Discrete Comput Geom 8:51–71 (1992)

Discrete & Computational
Geometry
© 1992 Springer-Verlag New York Inc.

Applications of Random Sampling to On-Line Algorithms in Computational Geometry*

Jean-Daniel Boissonnat,[1] Olivier Devillers,[1] René Schott,[2]
Monique Teillaud,[1] and Mariette Yvinec[3]

[1] INRIA, 2004 Route des Lucioles, B.P. 109,
06561 Valbonne Cédex, France
boissonn@alcor.inria.fr

[2] CRIN, B.P. 239, 54506 Vandœuvre, France

[3] LIENS, CNRS URA 1327, 45 rue d'Ulm,
75230 Paris Cédex 05, France

Abstract. This paper presents a general framework for the design and randomized analysis of geometric algorithms. These algorithms are on-line and the framework provides general bounds for their expected space and time complexities when averaging over all permutations of the input data. The method is general and can be applied to various geometric problems. The power of the technique is illustrated by new efficient on-line algorithms for constructing convex hulls and Voronoi diagrams in any dimension, Voronoi diagrams of line segments in the plane, arrangements of curves in the plane, and others.

1. Introduction

Randomized incremental construction is a new paradigm in computational geometry that has been successfully applied to a variety of problems [7], [12], [11]. These algorithms are rather simple, easy to code, and efficient in practice. Moreover, they do not require that the input data satisfy some probabilistic distribution but only that they are inserted in random order.

Clarkson and Shor [7] have given a general framework for the design and analysis of such randomized incremental constructions. In this framework, geometrical problems are stated in terms of objects, regions, and conflicts between objects

* This work has been supported in part by the ESPRIT Basic Research Action Nr. 3075 (ALCOM).

and regions. The input data form a set of objects, the regions are defined by small subsets of objects, and each region is in conflict with a (possibly empty) subset of the input objects. Precise definitions of objects, regions, and conflicts, of course, depend of the particular problem at hand. The output of the problem is assumed to be closely related to the set of regions defined by the input objects which are not in conflict with those objects. The algorithm incrementally constructs the set of regions defined by a current subset of the input objects which are not in conflict with that subset, and maintains in an additional data structure, called the conflict graph, the conflicts between the regions of the current construction, and the objects not yet introduced. When a new object is to be inserted, the regions in conflict with that object are replaced by new regions and the conflict graph is updated. This general framework has been shown to yield efficient algorithms to such problems as computing the convex hull or the Voronoi diagram of a set of points or computing the intersections of a set of line segments.

Because of the conflict graph, these algorithms are static in nature. A few attempts have been proposed to avoid the conflict graph and thus to obtain on-line algorithms in some specific applications [2], [3], [10]. The present paper extends these results and provides a general framework for on-line algorithms with good expected behavior when the objects are assumed to be introduced in random order. The basic idea, first introduced in [3], is the following. Instead of replacing old regions by new ones, we store, into a data structure called the Influence DAG, all the regions that appear at some stage of the construction as defined by the current subset of objects and without conflict with that subset. This structure is constructed on-line and designed to retrieve efficiently the conflicts between any new object to be inserted and the current set of regions.

Our main result bounds the cost of inserting a new object into a set of already inserted objects. Our results hold when averaging over all possible permutations of the set of inserted objects.

In the second part of the paper, we show that the major applications that can be solved using the Clarkson and Shor approach can also be solved using our framework. For all these applications, we obtain simple on-line algorithms with the same complexity as their static counterparts.

More specifically, the convex hull of a set of n points can be computed with logarithmic cost per added point in dimensions 2 and 3 and with cost $O(n^{\lfloor d/2 \rfloor - 1})$ per point in dimension $d > 3$. By the well-known correspondence between Delaunay triangulations (and Voronoi diagrams) in dimension d and convex hulls in dimension $d + 1$, we immediately deduce algorithms for constructing those structures. A more direct construction can also be derived and further extended to compute the Voronoi diagram of a set of line segments in the plane at the same cost. All these complexities are worst-case optimal for randomized algorithms.

Another application consists in computing an arrangement of planar curves (of bounded degree) in time $O(\log n + a/n)$ per insertion using $O(n + a)$ space, where a is the size of the output. This algorithm can also be used to locate a point in a planar subdivision in $O(\log n)$ query time after a preprocessing that requires $O(n \log n)$ time and $O(n)$ space.

This list is not exhaustive and several other applications will be mentioned.

2. The General Framework

2.1. *Definitions and Notations*

Following Clarkson and Shor [7], we formulate the geometric problems considered in this paper in terms of *objects* and *regions.*

The *objects* are members of a universe $\mathcal{O}$ and are the input data of the problem. Typically, objects are points, segments, lines, planes, or hyperplanes in a usual d-dimensional Euclidean space E^d.

The *regions* are defined by subsets of the universe $\mathcal{O}$ of cardinality less than a constant b. This assumes that there is a relation between the subsets of $\mathcal{O}$ of cardinality less than b and the regions; a region F is then said to be defined by a subset $\mathcal{X}$ of objects if $\mathcal{X}$ is in relation with F. Moreover, as in [7], we define *conflicts* between objects and regions. The subset of objects of $\mathcal{O}$ which are in conflict with a region is called the *influence range* of the region. Conflicts are defined in such a way that a region F defined by the subset $\mathcal{X}$ is not in conflict with the objects of $\mathcal{X}$.

Of course, the notion of conflict or, equivalently, the influence range of a region F has to be made precise for each specific problem. Still following Clarkson, we assume that, within this framework, the required geometric construction can be formulated as the following problem: given a finite subset $\mathcal{S}$ of the object universe $\mathcal{O}$, find all the regions that are defined by objects in $\mathcal{S}$ and that have no object of $\mathcal{S}$ in their influence range.

For a finite set of objects $\mathcal{S}$ we denote by $\mathcal{F}(\mathcal{S})$ the set of regions defined by the objects in $\mathcal{S}$. We call the number of objects of $\mathcal{S}$ that belong to the influence range of F *width* of a region F with respect to $\mathcal{S}$. Let $\mathcal{F}_j(\mathcal{S})$ be the subset of $\mathcal{F}(\mathcal{S})$ consisting of the regions that have width j with respect to $\mathcal{S}$.

2.2. *The I-DAG (Influence Directed Acyclic Graph)*

The different algorithms presented in this paper are incremental and introduce objects one by one. Let $\mathcal{S}$ be the set of objects which have already been introduced. At a given stage, the incremental algorithm inserts a new object in $\mathcal{S}$ and updates the set $\mathcal{F}_0(\mathcal{S})$ of regions of zero width defined by $\mathcal{S}$. This is performed through the maintenance of a dynamic structure called the Influence DAG (I-DAG for short) described below.

The I-DAG is a rooted directed acyclic graph whose nodes are associated with regions that, at some stage of the algorithm, have appeared as regions of zero width defined by the set of objects that have been introduced at that stage. Although the I-DAG is not, strictly speaking, a tree, we speak of leaves, fathers, sons, etc., in the obvious way. The nodes of the I-DAG associated with regions of current zero width with respect to $\mathcal{S}$ are marked. When a new object O is added to $\mathcal{S}$, one new node is created for each region in $\mathcal{F}_0(\mathcal{S} \cup \{O\})$ that is not a region of $\mathcal{F}_0(\mathcal{S})$.

