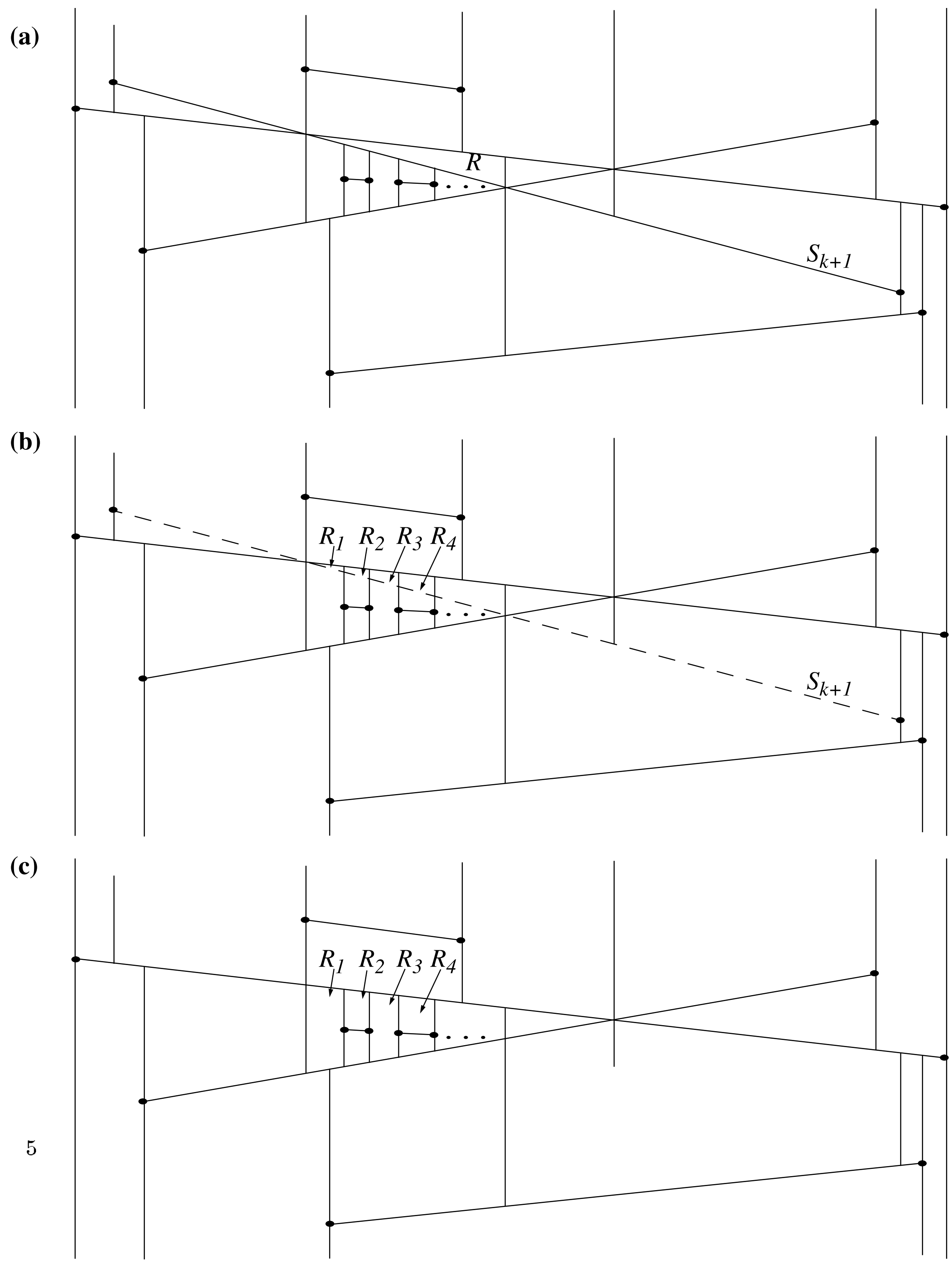

Figure 2: Figure 2:Trapezoidal Decomposition update

To summarize, the expected cost of maintaining a trapezoidal decomposition over a random sequence, ignoring the endpoint location, is in the order of

$$n + \frac{I}{n^2} \sum_{i=1}^{|\bar{u}|} r(k) = O(n+1),$$

where I is the number of intersections among the segments in N.

