Dynamic Maintenance of Geometric Structures Made Easy*

Otfried Schwarzkopf
Institut für Informatik, Fachbereich Mathematik
Freie Universität Berlin
Arnimallee 2–6, W1000 Berlin 33, Germany

Abstract

The problem of dynamically maintaining geometric structures is considered. A technique is proposed that uses randomized incremental algorithms which are augmented to allow deletions of objects. A model for distributions on the possible input sequences of insertions and deletions is developed and analyzed using Seidel's Backwards Analysis. While the analysis and the generic algorithm are given in terms of Clarkson and Shor's abstract model, it is further shown how to apply this to maintain Voronoi diagrams, convex hulls, and planar subdivisions. A strikingly simple algorithm for the maintenance of convex hulls in any dimension is given. The expected running time is $\mathcal{O}(\log m)$ per update for dimensions 2 and 3, $\mathcal{O}(m \log m)$ for dimension $d = 4, 5$, and $\mathcal{O}(m^{\lfloor \frac{d}{2} \rfloor - 1})$ for dimension $d \geq 6$, where m is the number of points in the current set.

1 Introduction

Many problems in Computational Geometry — especially in practical applications — call for dynamically maintaining some geometric data structures, while the objects defining this structure are inserted or deleted from the structure. Typical examples are the maintenance of *planar subdivisions* (i.e. the trapezoidization of the plane induced by a set of line segments), or the maintenance of *convex hulls* or *Voronoi diagrams*, while the set of line segments (points, sites resp.) defining these is changed dynamically. In the classical, deterministic setting, the dynamic problems tend to be much more complicated than the static ones (i.e. the computation of planar subdivisions, convex hulls or Voronoi diagrams given the set of line segments, points or sites), even in the two-dimensional case, see e.g. the algorithms for planar point location [21, 4, 9, 19, 22] or the work done on the maintenance of convex hulls [21, 20].

*This research was supported by the Deutsche Forschungsgemeinschaft under Grant Al 253/1–3, Schwerpunktprogramm "Datenstrukturen und effiziente Algorithmen" and by the ESPRIT I Basic Research Action of the European Community under contract No. 3075 (project ALCOM). Furthermore, part of this research has been done while the author visited Utrecht University.

However, practitioners soon discovered the fact that the so-called *randomized incremental algorithms* can be abused to deal with dynamic problems. Formally, a randomized incremental algorithm to compute some structure $\mathcal{A}(S)$ for a set $S = \{p_1, \ldots, p_n\}$ of n input objects is an algorithm which obtains a random permutation of the set S and incrementally adds one object of S after the other in this sequence, maintaining $\mathcal{A}(\{p_1, \ldots, p_i\})$ during the course of the computation. It is then possible to show that the expected time required to add all n objects is nicely bounded—where the expectancy does not depend on an input distribution, but only on the choice of the random permutation.

Ever since the introduction of randomized incremental construction into computational geometry by Clarkson and Shor [6] and Mulmuley [12], the paradigm has become very popular in the field, since it often leads to elegant, simple algorithms, lending themselves to implementation easily. Randomized incremental algorithms have been used for the computation of planar subdivisions induced by line segments (and generalizations thereof) [6, 12, 14, 25, 3, 1], for the computation of convex hulls [6, 24, 1], of Voronoi diagrams and Higher Order Voronoi diagrams [11, 10, 2, 15], for hidden surface removal [13], and for linear programming and the computation of smallest enclosing ellipses [24, 26].

In the first algorithms, such as those by Clarkson and Shor [6], it was necessary to maintain a *conflict graph* between the structure $\mathcal{A}(\{p_1, \ldots, p_i\})$ and the not yet inserted points $\{p_{i+1}, \ldots, p_n\}$. It was thus necessary to know the whole set S in advance. However, more recent algorithms got rid of this restriction. In fact, most of the above-mentioned papers describe *on-line* algorithms in the sense that they need not know the objects $\{p_{i+1}, \ldots, p_n\}$ in stage i, i.e. when computing the structure $\mathcal{A}(\{p_1, \ldots, p_i\})$.

And this is precisely what makes randomized incremental algorithms amenable for use in a dynamic setting. In fact, such on-line randomized incremental algorithms can be viewed as deterministic algorithms that incrementally add objects to some structure. The analysis then turns into an average-case analysis showing bounds on the av-

erage running time, where the average is now taken over a distribution on the possible input sequences. The only requirement on this distribution is that all permutations of the same set are equally likely, a much more sensible assumption than to require that points are drawn from some space with some fixed (in most cases uniform) distribution. The reader might wish to compare this view on randomized incremental algorithms with the difference between Quicksort and Randomized Quicksort. In fact, these "*Unrandomized*[1] randomized incremental algorithms" relate to randomized incremental algorithms in exactly the same way as Quicksort relates to Randomized Quicksort.

In view of this, it seems quite permissible to exploit such algorithms in practical situations: They are simple, efficient in practice, and we have a theoretical model (namely a distribution on the input sequences where every permutation of the same set is equally likely) which allows us to analyze them. However, the situation has been dissatisfying when deletions are concerned. In several cases, it is quite obvious how to perform deletions of objects in such "unrandomized" algorithms, and they seem to be used and perform quite well in practice. However, a theoretical model and an analysis of this kind of algorithms is missing. In this paper we define a model for distributions on the set of possible input sequences, consisting of insertions and deletions. Again, the user will be free to chose the base set $\mathcal{U}$ on which the algorithm will operate, and she will also be free to chose the order in which insertions and deletions follow each other. When performing an insertion, we require that every element currently not in the set of objects is added with equal probability, and for deletions we require that every object currently present is removed equally likely. Note that this model is stronger than to assume that the decision between insertions and deletions is done randomly. In fact, bounds on such a distribution of input sequences follow easily from ours.

We then analyze "unrandomized" randomized incremental algorithms in Clarkson and Shor's abstract setting [6]. We obtain generic algorithms for the abstract maintenance problem which perfectly generalize the results on randomized incremental algorithms in [6]. Furthermore, we show how to instantiate this generic algorithm to maintaining convex hulls in any dimension, and to maintaining planar subdivisions. Although we phrase our results within the abstract setting of [6], our proofs are completely different and often much simpler than those given there. In fact, one of the main contributions of the present paper is a new analysis of the expected size of the conflict graph in randomized incremental algorithms, based on Seidel's backwards analysis.

[1] Note that "unrandomized" is the opposite of "randomized" such as in "Randomized Quicksort". It has nothing to do with "derandomization" as considered elsewhere.

A rather similar model for sequences of insertions and deletions has independently been defined by Mulmuley [16, 17] (his definition is slightly different in that it does not allow that an object that has been deleted is added again). However, Mulmuley's paper aims in a different direction: While we are generalizing randomized incremental algorithms to include deletions, he generalizes *randomized search trees* to higher dimensions. His results are stronger, but they also use much more complicated data structures, while the methods used in the present paper surprise by their sheer simplicity.

