An Introduction to Randomization in Computational Geometry.

Olivier Devillers

published in Theoretical Computer Science, 157:35-52, 1996

An Introduction to Randomization in Computational Geometry.

Olivier Devillers

INRIA, B.P.93,
06902 Sophia-Antipolis cedex (France),
E-mail: olivier.devillers@sophia.inria.fr.

Abstract

This paper is not a complete survey on randomized algorithms in computational geometry, but an introduction to this subject providing intuitions and references.

In a first time, some basic ideas are illustrated by the sorting problem, and in a second time few results on computational geometry are briefly explained.

1 Introduction

Usually, computational geometry algorithms are complex and difficult to implement. A new possibility consists in designing simpler algorithms whose complexity is not optimal for all data but only when averaging on some random choices done by the algorithm. Although randomized algorithms appear in other domains, this survey only deals with applications in computational geometry. Classical algorithms deduce a result from some input data in a deterministic way, a randomized algorithm has to made choices between several possibilities to reach its goal. Randomness appears in the way to choose among the different possibilities. Complexity analysis is done by averaging among the different possibilities. It is important to notice that only the path chosen by the algorithm is random, the result is perfectly determined and no hypotheses are done on the data distribution such that "points obey a Poisson law"...

Several techniques exist to introduce randomness in an algorithm. For example, each time the algorithm has to make a choice between two different possibilities, it tosses a coin. For divide and conquer algorithms, the dividing part can be done in a random manner [Cla87]. Quicksort can be described in that manner as follows. To sort a set of n real values, choose one at random, call it the *pivot*, divide the set into three parts: the pivot, the values greatest than the pivot and those smallest than the pivot. Sort recursively the subsets and concatenate the results. Here, randomness appears in the choice of the pivot, that is in the way of dividing the problem in sub-problems.

A class of randomized algorithms, we will give a particular interest, is the one of incremental algorithms. In that kind of algorithm, randomness is not present a priori, in fact the behavior of these algorithms depends on the order

* *This work was supported by the ESPRIT Basic Research Action 7141 (ALCOMII).*

of insertion used for the data. If this order is part of the problem then such algorithms are not randomized, but if the order is not imposed, the algorithm has to choose between the paths corresponding to all different possible orders of insertion.

Clarkson and Shor [CS89] proposed a scheme of algorithms using a structure called the *conflict graph*. These algorithms are incremental, input objects are examined one by one, and for each new one the current result is updated. The conflict graph contains relations between current result and not yet examined objects, these relations are called *conflicts*. The randomized aspect is ensured by the choice of a random order to look at the input.

The conflict graph method, in spite of its incremental aspect, needs the knowledge of the whole input data at the beginning, because the not yet inserted part of the data is already present in the conflict graph. Another work [BDS$^+$92] allows the realization of *on-line* algorithms. The structure called *influence graph* contains the history of the successive partial results. Data can be inserted in a semi-dynamic way, finding the conflicts of the new item with all partial results. These conflicts are deduced one from others by looking to the history of the different partial results. This technique can apply to various problems: arrangements, convex hulls, visibility graphs, generalized Voronoi diagrams and, after few modifications, higher order Voronoi diagrams [BDT93]. All these algorithms are basic tools in computational geometry.

The next step consists in reaching a fully dynamic structure, handling insertions and deletions. A first possibility consists in remembering the whole history of the construction [Sch91], this approach often yields simple algorithms but the size of the history becomes the real parameter of the complexity and can be greater than the number of objects effectively present at the relevant time. Another technique reconstructs the history after each deletion. When an item is deleted, it is deleted not only at the final level of the history but also at all the preceding levels. This approach can be used for Delaunay triangulation in the plane [DMT92], arrangement of line segments [DTY92], convex hulls [CMS93] and also described in an abstract setting [DY93].

Paradoxicaly, the influence graph, whose aim was the (semi-)dynamization of static algorithms, allows us to improve some static algorithms. By merging the conflict graph and influence graph techniques, R. Seidel has obtained a simple and elegant algorithm to triangulate a simple polygon in $O(n \log^\star n)$ time [Sei91]. Instead of looking for all conflicts of the new object in the influence graph, R. Seidel proposes to start the search for conflicts not at the beginning of the history but at few key times in the history. In order to initialize this search, a conflict graph must be computed at these key times. In that particular case, the trapezoidal map of the edges of the polygon is computed using the influence graph; the conflict graphs at the key times are computed using the knowing of adjacency relationships between edges in the simple polygon.

Seidel's analysis uses in that case the fact that there is only one conflict at each step in the history. This approach can be generalized to the case of multiple conflicts, and allows other applications : skeleton of a simple polygon in $O(n \log^\star n)$ time and Delaunay triangulation of points knowing the minimum spanning tree in $O(n \log^\star n)$ time [Dev92]. This result improves previous known bounds. The skeleton or edge Vorono diagram or medial axis is a famous structure used in computational geometry but also in other domains. For the minimum spanning tree, the result is in fact more general: given a con-

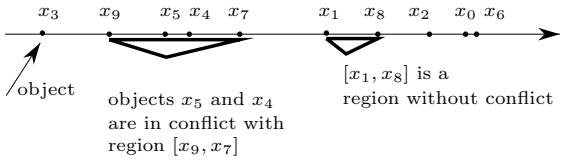

Figure 1: The sorting example

nected subset of bounded degree of the Delaunay triangulation spanning the set of points, the whole triangulation can be reconstructed in $O(n \log^\star n)$ time. The minimum spanning tree verifies this hypotheses. This result proves a kind of equivalence between Delaunay triangulation and minimum spanning tree, it was already known that the minimum spanning tree can be extracted from the Delaunay triangulation in linear time [PS85].

We will give below some applications of these algorithms in computational geometry, but before that the general principles will be illustrated using the example of sorting n real values. This paper only provides a general idea of the subject, readers interested in more details or in complete proofs can read other references [CS89, Mul93, BY95, Tei93].

2 Conflict graph and influence graph

We first need some definitions or more exactly, a description of the problem in a general framework matching different applications.