2.1 Point location

In the previous analysis, we ignored the cost of locating the endpoint in the updated segment. Lets consider a sequence of (N, δ), $\bar{u}$. Let N^{k} be the set of undeleted segments existing at time k. Here we introduce history(k). The history(k) will point to the trapezoids in $H(N^{\mathrm{k}})$. Therefore, history(0) consists of one node that corresponds to whole plane. Inductively, assume that history(k) has been defined. Now, consider the $(k+1)$th update of segment S_{k+1}. History($k+1$) is obtained from history(k) as follows:

1. Mark the leaves of history(k) corresponding to the trapezoids in the $H(N^{\mathrm{k}})$ that are destroyed during the $(k+1)th$ update as killed.

2. Allocate new nodes for the newly created trapezoids in $H(N^{\mathrm{k+1}})$.

3. Build a certain data structure link(S_{k+1}), which serves as a link between the decomposition $H(N^{\mathrm{k}})$.

If the $(k+1)th$ update is addition, every destroyed trapezoids in $H(N^{\mathrm{k}})$ has at most four children. Link(S_{k+1}) consists of pointers from the killed leaves of history(k) to their children.

If the $(k+1)$th update is deletion, the number of destroyed children need not be bounded by any number. The children of any killed trapezoids can be ordered from left to right. We let the link(S_{k+1}) consist of linearly ordered list of children, one list for each killed leaf of history(k). With each such linearly ordered list, we maintain a search tree. Given any point p lying within a killed trapezoid in $H(N^{\mathrm{k}})$, link(S_{k+1}), allows us to locate its child containing p in $O(log|N^{\mathrm{k}}|) = O(\log r(k))$ time.

Let p be a fixed query point. To locate p in $H(N^{\mathrm{k}})$, we begin at the root of history(k), which corresponds to the whole plane. We descend to its unique child containing p and keep repeating this process until we reach a leaf of history(k). This leaf corresponds to the trapezoid in $H(N^{\mathrm{k}})$ containing p. Each descent takes at most logarithmic time. The cost of locating a point would be proportional to the length of the search path in history(k), ignoring a logarithmic factor.

Analysis of end point location Given $j \leq k$, we define a 0-1 random variable V_j. V_j is defined to be 1 if the trapezoid in $H(N^{\mathrm{j}-1})$ containing the query point p is destroyed during the jth update. Otherwise, it is defined to be 0. Here we find out that the expected length of the search path is of the order of $\sum_{j=1}^{|\bar{u}|} \frac{1}{r(j)}$

If the signature is weakly monotonic signature, where $r(j) = \Omega(j)$, the expected length of the search path $O(\log k)$. This is the case for most update sequences but the worst case is $O(k)$.

Theorem The expected cost of maintaining a trapezoidal decomposition and the history over a random sequence id of the order of

$$n + \frac{I}{n^2} \sum_{i=1}^{|\bar{u}|} r(k) = O(n + I), \tag{1}$$

where I is the number of intersections among the segments in N. This ignores the cost of point location during addition. the expected cost of point location at time k is $O(\log k \sum_{j \leq k} 1/r(j))$ If δ is weakly monotonic, it is $O(log^2 k)$ for most sequences.

Therefore, the performance for this algorithm is not satisfactory if δ is not weakly monotonic. Since the history can become long and dirty if it contains too many segments that have already been deleted.

3 Voronoi diagrams

Let N^{k} denote the set of undeleted sites existing at time k. Let $G(N^{\mathrm{k}})$ denote their Voronoi diagram. The goal is to maintain $G(N^{\mathrm{k}})$. So if we have an update, $(k+1)$th, and a segment, $S = S_{k+1}$ to be the site involved in the update. The goal is to update $G(N^{\mathrm{k}})$ to $G(N^{\mathrm{k+1}})$ very quickly.

Update Lets consider when the update is deletion of segment S from N^{k}. Let Q_1, Q_2, Q_3, $...Q_j$ be the sites labeling the regions adjacent to the Voronoi region we want to delete, vor(S). The Voronoi diagram of $N^{\mathrm{k+1}}$ can be obtained from Voronoi diagram as follows

1. Separately compute the Voronoi diagram of the sites Q_1, Q_2, Q_3, $...Q_j$, which can be done in $O(j \log j)$. This decomposes the vor(S) into $O(j)$ convex regions.