Our paper is split in several sections as follows: In Section 2 we define our model for sequences of insertions and deletions, and provide an interesting view on it by considering the probability space of all possible sequences as a set of paths in a certain graph. This graph will make it quite obvious how to apply Backward Analysis to our problems.

Since our results will be given in the abstract setting by Clarkson and Shor, (i.e. in terms of their τ-function, which describes the expected complexity of a structure defined by a given number of objects), it is easy to lose track of our actual algorithms. Therefore, we first demonstrate our techniques on one problem: In fact, in Section 3 we show that a planar Voronoi diagram can be maintained with $\mathcal{O}(\log m)$ expected time for insertions and deletions, where m is the number of sites in the current stage.

In Section 4 we restate our results in Clarkson and Shor's abstract setting. Since most ideas have already been presented in the section before, we can content ourselves to filling in some points that get more complicated in the abstract setting (or higher dimensions).

Finally, in Section 5 we give two applications of the abstract results. As a trivial example, where we can instantiate the abstract assumptions quite easily, we show that planar subdivisions can be maintained with $\mathcal{O}(\log^2 m)$ expected time per insertion and deletion. Point location queries take time $\mathcal{O}(\log^2 m)$ with high probability. Again, m is the number of line segments currently in the structure.

As a more interesting example, we show how to maintain the intersection of halfspaces in any dimension with expected time $\mathcal{O}(m^{\lfloor \frac{d}{2} \rfloor - 1})$ per insertion or deletion (the actual result is in terms of the τ-function, so the running time decreases accordingly when the halfspaces are nicely distributed). Since n insertions can be used to build a convex polytope of complexity $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$, this expected running time is optimal (for less than five dimensions, the expected time per update is one log-factor off the optimal solution). By the usual lifting map, this implies algorithms for the maintenance of Voronoi diagrams.

Finally, we would like to add that we recently learned that Mulmuley [18] has obtained similar results. In particular, his generic algorithm is also based on the notion of history. His results can be found in these proceedings as well.

2 A randomized model for the dynamic setting

We wish to give a randomized model for sequences of insertions and deletions. Related models have independently been considered by Mulmuley [16, 17, 18], Devillers et al. [7], and Clarkson et al. [5].

So, let $\mathcal{U}$ be a set of U *objects* (For ease of presentation, we assume that $\mathcal{U}$ is a finite set. However, all our results can be generalized to an infinite universe). For every subset $S \subset \mathcal{U}$ we define $\mathcal{A}(S)$, the structure we are interested in. For definiteness, you might wish to think of $\mathcal{U}$ as a set of point sites in the plane, and of $\mathcal{A}(S) = Vor(S)$ as the Voronoi diagram of S.

Now, let $\delta \in \{+,-\}^*$ be a (possibly infinite) string over the alphabet $\{+,-\}$. This string is going to denote the sequence of insertions and deletions: $+$ denoting insertion and $-$ denoting deletion. In our model, the user is free to choose $\mathcal{U}$ and δ. Then, the algorithm will do the following: It starts with a set $S_0 = \emptyset$ and goes through stages 1, $2, \ldots, \ell$, where ℓ is the length of δ (if δ is an infinite string, definitions extend naturally). In stage i the i^{th} element of δ is considered. If this is a $+$-sign, then S_i is obtained from S_{i-1} by adding a random element of $\mathcal{U} \setminus S_{i-1}$. If the i^{th} element is a $-$, then S_i is obtained by removing a random element of S_{i-1}. We want to maintain $\mathcal{A}(S_i)$ during the course of this algorithm.

We are interested in giving bounds on the expected total structural change in $\mathcal{A}(S_i)$, averaged over all possible random choices for the insertions and deletions. Furthermore, we wish to find a generic algorithm to actually do these updates, assuming suitable algorithms for the basic subproblems.

We remark that when $\delta = +^U$, this models the usual randomized incremental construction.

To get some more intuition about this model, we take an alternative look on it. First, we observe that $n_i := \#S_i$, the size of S_i after stage i, is completely determined by the sequence δ; it does not depend on any random choices made by the algorithm. This implies that at stage i, all subsets of $\mathcal{U}$ of cardinality n_i can appear as S_i, and all these subsets are equally likely (by symmetry). We can thus consider a directed acyclic graph $\mathcal{G}$ which contains a node for every possible subset of $\mathcal{U}$ for every stage of the dynamic construction, see Figure 1.

Formally, we can define $\mathcal{G}$ as follows: It consists of levels $L_0, L_1, \ldots, L_\ell$, where level L_i contains $\binom{U}{n_i}$ nodes corresponding to the n_i-subsets of $\mathcal{U}$. In the following we identify these subsets with their corresponding nodes. Now, introduce an arc $e = (S_{i-1}, S_i)$ from node[2] S_{i-1} in level L_{i-1} to S_i in level L_i if and only if S_i can be obtained from S_{i-1} by the insertion (deletion, resp.) of one element p. Let $elem(e)$ denote this element p.

A path in $\mathcal{G}$, starting in L_0 and ending in L_ℓ, will be called a *randomized δ-update sequence* (the δ will be dropped when it is clear from the context). Our algorithm proceeds along some randomized δ-update sequence. Every randomized δ-update sequence is equally likely, and we wish to bound its expected cost, for some cost function *cost*. We will thus associate a cost $cost(e)$ with every arc e in $\mathcal{G}$. Our first interest, for instance, will be to bound the expected number of structural changes in $\mathcal{A}(S_i)$, so we will define $cost(S_{i-1}, S_i)$ as the number of structural changes between S_{i-1} and S_i. We then have to bound the expected sum of arc costs along a randomized δ-update sequence, averaged over all possible randomized δ-update sequences.

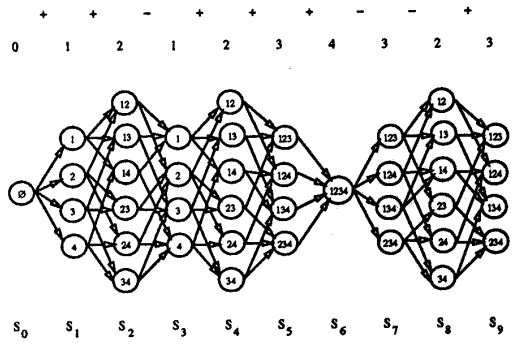

Figure 1: The graph $\mathcal{G}$ for $U = 4$, $\ell = 9$

3 Maintaining planar Voronoi diagrams

For concreteness, we consider the case of maintaining Voronoi diagrams in the plane first. Most of our ideas can be demonstrated on this example, and in the following sections we can content ourselves with filling in the points that become more complicated in the abstract setting.

So, assume that $\mathcal{U}$ is a set of U points in the plane, and let $\mathcal{A}(S) = Vor(S)$ denote the Voronoi diagram of S. We assume that the points in $\mathcal{U}$ are in general position, i.e. no four points lie on a circle, and no three points are colinear.