Our problem must be expressed in term of objects, regions and conflicts. *Objects* are the input of the problem : in the case of sorting, objects are real numbers. The set $\mathcal{S} = \{x_i\}_{i \in \{0,\ldots,n\}}$ must be sorted. A *region* can be seen as some geometric guy determined by a small number (bounded by a constant) of objects. In the case of sorting, a region is determined by two objects x_i and x_j and is in fact the interval $[x_i, x_j]$. Then, we have to define the conflict notion between regions and objects; still in our sorting example, a number conflicts $[x_i, x_j]$ just if it belongs to $]x_i, x_j[$ (see figure 1).

Once objects, regions and conflicts are defined, the algorithms proposed in the sequel will allow to compute the set of regions defined by the objects belonging to a given set $\mathcal{S}$ and without conflicts with these objects. Sorting $\mathcal{S}$ can be described in that way, since an interval $[x_i, x_j]$ is without conflicts if x_i and x_j are consecutive in the set of sorted numbers.

2.1 The approach "storing conflicts"

The first method used consists in computing the result for the k first objects and storing the conflicts between the already computed regions and the non yet inserted objects in a structure called conflict graph [CS89]. When the $(k+1)$th object is inserted, the conflict graph allows a direct access to the regions conflicting this object, these regions must be deleted from the current result, and the new created regions must be deduced from the deleted ones and the new object. The conflicts between new regions and non inserted objects must be computed too and stored in the conflict graph.

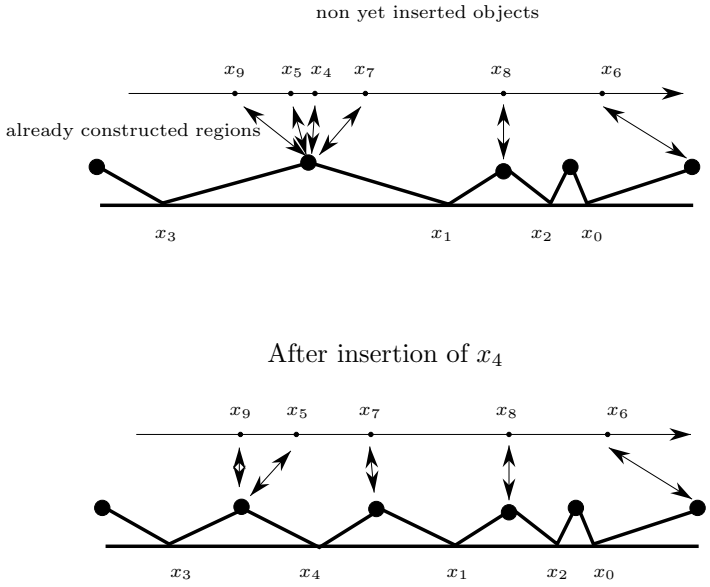

Figure 2: The conflict graph

To be more concrete, we will illustrate these ideas on the sorting example. The k first numbers $x_i, i \leq k$ have been sorted and conflicts are known, that is for every number $x_l, l > k$ we know to which interval $[x_i, x_j]$ it belongs to, among the intervals $[x_i, x_j]$ $i, j \leq k$ that have no conflict with the k first numbers. In that example a number conflicts only one region.

Now, we look at the insertion of x_{k+1}. We know, from the conflict graph, the interval $[x_i, x_j]$ containing it, so this interval is no longer *without conflict* thus we have to delete it and create two new intervals $[x_i, x_{k+1}]$ and $[x_{k+1}, x_j]$. Then the $k+1$ first numbers are sorted, and we need to update the conflict graph: numbers which were conflicting other intervals than $[x_i, x_j]$ are not affected, and those conflicting $[x_i, x_j]$ are now in conflict with either $[x_i, x_{k+1}]$ or $[x_{k+1}, x_j]$ (see figure 2). We have to notice that this last step of update of the conflict graph looks like *Quicksort* algorithm

> To recursively sort numbers in interval $[x_i, x_j]$ choose a random number among them, say x_{k+1}, and divide the numbers inside $[x_i, x_j]$ into the ones smaller than x_{k+1} and the ones bigger.

To summarize, the conflict graph algorithm for sorting is just another point of view on quicksort.

This conflict graph method has been very powerful for various applications but has the disadvantage to be static: all objects must be known in advance to initialize the conflict graph with an unique region conflicting all objects. The next paragraph will propose a semi-dynamic approach solving the same problems.

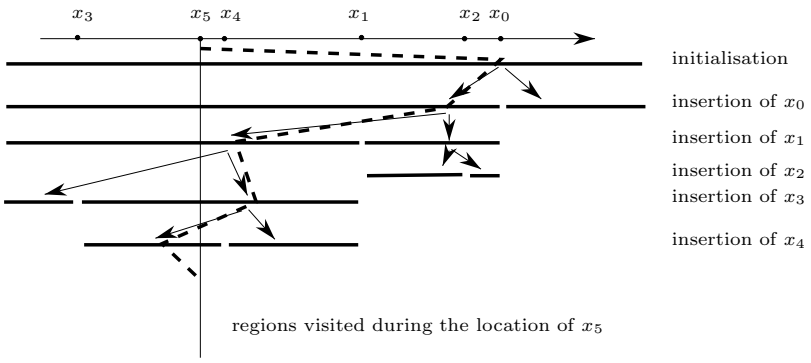

Figure 3: The influence graph

2.2 The historical approach

In fact, the conflict graph can be replaced by another structure which, instead of storing the conflicts of non yet inserted objects, locates the regions in conflict with the new object. This approach yields semi-dynamic algorithms, objects are not known in advance but only when they are inserted. The basic idea of the influence graph [BDS+92] consists in remembering the history of the construction. When the insertion of a new object makes the conflicting regions disappear, they are not deleted but just marked *inactive*. The regions created are linked to existing regions in the influence graph in order to locate further conflicts. This idea of using the history appeared in computational geometry with the Delaunay tree [BT86, BT93] and was used in various other works for example [Tei93, Sei91, GKS92].

