Chapter 6

Dynamic randomized algorithms

The geometric problems encountered in this chapter are again stated in the abstract framework of objects, regions, and conflicts introduced in chapter 4. A *dynamic* algorithm maintains the set of regions defined and without conflict over the current set of objects, when the objects can be removed from the current set as well as added. In contrast, on-line algorithms that support insertions but not deletions are sometimes called *semi-dynamic.*

Throughout this chapter, we denote by $\mathcal{S}$ the current set of objects and use the notation introduced in the previous two chapters to denote the different subsets of regions defined over $\mathcal{S}$. In particular, $\mathcal{F}_0(\mathcal{S})$ stands for the set of regions defined and without conflict over $\mathcal{S}$. To design a dynamic algorithm that maintains the set $\mathcal{F}_0(\mathcal{S})$ is a much more delicate problem than its static counterpart. In the previous chapter, we have shown how randomized incremental methods provide simple solutions to static problems. In addition, the influence graph techniques naturally lead to the design of semi-dynamic algorithms. In this chapter, we propose to show how the combined use of both conflict and influence graphs can yield fully dynamic algorithms.

The general idea behind our approach is to maintain a data structure that meets the following two requirements:

- It allows conflicts to be detected between any object and the regions defined and without conflict over the current subset.
- After deleting an object, the structure is identical to what it would have been, had the deleted object never been inserted.

Such a structure is called an *augmented influence graph,* and can be implemented using an influence graph together with a conflict graph between the regions stored in the influence graph and the current set of objects. In some cases, we might be able to do without the conflict graph.

In section 6.2, we describe the augmented influence graph and how to perform insertions and deletions. The randomized analysis of these operations is given in section 6.3. This analysis assumes a probabilistic model which is made precise and unambiguous in section 6.1. The general method is used in section 6.4 to design a dynamic algorithm that builds the vertical decomposition of a set of segments in the plane.

This chapter also uses the terminology and notation introduced in the previous two chapters. To ease the reading process, some definitions are recalled in the text or in the footnotes.

6.1 The probabilistic model

The current set of objects, denoted by S, is the result of a sequence of insertions and deletions. Due to the second requirement which we stated earlier, the data structure does not keep track of the deleted objects. Consequently, at any given time, the data structure depends only on S and on the order in which the objects in S were introduced. In fact, an object stored in the data structure may have been inserted and removed several times, yet the current state of the data structure only keeps track of the last insertion.

At any given time, the data structure only depends on the *chronological sequence* $\Sigma = \{O_1, O_2, \ldots, O_n\}$ which is the sequence of objects in S in the order of their last insertion.

The randomized analysis of a dynamic algorithm assumes that:

- If the last operation is an insertion, each object in the current set S is equally likely to have been inserted in this operation.
- If the last operation is a deletion, each object present prior to the deletion is equally likely to be deleted in this operation.

It follows from these two assumptions that the chronological sequence Σ is random, and that every permutation of the objects in S is equally likely to occur in Σ. Let the current set of object S be of size n, and let i be an integer in $\{1, \ldots, n\}$. Each object in S is the object O_i of rank i in Σ with uniform probability $1/n$. Moreover, in a deletion, the object O_i of rank i in Σ is deleted with uniform probability $1/n$.

We let S_i be the subset $\{O_1, O_2, \ldots, O_i\}$ of the first i objects in the chronological sequence. The probabilistic model implies that for i, $1 \leq i \leq n$, S_i is a random i-sample[1] of S and, for each pair (i, j) such that $1 \leq i \leq j \leq n$, S_i is a

[1]We may recall that a random i-sample is a random subset of size i of S. Its elements are chosen in a way that makes all the $\binom{n}{i}$ possible subsets of size i of S equally likely.

random i-sample of $\mathcal{S}_j$.

6.2 The augmented influence graph

The augmented influence graph obtained after a sequence of insertions and deletions that results in a set $\mathcal{S}$, is determined only by the chronological sequence Σ of the objects in $\mathcal{S}$ and is denoted $\mathcal{I}a(\Sigma)$. The augmented influence graph $\mathcal{I}a(\Sigma)$ is connected, directed, and acyclic. It has the same nodes and arcs as the graph built by an on-line algorithm which inserts the objects of the sequence Σ in the order of Σ. There is a node in the graph for each region that belongs to $\bigcup_{i=1}^{n} \mathcal{F}_0(\mathcal{S}_i)$, where $\mathcal{F}_0(\mathcal{S}_i)$ denotes the set of regions defined and without conflict over $\mathcal{S}_i$. Let us recall that a region is characterized by two subsets of objects: the subset of objects with which it conflicts, called the influence domain of the region, and the subset of objects with bounded size that determine the region. In the following, we call each object that belongs to this second subset a *determinant* of the region. We call *creator* of a region the object of highest chronological rank among the determinants of the region. As in the preceding chapter, we use the terminology of trees to describe the structure of the augmented influence graph, and often identify a node with the region that is stored therein. This lets us speak for instance of the parent or children of a region, or of the influence domain of a node. The arcs of the influence graph maintain the *inclusion property* that the influence domain of a node is a subset of the union of the influence domains of its parents. For dynamic algorithms, we demand that arcs of the augmented influence graph also ensure a *second inclusion property* stating that, apart from its creator, the set of determinants of a region is contained in the set of determinants of its parent regions.

In addition to the usual information stored in the influence graph, the augmented influence graph stores a conflict graph between the objects in the current set $\mathcal{S}$ and the regions stored in the nodes of the influence graph. This conflict graph is represented as in the preceding chapter by a system of interconnected lists: To each region F stored at a node of the influence graph corresponds a list $L'(F)$ of objects of $\mathcal{S}$ with which it conflicts. To each object O in the current set $\mathcal{S}$ corresponds a list $L(O)$ of regions stored in the entire influence graph that conflict with O. There is a bidirectional pointer between the entry corresponding to a region F in the list $L(O)$ of an object O and the entry corresponding to O in the list $L'(F)$.