The already-existing nodes are never deleted from the I-DAG but possibly unmarked. The new nodes are linked by edges to other nodes of the I-DAG. These edges are constructed in such a way that a new object O can be efficiently located in the structure; locating O means here to find all the nodes whose associated regions are in conflict with O. When a new node corresponding to a region F of $\mathscr{F}_0(\mathscr{S} \cup \{O\})$ is added to the I-DAG, we put edges between already-existing nodes and the new nodes, so that the influence range of F is included in the union of the ranges of its parents.

The I-DAG structure is characterized by the two following fundamental properties:

Property 1. At each stage of the incremental process, the regions of zero width ($\mathscr{F}_0(\mathscr{S})$) are leaves of the I-DAG.

Property 2. The influence range of the region associated with a node is included in the union of the ranges of its parents.

The construction of the I-DAG can be sketched as follows:

- We initialize $\mathscr{S}$ with the b first objects. A node of the I-DAG is created for each region of $\mathscr{F}_0(\mathscr{S})$ and made son of the root of the I-DAG. The influence range associated with the root is the whole objects universe $\mathscr{O}$.
- At each subsequent step, a new object O is added to $\mathscr{S}$ and the I-DAG is updated. The two following substeps are performed:
  Location Substep. This substep finds all the nodes of the I-DAG whose regions have zero width and are in conflict with O. This is done by traversing every path from the root of the I-DAG down to the first node which is not in conflict with object O.
  Creation Substep. From the information collected during the location substep, the creation substep creates a new node for each region in $\mathscr{F}_0(\mathscr{S} \cup \{O\})/\mathscr{F}_0(\mathscr{S})$ and links the new nodes to already-existing nodes in the structure, so that Properties 1 and 2 still hold. The details of this substep depend on each particular application.

2.3. *Randomized Analysis of the I-DAG*

This subsection aims to provide a randomized analysis of the space and time required to build the I-DAG structure. Randomization here concerns only the order in which the inserted objects are introduced in the structure. Thus, if the current set of objects is a set S of cardinality n, our results are expected values that correspond to averaging over the $n!$ possible permutations of the inserted objects, each equally likely to occur.

We first prove some probabilistic results that are purely combinatorial. These results allow us to prove the main results of this paper, stated as Theorems 2.4 and 2.9 below, that give quite general upper bounds on the size and the update time of the I-DAG as functions of the expected size of the output for a sample of the input.

For the sake of clarity, we will make some hypotheses that simplify the analysis somewhat. These conditions are fulfilled by a large class of geometric problems and allow us to express the results in a simple way. However, these hypotheses are not really necessary and will be removed in Section 2.4.

Probabilistic Lemma. We first introduce some additional notation and definitions. In some geometric situations (e.g., when computing an arrangement of line segments), the regions are not all defined by the same number of objects. We adapt our notation accordingly: $\mathscr{F}_j^i(\mathscr{S})$ will denote the subset of regions of $\mathscr{F}(\mathscr{S})$ defined by i objects of $\mathscr{S}$ $(i \le b)$ and having width j, and $\mathscr{F}_{\le j}^i(\mathscr{S})$ will denote the subset of regions defined by i objects and having width at most j.

An *r-random sample* of a finite set of objects $\mathscr{S}$, of cardinality n, is a subset of $\mathscr{S}$ of r objects chosen at random, i.e., with probability $1\Big/\binom{n}{r}$.

Let $f_0(r, \mathscr{S})$ be the expected size $E(|\mathscr{F}_0(\mathscr{R})|)$ of $\mathscr{F}_0(\mathscr{R})$ for random samples $\mathscr{R}$ of cardinality r.

The first lemma, due to Clarkson and Shor [7], bounds the number $|\mathscr{F}_{\le j}^i(\mathscr{S})|$ of regions with width at most j defined by $\mathscr{S}$. We recall it for completeness.

Lemma 2.1.

$$|\mathscr{F}_{\le j}^i(\mathscr{S})| = O(j^i f_0(\lfloor n/j \rfloor, \mathscr{S})).$$

Proof. The proof uses the technique of random sampling. Let $\mathscr{R}$ be an r-random sample of $\mathscr{S}$. A region F of $\mathscr{F}_j^i(\mathscr{S})$ has width zero with respect to $\mathscr{R}$, if and only if the i objects defining F belong to $\mathscr{R}$ while the j objects in conflict with F do not. So the probability $\text{Prob}(F \in \mathscr{F}_0(\mathscr{R}))$ that F belongs to $\mathscr{F}_0(\mathscr{R})$ is

$$\begin{aligned}
\text{Prob}(F \in \mathscr{F}_0(\mathscr{R})) &= \frac{\binom{n-i-j}{r-i}}{\binom{n}{r}} \\
&= \frac{(n-i-j)!\,r!\,(n-r)!}{(n-r-j)!\,(r-i)!\,n!} \\
&= \frac{r\cdots(r-i+1)}{n\cdots(n-i+1)}\,\frac{(n-r)\cdots(n-r-j+1)}{(n-i)\cdots(n-i-j+1)} \\
&= \frac{r\cdots(r-i+1)}{n\cdots(n-i+1)}\left(\frac{n-r-k+1}{n-i-k+1}\right)^k \quad \text{for } j \le k \\
&\ge \frac{r\cdots(r-i+1)}{n\cdots(n-i+1)}\left(1-\frac{1}{k}\right)^k \quad \text{if } r = \lfloor n/k \rfloor, \text{ for } 1 \le k \\
&\ge \frac{1}{4}\,\frac{r\cdots(r-i+1)}{n\cdots(n-i+1)} \quad \text{for } k \ge 2.
\end{aligned}$$

Then we compute the expected size of $\mathcal{F}^i_0(\mathcal{R})$

$$\begin{aligned} E(|\mathcal{F}^i_0(\mathcal{R})|) &= \sum_{j=0}^{n-i} \sum_{F \in \mathcal{F}^i_j(\mathcal{S})} \operatorname{Prob}(F \in \mathcal{F}^i_0(\mathcal{R})) \\ &\geq \sum_{j=0}^{k} \sum_{F \in \mathcal{F}^i_j(\mathcal{S})} \operatorname{Prob}(F \in \mathcal{F}^i_j(\mathcal{R})) \\ &\geq \left(\sum_{j=0}^{k} |\mathcal{F}^i_j(\mathcal{S})| \right) \frac{1}{4} \frac{r \cdots (r-i+1)}{n \cdots (n-i+1)}. \end{aligned}$$

By introducing f_0, we get

$$\begin{aligned} |\mathcal{F}^i_{\leq k}(\mathcal{S})| &\leq 4 \frac{n \cdots (n-i+1)}{r \cdots (r-i+1)} E(|\mathcal{F}^i_0(\mathcal{R})|) \\ &\leq O(k^i f_0(\lfloor n/k \rfloor, \mathcal{S})). \end{aligned}$$ □

The Update Conditions. We first assume that three update conditions are satisfied. As already mentioned, these hypotheses are mainly for clarity and will be removed later.