We now detail the case of sorting. The influence graph is in this case a binary tree whose nodes are intervals, the two sons of a node correspond to the splitting of that interval into two sub-intervals. When a new number x_{k+1} is inserted, it is located in the binary tree, the leaf containing it becomes an internal node, its interval $[x_i, x_j]$ is split into two with respect to the new inserted number (see Figure 3). Thus for sorting, the influence graph is nothing else than an usual binary search tree (without balancing scheme). In fact, the comparisons done in the two algorithms are exactly the same. If x_i and $x_j, i < j$ must be compared, they are compared during the insertion of x_i in the conflict graph and during the insertion of x_j in the influence graph. This likeness between the conflict and influence graphs is general, the conflict tests computed are the same, they are only delaied to achieve semi-dynamic algorithms.

2.3 Complexity

The algorithms above, as they are presented, are not randomized. They are incremental algorithms, updating a result (the set of regions without conflict) each time a new object is inserted. If a classical complexity analysis (in the worst case) is done, results are very bad, because the insertion of a new object may change a lot of things in the current result.

Now, we will randomize the algorithm, that is introduce some randomness,

in a very simple way: objects are no longer inserted in an order determined by the user, but in a random order. From a practical point of view, the objects can be shuffled before their insertion but they must be known in advance and the algorithm is static. To exploit the semi-dynamic capabilities of the influence graph, the objects must be processed in the order given by the user. In many applications this order is random enough to get efficient results.

Thus the main hypothesis is the choice of a random order to introduce the input. We do not give here general proofs of complexity [BDS+92, Dev92] but we illustrate the backward analysis technique [Sei91] on the example of sorting.

The cost of inserting the nth number x_n in the influence graph (that is a binary search tree) is analyzed. Just recall that all the intervals without conflict existing at a given time during the construction remain in the graph. Just after the insertion of the kth number, only one interval $[x_i, x_j]$ (without conflict at stage k) is in conflict with x_n, then the randomized hypothesis yields that x_i is the kth number with probability $\frac{1}{k}$ and the same for x_j. In one word: "the region conflicting x_n at stage k was created at stage k with probability $\frac{2}{k}$". Thus the number of regions conflicting x_n is easily obtained by summing over their creation stage.

Expected insertion cost for x_n

$$
\begin{aligned}
&= \text{Number of regions in the graph conflicting } x_n \\
&= \sum_{1 \le k < n} \text{Probability for region conflicting at stage } k \text{ was created at stage } k \\
&= \sum_{1 \le k < n} \frac{2}{k} \\
&= O(\log n)
\end{aligned}
$$

The cost of insertion of a number in the influence graph is $O(\log n)$ and the cost of sorting the n numbers is $O(n \log n)$.

Similar techniques apply in general cases. Time and space complexity are computed depending on the expected size of partial results. Expected is here in a randomized sense of course, the expected size of the result at stage k is the size of a random sample of k objects among the n objects. More formally:

Theorem : *If $f_0(k)$ is the expected number of regions without conflict determined by a random sample of size k of a set $\mathcal{S}$ of n objects, then regions without conflicts determined by $\mathcal{S}$ can be computed in time $O\left(\sum_{j=1}^{n} f_0(\lfloor \frac{n}{j} \rfloor)\right)$ and space $O\left(\sum_{j=1}^{n} \frac{f_0(\lfloor \frac{n}{j} \rfloor)}{j}\right)$, using conflict graph or influence graph technique provided that the objects are inserted in a random order.*

Corollary : *If $f_0(k) = O(k)$ then the regions without conflict can be determined in $O(n \log n)$ time and $O(n)$ space.*

3 Fully dynamic algorithms

As described above, using a technical trick (inserting the objects in a random order) we get semi-dynamic algorithms, simple to code and with good complexities.

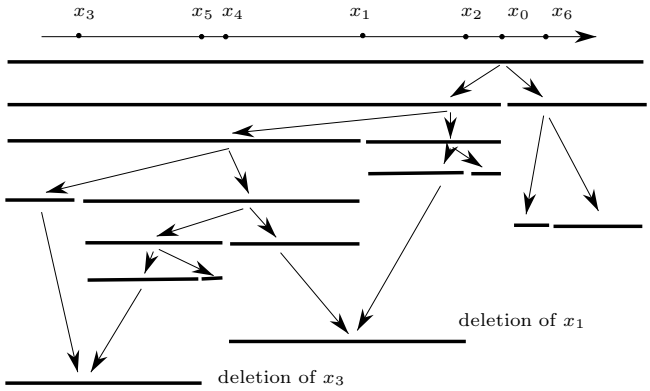

Figure 4: History of insertions and deletions

Then, numerous papers deal with fully dynamic algorithms: the objects may be removed from the structure. This important problem is very difficult in computational geometry, for example, the convex hull problem in the plane has no fully dynamic optimal (logarithmic) solution, but only a solution with $O(\log^2)$ complexity per insertion or deletion [OvL81].

Randomized algorithms gave the hope of algorithms which are not optimal for every sequence of insertions and deletions, but only "expected optimal" if the operations are randomized. More precisely, insertions may be assumed to be in a random order, and a deletion can be the deletion of any (already inserted) object with the same probability.

Several techniques were used to solve this problem. I will distinguish three: store the whole history, store the history of insertions only, or use coin flipping techniques. As for influence and conflict graphs, we will illustrate these techniques with the example of sorting to obtain dictionaries, balanced "on average", easier to manage than structures balanced in the worst case like red-black trees or AVL trees.

3.1 The historical approach, one more time

A natural idea is the following. The influence graph remembers all the successive sorted sequences of the numbers present in the structure. If the possibility of deletions is required, one can do exactly the same thing: remember all successive sequence, but know the size of the sequence is no necessarily growing. If $[x, y]$ and $[y, z]$ are two regions without conflicts (that is y is the only number between x and z) and y must be removed, a new node $[x, z]$ is created and is made child of $[x, y]$ and $[y, z]$ (see figure 4).