Inserting an object

Inserting an object O_n into a structure built for a set $\mathcal{S}_{n-1}$ is very similar to the operation of inserting an object in an on-line algorithm that uses an influence graph. The only difference is that, in addition to the insertion into the influence graph, we must also take care of updating the conflict lists. This can be done in two phases: a locating phase, and an updating phase.

Locating. The algorithm searches for all the nodes in the influence graph of $\mathcal{I}a(\Sigma)$ that conflict with O_n. Each time a conflict is detected, we add a conflict arc to the conflict graph, add O_n to the conflict list of the region that conflicts with it, and add this region to the list $L(O_n)$.

Updating. A node of the influence graph is created for each region in $\mathcal{F}_0(\mathcal{S}_n)$ determined by a set of objects that contains O_n. This node is also linked to parent nodes so that the two inclusion properties hold.

We may recall that a region in $\mathcal{F}_0(\mathcal{S}_n)$ is said to be created by O_n if it is determined by a set of objects that contains O_n. Similarly, a region of $\mathcal{F}_0(\mathcal{S}_{n-1})$ is said to be killed by O_n if it conflicts with O_n. More generally, a region stored in a node of the influence graph $\mathcal{I}a(\Sigma)$ has a *creator* in Σ, and a *killer* if it is not a leaf. The *creator* of F is, among all the objects that determine F, the one that has the highest rank in Σ. The *killer* of F is, among all the objects in Σ that conflict with F, the one with the lowest chronological rank.

For the rest of this chapter, we assume that the augmented influence graph satisfies the update condition 5.3.3. In particular, a node of the graph that stores a region created by O_n is linked only to nodes storing regions killed by O_n.

Deleting an object

To simplify the discussion, assume that the current set $\mathcal{S}$ has n objects, and that the current data structure is the augmented influence graph $\mathcal{I}a(\Sigma)$ corresponding to the chronological sequence $\Sigma = \{O_1, \ldots, O_n\}$. The object to be deleted is O_k, the object that has chronological rank k. The algorithm must modify the augmented influence graph to look as if O_k had never been inserted into Σ. The augmented graph must therefore correspond to the chronological sequence $\Sigma' = \{O_1, \ldots, O_{k-1}, O_{k+1}, \ldots O_n\}$.

For any integer l, $k \leq l \leq n$, let us denote by $\mathcal{S}'_l$ the subset $\mathcal{S}_l \setminus \{O_k\}$ of $\mathcal{S}$. In particular, observe that $\mathcal{S}'_k = \mathcal{S}_{k-1}$.

In what follows, an object is called a *determinant* of a region if it belongs to the set of objects that determine that region. The symmetric difference between the nodes of $\mathcal{I}a(\Sigma)$ and those of $\mathcal{I}a(\Sigma')$ can be described as follows.

1. The nodes of $\mathcal{I}a(\Sigma)$ that do not belong to $\mathcal{I}a(\Sigma')$ are determined by a set of objects that contain O_k. Therefore O_k is a determinant of those regions, and we say that such nodes (and the corresponding regions) are *destroyed* when O_k is deleted.

2. The influence graph $\mathcal{I}a(\Sigma')$ has a node that does not belong to $\mathcal{I}a(\Sigma)$ for

each region in $\bigcup_{l=k+1,\dots,n} \mathcal{F}_0(\mathcal{S}'_l)$ that conflicts with O_k. Let us say that such a node is *new* when O_k is deleted, and so is its corresponding region. A new region has a creator and, occasionally, a killer in the sequence Σ'. If the region belongs to $\mathcal{F}_0(\mathcal{S}'_l)$, conflicts with O_k, and is determined by a set of objects that contain O_l, then it is a new region after O_k is deleted, and its creator is O_l.

Nodes that play a particular role when O_k is deleted include of course the new nodes as well as the destroyed ones, but the nodes killed by O_k also have a special part to play. The nodes killed by O_k should not be mistaken for the nodes destroyed when O_k is deleted. Nodes killed by O_k correspond to regions of $\mathcal{F}_0(\mathcal{S}_{k-1})$ that conflict with O_k, whereas nodes destroyed when O_k is deleted correspond to regions that admit O_k as a determinant. The latter nodes disappear from the whole data structure when O_k is deleted. The former nodes are killed when O_k is inserted but remain in the data structure (occasionally becoming internal nodes), and they still remain after O_k is deleted.

Upon a deletion, the arcs in the influence graph $\mathcal{I}a(\Sigma)$ that are incident to the nodes destroyed by O_k disappear and the graph $\mathcal{I}a(\Sigma')$ has arcs incident to the new nodes. In particular, new nodes must be linked to some parents (which are not necessarily new nodes). Moreover, a few nodes of $\mathcal{I}a(\Sigma)$ that are not destroyed witness the destruction of some of their parents. Let us call these nodes *unhooked*. They must be rehooked to other parents.

Again, deletions can be carried out in two phases: a *locating phase*, and a *rebuilding phase*.

Locating. The algorithm must identify which nodes of the influence graph $\mathcal{I}a(\Sigma)$ are in conflict with O_k, which nodes have to be destroyed, and which are unhooked. Owing to both inclusion properties, this can be done by a traversal of the influence graph. This time, however, we not only visit the nodes that conflict with O_k, but also those which admit O_k as a determinant. The destroyed or unhooked nodes are inserted into a dictionary which will be looked up during the rebuilding phase.