1. The number of sons of a node of the I-DAG is bounded.
2. Given a region F and an object O, the test to decide whether or not O is in conflict with F can be performed in constant time.
3. If the new object O added to the current set $\mathcal{S}$ is found to be in conflict with k regions of $\mathcal{F}_0(\mathcal{S})$, then the creation substep requires $O(k)$ time.

Expected Storage

Lemma 2.2. *If $\mathcal{S}$ has cardinality n, the expected size of the I-DAG of $\mathcal{S}$ is*

$$O\left(\sum_{j=1}^{n} \frac{f_0(\lfloor n/j \rfloor, \mathcal{S})}{j} \right).$$

Proof. The expected number of nodes $\eta(\mathcal{S})$ in the I-DAG of $\mathcal{S}$ can be obtained by summing, for all the regions F of $\mathcal{F}(\mathcal{S})$, the probability that F occurs as a node in the I-DAG, which is $i!\,j!/(i+j)!$ (the i objects defining F must be inserted before the j objects in F). By Lemma 2.1

$$\begin{aligned}
\eta(\mathcal{S}) &= \sum_{i=1}^{b} \sum_{j=0}^{n-i} |\mathcal{F}_j^i(\mathcal{S})| \frac{i!\,j!}{(i+j)!} \\
&= \sum_{i=1}^{b} \left(|\mathcal{F}_0^i(\mathcal{S})| + \sum_{j=1}^{n-i} (|\mathcal{F}^i_{\le j}(\mathcal{S})| - |\mathcal{F}^i_{\le (j-1)}(\mathcal{S})|) \frac{i!\,j!}{(i+j)!} \right) \\
&= \sum_{i=1}^{b} \left(\sum_{j=0}^{n-i-1} |\mathcal{F}^i_{\le j}(\mathcal{S})|\, i \frac{i!\,j!}{(i+j+1)!} + |\mathcal{F}^i_{\le (n-i)}(\mathcal{S})| \frac{i!\,(n-i)!}{n!} \right) \\
&\le O\left(\sum_{j=1}^{n} \frac{f_0(\lfloor n/j \rfloor, \mathcal{S})}{j} \right)
\end{aligned}$$

According to update condition 1, this bound applies also to the size of the I-DAG □

Expected Time

Lemma 2.3. *Under the update conditions, if $\mathcal{S}$ has cardinality n, the expected time for inserting the last object in the I-DAG is*

$$O\left(\frac{1}{n} \sum_{j=1}^{n} f_0(\lfloor n/j \rfloor, \mathcal{S}) \right)$$

Proof. Under update condition 2, the computing time spent to locate the last inserted object O is proportional to the total number of nodes of the I-DAG visited when locating O. Due to update condition 1, the number of nodes visited when locating O is at most proportional to the number of nodes of the I-DAG associated with regions in conflict with O. Thus the expected time for locating the last inserted object O is at most proportional to the expected number, $\theta(\mathcal{S})$, of nodes of the I-DAG associated with regions in conflict with O.

Let F be a region of $\mathcal{F}_j^i(\mathcal{S})$. F is a region in conflict with O associated with a node of the I-DAG if O is one of the j objects in conflict with F and if the i objects defining F have been inserted before the j objects in conflict with F. This occurs with the probability $j/n \times i!\,(j-1)!/(i+j-1)!$. The expected number $\theta(\mathcal{S})$ of nodes visited during the last insertion is then obtained by summing, for all the F of $\mathcal{F}(\mathcal{S})$, the above probability. Using Lemma 2.1, this yields

$$\theta(\mathcal{S}) = \sum_{i=1}^{b} \sum_{j=1}^{n-i} |\mathcal{F}_j^i(\mathcal{S})| \frac{i!\,j!}{n(i+j-1)!} = O\left(\frac{1}{n} \sum_{j=1}^{n} f_0(\lfloor n/j \rfloor, \mathcal{S}) \right),$$

from a calculation similar to the proof of Lemma 2.2. □

Due to update condition 3, the computing time of the last creation substep is also dominated by a term proportional to the number of nodes of the I-DAG associated with a region in conflict with O and admits the same expected upper bound as $\theta(\mathscr{S})$.

Main Theorem. Lemmas 2.2 and 2.3 prove our main result:

Theorem 2.4. *If the set of already-inserted objects $\mathscr{S}$ has cardinality n, and if the update conditions are fulfilled, the I-DAG of $\mathscr{S}$ requires $O(\sum_{j=1}^{n} f_0(\lfloor n/j \rfloor, \mathscr{S})/j)$ expected memory space. The insertion of a new object can be done in $O(1/n \sum_{j=1}^{n} f_0(\lfloor n/j \rfloor, \mathscr{S}))$ expected update time.*

Corollary 2.5. *Under the update conditions, the total expected time to build an I-DAG for a set $\mathscr{S}$ of n objects is $O(\sum_{j=1}^{n} f_0(\lfloor n/j \rfloor, \mathscr{S}))$.*

Proof. Notice that in that corollary, $\mathscr{S}$ is no longer the current subset of inserted objects, but the final set of objects. From Theorem 2.4, we know that, if a subset $\mathscr{R}$ of $\mathscr{S}$ with cardinality r has been inserted at a given time, the expected time to insert the last object of $\mathscr{R}$ is

$$O\left(\sum_{j=1}^{r} \frac{1}{r} f_0(\lfloor r/j \rfloor, \mathscr{R})\right),$$

this expected time accounts for averaging over the $r!$ permutations of the objects of $\mathscr{R}$. Now the expected time to insert the rth object of $\mathscr{S}$ results from further averaging over the r-random samples of $\mathscr{S}$ which yields

$$O\left(\sum_{j=1}^{r} \frac{1}{r} f_0(\lfloor r/j \rfloor, \mathscr{S})\right).$$

The proof of Corollary 2.5 results from summing over r from 1 to n. □

Two immediate consequences of Theorem 2.4 are the following:

- If $f_0(x, \mathscr{S}) = \Theta(x)$, the space complexity is $O(n)$ and the time to insert a new object is $O(\log n)$.
- If $f_0(x, \mathscr{S}) = \Theta(x^\alpha)$ (with $\alpha \geq 1$), the space complexity is $O(n^\alpha)$ and the time to insert a new object is $O(n^{\alpha-1})$).

2.4. *Removing the Update Conditions*

In some cases the three update conditions can be removed. We will show in Section 3 that removing the update conditions (especially condition 1) will lead, in some cases, to simpler algorithms.

Constant Test Time and Linear Update Time. Update conditions 2 and 3 can be relaxed. If the time required to check if a region and an object are in conflict surpasses $O(1)$, the overcost will simply appear as a multiplicative factor in the overall complexity. If the time required by the creation substep surpasses a linear function of the number of conflicts, it is, in general, not difficult to charge the overcost to the overall complexity. This analysis has been done, for example, for the incremental construction of higher order Voronoi diagrams [2].