2. Remove the j edges bounding vor(S).

The result will be the Voronoi diagram of $N^{\mathrm{k+1}}$. The structural change in the Voronoi diagram, as the update happen, is proportional to j and ignoring a logarithmic factor. The structural change is the number of newly created and destroyed edges and veracities. **Analysis** We have seen that the structural change during the deletion of site $S \in N^{\mathrm{k}}$ is $O(j)$. Where j is the size of $vor(S)$ which is proportional to the number of vertices in $G(N^{\mathrm{k}})$ that are destroyed during deletion. A vertex v is destroyed during deletion iff the deleted site is one of the three sites defining v. Summing up the structural

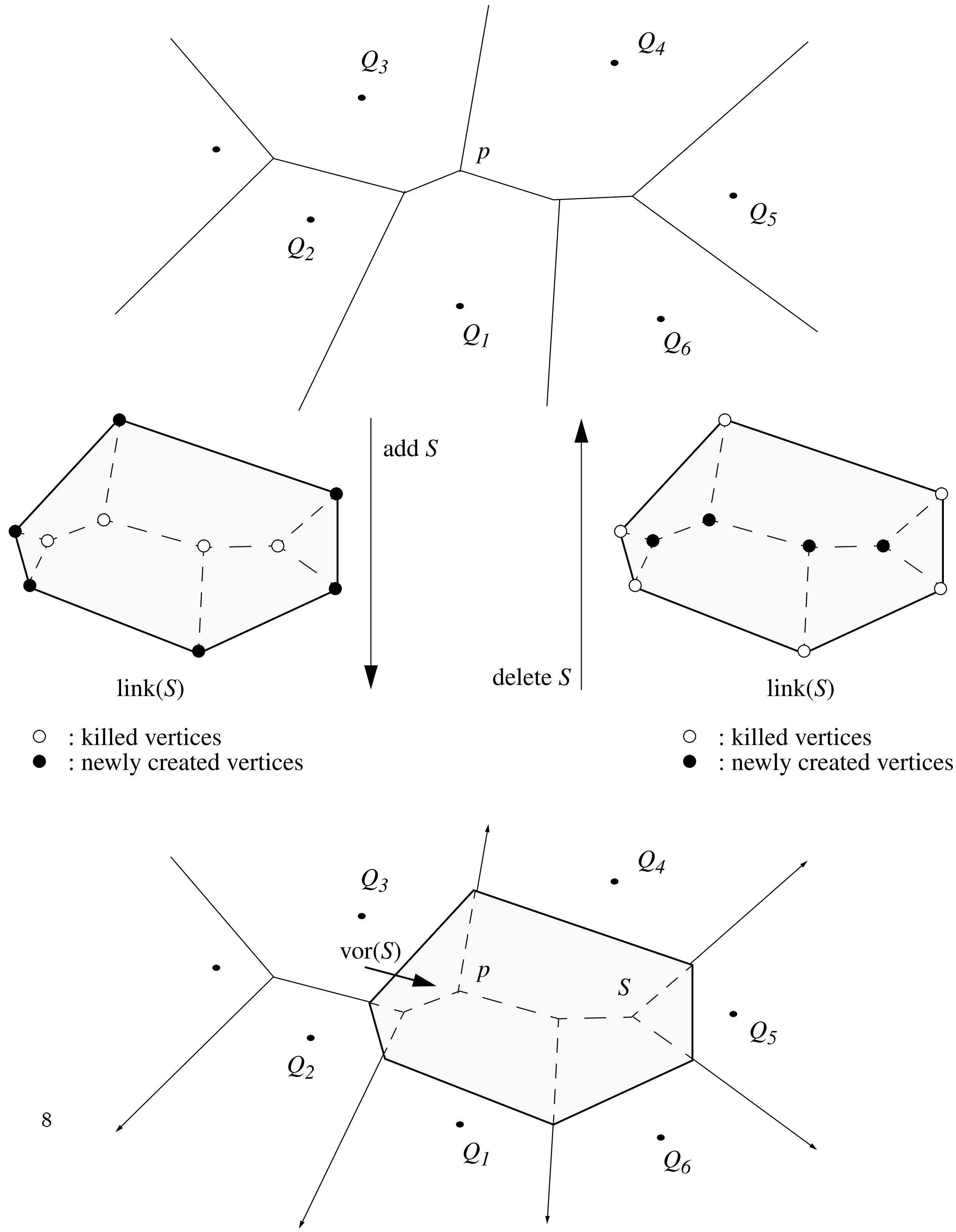

Figure 3: Figure 3: Vorornoi Diagrams update

change by letting the deleted site $S^{\mathrm{k}+1}$ range all over all sites in N^{k}. It follows that the sum, ϕ, is proportional to the size of $G(N\mathrm{k})$. Since the size of planar Voronoi diagram is linear in the number of sites, then the sum, ϕ, of structural change is $(|N^{\mathrm{k}}|)$. Since the deleted site S_{k+1} is randomly chosen from N^{k}, the expected structural change is $\phi/|N^{\mathrm{k}}| = O(1)$. If the update is addition, we use the backward analysis. We find that the expected structural change during each update in a random (N,δ)-sequence is $O(1)$. Therefor, the total expected structural change is $O(n)$.

3.1 Conflict search

Conflict search, here, means locating the conflicting vertex during the addition of new site. We use the history technique to help us the conflicting vertices.

1. Mark the leaves of history(k) pointing to the destroyed vertices in $G(N^{\mathrm{k}})$ as killed.

2. Locate new nodes for the newly created vertices of $G(N^{\mathrm{k}+1})$.

3. Link the killed and the newly created nodes according to their adjacencies.

Suppose we know a leaf of history(k) in conflict with the updated site, I. We descend to a leaf of history($k+1$) in conflict with I as follows. The killed and the newly created nodes of link(S_{k+1}) in conflict with I form its connected subgraph. We start the search at the known conflicting leaf of history(k) and restrict the search to the above connected subgraph of link(S_{k+1}). We stop the search as soon as we reach a newly created node in conflict with I. In the worst case, the cost of the search is proportional to the size of the connected subgraph.

Using history in conflict search