This approach has been first proposed by Otfried Schwarzkopf [Sch91, Sch92]. It has the great advantage of simplicity and is easy to implement. There are two main drawbacks, the first one is that the size of the graph depends on the size of the sequence of insertions and deletions and not on the number of objects really present in the structure, thus if we maintain a set of size about 1000 under a sequence of one million of insertions and deletions the structure will have a size of one million; this can be solved by cleaning the structure, each time the history is to big comparing to the number of objects present the structure is

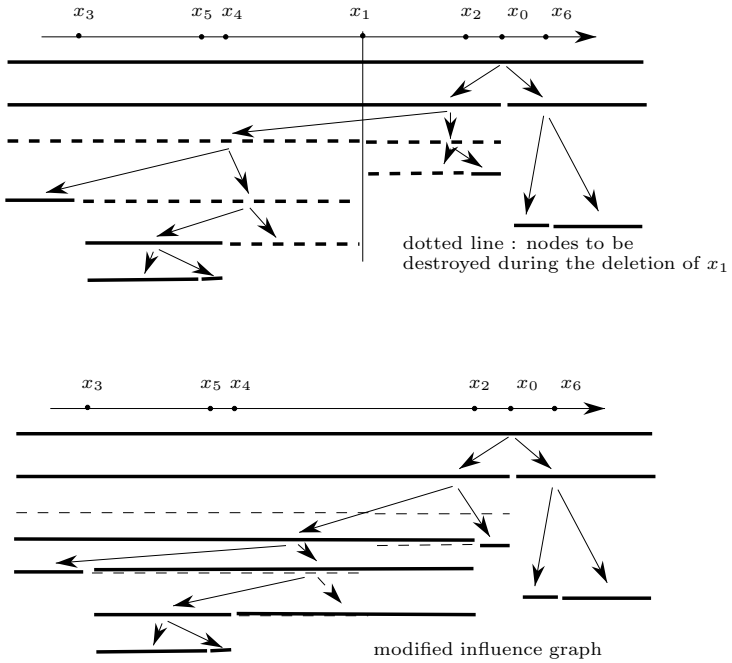

Figure 5: Insertions' history

recomputed from scratch. The second drawback is more technical, it is frequent with this kind of structure that the arity of the graph is not well controlled, a node of the graph may have many children and the right one must be found using an auxiliary location structure. The algorithm becomes less simple and the theoretical complexity is multiplied by a logarithmic factor.

3.2 Still the historical approach, but history can be modified

Other algorithms [DMT92, DTY92, CMS93, DY93] still use the history, but the history of insertions only. While only insertions are performed, the algorithm works as usual. When a deletion arises, the history of the structure is modified to take in account only the remaining objects. That is the object is deleted not only at the final stage, but also at all the intermediary stages. This approach, inspired by the "1984" novel of Georges Orwell, permits to manipulate the influence graph defined in the semi-dynamic setting. The structure is exactly the same, but the updates become more complicated in some cases.

For our favorite example, it is still simple. All nodes corresponding to intervals determined by the removed objects are destroyed, and the oldest son of these nodes is promoted to replace the destroyed node (see figure 5).

This approach needs to manipulate the influence graph not only by adding children to its leaves but also by modifying internal nodes. When an internal node is destroyed, its children need a new parent and they must be hang up to other nodes.

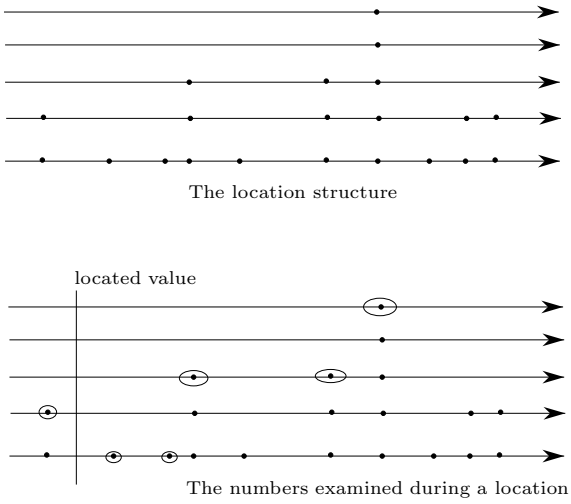

Figure 6: Coin tossing structure

3.3 Coin tossing

In some cases, another approach than the "historical" one can be used. The skip list uses a coin tossing method for sorting [Pug90]. Ketan Mulmuley proposes several methods for dynamic randomized geometric algorithms [MS91, Mul91b, Mul91c, Mul91d]. As in the previous sections, the main idea will be illustrated using the example of a dictionary structure. We will present the static version in a first step and then the way to dynamize the structure.

Let us assume that $x_1 < x_2 < \ldots < x_n$ are n sorted numbers. A sample of this set is constructed by tossing a coin for each number to decide if it belongs to the sample or not. Then, the selected set is sampled again in the same manner, and so on until there is no numbers left. The different levels of the structure are linked together, and if another number must be located in the set, it is successively located at each level. The location at one level is used as starting point for the location at the level below (see Figure 6).

The above description is static, but this structure can be dynamically maintained: when a new number is inserted, it is first located at the lower level, inserted at this level and then a coin is flipped to decide if it has to be inserted above, and so on while the coin tossing is succesful. When an object must be deleted, it is enough to delete it at each level where it appears. The structure is a kind of tree, with non fixed arity, the update is very easy and the structure is balanced on average and the arity is constant on average. In the case of dictionary structure, locations are done in logarithmic time, and update in constant time (after location).

4 Accelerated algorithms

4.0.1 Conflict and influence graphs can cooperate

As it has already been explained, the conflict graph knows all the conflict although the influence graph helps to deduce the conflicts at stage k from the conflicts at stage $k-1$. The idea introduced by Seidel [Sei91] consists in a collaboration of these two structures.

When an object is inserted in the influence graph, the conflicts at stage 1 are searched, the conflicts at stage 2 are deduced and so on. The idea is to initialize the search directly at an intermediary stage k and to deduce the conflicts at stage $k+1 \ldots$ to n. Of course the conflicts at stage k must be found using something else. These conflicts at stage k can be stored in a conflict graph, this graph is not updated during the construction, but just constructed for each key-time k. This combination of conflict and influence graph can be efficient only if there is a direct way to compute the conflict graph at the key-time.