Bounded Number of Sons. It is more interesting to attempt to remove update condition 1. The preceding analysis works because each node in the I-DAG is associated to a region, and all the relevant quantities can be expressed as functions of the number of nodes. If the condition is not fulfilled, we must count the number of edges of the I-DAG and, to that purpose, we introduce the notion of *bicycles.* A bicycle is a pair of regions of $\mathscr{F}(\mathscr{S})$ occurring as a father and one of its sons in the I-DAG associated with at least one permutation of the object set $\mathscr{S}$.

Notice that the maximum number of objects defining a bicycle is at most $2b$ and thus is still bounded. An object is in conflict with a bicycle if it does not belong to the set of objects that define the bicycle, and if it is in conflict with at least one of the two regions forming the bicycle. Thus the influence range of a bicycle is the union of the influence range of the two regions forming the bicycle, excepting the objects defining these regions.

In analogy with the notation used for regions, the additional notations $\mathscr{G}_j(\mathscr{S})$, $\mathscr{G}^i_{\leq j}(\mathscr{S})\cdots$ are naturally derived. We define $g_0(r, \mathscr{S})$ to be the expected size $E(|\mathscr{G}_0(\mathscr{R})|)$ of $\mathscr{G}_0(\mathscr{R})$ for r-random samples of $\mathscr{S}$.

With these definitions, the following lemma can be proved, using the random sampling technique, in a way similar to Lemma 2.1.

Lemma 2.6.

$$|\mathscr{G}^i_{\leq j}(\mathscr{S})| = O(j^i g_0(\lfloor n/j \rfloor, \mathscr{S})).$$

We can now compute the expected storage required by the I-DAG.

Lemma 2.7. *If $\mathscr{S}$ has cardinality n, the expected size of the I-DAG of $\mathscr{S}$ is*

$$O\left(\sum_{j=1}^{n} \frac{g_0(\lfloor n/j \rfloor, \mathscr{S})}{j}\right).$$

Proof. The size of the I-DAG is linearly related to the number of its edges. A necessary condition for a given bicycle G to occur as an edge in the I-DAG, is that the i objects defining G are inserted before the j objects in conflict with G (i.e., in conflict with one of the two regions associated with G). So the probability that $G \in \mathscr{G}(\mathscr{S})$ arises in the I-DAG is less than $i!\,j!/(i+j)!$. Calculations analogous to those appearing in the proof of Lemma 2.2 yield the result. $\square$

Lemma 2.8. *The expected time for inserting the nth object in the I-DAG is*

$$O\left(\frac{1}{n}\sum_{j=1}^{n} g_0(\lfloor n/j \rfloor, \mathscr{S})\right).$$

Proof. Let O be the nth object to be inserted. The number of times a node is visited is equal to the number of its parents which are in conflict with O. Thus the number of performed tests is no more than the number of bicycles in conflict with O occurring in the I-DAG.

Let G be a bicycle of $\mathscr{G}_j^i(\mathscr{S})$. G is in conflict with O and occurs as an edge in the I-DAG if the following two necessary conditions are satisfied: (1) the i objects defining G are inserted before the j objects in conflict with G; and (2) one of the j objects is O. The probability that G is in conflict with O is thus no more than $j/n \times i!\,(j-1)!/(i+j-1)!$. The result is then achieved as in Lemma 2.3. □

Lemmas 2.7 and 2.8 prove the main result of this section:

Theorem 2.9. *If the set of already-inserted objects $\mathscr{S}$ has cardinality n and if update conditions* 2 *and* 3 *are fulfilled (but not update condition* 1*), the I-DAG of $\mathscr{S}$ requires $O(\sum_{j=1}^{n} g_0(\lfloor n/j \rfloor, \mathscr{S})/j)$ expected memory space. The insertion of a new object can be done in $O(1/n \sum_{j=1}^{n} g_0(\lfloor n/j \rfloor, \mathscr{S}))$ expected update time.*

Corollary 2.10. *Under update conditions* 2 *and* 3, *the total expected time to build an I-DAG for a set $\mathscr{S}$ of n object is $O(\sum_{j=1}^{n} g_0(\lfloor n/j \rfloor, \mathscr{S}))$.*

2.5. *Faster Object Location*

If the following additional property is verified, it is possible to get a better complexity result for the search of a single region in conflict with a new object.

Property 3. The influence range of the region associated to a node is included in the union of the ranges of its children.

If Property 3 holds, then a conflict with a given new object is found by a simple path from the root of the I-DAG to a leaf.

Theorem 2.11. *If Property* 3 *holds then a conflict with* any *new object can be found in $O(\log n)$ time provided that the n objects that were inserted in the I-DAG have been inserted in random order.*

Proof. Let o be the new object and F be a region on the path from the root to a leaf of the I-DAG in conflict with o. Suppose that F has zero width after the insertion of the kth object. If F is defined by i objects, the conditional probability that F has been created during the insertion of the kth object is $i/k \le b/k$. Indeed, for F to be created at step k, the kth inserted object must be one of the i objects defining F. (It is important to notice that the above probability is conditioned by the fact that the k first objects are given.)

Averaging this probability over the $\binom{n}{k}$ possible subsets of k objects introduced first in the I-DAG yields a probability less than b/k, that the node on the path would change after the insertion of the kth object. Thus the number of visited nodes is less than $\sum b/k = O(\log n)$. □

Let us make some remarks about Theorem 2.11. First, it is important to notice that the cost studied here is expected over all possible orders to insert the objects in the I-DAG, but there is no hypothesis on the new object, by opposition to Lemma 2.3 where all objects, including the last one, are supposed to satisfy the randomization hypothesis. Second, in many applications, it is possible to find all conflicts with a new object from a single one in time proportional to the actual number of conflicts (e.g., by the use of some neighborhood notions). The faster location may be used as a first step of the insertion of a new object in the I-DAG. A last remark concerns the worst case; though this article takes interest in randomized complexities, the faster location has a worst-case running time $O(n)$ which is better than the general location step.

2.6. *Queries*

In some applications, queries consist of finding the regions having zero width which are in conflict with a given element of the object universe: this is just a special instance of a location substep. In such cases, the I-DAG can be used and the randomized analysis of Theorem 2.4 holds, provided that the query object q together with the set of objects $\mathcal{S}$ introduced in the I-DAG satisfy the randomization hypothesis, i.e., the $(n + 1)!$ permutations of $\{q\} \cup \mathcal{S}$ are likely to occur.

In other applications, regions and queries are such that the answer to a query consists of exactly one region of $\mathcal{F}_0(\mathcal{S})$ for any set $\mathcal{S}$. Such a query will be answered by a location substep that will traverse only one path from the root to a leaf. This yields the following strong variant of Lemma 2.3.