1. Locate a node of histroy(1) in conflict with S. This takes $O(1)$ time.

2. Assume that this node is killed at time j.

3. Using link(S_j), locate a node created at time j in conflict with S.

4. Repeat this process until we reach a leaf of history(k). This will correspond to a vertex of $G(N^{\mathrm{k}})$ in conflict with S.

In the worst case, the cost of this conflict search is proportional to the total number of nodes in history(k) in conflict with S.

Analysis If the update sequence under consideration is random, the expected number of nodes in history(k) in conflict with S_{k+1} is $O(\log k)$. This assumes that the signature is weakly monotonic.

Assuming that δ is weakly monotonic, the expected cost of maintaining a Voronoi diagram over a random sequence is $O(n \log n)$. The expected cost of conflict search through history(k) during the addition of S_{k+1} is $O(\log k)$.

4 History and configuration spaces

In this section, I will briefly go over the notations and the analysis because I think that all information presented in this section are repetitive from previous sections. Consider dynamically maintaining a geometric complex $H(M)$. Let $\bar{u}$ be any sequence. The history at time k during the execution of $\bar{u}$ is data structure that results by remembering the differences between the successive partitions in the sequence. The leaves of history(k) correspond to the faces of $H(N^{\mathrm{k}})$ of all or some specified dimensions. It is assumed that these faces are configurations that are defined by abounded number of objects in N^{k}.

4.1 Expected performance

The total number of nodes in the history of $\bar{u}$ is bounded by the total structural change during $\bar{u}$. Let $\Pi^0(N^{\mathrm{k}})$ be the set of active configurations at time k. We want to get the number of configurations in $\Pi^0(N^{\mathrm{i}})$ that are destroyed during update. First calculate the expected value, conditional on a fixed N^{i}. A configuration $\sigma \in \Pi^0(N^{\mathrm{i}})$ can be destroyed iff the deleted objects is one of the most d triggers associated with σ. If we sum the number of destroyed configurations by letting the deleted object S ranges over all of N^{k}, then the sum is at most $d\pi^0(N^{\mathrm{i}})$. Where the size of $\Pi^0(N^{\mathrm{i}})$ is $\pi^0(N^{\mathrm{i}})$. Each object in N^{i} is equally likely to be deleted. Then the expected number of destroyed configurations in $\Pi^0(N^{\mathrm{i}})$, conditional on a fixed N^{i} is at most