3.1 Bounding the total structural change

For an arc $e = (S_{i-1}, S_i)$ define $del(e)$ as the set of vertices v that are in $Vor(S_{i-1})$, but not in $Vor(S_i)$, and let $ins(e)$ denote the set of vertices in $Vor(S_i)$ but not in $Vor(S_{i-1})$. We then define $cost(e) := \#del(e) + \#ins(e)$ (the structural change in the Voronoi diagram induced by

[2]Note that we abuse notation and denote a node in $\mathcal{G}$ by S_i, the same symbol that denoted a random variable before.

the insertion (or deletion, resp.)—all other changes (edges, faces) are linear in $cost(e)$ as defined).

If we now introduce a random variable W_i, denoting the cost in stage i, we can express the expected total structural change as $E[\sum_{i\geq 1} W_i] = \sum_{i\geq 1} E[W_i]$ (by the additivity of expectancy). Furthermore, we can express $E[W_i]$ as

$$E[W_i] = \frac{1}{\#E_i} \sum_{e\in E_i} cost(e),$$

where E_i denotes the set of all arcs e connecting a node on level L_{i-1} with one on level L_i in $\mathcal{G}$. (Note that—due to symmetry—all arcs in E_i are equally likely).

We make the following

Observation 1 *The in- and outdegrees of nodes in $\mathcal{G}$ are completely determined by δ. In fact, the outdegree of a node $S_i \in L_i$ is $U - n_i$ if stage $i+1$ is an insertion, and n_i if it is a deletion. The indegree of $S_i \in L_i$ is n_i if stage i is an insertion, and $U - n_i$ if it is a deletion.*

To bound $E[W_i]$ we use two different arguments, depending on whether stage i is an insertion or a deletion. For the first case, we employ Seidel's *backward analysis*: Consider a fixed node $S_i \in L_i$. What is the expected value of $cost(e)$, averaged over all n_i arcs pointing into S_i? We use the fact that $cost(e) \in \mathcal{O}(\#ins(e))$, the number of vertices created by the insertion of $elem(e)$. Clearly, $ins(e)$ is the number of vertices of the Voronoi cell of $elem(e)$ in $Vor(S_i)$. Since the average size of a cell in $Vor(S_i)$ is constant, the expected cost of an arc pointing into S_i is also in $\mathcal{O}(1)$. Further averaging over all nodes in L_i shows that $E[W_i]$ is constant.

We now turn to the case of deletions. We basically employ the same argument, but use it the other way round (i.e. "backward backward analysis"). We consider a fixed node $S_{i-1} \in L_{i-1}$. What is the expected cost $E[cost(e)]$, averaged over all n_{i-1} arcs starting in S_{i-1}? Here we use that $cost(e) \in \mathcal{O}(\#del(e))$, and that $\#del(e)$ is the number of vertices of the cell of $elem(e)$ in $Vor(S_{i-1})$. Again, we average over all cells in $Vor(S_{i-1})$, so $E[W_i]$ is constant as well.

We thus obtain

Lemma 2 *The expected structural change occurring in a Voronoi diagram during a randomized δ-update sequence is $\mathcal{O}(1)$ per stage.*

3.2 Finding conflicts

Now, let's turn to the actual implementation of these operations. It is well known that a point p can be inserted into the Voronoi diagram of a set S in time linear in the number of vertices affected, as soon as we know some vertex v destroyed[3] by the insertion of p. We consider the problem of finding such a vertex below. The case of deletions is—in some sense—even simpler: Since we already know *where* a point p to be deleted *lies* within the diagram, we only have to refill the cell of p. This can be done in time linear in the number of destroyed vertices, e.g. by using a randomized incremental algorithm to compute the Voronoi diagram within this cell. Hence by Lemma 2, the work done by the algorithm—not counting the problem of determining a conflict for points to be inserted—is constant per stage.

The only remaining problem is the following: For a point p to be inserted into $Vor(S_i)$, find a vertex $v \in Vor(S_i)$ which is destroyed by the insertion of p (a *conflict*).

To do so, we first consider the usual technique of keeping the old versions of $Vor(S_i)$, i.e. when adding or deleting a point from $Vor(S_i)$ we do not really remove vertices, but mark them as *old*, and attach a pointer to them which points to the stage which causes their destruction. When a new point p has to be inserted, in stage j, say, we do the following: We find a vertex v_1 in $Vor(S_1)$ (in constant time) which is in conflict with p. Either v_1 still exists in the current diagram, or it has a pointer to a point q whose insertion (or deletion) destroyed v_1. Assume that q was inserted (deleted) in stage i. We perform a graph search on $Vor(S_{i-1})$, starting with v_1 (which is a vertex in $Vor(S_{i-1})$) and only considering vertices which are in conflict with both q and p. This is guaranteed to lead us to a vertex $v_i \in Vor(S_i)$ which is in conflict with p (see e.g. [10]). We then proceed with v_i as with v_1, until we find a conflict in the current Voronoi diagram.

We remark that for the case of $\delta = +^U$, i.e. a sequence of insertions only, this approach is pretty much the same as maintaining a conflict graph for all not-yet inserted points. In our setting, however, we cannot afford to do the latter, since for $\ell < U$, we do not know which of the points in $\mathcal{U}$ are eventually going to be inserted.

It remains to bound the running time of our technique. Fix some $2 \leq j \leq \ell$. If stage j is an insertion (of point p, say), we have to find a conflict for p in $Vor(S_{j-1})$. To do so, we *trace* a conflict for p through $Vor(S_1), \ldots, Vor(S_{j-1})$ as described above. Let W_{ij} be a random variable denoting the amount of work done (i.e., the number of edges traversed) to trace a conflict for p from $Vor(S_{i-1})$ to $Vor(S_i)$. The total time to find a conflict for p will then be $E[\sum_{i=1}^{j-1} W_{ij}]$. Now fix some $1 \leq i < j \leq \ell$. We want to bound $E[W_{ij}]$.

Consider first the case that stage i is an insertion, of point

[3] A vertex $v \in Vor(S)$ which is destroyed by the insertion of a point p is called a *conflict* of p, and p and v are said to be *in conflict*. This relation can be tested in constant time. In fact, $v \in Vor(S)$ and p are in conflict iff p lies within the circumcircle of the Delaunay-triangle dual to v

q, say. What is the amount of work W_{ij} to be done in this case? According to the above, it is bounded by the number of vertices in $Vor(S_{i-1})$ that are in conflict with both p and q. This implies that

$$E[W_{ij}] = E[\#\{v \in Vor(R) \mid v \text{ in conflict with } p \text{ and } q\}],$$

where the expectancy is taken over all choices of $R \subset \mathcal{U}$, $p, q \in \mathcal{U}$ with $|R| = n_{i-1}$, $p \neq q$, and $\{p, q\} \cap R = \emptyset$.