4.0.2 Complexity

In the case where $f_0(r)$, the expected number of regions without conflict determined by a random sample of size r of a set $\mathcal{S}$, is $O(r)$, the location time in the influence graph is $\sum_{j=1}^{n} \frac{1}{j} = \log n$ summing over the different stages where the new object has been located. In our context, the complexity of a partial location between stages k and n is $\sum_{j=k}^{n} \frac{1}{j} = \log n/k$.

4.0.3 Choosing the key time

The algorithm uses phases of insertion in the influence graph and phases of construction of the conflict graph at key-times. We will assume that the construction of conflict graphs can be done in linear time, and we will balance the different phases. More precisely, we will insert objects during a linear time, then we construct a conflict graph and we will have a new linear phase of insertion...

Inserting an object in the influence graph has a logarithmic cost, thus we can proceed $\frac{n}{\log n}$ objects in the first phase. Whence the conflict graph at the key-time $\frac{n}{\log n}$ is known, the insertion cost is bounded, not only by $O(\log n)$, but by $O(\log \frac{n}{n/\log n}) = O(\log \log n)$. To obtain a linear cost for the second insertion phase, we choose the next key-time at the $\frac{n}{\log \log n}$th object. The ith key-time is chosen after the $\frac{n}{\log^{(i)} n}$ objects (where $\log^{(i)}$ means $\log \log \ldots \log$ i times).

The whole set of objects is processed when $\log^{(i)} n < 1$ that is when $i = \log^\star n$ (that is the definition of $\log^\star$). The algorithm has $\log^\star n$ insertion phases and $\log^\star n$ construction of conflict graph phases, all in linear time. The whole complexity is thus $O(n \log^\star n)$.

We will see in the sequel several applications of this principle improving complexity from $O(n \log n)$ to $O(n \log^\star n)$ for triangulation and skeleton of a simple polygon, Voronoi diagram of points knowing the EMST, and possibly other applications in the future.

It is proved that this kind of strategies cannot allow better complexity results than $O(n \log^\star n)$ [CMS93] although the best lower bound for these problem is linear.

5 Applications

We will now describe applications of the influence graph technique. We just describe how the general framework apply by defining the objects, regions and conflicts in each case.

5.1 Convex hull

Convex hull is the favorite paradigm of computational geometers. Although the description of the problem is fairly simple, its solution takes in account all aspects of computational geometry.

5.1.1 Semi-dynamic algorithms

To apply the general framework of the influence graph to planar convex hull, just define objects as points, regions as half planes defined by two points and say that a point conflicts an half-plane if it belongs to it.

The convex hull problem reduces to the computation of non conflicting regions, indeed the region defined by p and q has no conflict if and only if pq is an edge of the convex hull. Here $f_0(k) = O(k)$ (the size of the convex hull is linear) and application of the influence graph theorem yields:

Theorem : *The influence graph maintains the convex hull of n points in the plane in expected time $O(\log n)$ per insertion, if the order of insertion is random.*

In higher dimensions, objects are points and regions are half spaces. If there is no hypothesis on the data, $f_0(k)$ is bounded by the worst case size of a convex hull in dimension d, that is $O(n^{\lfloor \frac{d}{2} \rfloor})$.

Theorem : *The influence graph maintains the convex hull of n points in dimension d in expected time $O(\log n)$ per insertion if $d \leq 3$ and $O(n^{\lfloor \frac{d}{2} \rfloor - 1})$ per insertion if $d \geq 4$. The order of insertion is assumed to be random.*

Using duality, all these results apply to the problem of intersecting n half-spaces in dimension d.

k-sets, k-levels These results can also extend to k-sets and k-levels if we look at the regions having at most k conflicts [BDT93, Mul91a].

5.1.2 Dynamic algorithms

The above algorithm is semi-dynamic, points can be added but not removed. The difficulty of dynamizing convex hull is that during the deletion of a point many other points may appeared on the hull. Thus it is necessary to store all the points, even those which are not on the current convex hull.

Clarkson, Mehlhorn and Seidel [CMS93] have proposed a solution based on the history of insertion technique.

Theorem : *The convex hull of n points in dimension d can be dynamically maintained in an expected update time $O(n^{\lfloor \frac{d}{2} \rfloor - 1})$ if $d \geq 4$ and $O(\log n)$ if $d \leq 3$. The points are assumed to be inserted in a random order, and a deleted point can be any point equally likely.*

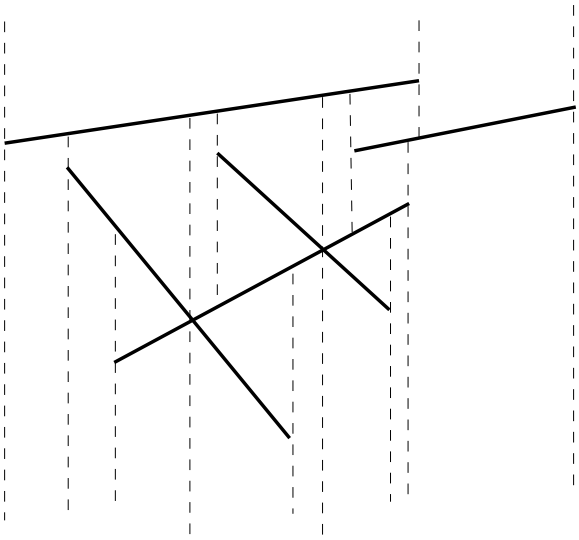

Figure 7: Trapezoidal map of an arrangement of line segments

5.2 Arrangements

Another classical problem in computational geometry is the computation of the *arrangement* of curves or surfaces, that is the subdivision of the plane or the space induced by a set of curves or surfaces.

5.2.1 Arrangements of line segments

One of the simplest versions of this problem is "given a set of line segments in the plane, compute all the intersection points". The difficult point is that the size of the result can change a lot depending on the input, for a set of n line segments, the number of intersection points t may vary between 0 and n^2. In such a case the most interesting is to get *output sensitive* algorithms. A very classical computational geometry algorithm is Bentley Ottmann plane sweep [PS85] that solves this problem in time $O((n+t)\log n)$. There exists also a determinist optimal, but complicated algorithm [CE92] of complexity $\Theta(n \log n + t)$. For this problem also, the influence graph yields to a simple algorithm, optimal on average, which is a very good candidate to effective implementation.