Theorem 2.12. *Assume that regions and queries are such that the answer to a query consists of exactly one region of $\mathcal{F}_0(\mathcal{S})$ for any set $\mathcal{S}$. Then* any *query can be answered in $O(\log n)$ time provided that the n objects that were inserted in the I-DAG have been inserted in random order.*

Proof. The proof is similar to the proof of Theorem 2.11. □

3. Applications

3.1. *Convex Hulls*

We consider the geometric problem of computing the convex hull of a set of points. In the plane, optimal deterministic on-line algorithms are known with time complexity $O(\log n)$ per update (see, for instance, [1]). In d-dimensional space, the worst-case complexity of the convex hull of n points is $\Omega(n^{\lfloor d/2 \rfloor})$. The best-known deterministic algorithm, due to Seidel [14], [9], requires $O(n \log n + n^{\lfloor (d+1)/2 \rfloor})$ time and $O(n^{\lfloor d/2 \rfloor})$ space. The algorithm is incremental but its complexity is amortized over the n insertions. This result is optimal in the worst-case for d larger than 2 and even. An output sensitive algorithm with time complexity $O(n^2 + f \log n)$ where f is the number of faces of the convex hull is reported in [13].

Randomized algorithms have also been recently proposed by Clarkson and Shor [7] and by Mulmuley [12]. These algorithms are incremental and optimal but static. A more on-line algorithm for $d \leq 3$ can be found in [10]; however, the analysis is only amortized over the n insertions.

We present, in this section, two (randomized) algorithms that are both on-line and optimal in any dimension. For brevity, we expose here only the two-dimensional case. The extension to higher-dimensional spaces is quite straightforward.

First Algorithm. Objects are points of the plane. Regions are defined by three points. The region PQR associated with P, Q, and R consists of the union of two half-planes: one bounded by line (PQ) and not containing R and the other bounded by line (QR) and not containing P (see Fig. 1). An object is in conflict with a region if and only if it lies in the region.

Now let P, Q, and R be three points in $\mathcal{S}$, the set of points already inserted. The region associated with these points has zero width if and only if P, Q, and R are three consecutive vertices of the convex hull. So computing the convex hull of $\mathcal{S}$ is equivalent to computing the zero width regions.

Let us now describe the algorithm. Suppose that the I-DAG has been constructed for the points in $\mathcal{S}$ and that we want to insert a new point M. The location substep gives the regions of $\mathcal{F}_0(\mathcal{S})$ containing M. If M belongs to the interior of the convex hull, there is no such region, $\mathcal{F}_0(\mathcal{S} \cup \{M\}) = \mathcal{F}_0(\mathcal{S})$, and the I-DAG is not modified. Otherwise, let P_1 and P_k be the two vertices adjacent to M in the new convex hull and let $P_2, \ldots, P_{k-1}$ be the chain of vertices which

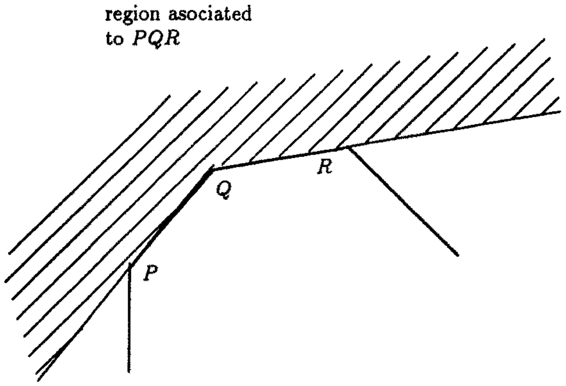

Fig. 1. Definition of regions for the convex hull problem.

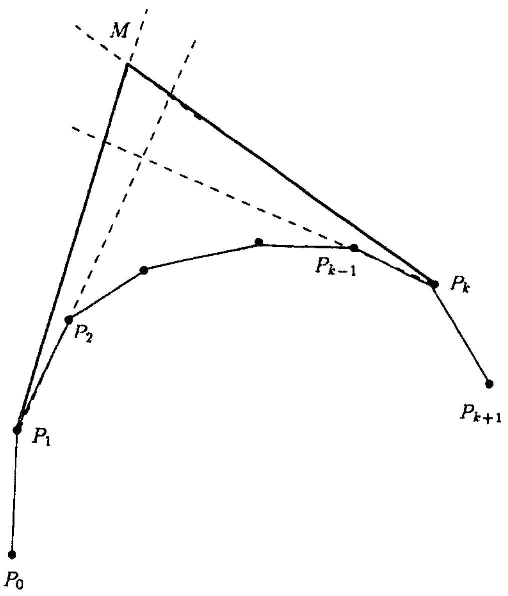

Fig. 2. Inserting a new point in the convex hull.

are no longer vertices of the convex hull after the insertion of M (see Fig. 2). The regions of $\mathscr{F}_0(\mathscr{S})$ containing M are $P_{l-1}P_lP_{l+1}$ for $1 \leq l \leq k$. By a simple test on these regions, we can determine $P_0P_1P_2$ and $P_{k-1}P_kP_{k+1}$ (M belongs to only one of the two half-planes defining the region). The I-DAG is then modified in the following manner: the width of the selected regions is incremented and three new regions are created, namely,

- P_0, P_1, M as a son of P_0, P_1, P_2;
- M, P_k, P_{k+1} as a son of P_{k-1}, P_k, P_{k+1};
- P_1, M, P_k as a son of both P_0, P_1, P_2 and P_{k-1}, P_k, P_{k+1}.

It is clear that the properties of the I-DAG are preserved and that the update conditions are satisfied. Here $f_0(r, \mathscr{S})$ is the expected size of the convex hull of r points of $\mathscr{S}$ which is clearly $O(r)$, so applying Theorem 2.4 we deduce:

Proposition 3.1. *The convex hull of n points in the plane can be computed on-line with $O(n)$ expected space and $O(\log n)$ expected update time.*

These results can be generalized to any dimension. The regions are defined by $d + 1$ points and are unions of two half-spaces. The zero width regions correspond to $(d - 2)$-faces of the convex hull of r points is $O(r^{\lfloor d/2 \rfloor})$.

Proposition 3.2. *The convex hull of n points in d-space can be computed on-line with $O(n^{\lfloor d/2 \rfloor})$ expected space and $O(\log n)$ expected update time if $d \leq 3$, and $O(n^{\lfloor d/2 \rfloor - 1})$ expected update time if $d > 3$.*

These results are optimal.

As far as queries are concerned, the above results show that we can decide if a point lies inside or outside the convex hull of n points in $O(\log n)$ expected time in the plane and $O(n^{\lfloor d/2 \rfloor - 1})$ expected time in d-space.

Second Algorithm. It might look more natural to take half-spaces as regions. In that case, as will become clear below, the number of sons is not bounded and update condition 1 is not satisfied. However, the result of Section 2.4 proves that the resulting algorithm has the same complexity as the one above.

We describe the algorithm for the two-dimensional case. It can be generalized to any dimension with no difficulty. $\mathcal{O}$ still denotes the set of points of the plane. Regions are now defined by only two points. The regions PQ and QP are the two half-planes limited by the line (PQ). A point is in conflict with the region PQ if it lies inside the corresponding half-plane. If a region PQ has zero width, then $[PQ]$ is an edge of the convex hull.