Rebuilding. The first thing to do is to effectively remove all the destroyed nodes. Those nodes can be retrieved from the dictionary, and all the incident arcs in the graph are also removed from the graph. The conflict lists of the nodes which conflict with O_k are also updated accordingly. We shall not detail these low-level operations any further, as they should not raise any problems. Next, we must create the new nodes, as well as their conflict lists; we must also hook these new nodes and rehook the nodes that were previously unhooked. The detail of these operations depends on the nature of the specific problem in hand. The general design is always the same, however: the algorithm *reinserts* one by one, and in chronological order, all the objects O_l whose rank l is higher than k and that are creators of at least one new or unhooked region. To *reinsert* an object involves creating a node for each new region created by O_l, hooking this node into the influence graph, setting up its conflict list, and finally rehooking all the unhooked nodes created by O_l.

To characterize the objects O_l that must be reinserted during the deletion of O_k, we must explain what *critical regions* and the *critical zone* are. For each $l \geq k$, we call *critical* those regions in $\mathcal{F}_0(\mathcal{S}'_{l-1})$ that conflict with O_k. We call *critical zone*, and denote by $\mathcal{Z}_{l-1}$, the set of those regions.

Lemma 6.2.1 *Any object O_l of chronological rank $l > k$ that is the creator of a new or unhooked node when O_k is deleted conflicts with at least one critical region in $\mathcal{Z}_{l-1}$.*

Proof. If O_l is the creator of a new node, then there is a region F in $\mathcal{F}_0(\mathcal{S}'_l)$ that is determined by O_l and conflicts with O_k. In the influence graph $\mathcal{I}a(\Sigma')$, this region is linked to parents which, according to condition 5.3.3, are associated with regions in $\mathcal{F}_0(\mathcal{S}'_{l-1})$ which conflict with O_l. Still according to this condition, at least one of these nodes conflicts with O_k, which proves the existence of a region G in $\mathcal{F}_0(\mathcal{S}'_{l-1})$ that conflicts with both O_l and O_k.

If O_l is the creator of a unhooked node, then there is a region F in $\mathcal{F}_0(\mathcal{S}'_l) \cap \mathcal{F}_0(\mathcal{S}_l)$ whose determinants include O_l. The region F is linked in the influence graph $\mathcal{I}a(\Sigma')$ to parents, at least one of which is either new or killed by O_k (otherwise the region does not need to be rehooked). Update condition 5.3.3 assures us that this parent conflicts with O_l, which proves that there is a region G in $\mathcal{F}_0(\mathcal{S}'_{l-1})$ that conflicts both with O_l and O_k. □

For each $l > k$, we must thus determine whether there is a critical region in $\mathcal{Z}_{l-1}$ that conflicts with O_l. If so, then O_l is reinserted, and we must find all the critical regions that conflict with O_l. Dynamic algorithms are efficient mostly when reinserting O_l involves traversing only a *local* portion of the influence graph that contains all the critical regions which conflict with O_l.

Before starting the rebuilding phase, the critical zone is initialized with those regions killed by O_k. At each subsequent reinsertion, the critical zone changes. To determine the next object that has to be reinserted and the critical regions that conflict with this object, we maintain in a priority queue $\mathcal{Q}$ the set of killers, according to the sequence Σ', of current critical regions. Killers are ordered within $\mathcal{Q}$ by their chronological rank, and each one points to a list of the critical regions that it kills.

The priority queue $\mathcal{Q}$ is first initialized with those regions killed by O_k. For each critical region F in $\mathcal{Z}_{k-1}$, we identify its killer in Σ' as the object, other than O_k, with the lowest rank in the list $L'(F)$.

At each step of the rebuilding process, the object with the smallest chronological rank O_l is extracted from $\mathcal{Q}$, and we also get all the critical regions that conflict with O_l. The object O_l is then reinserted, and the details of this operation depend of course on the problem in hand. The main obstacle is that we might have to change more than the critical zone of the influence graph. Indeed, the new regions created by O_l always have some critical parents, even though they may also have non-critical parents. Moreover, parents of an unhooked region are new, but the unhooked region itself is not. To correctly set up the arcs in the influence graph that are incident to new nodes, the algorithm must find in $\mathcal{I}a(\Sigma)$ the unhooked nodes and the non-critical parents of the new nodes. At this phase, the dictionary set up in the locating phase is used. After reinserting O_l, the priority queue $\mathcal{Q}$ is updated as follows: the regions in $\mathcal{Z}_{l-1}$ that conflict with O_l are not critical any more; however, any new region created by O_l belongs to $\mathcal{Z}_l$. Then for each of these regions F, the killer of F in Σ' is identified as the object in $L'(F)$ with the smallest chronological rank. This object is then searched for in $\mathcal{Q}$ and inserted there if it is not found. Then F is added to the list of regions killed by O_l.

6.3 Randomized analysis of dynamic algorithms

The randomized analysis of the augmented influence graph and the insertion and deletion operations are based on the probabilistic model described in section 6.1. The first three lemmas in this paragraph analyze the expected number of elementary changes to be performed upon a deletion.

Lemma 6.3.1 *Upon deleting an object, the number of nodes that are destroyed, new, or unhooked is, on the average,*

$$O\left(\frac{1}{n}\sum_{l=1}^{n}\frac{f_0(l,S)}{l}\right),$$

where, as usual, $f_0(l,S)$ stands for the number of regions defined and without conflict over a random sample of size l from S.