Describing the algorithm reduces to describing the influence graph. In fact, the algorithm constructs the trapezoidal map of the arrangement: for each segment end-point and each intersection point, two vertical rays are drawn, and stopped at the first encountered line segments above and below. If objects are line segments, regions are trapezoids (defined by at most 4 segments) and a segment conflicts a trapezoid if they intersect. Then the trapezoids of the map are exactly the regions without conflicts. Thus the influence graph mechanism works! Here, $f_0(r)$ is r plus the average number of intersection points in a random sample of size r. An intersection point of the whole arrangement appears in the sample if the two segments defining it have been chosen, thus with probability $\left(\frac{r}{n}\right)^2$. So, $f_0(r) = r + \frac{r^2}{n^2}t$ and we conclude.

Theorem : *The influence graph maintains the trapezoidal map of an arrangement of n line segments in expected insertion time $O(\log n + \frac{t}{n})$ if the insertion sequence is randomized (t is the size of the arrangement).*

The trapezoidal map can be generalized for arrangements of planar curves of bounded degree, and thus the influence graph method too. The differences are a more complex description of the algorithm, and also an increase of the constants hidden in the big O.

5.2.2 Arrangements of triangles

The natural generalization of the preceding work to the three dimensional case consists in using vertical walls passing through triangles' edges and through intersection of triangles. But such a decomposition must be refined because the cells have not a bounded description. Boissonnat and Dobrindt get an interesting result, slightly sub-optimal, for the computation of the lower envelope of an arrangement of triangles (the visible part from infinity) [BD92]:

Theorem : *The influence graph computes the lower envelope of n triangles in expected time $O(n^2\alpha(n)\log n)$ if the insertion sequence is randomized (α is the inverse of the Ackermann function and is extremly slowly growing)*

5.2.3 Computing a single cell

The above algorithm can also be adapted to compute only one cell of the arrangement of line segments, this can be very interesting, if of course we are interested in only one cell, and if the size of this cell is smaller than the size of the whole arrangement. Chazelle *et al.* have proposed to maintain the cell containing a given target point. At each insertion, the possible splitting of the cell by the new segment is examined. The final expected complexity of this algorithm is $O(n\alpha(n)\log n)$ [CEG$^+$93].

De Berg, Dobrindt and Schwarzkopf [dBDS94] proposed another solution, this possible splitting of the cell is not examined for each new segment but only at several moments. This approach yields the same complexity and can be generalized in three dimensions.

5.2.4 Simple polygon triangulation

The triangulation of a simple polygon is also a classical problem of computational geometry. During the past years the complexity of this problem decrease from $O(n^2)$ to $O(n\log n)$, $O(n\log\log n)$, $O(n\log^\star n)$ and finally Chazelle proposed a deterministic solution in linear time [Cha93], but, one more time, this algorithm is optimal only in theory and is fairly impractical. The trapezoidal decomposition simply yields a triangulation of the set of line segments, thus the randomized algorithm for trapezoidal map allows to triangulate a polygon in expected $O(n\log n)$ time.

The accelerated algorithms arise here. From the trapezoidal map of a sample of the segments and the knowledge of the simple polygon, it is easy to construct the conflict graph with the line segments which are not in the sample, it suffices to follow the adjacencies in the trapezoidal map when walking on the boundary of the simple polygon. The cost of such walk is clearly linear in the size of the conflict graph. This is exactly the case described for applying the accelerated

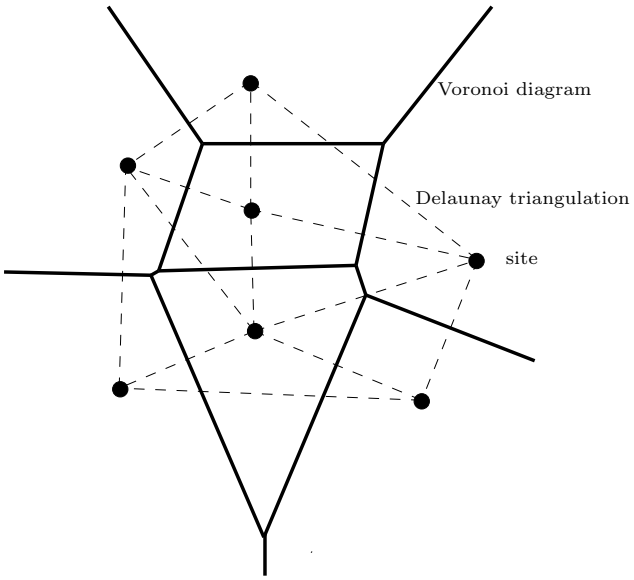

Figure 8: Voronoi diagram of 7 sites

framework, and we obtain a simple randomized and efficient algorithm even if it is slightly sub-optimal [Sei91].

Theorem : *The trapezoidal map of a simple n-gon can be computed in expected time $O(n \log^\star n)$. A triangulation can be deduced in $O(n)$ time.*

Recently, this approach has been combined with the single cell algorithm of De Berg, Dobrindt and Schwarzkopf [dBDS94] to yields the following theorem [dBDDS95]:

Theorem : *A single cell of the intersection of two simple polygon can be computed in expected time $O(n(\log^\star n)^2)$, where n is the total size of the polygons.*

5.3 Voronoi diagrams

5.3.1 Points Voronoi diagrams

We recall the definition of the Voronoi diagram in the plane. Given a set of points in the plane called sites, the Voronoi diagram is the partition of the plane such that a cell is composed of the set of points which are closer to one site than to the others (Figure 8). This structure can answer the classical *post office problem* "Where is the nearest post office to give my letter?" or more formally, "what is the nearest site from a given query point?". The dual graph of the Voronoi diagram, the Delaunay triangulation, is obtained by linking two sites if their Voronoi regions are adjacent.