In addition to the standard information stored in each node of the I-DAG, we also maintain at each leaf, which is associated with an edge E of the convex hull, two pointers towards the two leaves associated with the two edges of the convex hull adjacent to E. When a new point M is inserted, the location substep provides all the half-planes with current width 0 in conflict with M. These half-planes correspond to a chain of edges of the convex hull $P_1P_2, \ldots, P_{k-1}P_k$ (see Fig. 2). The two extremal edges P_1P_2 and $P_{k-1}P_k$ are identified by testing if their two neighbors are not both in conflict with M. Two new regions P_1M and MP_k are created. In order to satisfy Property 2, P_0P_1 and P_1P_2 are made parents of P_1M; similarly, $P_{k-1}P_k$ and P_kP_{k+1} are made parents of MP_k. The neighborhood relationships are updated: P_0P_1 and P_1M become adjacent and, similarly, P_kP_{k+1} and MP_k, and P_1M and MP_k.

Notice that the width of P_0P_1 is still zero and that this region may have other sons in the future: update condition 1 is not satisfied.

As described in Section 2.4, we introduce the notion of a bicycle. Here a bicycle is defined by two regions PQ and QR sharing a point of definition (a bicycle here is a region of the previous section; see Fig. 1). The zero width bicycles are of two kinds. The first ones are associated with two regions with zero width: P_0P_1 and P_1M in Fig. 2. They correspond to two consecutive edges of the convex hull. The second ones are associated with a region of zero width and a region in conflict with the additional point of definition of the other: P_1P_2 and P_1M in Fig. 2. They correspond to an edge E of the convex hull and to an edge E', incident to one of the end-points of E and whose supporting line separates a 1-set of $\mathcal{S}$. Using the results on the number of k-sets (see, for instance, [9]) we conclude that $g_0(r)$ is $O(r^{\lfloor d/2 \rfloor})$. Thus Theorem 2.9 implies that this simpler algorithm has the same complexity as the algorithm of the previous section.

3.2. *Voronoi Diagrams*

Using the well-known correspondence between Voronoi diagrams in d-dimensions and convex hulls in $d + 1$-dimensions we immediately deduce, from the previous

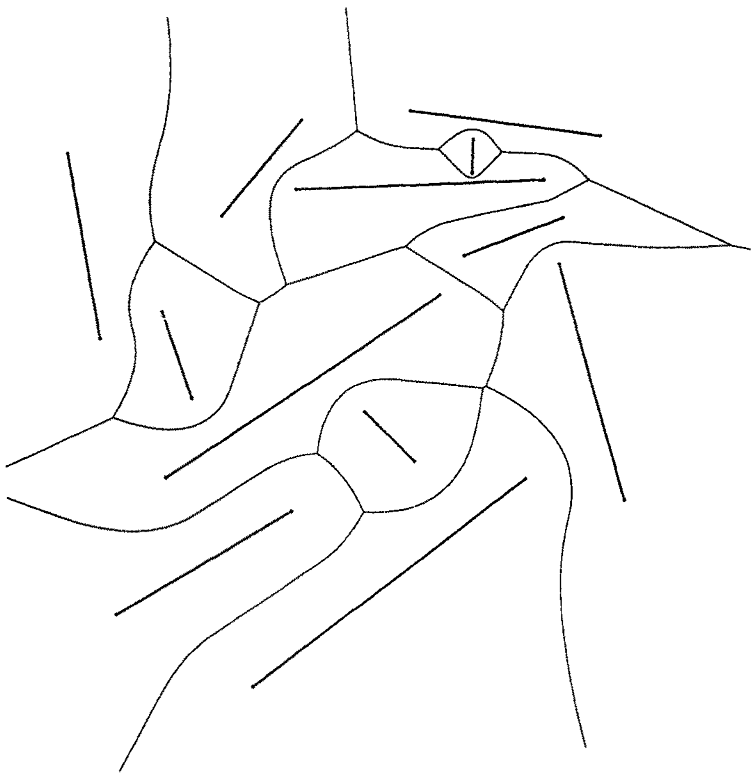

Fig. 3. Example of Voronoi diagram of line segments.

section, two optimal on-line algorithms to construct the Voronoi diagrams of points in any dimension. A direct presentation that does not use this correspondence has already been described in detail in [4].

Let us consider now the case of Voronoi diagrams of line segments in two dimensions (Fig. 3). Here $\mathcal{O}$ is the set of all line segments of the Euclidean plane. Let P, Q, R, and S be four segments and let Γ be the portion of the bisector of P and Q extending between the two points equidistant from P, Q, R and P, Q, S, respectively (see Fig. 4). The region associated with P, Q, R, and S is the union of the disks tangent to P and Q whose centers lie on Γ. A line segment and a region are in conflict if and only if they intersect. A region has zero width if and only if Γ is an edge of the Voronoi diagram.

The update algorithm is as follows. We find that in the I-DAG all regions in conflict with the new segment M: these regions correspond to the edges which disappear in the new diagram. Let E be such an edge. If one of the two end-points of E is still a valid vertex of the Voronoi diagram (i.e., if the segment M does not intersect the corresponding disk) we compute in constant time the portion of E that remains in the new diagram. The new region associated with that new edge becomes a son of the region associated with E. We then connect the new vertices

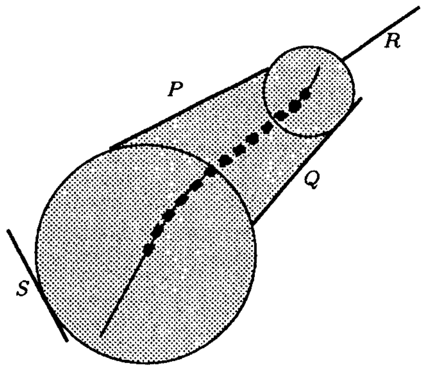

Fig. 4. Definition of regions for Voronoi diagrams of segments. Γ is in dotted lines

of the Voronoi diagram (which are new end-points lying on old edges) by edges supported by new bisectors (see Fig. 5). The region corresponding to a new edge e' is made son of the regions associated with the unique path of disappearing edges that joins the two end points of e' (the set of disappearing edges form a tree, as can easily be shown).

The update conditions are satisfied and, by the Euler relation, f_0 is linearly related to the number of segments.

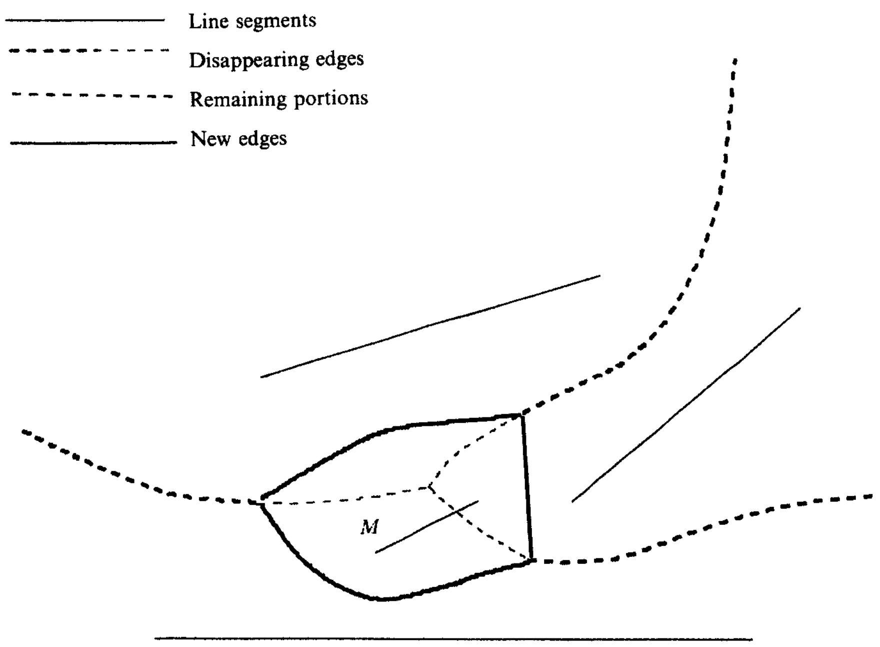

Fig. 5. Insertion of a new segment in a Voronoi diagram.