Proof. We bound the number of destroyed, new, and unhooked nodes separately.
1. The number of destroyed nodes. A node in $\mathcal{I}a(\Sigma)$ corresponding to a region F in $\mathcal{F}_j^i(S)$ is destroyed during a deletion if the object deleted is one of the i objects that determine the region F. Let F be a region in $\mathcal{F}_j^i(S)$. Given that F corresponds to a node in the influence graph built for S, this node is destroyed during a deletion with a conditional probability $i/n \leq b/n$. From theorem 5.3.2, we know that the expected number of nodes in the influence graph is

$$O\left(\sum_{l=1}^{n}\frac{f_0(l,S)}{l}\right),$$

so the number of nodes destroyed when deleting an object is, on the average,

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

2. The number of new nodes. The regions that correspond to the new nodes in the influence graph when O_k is deleted are exactly the regions created by O_l, for some l such that $k < l \leq n$, that belong to $\mathcal{F}_0(\mathcal{S}'_l)$ and conflict with O_k. Let F be a region of $\mathcal{F}^i_j(\mathcal{S})$. This region F belongs to $\mathcal{F}_0(\mathcal{S}'_l)$ with the probability $p^i_j(l-1)$ that was given in subsection 5.2.2. Assuming this, F is created by O_l with conditional probability $i/(l-1)$, and F conflicts with O_k with conditional probability $j/(n-l+1)$. Therefore, for a given k, the number of new nodes in the influence graph upon the deletion O_k is, on the average (using corollary 4.2.7 to the moment theorem),

$$\begin{aligned}\sum_{i=1}^{b}\sum_{j=1}^{n}\sum_{l=k+1}^{n}|\mathcal{F}^i_j(\mathcal{S})|\; p^i_j(l-1)\,\frac{i}{l-1}\,\frac{j}{n-l+1} &= O\left(\sum_{l'=k}^{n-1}\frac{m_1(l',\mathcal{S})}{l'(n-l')}\right)\\ &= O\left(\sum_{l'=k}^{n-1}\frac{f_0(\lfloor l'/2\rfloor,\mathcal{S})}{l'^2}\right).\end{aligned}$$

Averaging over all ranks k, the number of new nodes in the influence graph after a deletion is

$$O\left(\frac{1}{n}\sum_{k=1}^{n}\sum_{l'=k}^{n-1}\frac{f_0(\lfloor l'/2\rfloor,\mathcal{S})}{l'^2}\right)=O\left(\frac{1}{n}\sum_{l=1}^{n-1}\frac{f_0(l,\mathcal{S})}{l}\right).$$

3. The number of unhooked nodes. Unhooked nodes are the non-destroyed children of destroyed nodes. If condition 5.3.3 is satisfied, the number of children of each node in the augmented influence graph is bounded by a constant. It follows that the number of unhooked nodes is at most proportional to the number of destroyed nodes. □

The update condition 5.3.3 assumes that the number of children of a node is bounded by a constant. However, the number of parents of a node is not necessarily bounded by a constant and the following lemma is useful to bound the number of arcs in the influence graph that are removed or added during a deletion.

Lemma 6.3.2 *The number of arcs in the influence graph that are removed or added during a deletion is, on the average,*

$$O\left(\frac{1}{n}\sum_{l=1}^{n-1}\frac{f_0(l,\mathcal{S})}{l}\right).$$

Proof. The simplest proof of this lemma involves the notion of biregion encountered in exercise 5.7. A *biregion* defined over a set of objects $\mathcal{S}$ is a pair of regions defined over $\mathcal{S}$ which can possibly be related as parent and child in the influence graph, for an appropriate permutation of $\mathcal{S}$. A *biregion* is determined by at most $2b$ objects, and the notion of conflict between objects and regions can be extended to biregions: an object conflicts with a biregion if it is not a determinant of any of the two regions but conflicts with at least one of the two regions. Biregions obey statistical laws similar to those obeyed by regions. In particular, a biregion determined by i objects of $\mathcal{S}$ which conflicts with j objects of $\mathcal{S}$ is a biregion defined and without conflict over a random l-sample of $\mathcal{S}$, with the probability $p_j^i(l)$ given by lemma 4.2.1. A biregion defined and without conflict over a subset $\mathcal{S}_l$ of $\mathcal{S}$ corresponds to an arc in the influence graph whenever the objects that determine the parent region are inserted before those that determine the child region and at the same time conflict with the parent region. This only happens with a probability $\alpha \in [0,1]$ (which depends on the number of objects determining the parent and the child, and the number of objects that at the same time determine the child and conflict with the parent).

A biregion determined by i objects in $\mathcal{S}$ and conflicting with j objects in $\mathcal{S}$ corresponds to an arc in the influence graph $\mathcal{I}a(\Sigma)$ that was created by O_l, with a probability smaller than $\frac{i}{l}\, p_j^i(l)$ (see also exercise 5.7); this arc, created by O_l, conflicts with O_k with a probability smaller than

$$\frac{1}{l}\,\frac{j}{n-l}\,p_j^i(l).$$

A computation similar to that in the proof of lemma 6.3.1 shows that the expected number of arcs in the influence graph that are created or removed during a deletion (which are those adjacent in the influence graph to new nodes or to destroyed nodes) is

$$O\left(\frac{1}{n}\sum_{l=1}^{n-1}\frac{\mathit{ff}_0(l,\mathcal{S})}{l}\right),$$

where $\mathit{ff}_0(l,\mathcal{S})$ is the expected number of biregions defined and without conflict over a random l-sample of $\mathcal{S}$. It remains to show that $\mathit{ff}_0(l,\mathcal{S})$ is proportional to $f_0(l,\mathcal{S})$. Let $\mathcal{S}_l$ be a subset of size l of $\mathcal{S}$. The parent region in a biregion that is defined and without conflict over $\mathcal{S}_l$ is a region defined over $\mathcal{S}_l$ that conflicts with exactly one object in $\mathcal{S}_l$, and is therefore a region in $\mathcal{F}_1(\mathcal{S}_l)$. Conversely, if the update condition 5.3.3 is true, every region in $\mathcal{F}_1(\mathcal{S}_l)$ is the parent in a bounded number of biregions defined and without conflict over $\mathcal{S}_l$. It follows that $\mathit{ff}_0(l,\mathcal{S})$ is within a constant factor of the expectation $f_1(l,\mathcal{S})$ of the number of regions defined and conflicting with one element over a random l-sample. From corollary 4.2.4 to the sampling theorem, this expected number is $O(f_0(l,\mathcal{S}))$. □