$$d\pi^0(N^{\mathrm{i}})/|N^{\mathrm{i}}| = d\pi^0(N^{\mathrm{i}})/r(i).$$

The expected number of newly created configurations, conditional on a fixed N^{i}, is

$$\pi^1(N^{\mathrm{i}})/(|N^{\mathrm{i}}|) = \pi^1(N^{\mathrm{i}})/r(i).$$

Now lets assume that N^{i} is arandom sample of N of size $r(i)$. For $c \geq 0$, let $e^{\mathrm{c}}(r)$ denote the expected value of $\pi^{\mathrm{c}}(R)$, where R is a random sample of N of size r. Here π^{c} is the size of $\Pi^{\mathrm{c}}(R)$, the set of configurations in $\Pi(R)$ with level c. Note that $e^0(r)$ is just $e(r)$. The expected structural changes during the $(i+1)$th update of a random sequence, assuming that it is deletion is bounded by

$$\frac{de(r(i)) + e^1(r(i))}{r(i)}.$$

The expected structural change, conditional on a fixed N^{i+1} is bounded by

$$\frac{de(r(I+`)) + e^1(r(i+1))}{r(i+1i)}.$$

The expected value of the total structural change during a random sequence is

$$\sum_{k=1}^{|\bar{u}|} \frac{dc(r(k)) + e^1(r(k))}{r(k)}.$$

Here d is the maximum degree of a configuration in $\Pi(N)$. Let $a > 0$ be a fixed integer. Consider any configuration space $\Pi(R)$. Let d be the maximum degree of a configuration in $\Pi(R)$. Let r be the size of R. Let Q be a random sample of R of size $r/2$. Then

$$\pi^a(R) = O(2^{a+d}E[\pi^0(Q)]).$$

The probability that any given element in R occurs in Q is $1/2$. It follows that

$$prob(\sigma \in \Pi^0(Q)) \approx \frac{1}{2^{d(\sigma)+a}},$$

where $\approx$ denotes asymptotic equality with a constant factor ignored. This equation is exact if Q is chosen by flipping a fair coin. If Q is a random subset of size exactly $r/2$, then one can calculate the exact expression in termsof binomial coefficients. The expected structural change during a random a sequence is bounded, within a constant factor, by

$$\sum_{k=1}^{|\bar{u}|} \frac{e(r(k)) + e(r(k)/2)}{r(k)} = O\left(\sum_k \frac{\bar{e}(r(k))}{r(k)}\right).$$

If $\Pi(N)$ is trvial configuration space then the expected structural change during a random sequence is $O(\sum_k 1/r(k))$. In particular, it is $O(\log n$ if δ is weakly monotonic. For most update sequences, the bound is $O(\log n)$. Lets consider a random sequence. Suppose the $(k+1)$th update during this sequence is addition of S_{k+1}. Within a constant factor, by $\sum_{i\leq k} \bar{e}(r(I))/r(i)^2$. In particular, this is of the order of $\sum_{I\leq k} \bar{e}(i)/i^2$, if δ is weakly monotonic.

5 Rebuilding history

The performance of the previous history-based algorithm is not satisfactory enough if the signature of the update sequence is not weakly monotonic. The source of this problem is that history(k) can depend on several dead objects in the past whose number can exceed the number of active objects at time k. An object occurring in history(k) is called dead if the deletion of that object has already occurred in the update sequence $\bar{u}$. Rebuilding