A characteristic property of the Delaunay triangulation is that the circle circumscribing a triangle does not contain any sites, or in the dual, the center of the circle is at the same distance from three points (triangle vertices) and closer to these sites than to any other sites. Using this property, it suffices to say that objects are sites, regions are triangles and that a point conflicts a

triangle if the point is inside the circle circumscribing the triangle so that the Delaunay triangulation corresponds to the triangles without conflicts. Then the result follows [BDS+92, Dev92]. The algorithm can be fully dynamized using the history of insertions [DMT92].

Theorem : *The influence graph maintains the Delaunay triangulation and the Voronoi diagram of n sites in the plane in $O(\log n)$ expected insertion time $O(\log \log n)$ expected deletion time if the insertion sequence is randomized and if the deleted site is any present site with the same probability.*

Theorem : *The influence graph maintains the Delaunay triangulation and the Voronoi diagram of n sites in dimension $d \geq 3$ in $O(n^{\lfloor \frac{d+1}{2} \rfloor - 1})$ expected insertion time if the insertion sequence is randomized.*

Order k Voronoi diagram In order k Voronoi diagrams cells no longer corresponds to points closer from one site than others but closer from k sites from other. This diagrams can be computed with the same kind of methods [BDT93, AS92].

5.3.2 Voronoi diagrams of line segments

Voronoi diagrams can also be defined for other kind of sites than points, the most frequent case is the Voronoi diagram of line segments or polygons (figure 9).

This diagram is also called skeleton or medial axis. One more time, the influence graph technique applies. The objects are the line segments, regions are associated to the edges of the Voronoi diagram, which are defined by 4 objects, and finally, an object conflicts a region when the object intersects the union of the circles centered on the Voronoi edge and touching the closest obstacle (Figure 10).

Theorem : *The influence graph maintains the edge Delaunay triangulation and the Voronoi diagram of n line segments in the plane in $O(\log n)$ expected insertion time if the insertion sequence is randomized.*

5.3.3 Dog and postmen problem

There are two ways of dynamizing a problem, up to now we have modified the data by insertions or deletions, but one can want a continuous modification by moving the data along trajectories depending on the time. This two notions of dynamization can be combined to solve the following problem.

The sites are postmen moving in the plane at constant speed along lines and the query is "a dog, capable of maximal speed v, wakes up at point (x, y) at time t and wants to bite a postman as soon as possible. What is the best postman?"

The algorithm can allow the insertion and deletion of postmen [DG93].

Theorem : *The Voronoi diagram of a set of n sites moving with constant speed in the plane can be maintained in expected time $O(f_{\mathcal{S}}(n)/n + \log n)$ per insertion or deletion, where $f_{\mathcal{S}}(r)$ is the average size of the diagram for a sample of r sites and with usual randomized hypotheses. Queries like "What is the nearest postman of a given point at time t" and dog queries can be answered in $O(\log^3 n)$ if the dog is faster than all postmen.*

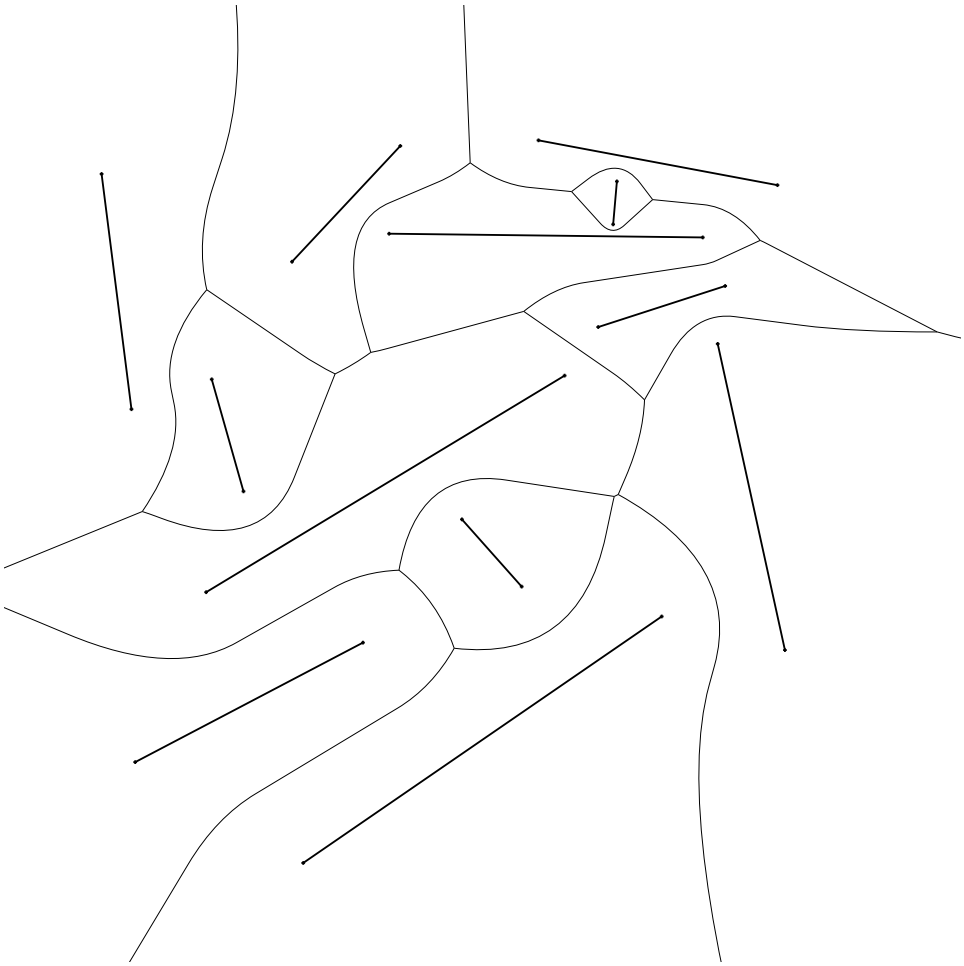

Figure 9: Example of Voronoi diagram of line segments

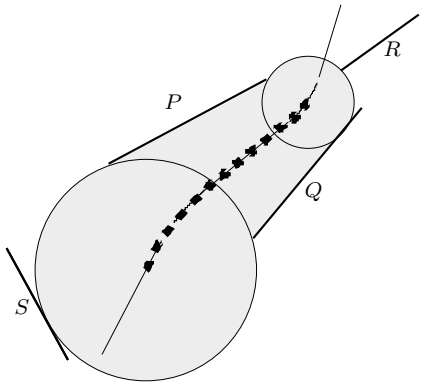

Figure 10: Region definition for the Voronoi diagram of line segments

5.3.4 Voronoi diagram and minimum spanning tree

The Euclidean minimum spanning tree (EMST) of n sites in the plane is the tree having as nodes the sites and where the edges linking this nodes are chosen to minimize the total length of all edges. It is known that these edges belong to the Delaunay triangulation and the most efficient method to compute the EMST consists in computing the Delaunay triangulation and extracting the EMST in linear time [PS85].