Lemma 6.3.3 *The total size of all the conflict lists attached to the nodes that are new or destroyed when an object is deleted is, on the average,*

$$O\left(\sum_{l=1}^{n} \frac{f_0(l,S)}{l^2}\right).$$

Proof.
1. Conflict lists of destroyed nodes. A region F of $\mathcal{F}_j^i(S)$ corresponds to a node of the influence graph $\mathcal{I}a(\Sigma)$ with probability

$$\sum_{l=1}^{n} p_j^i(l)\frac{i}{l}$$

as implied by lemma 5.2.2. The conflict list attached to this node has length j and this node is destroyed during the deletion of an object with probability i/n. The total size of the conflict lists attached to destroyed nodes is thus

$$\begin{aligned}\sum_{l=1}^{n}\sum_{i=1}^{b}\sum_{j=1}^{n} |\mathcal{F}_j^i(S)|\ p_j^i(l)\,\frac{i}{l}\,\frac{i}{n}\,j &= O\left(\frac{1}{n}\sum_{l=1}^{n}\frac{m_1(l,S)}{l}\right)\\ &= O\left(\frac{1}{n}\sum_{l=1}^{n}(n-l)\,\frac{f_0(\lfloor\frac{l}{2}\rfloor,S)}{l^2}\right)\\ &= O\left(\sum_{l=1}^{n}\frac{f_0(l,S)}{l^2}\right),\end{aligned}$$

as follows from corollary 4.2.7 to the moment theorem.
2. Conflict lists attached to new nodes. A region F of $\mathcal{F}_j^i(S)$ is a new region created by O_l when O_k is deleted, if it is a region of $\mathcal{F}_0(S_l')$ determined by O_l that conflicts with O_k. The conflict list attached to the new node corresponding to F has $j-1$ elements. The total size of the conflict lists attached to new nodes when deleting O_k is thus, on the average,

$$\sum_{i=1}^{b}\sum_{j=1}^{n}\sum_{l=k+1}^{n} |\mathcal{F}_j^i(S)|\ p_j^i(l-1)\,\frac{i}{(l-1)}\,\frac{j}{(n-l+1)}\,(j-1).$$

Applying corollary 4.2.7, this size is

$$O\left(\sum_{l=k}^{n-1}\frac{m_2(l,S)}{l(n-l)}\right) = O\left(\sum_{l=k}^{n-1}(n-l)\,\frac{f_0(\lfloor\frac{l}{2}\rfloor,S)}{l^3}\right).$$

Averaging over all ranks of k, the above sum becomes

$$O\left(\frac{1}{n}\sum_{k=1}^{n}\sum_{l=k}^{n}(n-l)\,\frac{f_0(\lfloor\frac{l}{2}\rfloor,S)}{l^3}\right) = O\left(\sum_{l=1}^{n}\frac{f_0(l,S)}{l^2}\right). \qquad \square$$

Lastly, setting up the priority queue $\mathcal{Q}$ of killers of critical regions involves the regions of the influence graph $\mathcal{I}a(\Sigma)$ that are killed by O_k. The conflict lists of these regions are traversed in order to set up the conflict lists of the new children of these nodes. The following lemma is therefore needed in order to fully analyze dynamic algorithms.

Lemma 6.3.4 *The number of nodes in the influence graph $\mathcal{I}a(\Sigma)$ that are killed by a random object in $\mathcal{S}$ is, on the average,*

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

The total size of the conflict lists attached to the nodes killed by a random object is, on the average,

$$O\left(\sum_{l=1}^{n}\frac{f_0(l,\mathcal{S})}{l^2}\right).$$

Proof. A node in $\mathcal{I}a(\Sigma)$ that is killed by an object O_k of rank k corresponds to a region in $\mathcal{F}_0(\mathcal{S}_{k-1})$ that conflicts with O_k. A region F in $\mathcal{F}_j^i(\mathcal{S})$ is a region in $\mathcal{F}_0(\mathcal{S}_{k-1})$ that conflicts with O_k with probability

$$p_j^i(k-1)\frac{j}{n-k+1}.$$

Hence, the average number of nodes in $\mathcal{I}a(\Sigma)$ killed by a random object of $\mathcal{S}$ is

$$\begin{aligned}
\frac{1}{n}\sum_{k=2}^{n}\sum_{i=1}^{b}\sum_{j=1}^{n}|\mathcal{F}_j^i(\mathcal{S})|\ p_j^i(k-1)\frac{j}{n-k+1} &= O\left(\frac{1}{n}\sum_{k=2}^{n}\frac{m_1(k-1,\mathcal{S})}{n-k+1}\right)\\
&= O\left(\frac{1}{n}\sum_{k=1}^{n}\frac{f_0(k,\mathcal{S})}{k}\right),
\end{aligned}$$

as can be deduced from corollary 4.2.7 to the moment theorem.

The total size of the conflict lists attached to nodes killed by a random object is, on the average,

$$\begin{aligned}
\frac{1}{n}\sum_{k=2}^{n}\sum_{i=1}^{b}\sum_{j=1}^{n}|\mathcal{F}_j^i(\mathcal{S})|\,p_j^i(k-1)\frac{j}{n-k+1}(j-1) &= O\left(\frac{1}{n}\sum_{k=2}^{n}\frac{m_2(k-1,\mathcal{S})}{n-k+1}\right),\\
&= O\left(\sum_{k=1}^{n}\frac{f_0(\lfloor k/2\rfloor,\mathcal{S})}{k^2}\right)\\
&= O\left(\sum_{l=1}^{n}\frac{f_0(l,\mathcal{S})}{l^2}\right).
\end{aligned}$$

□

The detailed operations required to insert or delete an object in an augmented influence graph depend upon the problem under consideration. In particular, deletions demand a number of operations (insertions, deletions, or queries) on a dictionary of nodes, or even several dictionaries. To be able to present a general analysis, we introduce here an update condition for dynamic algorithms that use an augmented influence graph. This condition is similar to those introduced in the previous chapter to analyze incremental algorithms, namely condition 5.2.1 for algorithms using a conflict graph and condition 5.3.3 for algorithms using an influence graph. The condition we introduce here is a reasonable one, which will be fulfilled by all the dynamic algorithms described in this book.