We change this last expression slightly, by fixing some $R' \subset \mathcal{U}$ with $|R'| = n_{i-1}+2 = n_i+1$ and considering $u = E[\#\{v \in Vor(R' \setminus \{p, q\}) \mid v \text{ in conflict with } p \text{ and } q\}]$, where the expectancy is taken over all $\{p, q\} \subset R'$. Now observe that $v \in Vor(R' \setminus \{p, q\})$ is in conflict with both p and q exactly if v is a close-type vertex of the order-3 Voronoi diagram of R' and the two points in R' closest to v are p and q.[4] To exploit this observation, let $V_3(R')$ denote the set of all close-type vertices of the order-3 Voronoi diagram of R'. To every vertex $v \in V_3(R')$ define $near(v)$ as the set of the two points in R' closest to v. With this notation, $u = E[\#\{v \in V_3(R') \mid near(v) = \{p, q\}\}]$, where the expectation is over all choices of $\{p, q\} \subset R'$. Since the order-3 Voronoi diagram has size $\mathcal{O}(n_i)$ (cf. [8]), and since there are $(n_i + 1)n_i$ possible choices for p and q, we find that $u \in \mathcal{O}(\frac{1}{n_i})$. Averaging over all $R' \subseteq \mathcal{U}$, we obtain $E[W_{ij}] \in \mathcal{O}(\frac{1}{n_i})$.

Now consider the case that stage i is a deletion. We employ similar arguments: $E[W_{ij}]$ can be bounded by $E[\#\{v \in Vor(R) \mid v \text{ in cell of } q \text{ and in conflict with } p\}]$ where the expectancy is taken over all $R \subset \mathcal{U}, \#R = n_{i-1}$, $q \in R, p \in \mathcal{U} \setminus R$. Again, we rewrite this as $E[\#\{v \in R' \mid v \text{ in conflict with } p \text{ and } q\}]$, with R', p, and q as in the case of an insertion, and get the same bound as in that case.

For the situation of randomized incremental construction, we are now done, since for $\delta = +^{\ell}$, we have $n_i = i$, so by summing over all pairs $1 \leq i < j \leq \ell$ we obtain a bound of

$$\sum_{1 \leq i < j \leq \ell} \mathcal{O}(\frac{1}{i}) = \mathcal{O}(\ell \log \ell),$$

so we have a new proof for the expected size of the conflict graph in randomized incremental construction.

In our case, however, there are two additional problems: Consider a situation where we are maintaining sets of maybe about 1000 points, and do so for several 10^6 updates. Clearly it is ridiculous to store the complete history of updates—even when the expected storage per update is only constant! Furthermore, for a sequence δ that keeps n_i considerably smaller than i—because deletions and insertions occur nearly equally often—, we don't have any reasonable bounds on the expected time to perform an insertion: in fact, if n_i remains bounded by a constant, an insertion in stage j costs $\Theta(1/n_i) = \Theta(1)$ work for every stage i, for $i = 1, \ldots, j-1$, resulting in $\Theta(j)$ update time. This is clearly unacceptable.

So, something is wrong with our approach of keeping the history of the randomized update sequence. Devillers et al. [7] suggest to deal with deletions by tampering with the history: When a point q is deleted from the current structure, they consider the total history of computation, and eliminate all traces of q from it. When a new point p searches for a conflict in the history, it will believe that q never existed.

This approach can complicate a randomized incremental construction considerably, thus losing its most desirable property. We suggest different solutions, which have the advantage of being quite simple in the implementation. In fact, we propose not to tamper with the past, but to get rid of it in certain intervals.

The key idea is that the history stored for searches should not be longer than the current size of the set, n_i. We will thus have to build a new history when the old history gets too long for the size of the stored set. A simple way to do this would be the following: Introduce a variable a_i, denoting the age of the current history in stage i. In the beginning, we set $a_0 := 0$ and increment a_i in every stage. After stage i, we then check whether n_i is at least, say, $a_i/2$. If it is not, we disregard the current history, and build a new one by reconstructing $Vor(S_i)$ from scratch using a randomized incremental algorithm (i.e. by a recursive call to the same algorithm, but only using insertions). This will give us a new history of length $a_i = n_i$, so we can proceed with this history until it gets again too long (or the set S_i too small). It can be checked that the work done to rebuild these new histories can be charged to the deletions, giving expected *amortized* time $\mathcal{O}(\log n_i)$ for stage i.

However, this implies that some stages in the randomized update sequence *must* take long—not only in the expected case, but at every execution of the algorithm (since the cardinality n_i does not depend on any random choices, so the stages when a reconstruction occurs are fixed). Since this is undesirable (it just does not fit into the framework of randomized incremental algorithms), we suggest a different solution.

3.3 A technique to rebuild the past

In fact, we nearly do what we indicated above: We keep a variable a_i, giving the age of the current history in stage i, and which is incremented in every stage. In the beginning,

[4] Recall that three points $\{r, s, t\} \subset S$ form a vertex v of the order-k Voronoi diagram of S iff there are exactly $k-1$ or $k-2$ points of S in the interior of the circle through r, s, and t. The former are called *close-type* vertices, the latter *far-type* vertices (this is not really important, we only use it to deduce that there is at most a linear number of these vertices, and a different proof of that fact in an abstract setting will be given in the next section anyway).

$a_i = i$. Before every deletion—in stage i, say—we check whether $n_i > \frac{a_i}{2}$. If this is not the case, we start reconstructing $Vor(S_i)$ from scratch, recursively calling the algorithm for a sequence of insertions of the points now in S_i. However, since we cannot afford to wait for the completion of this reconstruction, we continue processing the randomized update sequence, doing the reconstruction in the background—but six times as fast! In other words, every time we process one point in the sequence, we also process six points in the background reconstruction[5]. When the background process has reached the set S_i (this will happen in stage $i + n_i/6$), it continues by following the original sequence starting in stage i. Since it is six times as fast as the original process, the background process will reach the original process in stage $i + n_i/5$. At that point, the original process is discarded, and the background process takes over. Clearly, this technique only slows us down by a constant factor (in fact, a factor of seven), since we now process seven points while we processed one before. We only have to show that a background process never has to span another background process before it reaches the original process. But the background process reaches the old process after processing $n_i + n_i/5$ stages, and at least n_i of these are insertions, so in stage $(6/5)n_i$ of the new process there are at least $(1 - 1/5)n_i > \frac{1}{2}(6/5)n_i$ points.

Now consider the expected time to find a conflict for a point that has to be inserted in stage j. As shown above, we can bound that time by

$$\sum_{1 \le i < j} \mathcal{O}(1/n_i)$$

where stages are now numbered according to the current history. However, we now have that $n_i \ge \frac{4}{11} i$ (since we start a reconstruction when n_i falls below $i/2$, and then at most $n_i/5$ points are deleted until the reconstructed history takes over—so the worst stage is $i + (i/10)$ with $(1/2 - 1/10)i$ points), so $\mathcal{O}(1/n_i) = \mathcal{O}(1/i)$, and we can bound the expected time by $\mathcal{O}(\log j) = \mathcal{O}(\log n_j)$.