The lower bounds for both problems, the EMST and the Delaunay triangulation, are $\Omega(n \log n)$. An interesting theoretical problem is the complexity of the transformation between these two structures. The accelerated scheme combining conflict and influence graphs applies to the Delaunay triangulation provided that a spanning subgraph of bounded degree of the Delaunay triangulation is known, and the minimum spanning tree is such a subgraph.

Theorem : *The Voronoi diagram (the Delaunay triangulation) of n sites whose EMST is known can be computed in expected time $O(n \log^\star n)$.*

5.3.5 Medial axis of a simple polygon

The accelerated method applies also to the Voronoi diagram of line segments. If the set of line segments is known to form a simple polygon, this information can be exploited to compute the conflicts between the polygon and the Voronoi diagram of a sample of the polygon edges [Dev92] :

Theorem : *The skeleton of a simple n-gon can be computed in expected time $O(n \log^\star n)$.*

References

[AS92] F. Aurenhammer and O. Schwarzkopf. A simple on-line randomized incremental algorithm for computing higher order Voronoi diagrams. *Internat. J. Comput. Geom. Appl.*, 2:363–381, 1992.

[BD92] J. D. Boissonnat and K. Dobrindt. Randomized construction of the upper envelope of triangles in R^3. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 311–315, 1992.

[BDS+92] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete Comput. Geom.*, 8:51–71, 1992.

[BDT93] J.-D. Boissonnat, O. Devillers, and M. Teillaud. An semidynamic construction of higher-order Voronoi diagrams and its randomized analysis. *Algorithmica*, 9:329–356, 1993.

[BT86] J.-D. Boissonnat and M. Teillaud. A hierarchical representation of objects: The Delaunay tree. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 260–268, 1986.

[BT93] J.-D. Boissonnat and M. Teillaud. On the randomized construction of the Delaunay tree. *Theoret. Comput. Sci.*, 112:339–354, 1993.

[BY95] J-D. Boissonnat and M. Yvinec. *Géométrie algorithmique*. Ediscience international, Paris, 1995.

[CE92] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39:1–54, 1992.

[CEG+93] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir, and J. Snoeyink. Computing a face in an arrangement of line segments. *SIAM J. Comput.*, 22:1286–1302, 1993.

[Cha93] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete Comput. Geom.*, 10:377–409, 1993.

[Cla87] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete Comput. Geom.*, 2:195–222, 1987.

[CMS93] K. L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Comput. Geom. Theory Appl.*, 3(4):185–212, 1993.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[dBDDS95] M. de Berg, O. Devillers, K. Dobrindt, and O. Schwarzkopf. Computing a single cell in the union of two simple polygons. Research Report 2626, INRIA, BP93, 06902 Sophia-Antipolis, France, 1995.

[dBDS94] M. de Berg, K. Dobrindt, and O. Schwarzkopf. On lazy randomized incremental construction. In *Proc. 26th Annu. ACM Sympos. Theory Comput.*, pages 105–114, 1994.

[Dev92] O. Devillers. Randomization yields simple $O(n \log^* n)$ algorithms for difficult $\Omega(n)$ problems. *Internat. J. Comput. Geom. Appl.*, 2(1):97–111, 1992.

[DG93] O. Devillers and M. Golin. Dog bites postman: Point location in the moving Voronoi diagram and related problems. In *Proc. 1st Annu. European Sympos. Algorithms (ESA '93)*, volume 726 of *Lecture Notes in Computer Science*, pages 133–144. Springer-Verlag, 1993.

[DMT92] O. Devillers, S. Meiser, and M. Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. *Comput. Geom. Theory Appl.*, 2(2):55–80, 1992.

[DTY92] O. Devillers, M. Teillaud, and M. Yvinec. Dynamic location in an arrangement of line segments in the plane. *Algorithms Rev.*, 2(3):89–103, 1992.

[DY93] K. Dobrindt and M. Yvinec. Remembering conflicts in history yields dynamic algorithms. In *Proc. 4th Annu. Internat. Sympos. Algorithms Comput. (ISAAC 93)*, volume 762 of *Lecture Notes in Computer Science*, pages 21–30. Springer-Verlag, 1993.

[GKS92] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[MS91] K. Mulmuley and S. Sen. Dynamic point location in arrangements of hyperplanes. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 132–141, 1991.

[Mul91a] K. Mulmuley. On levels in arrangements and Voronoi diagrams. *Discrete Comput. Geom.*, 6:307–338, 1991.

[Mul91b] K. Mulmuley. Randomized multidimensional search trees: dynamic sampling. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 121–131, 1991.

[Mul91c] K. Mulmuley. Randomized multidimensional search trees: further results in dynamic sampling. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 216–227, 1991.

[Mul91d] K. Mulmuley. Randomized multidimensional search trees: lazy balancing and dynamic shuffling. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 180–196, 1991.

[Mul93] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, New York, 1993.

[OvL81] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, New York, NY, 1985.

[Pug90] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 35:668–676, 1990.

[Sch91] O. Schwarzkopf. Dynamic maintenance of geometric structures made easy. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 197–206, 1991.

[Sch92] O. Schwarzkopf. *Dynamic Maintenance of Convex Polytopes and Related Structures*. Ph.D. thesis, Fachbereich Mathematik, Freie Universität Berlin, Berlin, Germany, June 1992.

[Sei91] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1:51–64, 1991.

[Tei93] M. Teillaud. *Towards dynamic randomized algorithms in computational geometry*, volume 758 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.