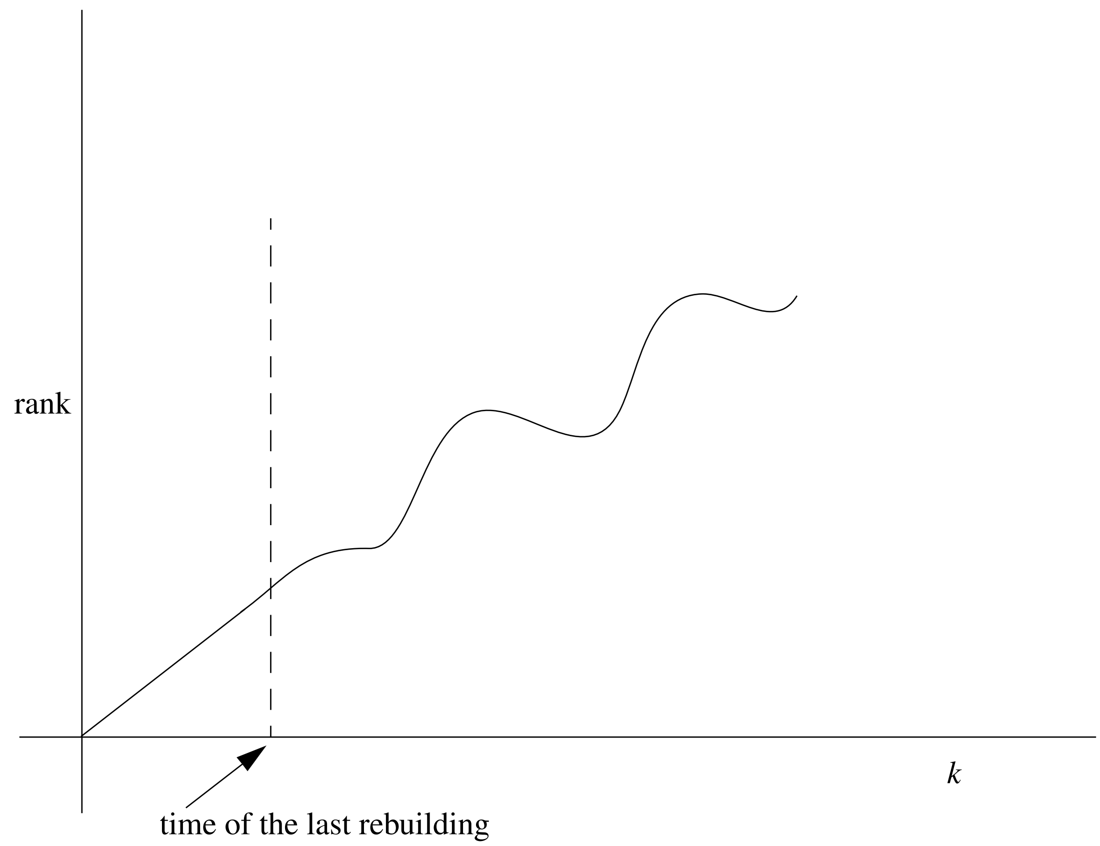

Figure 4: Figure 4: History rebuilding

history works as follow; every time the number of dead objects in the history exceeds, say, twice the number of active objects in the history, we rebuild history from scratch.

Rebuilding is done just by adding the currently active objects in the same order in which they where added previously, although it might be better to add them in random order. The resulting history is as if the dead objects were never encountered during the previous updates. We keep rebuilding history in this fashion whenever required. We want to know the cost of rebuilding. Rebuilding works by adding the currently active objects in the same order they were added originally. The number of these objects is clearly less than the number of dead objects. We could see that the total number of additions during all rebuildings is less than the length of $\bar{u}$. If $a(l)$ denote the number of active objects at the time of the lth rebuilding, then $\sum_l a(l) \leq |\bar{u}^2|$. For example, consider the dynamic maintenance of Voronoi diagrams. The lth rebuilding involves $a(l)$ additions of sites. The cost of rebuilding is expected to be $O(a(l) \log a)$.

Theorem The expected cost of maintaining a planar Voronoi diagram over a random sequence is $O(n \log n)$.

For trapezoidal decomposition, let $I(l)$ denote the number of intersections among the $a(l)$ segments that are added during the lth rebuilding. The expected cost of the lth rebuilding is $O(I(l) + a(l))$ for semidynamic case. The expected value of $I(l)$ is $O(a(l)^2 I/n^2)$, where I is the number of intersections among the segments in N .

Theorem The expected cost of maintaining a trapezoidal decomposition over a random sequence is $O(n \log n + I)$.

6 Deletions in history

History rebuilding works by removing dead objects periodically. Here we introduce a new technique. We want to minimize the number of dead objects at any time. With this method, the number of dead objects is 0.

Deletion in history works by deleting the object from history whenever we encounter a delete operation. Once this is done, the history should look as if the deleted objects where never added in the first place. Let M be the set of active objects at any given time. The size of M is denoted by m. Let M^j be the set of first j objects.

The history associated with M is denoted by history(M). Deletion from history(M) can be done through a series of operations called **rotations**. Let $S \in M$ be the object we want to delete from history(M). Let M' denote the set $M \setminus S$, with naturally inherited order. The goal is to obtain history(M') from history(M). Deletion of S from history(M) is carried out as follows.