We have thus proven the following

Theorem 3 *Under a randomized δ-update sequence, the Voronoi diagram can be maintained with expected time $\mathcal{O}(\log n_i)$ per stage.*

We only state that the following—much simpler, albeit less general—scheme also solves the problem of too long histories:

> When deleting a point from S_{i-1}, toss a coin, and with probability $2/n_i$ forget the current history and construct a new one by reconstructing $Vor(S_i)$ with a recursive call to the algorithm (but now only processing insertions)

In fact, this technique can be implemented in such a fashion that the expected cost of deletions reduces to a constant, thus outperforming more complicated structures such as [7], or [5].

[5]Note that it does not suffice to give six times as much CPU time to the background process as to the foreground process, since—due to randomization—the foreground process might process its points too fast

4 A randomized update sequence in the abstract setting

In this section we are going to repeat our arguments for the abstract model introduced by Clarkson and Shor [6].

So, let $\mathcal{U}$ denote a set of *objects*, and let $\mathcal{F}$ denote an (abstract) set of *regions*. With every region v we associate a set $start(v) \subseteq \mathcal{U}$ and a set $stop(v) \subseteq \mathcal{U}$ with $\#start(v) \le b$ for some constant $b > 0$, and $start(v) \cap stop(v) = \emptyset$. An element of $stop(v)$ is called a *conflict* of v. We further define for $t \ge 0$ and $S \subset \mathcal{U}$

$$\mathcal{F}_t(S) := \{v \in \mathcal{F} \mid start(v) \subseteq S, \#\{stop(v) \cap S\} = t\},$$

and let further $\mathcal{F}(S) := \bigcup_{t \ge 0} \mathcal{F}_j(S)$, and set $\mathcal{A}(S) := \mathcal{F}_0(S)$. Furthermore, we let

$$\tau(n) := \max_{1 \le r \le n} E[\#\mathcal{A}(R) \mid R \subseteq \mathcal{U}, \#R = r].$$

We wish to maintain $\mathcal{A}(S)$ in the course of a randomized update sequence.

For the case of planar Voronoi diagrams considered before, $\mathcal{U}$ is a set of points in the plane, $\mathcal{F}$ is the set of all possible Voronoi vertices appearing in the Voronoi diagram of any subset of $\mathcal{U}$. (So a *region* is a *vertex* here! From now on we will only employ the abstract terms as used by Clarkson and Shor.) For $v \in \mathcal{F}$, $start(v)$ is the set $\{p_1, p_2, p_3\}$ of three points defining the vertex v (so $b = 3$), and $stop(v)$ is the set of points in $\mathcal{U}$ lying in the interior of the circumcircle of p_1, p_2, and p_3. In particular, a vertex v appears in the Voronoi diagram of a set $S \subseteq \mathcal{U}$ iff $start(v) \subseteq S$ and $stop(v) \cap S = \emptyset$, so the structure $\mathcal{A}(S)$ defined as above is in fact the Voronoi diagram of S.

4.1 The total structural change

Before we start, we give the following lemma, which follows from Theorem 3.2 in [6].

Lemma 4 [Clarkson, Shor] *The expected number of regions with a constant number of conflicts is of the same order as the expected number of regions without conflicts, i.e. for constant $t \ge 0$ we have*

$$E[\#\mathcal{F}_t(S)] \in \mathcal{O}(\tau(\#S)),$$

where the expectation is with respect to all choices of $S \subseteq \mathcal{U}$.

We can now start to bound the expected cost for the maintenance of $\mathcal{A}(S_i)$ during a randomized update sequence. As in Section 3 we start with the expected total structural change:

For an arc $e = (S_{i-1}, S_i)$, let $del(e) := \mathcal{A}(S_{i-1}) \setminus \mathcal{A}(S_i)$ and $ins(e) := \mathcal{A}(S_i) \setminus \mathcal{A}(S_{i-1})$, and let $cost(e) = \#del(e) + \#ins(e)$ be the structural change along arc e. As before, we let W_i denote the cost in stage i, and have

$$E[W_i] = \frac{1}{\#E_i} \sum_{e \in E_i} cost(e)$$

Assume first that stage i is an insertion. Consider a fixed node $S_i \in L_i$. What is the expected value of $\#ins(e)$, averaged over the arcs pointing into S_i? Since $v \in ins(e)$ iff $elem(e) \in start(v)$, this is

$$\begin{aligned} & \frac{1}{n_i} \sum_{p \in S_i} \#\{v \in \mathcal{A}(S_i) \mid p \in start(v)\} \\ = & \frac{1}{n_i} \sum_{v \in \mathcal{A}(S_i)} \#\{p \in S_i \mid p \in start(v)\} \\ \leq & \frac{b}{n_i} \#\mathcal{A}(S_i) \in \mathcal{O}(\frac{\tau(n_i)}{n_i}) \end{aligned}$$

What is $E[\#del(e)]$? Here we have $v \in del(e)$ iff $elem(e) \in stop(v)$. We define $f(v) := stop(v) \cap S_i$, so $v \in del(e)$ iff $v \in \mathcal{F}_1(S_i)$ and $f(v) = elem(e)$. Here, the expected value of $\#del(e)$ over all arcs pointing into S_i is bounded by

$$\begin{aligned} & \frac{1}{n_i} \sum_{p \in S_i} \#\{v \in \mathcal{F}_1(S_i) \mid f(v) = p\} \\ & = \frac{1}{n_i} \#\mathcal{F}_1(S_i) \in \mathcal{O}(\frac{\tau(n_i)}{n_i}) \end{aligned}$$

The case that stage i is a deletion is done analogously, and we obtain

Lemma 5 *The expected structural change in stage i of a randomized δ-update sequence is bounded by $\mathcal{O}(\tau(n_i)/n_i)$.*

4.2 Implementing the algorithm

If we wish to go further in our discussions, we have to make some assumptions now. We require that

(*i*) an insertion of object p along arc (S_{i-1}, S_i) can be performed in time (nearly) linear in the size of the structural change along (S_{i-1}, S_i), once a region $v \in S_{i-1}$ with $p \in stop(v)$ (a conflict of p) is known, and

(*ii*) a deletion of object p along arc (S_{i-1}, S_i) can be performed in time (nearly) linear in the size of the structural change along (S_{i-1}, S_i).

As in Section 3.2 we can then immediately conclude that the expected work required to maintain $\mathcal{A}(S_i)$ along a randomized update sequence is $\sum_{i \geq 1} \mathcal{O}(\frac{\tau(n_i)}{n_i})$, not counting the work for finding conflicts for objects to be inserted.

For some problems (e.g. Voronoi diagrams in three and more dimensions, convex hulls in at least four dimensions, arrangements of hyperplanes) finding a conflict can be done by other means in a time which is dominated by the above. For other problems, however, we might be bound to find conflicts in a fashion analogous to Section 3.2:

4.3 Finding a conflict

We assume that

(*iii*) we can find a conflict for an object p to be inserted in stage j by "tracing" it through stages 1 to $j-1$. The cost W_{ij} of tracing p from $\mathcal{A}(S_{i-1})$ to $\mathcal{A}(S_i)$ is bounded by $\#\{v \in \mathcal{A}(S_{i-1}) \mid v \notin \mathcal{A}(S_i) \text{ and } p \in stop(v)\}$.