Proposition 3.3. *The Voronoi diagram of n line segments in the plane can be computed with $O(n)$ expected space and $O(\log n)$ expected update time.*

3.3. Arrangements of Planar Curves

Let us consider first the case of line segments. The best-known deterministic solution to this problem is due to Chazelle and Edelsbrunner [5]. It requires $O(n \log n + a)$ time and $O(n + a)$ space in the worst-case, if a is the number of intersecting pairs. Incremental randomized algorithms have recently been proposed by Clarkson and Shor [7] and by Mulmuley [12]. They both use a conflict graph (and thus are static) and have the same running time as the Chazelle and Edelsbrunner algorithm.

The general framework of Section 2 can be applied to solve this problem. Our algorithm builds the trapezoidal diagram of $\mathscr{S}$ [7], obtained by drawing a vertical line through each vertex of the arrangement of the segments and by keeping only the portions of the lines extending above and below the corresponding vertex and not intersected by any segment of $\mathscr{S}$, see Fig. 6. Objects are here line segments and regions are trapezoids (i.e., a cell of the trapezoidal diagram). A trapezoid is defined by at most four segments. A line segment and a region are in conflict if and only if they intesect. For each leaf of the I-DAG we also store some neighbors of the corresponding zero width trapezoid, more precisely, we store the adjacency relationships through the vertical edges of the trapezoidal diagram. In general position, a leaf of the I-DAG has at most four such neighbors.

When a new segment S is inserted, we traverse the I-DAG to collect the set $\mathscr{L}(S)$ of all the zero width regions in conflict with S. Each such region is subdivided into at most four subregions, and these subregions are eventually merged to form new trapezoids (vertical segments intersecting S have to be shortened), which is easily done using the adjacency relationships stored in the nodes of the I-DAG.

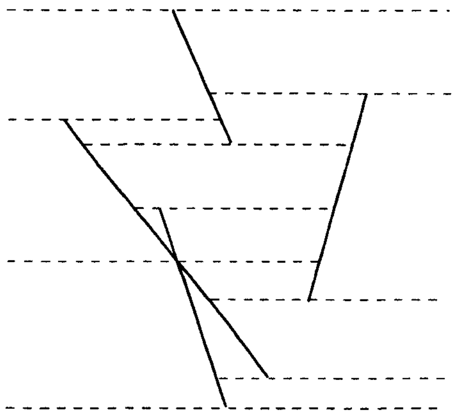

Fig. 6. Trapezoidal diagram.

The resulting trapezoids are the new nodes we attach to the I-DAG. The parents of a new node are the trapezoids of $\mathcal{L}(S)$ which intersect that node.

A node has at most four sons so that the update conditions are fulfilled (but the number of parents for a trapezoid is unbounded). An easy lemma, proved in [7], shows that $f_0(r, \mathcal{S}) = O(r + ar^2/n^2)$.

This result can be readily extended to planar arrangements of curves of bounded degree.

Proposition 3.4. *An arrangement of n planar curves (of bounded degree) can be computed on-line with $O(n + a)$ expected space and $O(\log n + a/n)$ expected update time, where a is the complexity of the arrangement.*

The trapezoids with zero width partition the plane, so that Theorem 2.12 applies:

Proposition 3.5. *A point can be located in an arrangement of n planar curves (of bounded degree) in $O(\log n)$ time using $O(n + a)$ expected space and $O(n \log n + a)$ expected preprocessing time, where a is the complexity of the arrangement.*

3.4. *Other Applications*

The I-DAG can be used to solve several other problems and provide simple on-line algorithms with the same worst-case complexities as the best (in general, static) deterministic algorithms. We simply mention some of them.

- Computing abstract Voronoi diagrams (see [11]).
- Computing arrangements of triangles or surface patches in space (see [6]).
- Computing the intersection of n half-spaces: this problem is dual to constructing convex hulls.
- Computing the union of n balls in d-space: consider the d-dimensional space as an hyperplane of a $d + 1$-space and use an inversion with a point outside the hyperplane as its pole: the problem reduces to that of computing the intersection of n half-spaces.
- Computing the visibility graph of a set of line segments in the plane (see [15]): take as regions the triangles containing two edges of the visibility graph incident to a common vertex and consecutive when sorted by polar angle around this vertex. A line segment is in conflict with a region if it intersects the region.

4. About Our Complexity Results

4.1. *Randomization*

Our analysis of the space and time required to build the I-DAG structure is randomized. As previously noted, randomization concerns here only the order in which the inserted objects are introduced in the structure. No assumption is made as to the distribution of the input. Our results are expected values that correspond to averaging over the $n!$ possible permutations of the n inserted objects, each supposed to be equally likely to occur.

4.2. *Amortization*

We have been able to bound the cost of inserting the kth object in the I-DAG. This cost is not amortized, as opposed to the results in [2] and [10], but the kth object may be any one of the inserted objects with the same probability.

It must be noticed, however, that the bound given in Theorem 2.4 cannot be a bound for the cost of inserting a given object. Indeed, let us consider the construction of the Delaunay triangulation (the dual of the Voronoi diagram) of a set of n points in the plane. We take $n - 1$ points close to two line segments and one point, say M, between the two segments (see Fig. 7). For appropriate positions of the points, the insertion of M will modify most of the triangles; thus, the expected cost of inserting M in the I-DAG at step k is $\Omega(k)$: whatever k is.

This is not in contradiction with our result. Indeed, the cost of inserting a given object appears weighted by the probability factor $1/n$ in the expected cost of step k. Our bound on the cost of the kth insertion proves that objects requiring expensive updates are rare whatever the set $\mathcal{S}$ of input data may be.