1. Let l be the number of objects in M with order higher than S

2. For $I \leq l$ define $M(i)$ to be the ordered set obtained from M by moving S higher in order by i places.

3. We obtain history(M') from history(M) by moving S higher in order step by step until it has the highest order.

4. Deleting S is easy, then.

This approach has one problem, it always needs l distinct rotations. This implies as $\Omega(l)$ lower bound on the cost of deletion. We can get around this difficulty as follows. Let S' be the object just higher than S in order on B. Let $C = M(i+1)$ be the order obtained by switching the orders of S and S' in B. So S and S' commute if history(B) and history(C) are identical. Begin with history(M). Let (S') be the first object commuting with S with higher order than S. Rotate S and S' and keep on repeating this process until there is no noncommuting object with higher order than S. At this point perform the deletion. *How do we know S'?*. Let S and S' be any two objects in B such that S immediately precedes S' in the order. Let $C = B(S, S')$ be the order obtained from B by witching the orders of S and S'. Commutativity Condition: History(B) and history(C) are identical data structures if no descendent of S in history(B) is killed by S'.

The deletion algorithm can be further refined as follows. Suppose we want to delete an object S from history(M). Let Γ be the set of descendents of S in history(M). Let killer(Γ) be the priority queue of the killers of the nodes in Γ.

General Deletion Algorithm

1. Remove the first object S' in killer(Γ). By the commutativiity condition, S commutes with all objects in M, whose current ranks lie between S and S'.

2. Let $\Gamma(S')$ be the subset of Γ killed by S'. The subset $\Gamma(S')$ is available now. Usign this information, switch the orders of S and S' in M. Update history(M) via rotation.

3. Replace the subset $\Gamma(S')$ of Γ by the new descendents of S in history(M) and update the priority queue killer(Γ).

4. If killer(Γ) is not empty, go to step 1.

5. End deletion: Now the order of S in M can be assumed to be the highest. Remove link(S) and the descendents of S in history(M).

Analysis The cost of this deletion algorithm is controlled by the structural change. This is defined as the sum of the structural changes during all rotations. Its expected value is bounded in the following theorem.

Theorem Let $e(i, M)$ denote the expected size of $\Pi^0(B)$, where B is a random sample M of size i. Suppose the ordering of the objects in M is random. Then the total number expected structural change during a deletion of random object from history(M) is bounded within a constant factor, by $\frac{1}{m}\sum_{i=1}^{m} e(i,M)/i$. The sum $\sum_{i=1}^{m} e(i,M)/i$ is the expected size of history(M). What the theorem is saying is that the average structural change during a random deletion from M is approximately $(1/m)$th fraction of the expected size of history.

6.1 Convex polytopes

Let M denote the set of undeleted half-spaces existing at any given time. Let $H(M)$ denote the convex polytope formed by intersecting these half-spaces. Suppose we want to delete a half-space $S \in M$. This will be done by applying the general deletion scheme. First we bring S to the end of ordering on M via rotations and then perform deletion.

Let A be any ordered set of half-spaces. Let R and R' be any two consecutive half-spaces in the ordering on A. Assume that they do not commute. Let $\bar{A}$ be the order obtained from A by switching the orders of R and R'. Update the history(A) to history($\bar{A}$) as follows;

1. let j be the order of R in A. The order of R' is $j+1$.
2. Assume that $H(A^{j-1})$ is bounded.
3. Let $\Gamma(R')$ denote the set of descendents of R in history(A) that are killed by R'. The deletion algorithm ensures that $\Gamma(R')$ is available to us prior to rotation. The nodes in $\Gamma(R')$ correspond to the vertices of cap(R) that are adjacent to R and are in conflict with R'. These vertices lie on the boundary ∂R and they also lie within the component $\bar{R}'$

Analysis History(M) can be used for locating the optimum of linear function on $H(M)$. The cost of query location is $O(\log^2 m)$ for most ordering of the objects in M.
Theorem The expected cost of a random update on history(M) and the convex polytope $H(M)$ is $O(\log m)$. The cost of locating the optimum of linear function on $H(M)$ is $O(\log^2 m)$ for most update sequences.