As before, we see that $E[W_{ij}] = E[\#\{v \in \mathcal{A}(R) \mid p, q \in stop(v)\}]$, where the latter expectation is taken with respect to all choices of $R \subset U$, $p, q \in U$ with $\#R = n_{i-1}$, $p \neq q$, and $\{p, q\} \cap R = \emptyset$. We fix $R' \subset U$, $\#R' = n_i + 1$ and consider $u = E[\#\{v \in \mathcal{A}(R' \setminus \{p, q\}) \mid p, q \in stop(v)\}]$, where the expectation is over all choices of $p, q \in R'$. If we define $f(v) := stop(v) \cap R'$, we have that $v \in \mathcal{A}(R' \setminus \{p, q\})$ with $p, q \in stop(v)$ exactly if $v \in \mathcal{F}_2(R')$ and $f(v) = \{p, q\}$, so $u = E[\#\{v \in \mathcal{F}_2(R') \mid f(v) = \{p, q\}\}]$. By Lemma 4 we have $\#\mathcal{F}_2(R') \in \mathcal{O}(\tau(n_i + 1))$, which implies $u \in \mathcal{O}(\frac{\tau(n_i)}{n_i^2})$. We conclude that $E[W_{ij}] \in \mathcal{O}(\frac{\tau(n_i)}{n_i^2})$.

We thus have

Lemma 6 *Under assumption (iii), the conflict finding step for stage j of a randomized update sequence can be implemented with expected time*

$$\sum_{1 \leq i < j} \mathcal{O}(\frac{\tau(n_i)}{n_i^2})$$

Note that the total time for a randomized update sequence of length ℓ is bounded by

$$\sum_{1 \leq i < j \leq \ell} \mathcal{O}(\frac{\tau(n_i)}{n_i^2}) \leq \mathcal{O}(\ell) \sum_{i=1}^{\ell} \frac{\tau(n_i)}{n_i^2},$$

a result that perfectly generalizes Lemma 3.9 in [6].

For degenerate sequences δ, n_i can remain quite small even for big i, which gives a bad bound in the above formula if $\tau(n_i) \in o(n_i^2)$. We verify that the technique suggested in Section 3.3 can be employed in this abstract setting as well,

and we can thus deduce that we can always force $n_i \geq \frac{4}{11}i$, where i is now the length of the current history, so

$$\mathcal{O}\Big(\sum_{1\leq i<j} \frac{\tau(n_i)}{n_i^2}\Big) = \mathcal{O}\Big(\sum_{1\leq i<j} \frac{\tau(i)}{i^2}\Big)$$

4.4 Point location queries

In many situations, we wish to use our structure to perform some queries that cannot easily be expressed with the notions introduced above.

For instance when maintaining the bottom-vertex triangulation of $Vor(S_i)$, we wish to be able to do queries of the form: given a point x, find the bv-simplex in $Vor(S_i)$ containing x. This is *not* the same as a conflict for x, since the latter only requires that x lies in the circumcircle of the simplex. Another example is the maintenance of arrangements of hyperplanes. Here, objects are hyperplanes and regions are bv-simplices of arrangements of hyperplanes. The typical query, however, uses *points* as query objects, so this cannot be expressed by the abstract notion. In this section we set out to fill this gap by introducing an abstract notion of query object.

So, let Φ be a set of (abstract) *query objects*. We assume a function $loc_\phi(S)$ which associates with $\phi \in \Phi$ and $S \subset \mathcal{U}$ a region $loc_\phi(S) \in \mathcal{A}(S)$. We require

(*iv*) $loc_\phi(S) \in \mathcal{A}(S') \implies loc_\phi(S') = loc_\phi(S)$

We wish to supplement our algorithm in a fashion which makes it possible to find the region $loc_\phi(S_i)$ for any query object ϕ at any stage i of the randomized update sequence fast. We further assume

(*v*) Given $v \in \mathcal{A}(S)$ and a $\phi \in \Phi$, then we can decide in constant time whether $loc_\phi(S) = v$

(*vi*) Given $loc_\phi(S_{i-1})$, then $loc_\phi(S_i)$ can be found in time $T_r(n_i)$, for some suitable function T_r

We implement queries again by *tracing* ϕ through the current history, i.e. through stages 1 to j. We first find a region $v_1 \in \mathcal{A}(S_1)$ in constant time. Either $v_1 \in \mathcal{A}(S_j)$ (and then $loc_\phi(S_j) = v_1$ by (*iv*)), or v_1 is deleted in stage i. Since v_1 is then a region in $\mathcal{A}(S_{i-1})$ by (*iv*), we can find a region $v_i \in \mathcal{A}(S_i)$ in time $T_r(n_i)$ by (*vi*). We continue in the same fashion with v_i.

What is the expected number of stages i where we have to relocate $loc_\phi(S_i)$? Let Pr_i denote the probability that $loc_\phi(S_{i-1}) \neq loc_\phi(S_i)$. The expected number of stages to be considered is then $\sum_{1\leq i\leq j} Pr_i$.

Consider the case of an insertion in stage i, and fix some $S_i \in L_i$. When picking a random arc $e = (S_{i-1}, S_i)$ pointing into S_i, the probability that $loc_\phi(S_{i-1}) \neq loc_\phi(S_i)$ is bounded by $\frac{b}{n_i}$. This can be seen by observing that $loc_\phi(S_{i-1}) \neq loc_\phi(S_i)$ iff $loc_\phi(S_i) \notin \mathcal{A}(S_{i-1})$, which is equivalent to $elem(e) \in start(loc_\phi(S_i))$.

The case of a deletion is analogous: When $S_{i-1} \in L_{i-1}$ is fixed, the probability that for a random arc $e = (S_{i-1}, S_i)$ we have $loc_\phi(S_{i-1}) \neq loc_\phi(S_i)$ is $\leq \frac{b}{n_i}$, since this is equivalent to $elem(e) \in start(loc_\phi(S_{i-1}))$.

Further averaging over S_i (S_{i-1}, resp.), we find $Pr_i \in \mathcal{O}(\frac{1}{n_i})$. Using the technique of Section 3.3, we enforce $n_i \in \Theta(i)$, so we have here

$$\sum_{i=1}^{j} Pr_i = \sum_{i=1}^{j} \mathcal{O}(1/i) = \mathcal{O}(\log j) = \mathcal{O}(\log n_j),$$

so we have

Lemma 7 *Under assumptions (iv), (v), and (vi), abstract queries in stage j of a randomized δ-update sequence can be done in expected time $\mathcal{O}(\log n_j \cdot T_r(n_j))$.*

Note, however, that this is a rather weak result: For a fixed ϕ, we have proven that the expected query time (averaged over all randomized update sequences) is small. We cannot even guarantee that it is *possible* to construct a history (randomized update sequence) that gives such a small query time for *all* possible queries $\phi \in \Phi$!