Update condition 6.3.5 (for augmented influence graphs) *A dynamic algorithm that uses an augmented influence graph satisfies the* update condition *when:*

1. *The augmented influence graph satisfies the update condition 5.2.1 for algorithms using an influence graph.*

2. *During a deletion:*

a. *The number of operations on a dictionary of nodes (insertions, deletions, or queries) is at most proportional to the total number of nodes killed (by the deleted object), destroyed, new, or unhooked.*

b. *The conflict lists of the new nodes are initialized using a time proportional to the total size of the conflict lists of the nodes killed (by the deleted object), destroyed, and new.*

c. *All the operations performed to update the augmented influence graph that do not pertain to dictionaries, conflict lists, or the priority queue $\mathcal{Q}$, are elementary and their number is proportional to the total number of destroyed or new nodes, and of arcs incident to these nodes.*

The complexity of a deletion depends partly on the data structures used to implement the dictionaries and the priority queue $\mathcal{Q}$ of the killers of critical regions. These data structures store a set of elements that belong to a finite, totally ordered universe, whose cardinality is bounded by a polynomial in the number n of current objects. Therefore, we can use the data structures described in section 2.3, or more simply we may use a standard balanced binary tree. In order to take all these cases into account, the analysis given below introduces two parameters. The first parameter, t, is the complexity of a query, insertion, or deletion performed on a dictionary of size $O(n^c)$, where c is some constant. The second parameter, t', is the complexity of a query, insertion, or deletion performed on a priority queue of size $O(n)$. Parameter t is $O(\log n)$ if a balanced tree is used, and $O(1)$ if the perfect dynamic hashing method of section 2.3 is used.

Parameter t' is $O(\log n)$ if we use a balanced binary tree, but it is $O(\log\log n)$ if we use a stratified tree along with perfect dynamic hashing (see section 2.3). Moreover, as we will see further on, if $f_0(l,S)$ grows at least quadratically, then implementing $\mathcal{Q}$ with a simple array of size n will suffice, and t' can be ignored in the analysis.

Theorem 6.3.6 *This theorem details the performances of a dynamic algorithm that uses an augmented influence graph and satisfies the update condition 6.3.5. Let S be the current set of objects, and n the size of S.*

1. *The data structure requires an average storage of*

$$O\left(n\sum_{l=1}^{n}\frac{f_0(l,S)}{l^2}\right).$$

2. *Adding an object can be performed in expected time*

$$O\left(\sum_{l=1}^{n}\frac{f_0(l,S)}{l^2}\right).$$

3. *Deleting an object can be performed in expected time*

$$O\left(\min\left(n,\ \frac{t'}{n}\sum_{l=1}^{n}\frac{f_0(l,S)}{l}\right)+\frac{t}{n}\sum_{l=1}^{n}\frac{f_0(l,S)}{l}+\sum_{l=1}^{n}\frac{f_0(l,S)}{l^2}\right).$$

As always, $f_0(l,S)$ is the number of regions defined and without conflict over a random l-sample of S, t is the complexity of any operation on a dictionary, and t' is the complexity of an operation on the priority queue $\mathcal{Q}$.

Proof.

1. The storage needed by the augmented influence graph $\mathcal{I}a(\Sigma)$ is proportional to the total size of the conflict lists attached to the nodes of $\mathcal{I}a(\Sigma)$. Each element in one of these conflict lists corresponds to a conflict detected by an on-line algorithm processing the objects in S in the chronological order of the sequence Σ. The expected number of conflicts, for a random permutation of Σ, is thus given by theorem 5.2.3 which analyzes the complexity of incremental algorithms that use a conflict graph.

2. The randomized analysis of an insertion into the augmented conflict graph is identical to that of the incremental step in an on-line algorithm that uses an influence graph. Indeed, the two algorithms only differ in that one updates conflict lists. Each conflict between the inserted object and a node in the current graph is detected by both algorithms. In the dynamic algorithm, detecting such a conflict implies adding the inserted object into the conflict list of the conflicting node, which can be carried out in constant time. The expected complexity of an insertion is thus given by theorem 5.3.4.

3. Let us now analyze the average complexity of deleting an object, say O_k. When locating O_k in the augmented influence graph, the nodes that are visited are exactly the destroyed nodes, and their children, and the nodes that conflict with O_k. Since every node has a bounded number of children, the cost of the traversal is proportional to the number of nodes destroyed or conflicting with O_k. The number of nodes in the influence graph that conflict with O_k is, on the average over all possible sequences Σ,

$$O\left(\sum_{l=1}^{k-1} \frac{f_0(\lfloor \frac{l}{2} \rfloor, S)}{l^2}\right),$$

which we know from the proof of theorem 5.3.4. Averaging over the rank of the deleted object, we get

$$O\left(\sum_{l=1}^{n} \frac{f_0(l, S)}{l^2}\right).$$

From lemma 6.3.1, the latter expression is also a bound on the expected number of nodes destroyed and thus on the global cost of traversing the influence graph.

If the update condition 6.3.5 is realized, lemmas 6.3.3 and 6.3.4 show that the conflict lists of the new regions can be set up in time

$$O\left(\sum_{l=1}^{n} \frac{f_0(l, S)}{l^2}\right).$$

Lemma 6.3.1 and the update condition 6.3.5 (2a) assert that the term

$$O\left(\frac{t}{n} \sum_{l=1}^{n} \frac{f_0(l, S)}{l}\right)$$

accounts for the average complexity of all the operations performed on the dictionaries of nodes.

Since t is necessarily $\Omega(1)$, lemmas 6.3.1 and 6.3.2, together with condition 6.3.5 (2c), assert that the former term also accounts for all the operations that update the augmented influence graph, not counting those on the conflict lists or the priority queue.