6.2 Trapezoidal decompositions

This is another point locating algorithm for trapezoidal decompositions that is based on the technique of deletion via rotations. The query time here is better by a query factor. Let A be any ordered set of segments. Let R and R' be any two consecutive segments in the ordering on A. Assume that they do not commute. Let $\bar{A}$ be the order obtained from A by switching the orders

of R and R'.
History update

1. Let j be the order of R in A. The order of R' is $j+1$.

2. Let $\Gamma(R')$ denote the set of the descendents of R in history(A) that are killed by R'.

3. Let π denote the nodes corresponding to the trapezoids in $H(A^{j-1})$ in conflict with both R and R'. the nodes in π can be accessed through the parent pointers associated with the trapezoids in $\Gamma(R')$. this takes time proportional to $\sum_{\triangle}$ face-length($\triangle$), where $\triangle$ ranges over all trapezoids in $\Gamma(R')$.

Analysis The expected cost of rotation during a random deletion is bounded within a logarithmic factor by

$$\frac{1}{m}\sum_i 1+\frac{i}{m^2}I(m) \leq 1+\frac{I(M)}{m}.$$

The expected cost of random addition to M is bounded. It is $O(\log m + I(M')/m)$, where M' is the resulting set of segments after addition, and $I(M')$ denotes the number of intersection among the segments in M'. The logarithmic term reflects the cost of locating an endpoint of the added segment.
Theorem The expected cost of random deletion from history(M) is $O([1+I(M)/m]\log m)$. The expected cost of random addition to history(M) is $O(\log m + I(M')/m)$, where M' is the resulting set of segments after addition. It follows that the expected cost of executing a random sequence is $O(n\log n + I\log n)$, where I is the number of intersections among the segments in N. The cost of point location using history(M) is $O(\log m)$ for most update sequences.

7 Dynamic shuffling

All the previous techniques for point location queries provide no guarantee on the query time for all update sequences. For example, consider point location in trapezoidal decompositions. Let M denote the set of undeleted segments existing at any time. The cost of point location using history(M) is $O(\log m)$ on most update sequences. That is if the order of addition of segments in M were random, then the cost of point location would be $O(\log m)$ with very high probability. The order of additions is governed by the actual update sequences. This would be the same as saying that the cost of point location is $O(\log m)$ for all update sequences but a $a(1/g(m))th$ fraction of the update sequences, where $g(m)$ is a large degree polynomial.

This section introduces a technique called dynamic shuffling, which generalizes the dynamization technique underlying randomized binary trees. It will ensure good query cost with high probability for ever update sequence.

Let M denote the set of undeleted objects existing at any time. We recall how history(M) was defined. We imagined adding the objects in M in the same order as the actual update sequence. History(M) recorded the history of this sequence of addition. We choose a random ordering of objects in M. This order is completely independent of the actual order in which the objects in M were added. We now record the history of this imaginary sequence of additions.

Deletion of object S from shuffle(M), History(M) using the shuffle technique, is just deletion from history(M). First bring S to the end of the order and delete it.

Addition is different. Consider adding a new segment S to shuffle(M). Let M'=$M \cup S$. We choose a priority for S randomly and independently from the uniform distribution on the interval[0,1]. After this, the addition operation is just reversing of the deletion operation. First, imagine that S has the highest priority in M. Accordingly, we add S to the end just like we add S to history(M). This will yield shuffle(M') where S has the highest priority. After this we use rotation, like the deletion operation but backward direction until S is in the proper place in the priority order.

Analysis We want to get the expected cost of maintaining this structure during a random sequence. Since shuffle(M) records the history of random sequence we conclude;

Theorem the total expected structural change during the deletion of a random object from shuffle(M) is bounded, within a constant factor, by $(\frac{1}{m})\sum_{i=1}^{m} e(i,M)/i$.

Theorem The expected total structural change in rotations, as we pass from shuffle(M) to shuffle(M'), is bounded, by a constant factor, $(1/m')\sum_{i=1}^{m'} e(i,M')/i$. Here, it is assumed that each object in M' is equally likely to occr as an added object. Combining the the last two theorems;

Theorem The cost of maintaining shuffle(M) on a random sequence is

$$\sum_{j=1}^{|\bar{u}|} \frac{1}{r(j)} \sum_{i=1}^{r=j} \frac{e(i)}{i}, \text{ where } e(i) = e(i,N).$$