However, we can demonstrate that this is in fact true. Using Chernoff bounds, we prove that the query time is small *with high probability* for *all* possible queries. For reasons of space, however, this proof is omitted in this version.

5 Applications

5.1 Maintaining planar subdivisions

We can maintain the trapezoidization of the plane induced by a set of non-intersecting line segments (see e.g. [12, 25, 1]) under a sequence of insertions and deletions of line segments. At the same time, we maintain a point location structure to answer queries of the form: Given a point in the plane, find the trapezoid in the current trapezoidization containing it.

We take $\mathcal{U}$ as a set of line segments in the plane, and define $\mathcal{F}$ as the set of all possible trapezoids occurring in the trapezoidization of any $S \subseteq \mathcal{U}$. We verify that every trapezoid $v \in \mathcal{F}$ is defined by at most four segments (that we call $start(v)$), and define $stop(v)$ as the set of all segments intersecting v. Then $\mathcal{A}(S) = \mathcal{F}_0(S)$ consists exactly of the trapezoids occurring in the trapezoidization of S, so maintaining $\mathcal{A}(S)$ is in fact what we wish to do.

Since we have $\tau(n) = \mathcal{O}(n)$ here, the expected structural change in any stage of a randomized update sequence is of constant size. Furthermore, a segment can be inserted

into the current trapezoidization in time linear in the size of the structural change, as soon as the trapezoid containing one of the endpoints of the segment is known. The same is true for deletions, so we have assumptions (*i*) and (*ii*), and updates can be done in expected constant time per stage, assuming a solution to the point location problem.

As for the latter, we define Φ as the set of all points in the plane, and $loc_\phi(S)$ as the trapezoid in $\mathcal{A}(S)$ containing ϕ. We can easily check that (*iv*) and (*v*) hold, and convince ourselves that it is possible to store the trapezoids in such a way that (*vi*) holds with $T_\tau(n_i) = \mathcal{O}(\log n_i)$. We can thus employ Lemma 7 and conclude with

Theorem 8 *Under a randomized update sequence, the trapezoidization of a set of line segments in the plane can be maintained with expected cost $\mathcal{O}(\log^2 n_i)$ per stage, where n_i is the current number of line segments. The structure can be used to answer point location queries in time $\mathcal{O}(\log^2 n_i)$ with high probability.*

5.2 Maintaining convex hulls in higher dimensions

Using our techniques, we can show how to maintain *convex hulls* under a sequence of insertions and deletions of points in any dimension.

So, let us fill our abstract setting with some flesh: We prefer to describe our algorithm in the dual setting, so we consider a set $\mathcal{U}$ of halfspaces in d-dimensional space, that we assume to lie in general position, i.e. no $d+1$ bounding hyperplanes intersect in a point. We define $\mathcal{F}$ as the set of all d-tuples of halfspaces (or maybe the vertex v they define). For $v \in \mathcal{F}$ we define $start(v)$ as the d-set of halfspaces defining v, and $stop(v)$ as the set of all halfspaces $h \in \mathcal{U}$ with $v \notin h$ (considering v as a point in d-space).

It is then easily verified that $\mathcal{A}(S)$ thus defined is the set of vertices of the intersection of the halfspaces in S.

Since we have $\tau(n) \in \mathcal{O}(n^{\lfloor \frac{d}{2} \rfloor})$, the expected structural change in stage i of a randomized update sequence is $\mathcal{O}(n_i^{\lfloor \frac{d}{2} \rfloor - 1})$ by Lemma 5.

We further consider how to implement the maintaining operations. From [6] we borrow the operation which inserts a halfspace in time bounded by the structural change, thus fulfilling assumption (*i*). A conflict can be found in time $\mathcal{O}(n_i)$ using randomized linear programming (see [24]), and for $d \geq 4$ this time is dominated by $\mathcal{O}(n_i^{\lfloor \frac{d}{2} \rfloor - 1})$. For two and three dimensions we use the approach of Section 4.3 and verify that assumption (*iii*) can be fulfilled in the same way as in Section 3.2, giving time $\mathcal{O}(\log n_i)$ for the conflict finding step.

The only remaining problem is to show how to perform deletions in time bounded by $\mathcal{O}(n_i^{\lfloor \frac{d}{2} \rfloor - 1})$. When a halfspace h is deleted from S_{i-1}, the polytope $\mathcal{A}(S_i)$ can be constructed by adding some "cap" C to the polytope $\mathcal{A}(S_{i-1})$. In fact, this cap C is the intersection of all halfspaces in S_i with h^-, where h^- is the closure of the complement of h.

We could now employ Seidel's algorithm [23] which can construct this cap in time $\mathcal{O}(n_i^2 + K \log n_i)$, where K is the complexity of the cap (and thus the size of the structural change in stage i). For $d \geq 6$, the n_i^2 term is dominated by the structural change. Furthermore, Mulmuley has shown [18, 17] how to get rid of this term for the case of Voronoi diagrams.

However, we believe that this approach is overly complicated in our setting, and suggest the following, quite simple, solution: To delete a halfspace h from $\mathcal{A}(S_{i-1})$, construct the cap C by a randomized incremental algorithm (i.e. a recursive call to the same algorithm, performing insertions only), adding the halfspace h^- first, and then a random permutation of the halfspaces in S_i.

We now analyze this technique. From Lemma 5 and Lemma 6 we know that we can construct C in $m := n_i + 1 = n_{i-1}$ stages, taking time $\mathcal{O}(m) \sum_{1 \leq r \leq m} \frac{\gamma(r)}{r^2}$, where $\gamma(r)$ is the function τ for this subproblem. In fact, $\gamma(r)$ is the expected complexity of the intersection of $h^- \cup R$, where $R \subseteq S_i$ is a sample of S_i of size r. To bound $\gamma(r)$, we fix S_{i-1} and observe that $h \in S_{i-1}$ has been chosen randomly, so

$$\begin{aligned}
\gamma(r) &= \frac{1}{m} \sum_{h \in S_{i-1}} \frac{1}{\binom{m-1}{r}} \sum_{\substack{R \subseteq S_{i-1} \setminus h \\ \#R = r}} \#\mathcal{A}(R \cup h^-) \\
&= \frac{1}{m} \frac{1}{\binom{m-1}{r}} \sum_{h \in S_{i-1}} \sum_{\substack{R' \subseteq S_{i-1} \\ \#R' = r+1 \\ h \in R'}} \#\mathcal{A}(R' \setminus h \cup h^-) \\
&= \frac{1}{m} \frac{1}{\binom{m-1}{r}} \sum_{\substack{R' \subseteq S_{i-1} \\ \#R' = r+1}} \sum_{h \in R'} \#\mathcal{A}(R' \setminus h \cup h^-) \\
&\leq \frac{1}{m} \frac{1}{\binom{m-1}{r}} \sum_{\substack{R' \subseteq S_{i-1} \\ \#R' = r+1}} \#\mathcal{F}_1(R') + \#\mathcal{F}_0(R')
\end{aligned}$$