It remains to analyze the management of the priority queue $\mathcal{Q}$ of critical region killers. The number of insertions and queries in the priority queue is proportional to the total number of critical regions encountered during the rebuilding phase.

These regions are either killed by the deleted object, or they are new regions. Their average number is thus

$$O\left(\frac{1}{n}\sum_{l=1}^{n}\frac{f_0(l,S)}{l}\right),$$

as asserted by lemmas 6.3.1 and 6.3.4. The average number of minimum queries to be performed on the queue Q is

$$O\left(\min\left(n,\frac{1}{n}\sum_{l=1}^{n}\frac{f_0(l,S)}{l}\right)\right),$$

since the number of objects to be reinserted is bounded from above by n on the one hand, and by the number of unhooked or new nodes (estimated by theorem 6.3.1) on the other hand. □

Consequently,

- If $f_0(l,S)$ grows slower than quadratically (with respect to l), we use a hierarchical structure for the priority queue, characterized by a parameter t' which bounds the complexity of any operation on this structure (insertion, membership or minimum query). Managing the queue has therefore the associated expected cost

$$O\left(\frac{t'}{n}\sum_{l=1}^{n}\frac{f_0(l,S)}{l}\right).$$

- If on the contrary, $f_0(l,S)$ grows at least quadratically, we use for Q a simple array of size n. This allows insertions and deletions to be performed into the queue in constant time. The cost of finding the minima during the whole rebuilding phase is then $O(n)$, and managing the queue has in this case the associated expected cost

$$O\left(n+\frac{1}{n}\sum_{l=1}^{n}\frac{f_0(l,S)}{l}\right).$$

- When $f_0(l,S)$ is $O(l)$, the expected number of destroyed or new nodes visited during a deletion is $O(1)$. Updating the conflict lists costs $O(\log n)$ anyway. Both the priority queue and the dictionaries can be implemented simply by balanced binary trees ($t = t' = \log n$) to yield a randomized complexity of $O(\log n)$ for a deletion.

6.4 Dynamic decomposition of a set of line segments

The vertical decomposition of a set of line segments in the plane is a structure defined in section 3.3. It can be built using a conflict graph by a static incremental algorithm, as explained in subsection 5.2.2, or by a semi-dynamic incremental algorithm using an influence graph, as detailed in subsection 5.3.2. By combining both structures using the general method explained in section 6.2, we can dynamically maintain the vertical decomposition of the segments under insertion or deletion of a segment. The algorithm described here is a generalization of the decomposition algorithms given in subsections 5.2.2 and 5.3.2. It is advisable to thoroughly understand both these algorithms before reading further.

Let us first recall that, for this problem, the objects are segments and the regions defined over a set $\mathcal{S}$ of segments are the trapezoids appearing in decompositions of subsets of $\mathcal{S}$. A trapezoid is determined by at most four segments. A segment conflicts with a trapezoid if it intersects the interior of the trapezoid.

Let $\mathcal{S}$ be the current set of segments and let $\Sigma = \{O_1, O_2, \ldots, O_n\}$ be the chronological sequence of segments in $\mathcal{S}$. We may also denote by $\mathcal{S}_l$ the subset of $\mathcal{S}$ consisting of the first l segments in Σ. The dynamic algorithm maintains an augmented influence graph $\mathcal{I}a(\Sigma)$, whose nodes correspond to the trapezoids that are defined and without conflict over the current subsets $\mathcal{S}_l$, $l = 1, \ldots, n$. The nodes and arcs in this graph are identical to those built in the influence graph by the on-line algorithm described in subsection 5.3.2. In addition, the augmented influence graph includes a conflict graph between the trapezoids corresponding to the nodes of the influence graph, and the segments in $\mathcal{S}$. The conflict graph is implemented using the interconnected list system, as explained in section 6.2. The structure does not encode all the adjacency relationships between the trapezoids but only those between the trapezoids in the current decomposition (corresponding to leaves of the influence graph).

Adding the n-th segment does not create problems and can be carried out exactly as explained in subsection 5.3.2. The only difference is that the conflict lists are updated when the conflicts are detected during the locating phase when inserting O_n.

Let us now explain how to delete segment O_k, of rank k in the chronological sequence Σ. As before, we denote by $\mathcal{S}'_l$ the subset $\mathcal{S}_l \setminus \{O_k\}$ of $\mathcal{S}$ and by Σ' the chronological sequence $\{O_1, \ldots, O_{k-1}, O_{k+1}, \ldots O_n\}$.

The algorithm proceeds along the usual lines and performs the two phases: locating and rebuilding. The locating phase detects the nodes of $\mathcal{I}a(\Sigma)$ that conflict with O_k, and the destroyed and unhooked nodes. In the rebuilding phase, the algorithm processes the segments of rank $l > k$ that are the creators of new or unhooked nodes. For this, the algorithm manages a priority queue which contains, at each step of the rebuilding process, the killers in Σ' of the critical regions. For each such object O_l, a killer of a critical region, the algorithm builds the new nodes created by O_l and rehooks the unhooked nodes created by O_l. Figure 6.1 shows how the influence graph built for the four segments $\{O_1, O_2, O_3, O_4\}$ is modified when deleting O_3. The reader may observe again how the graph was created incrementally, in figures 5.6, 5.7 and 5.8. In this example, nodes B and H are killed by O_3, nodes J,K,L,M,N,O,P,Q,S,U,V are destroyed, nodes R,T,W are unhooked (they are created by O_4), and B' is a new node (its creator is O_4).

The subsequent paragraphs describe in great detail the specific operations needed.

Locating. This phase is trivial: all the nodes that conflict with the object O_k to be deleted, or that are determined by a subset containing O_k, are visited together with their children. The algorithm builds a dictionary $\mathcal{D}$ of unhooked or destroyed nodes, which will be used during the rebuilding phase.

Rebuilding. The priority queue $\mathcal{Q}$, which contains the killers of critical regions, is initialized with the nodes in $\mathcal{I}a(\Sigma)$ that are killed by O_k.