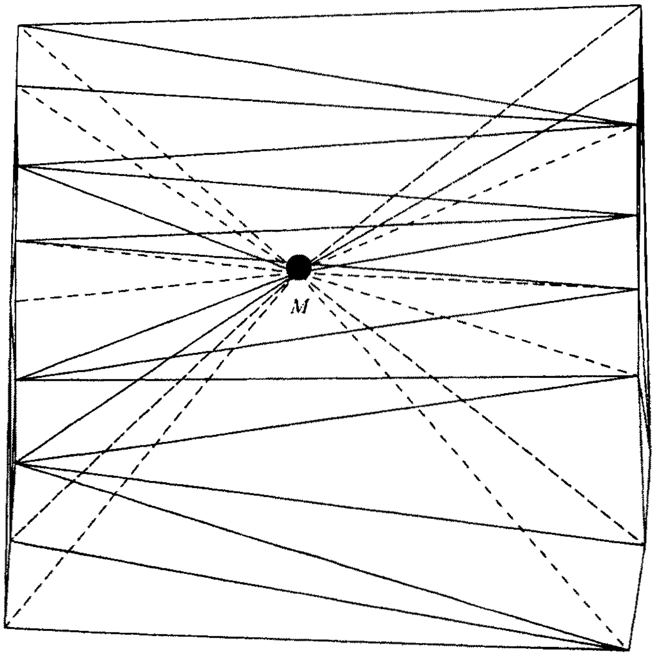

Fig. 7. Cost of inserting point O. The triangulation before (resp. after) the insertion of O is in plain lines (resp. in dotted lines).

4.3. Output Sensitivity

An algorithm is said to be output sensitive if, for a given set of input data, its complexity depends on the actual size of the output. It is clearly impossible, in general, to have incremental algorithms that are sensitive to the final output, because at some stage of the incremental construction the intermediate results may be greater than the final one. We may illustrate this with the example of the Voronoi diagram in three dimensions. Let $\mathcal{S}$ be a set of n points lying on two non-coplanar line segments. The Voronoi diagram of $\mathcal{S}$ is quadratic but, if a point O between the two segments is added to $\mathcal{S}$, the diagram of $\mathcal{S} \cup \{O\}$ becomes linear.

In view of this fact, it is interesting to define *on-line output sensitive* algorithms as algorithms whose update complexities depend on the actual size of the current output.

Our algorithms are not on-line output sensitive because the expected complexity of each step depends on $f_0(r, \mathcal{S})$ (or $g_0(r, \mathcal{S})$ for some $r \leq n$. Let us consider again the case of the Voronoi diagram of the set of points above. Inserting a $n + 2$th point to $\mathcal{S} \cup \{O\}$ will take $O(n^2)$ expected time using the I-DAG, although the current output is $O(n)$.

However, in many situations, the expected value $f_0(r, \mathcal{S})$ is a well-behaved function of the size r of the random sample which is sensitive to the actual size of the output for $\mathcal{S}$. In such a case, the expected complexity of the I-DAG is on-line output sensitive.

A first illustration is the case of an arrangement of planar curves which has been described in Section 3.3. As a second illustration, let us consider the case of the Voronoi diagram in higher dimensions. For some distributions of the input data, the diagrams built on the entire set of points, as well as on most of the samples, have a linear size. For example, the expected size of the Voronoi diagram, of a set $\mathcal{S}$ of n points evenly distributed in the unit d-ball, is $O(n)$ and the expected size of the Voronoi diagram for an r-random sample $f_0(r, \mathcal{S})$ is $O(r)$ [8]. This result readily implies that the Voronoi diagram of n points evenly distributed in the unit d-ball can be computed on-line with $O(n)$ space and $O(\log n)$ update time in any dimension.

5. Conclusion

We have presented a general framework for the design and analysis of efficient on-line algorithms. The algorithms are randomized, simple, and, in some cases, output sensitive. They have been coded easily and preliminary experiments have provided strong evidence that they are very efficient in practice. Experimental results are reported in [4] and [2].

This framework has been applied successfully to various problems: convex hulls and Voronoi diagrams in any dimension, Voronoi diagrams of segments in the plane, arrangements of curves in the plane, arrangements of surfaces in space, visibility graphs, unions of disks, and others.

Our technique assumes that the geometric structure to be computed is closely related to the regions of zero width. We may be interested in computing, instead, regions of width $\leq k$ to construct, e.g., k-sets or Voronoi diagrams of order k. It is possible to generalize the I-DAG, and to obtain results similar to the ones described here. The complexity results will depend on the expected size, f_k, of the regions of width $\leq k$ of random samples. Details can be found in the companion paper [2].

Finally, we leave as an open question whether it is possible to allow deletions as well as insertions in the I-DAG.

Acknowledgments

The authors would like to thank Serge Vaudenay who implements efficiently the Voronoi diagram of line segments and points out Theorem 2.11.

References

1. D. Avis, H. ElGindy, and R. Seidel, Simple on-line algorithms for convex polygons, *Computational Geometry* (G. T. Toussaint, ed.), North-Holland, Amsterdam, 1985, pp. 23–42.
2. J.-D. Boissonnat, O. Devillers, and M. Teillaud, A semi-dynamic construction of higher-order Voronoi diagrams and its randomized analysis, *Algorithmica* (to appear).
   Full paper available as Technical Report INRIA 1207. Abstract published in *Second Canadian Conference on Computational Geometry*, Ottawa 1990.
3. J.-D. Boissonnat and M. Teillaud, A hierarchical representation of objects: the Delaunay tree, *Second ACM Symposium on Computational Geometry*, Yorktown Heights, June 1986.
4. J.-D. Boissonnat and M. Teillaud, On the randomized construction of the Delaunay tree, *Theoretical Computer Science* (to appear).
   Full paper available as Technical Report INRIA 1140.
5. B. Chazelle and H. Edelsbrunner, An optimal algorithm for intersecting line segments in the plane, *IEEE Symposium on Foundations of Computer Science*, 1988, pp. 590–600.
6. K. L. Clarkson, H. Edelsbrunner, L. Guibas, M. Sharir, and E. Welzl, Combinatorial complexity bounds for arrangements of curves and surfaces, *Discrete and Computational Geometry*, **5** (1990), 99–160.
7. K. L. Clarkson and P. W. Shor, Applications of random sampling in computational geometry, II, *Discrete and Computational Geometry*, **4** (5) (1989).
8. R. A. Dwyer, Higher-dimensional Voronoi diagrams in linear expected time, *5th ACM Symposium on Computational Geometry*, Saarbrücken, June 1989.
9. H. Edelsbrunner, *Algorithms on Combinatorial Geometry*, Springer-Verlag, New York, 1987.
10. L. J. Guibas, D. E. Knuth, and M. Sharir, Randomized incremental construction of Delaunay and Voronoi diagrams, *ICALP 90*, Springer-Verlag, New York, 1990, pp. 414–431.
11. K. Mehlhorn, S. Meiser, and C. Ó'Dúnlaing, On the construction of abstract Voronoi diagrams, *STACS 90* (C. Choffrut and T. Lengauer, eds.), Springer-Verlag, New York, 1990, pp. 227–239.
12. K. Mulmuley, On obstruction in relation to a fixed viewpoint, *IEEE Symposium on Foundations of Computer Science*, 1989, pp. 592–597.
13. R. Seidel, Constructing higher-dimensional convex hulls at logarithmic cost per face, *ACM Symposium on Theory of Computing*, 1986, pp. 404–413.
14. R. Seidel, A Convex Hull Algorithm Optimal for Point Sites in Even Dimensions, Technical Report 14, Department of Computer Science, University British Columbia, Vancouver, BC, 1981.
15. E. Welzl, Constructing the visibility graph for n line segments in $o(n^2)$ time, *Information Processing Letters*, **20** (1985), 167–171.

Received July 1, 1990, and in revised form May 23, 1991.