The last equation follows because the last sum in the line before was a summation over all possible caps in R', thus giving the total number of vertices with exactly one conflict. By Lemma 4 (or the usual reference to k-sets) this is $\mathcal{O}(\tau(r))$. Further massaging the last line gives $\gamma(r) = \mathcal{O}(\tau(r)/r) \leq \mathcal{O}(r^{\lfloor \frac{d}{2} \rfloor - 1})$. The total time to construct the cap C is thus

$$\mathcal{O}(m) \sum_{1 \leq r \leq m} \frac{\gamma(r)}{r^2} \leq \mathcal{O}(m) \sum_{1 \leq r \leq m} \frac{r^{\lfloor \frac{d}{2} \rfloor}}{r^3}$$

$$= \begin{cases} \mathcal{O}(n_i^{\lfloor \frac{d}{2} \rfloor - 1}), & \text{if } d \geq 6; \\ \mathcal{O}(n_i \log n_i), & \text{if } 4 \leq d \leq 5, \\ \mathcal{O}(n_i), & \text{if } 2 \leq d \leq 3, \end{cases}$$

For $d \geq 6$ this is optimal, for $d = 4, 5$ it is quite satisfactory, but for two and three dimensions the running time is unacceptable. In that case we cannot afford to consider all n_i halfspaces when computing the cap C, and somehow have to identify those which can support a facet of C. To do so, we associate with every vertex $v_0 \in \mathcal{A}(S_i)$ a list $h(v_0)$ of all $h \in S_i$ with the property that the plane through v_0 parallel to the bounding plane of h supports $\mathcal{A}(S_i)$. It can then be seen that a halfspace $h \in S_i$ can only support a facet of C if h appears in the list $h(v)$ for some v incident to the deleted facet. By the methods used before we show that the expected number of these halfspaces is constant, which implies that we can compute the cap C in expected time $\mathcal{O}(\log n_i)$. Furthermore, the lists $h(v)$ can be maintained in expected time $\mathcal{O}(\log n_i)$.

We thus conclude with the

Theorem 9 *The intersection of d-dimensional halfspaces under a randomized update sequence of insertions and deletions of halfspaces can be maintained with expected time $\mathcal{O}(n_i^{\lfloor \frac{d}{2} \rfloor - 1})$ per stage for $d \geq 6$, $\mathcal{O}(n_i \log n_i)$ for $d = 4, 5$, and $\mathcal{O}(\log n_i)$ for $d = 2, 3$, where n_i is the number of halfspaces currently in the set.*

By the usual lifting map, this implies the analogous result for Voronoi diagrams in one dimension less.

Acknowledgments: I would like to thank Helmut Alt, Johannes Blömer and Emo Welzl for several helpful discussions, as well as for their comments on previous versions of this work.

References

[1] Jean-Daniel Boissonnat, Olivier Devillers, René Schott, Monique Teillaud, and Mariette Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete & Computational Geometry.* (to appear).

[2] Jean-Daniel Boissonnat, Olivier Devillers, and Monique Teillaud. A randomized incremental algorithm for constructing higher order Voronoi diagrams. *Algorithmica.* (to appear), an abstract has been published in the 2nd *Canadian Conference on Computational Geometry*, 1990.

[3] Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, Micha Sharir, and Jack Snoeyink. Computing a face in an arrangement of line segments. In 2nd *Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 441–448, 1991.

[4] Siu Wing Cheng and Ravi Janardan. New results on dynamic planar point location. In 31st *Symp. Foundations of Computer Science*, pages 96–105, 1990.

[5] Kenneth L. Clarkson, Kurt Mehlhorn, and Raimund Seidel. Four results on randomized incremental construction. (manuscript), 1991.

[6] Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4:387–421, 1989.

[7] Olivier Devillers, Stefan Meiser, and Monique Teillaud. Fully dynamic delaunay triangulation in logarithmic expected time per operation. In *Workshop on Algorithms and Data Structures*, 1991.

[8] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, 1987.

[9] Otfried Fries, Kurt Mehlhorn, and Stefan Näher. Dynamization of geometric data structures. In *Proc.* 1st *Symp. on Computational Geometry*, pages 168–176, 1985.

[10] Leo J. Guibas, Donald E. Knuth, and Micha Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. In *Proceedings of ICALP*, pages 414–431. Springer-Verlag, 1990. (also to appear in *Algorithmica*).

[11] Kurt Mehlhorn, Stefan Meiser, and Colm Ó'Dúnlaing. On the construction of Abstract Voronoi diagrams. *Discrete & Computational Geometry*, 6:211–224, 1991.

[12] Ketan Mulmuley. A fast planar partition algorithm, I. In 29th *Symp. Foundations of Computer Science*, pages 580–589, 1988.

[13] Ketan Mulmuley. An efficient algorithm for hidden surface removal. In *Proceedings of the ACM SIGGRAPH*, 1989.

[14] Ketan Mulmuley. A fast planar partition algorithm, II. In *Proc.* 5th *Symp. on Computational Geometry*, pages 33–43, 1989.

[15] Ketan Mulmuley. On levels in arrangements and Voronoi diagrams. *Discrete & Computational Geometry*, 6:307–338, 1991.

[16] Ketan Mulmuley. Randomized, multidimensional search trees: dynamic sampling. In *Proc.* 7th *Symp. on Computational Geometry*, pages 121–131, 1991.

[17] Ketan Mulmuley. Randomized multidimensional search trees: Further results in dynamic sampling. In 32nd *Symp. Foundations of Computer Science*, 1991.

[18] Ketan Mulmuley. Randomized multidimensional search trees: Lazy and dynamic shuffling. In 32nd *Symp. Foundations of Computer Science*, 1991.

[19] Mark Overmars. *The Design of Dynamic Data Structures.* Lecture Notes in Computer Science. Springer-Verlag, 1983.

[20] Mark Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23:166–204, 1981.

[21] F. P. Preparata and M. I. Shamos. *Computational Geometry.* Springer-Verlag, 1985.

[22] Franco Preparata and Roberto Tamassia. Fully dynamic point location in a monotone subdivision. *SIAM Journal on Computing*, 18:811–830, 1989.

[23] Raimund Seidel. Constructing higher dimensional convex hulls at logarithmic cost per face. In 18th *Symp. on Theory of Computing*, pages 404–413, 1986.

[24] Raimund Seidel. Linear programming and convex hulls made easy. In *Proc.* 6th *Ann. ACM Symp. Computational Geometry*, pages 211–215, 1990.

[25] Raimund Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, 1, 1991.

[26] Emo Welzl. Constructing smallest enclosing disks (balls and ellipsoids). Technical report, Freie Universität Berlin, 1991. (to appear).