At each step in the rebuilding process, the algorithm extracts from the priority queue $\mathcal{Q}$ the object O_l of smallest chronological rank. It also retrieves the list of the critical regions that conflict with O_l.

Each of these regions is split into at most four *subregions* by O_l, and the walls stemming from its endpoints and its intersection points. These subregions are not necessarily trapezoids in the decomposition $\mathcal{D}ec(\mathcal{S}'_l)$. Indeed, the walls cut by O_l have to be shortened, keeping only the part that is still connected to the endpoint or intersection point from which it stems. The other part of the wall must be removed and the adjacent subregions separated by this part must be joined. The join can be one of two kinds: *internal* when the portion of wall to be removed separates two critical regions, and *external* when it separates a critical region from a non-critical region (see figure 6.2).

To detect which regions to join,[2] the algorithm visits all the critical regions that conflict with O_l, and stores in a secondary dictionary $\mathcal{D}'_l$ the walls incident to these regions that are intersected by O_l. Any wall in this dictionary that separates two critical regions gives rise to an internal join, and any wall incident to only one critical region gives rise to an external join.

In a first phase, the algorithm creates a temporary node for each subregion resulting from the splitting of a critical region by O_l or the walls stemming from O_l. The node that corresponds to a subregion F_i of the region F is hooked in the graph as a child of F. Its conflict list is obtained by selecting, from the conflict

[2]The algorithm cannot traverse the sequence, ordered by O_l, of critical regions for two reasons: (1) it does not maintain the vertical adjacencies between the internal nodes of the influence graph, and the adjacencies between either the trapezoids of the decomposition $\mathcal{D}ec(\mathcal{S}'_{l-1})$ or the critical regions of $\mathcal{Z}_{l-1}$ are not available, and (2) the intersection of O_l with the union of the regions in $\mathcal{Z}_{l-1}$ may not be connected (see for instance figure 6.4).

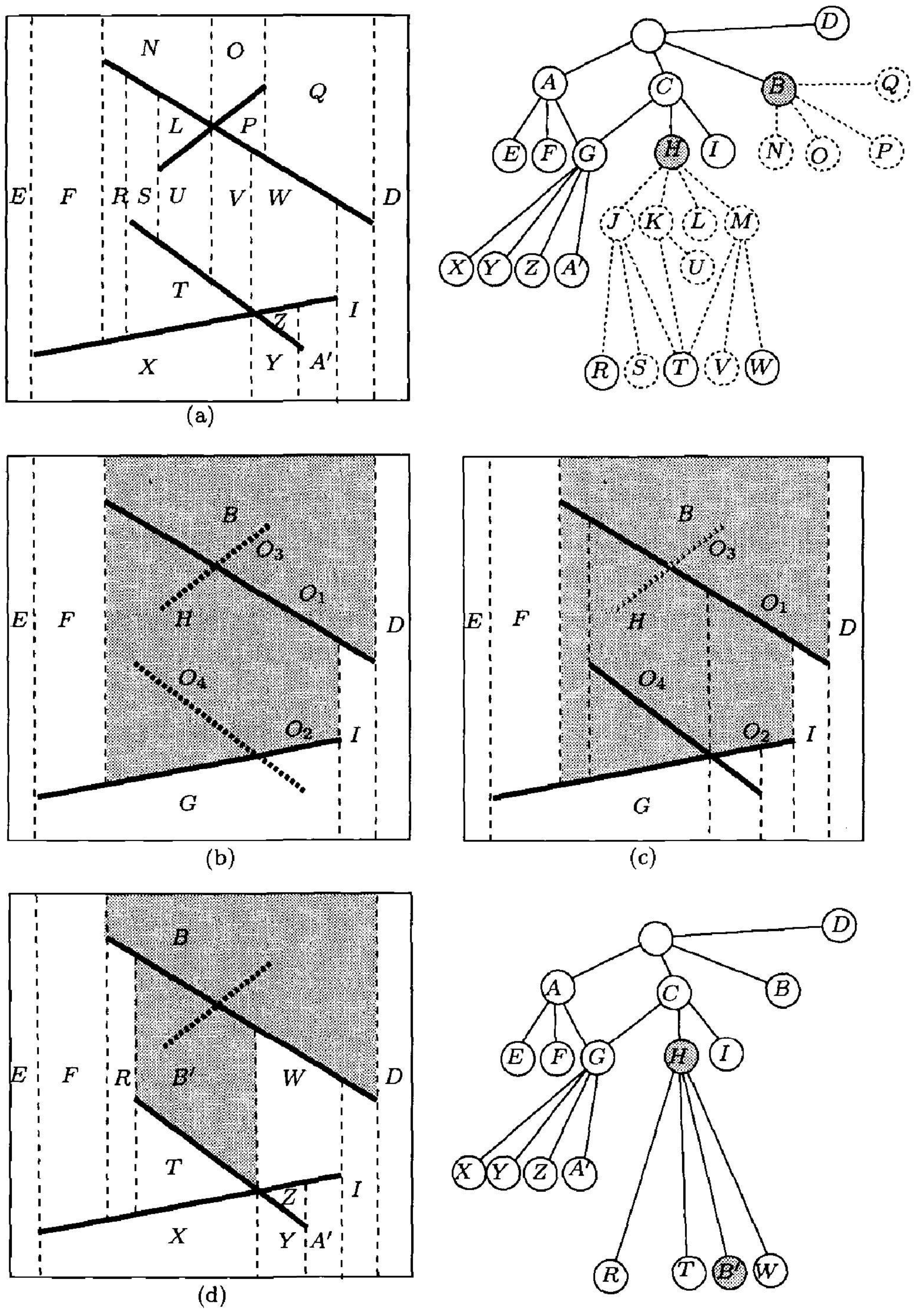


Figure 6.1. Deleting a segment.

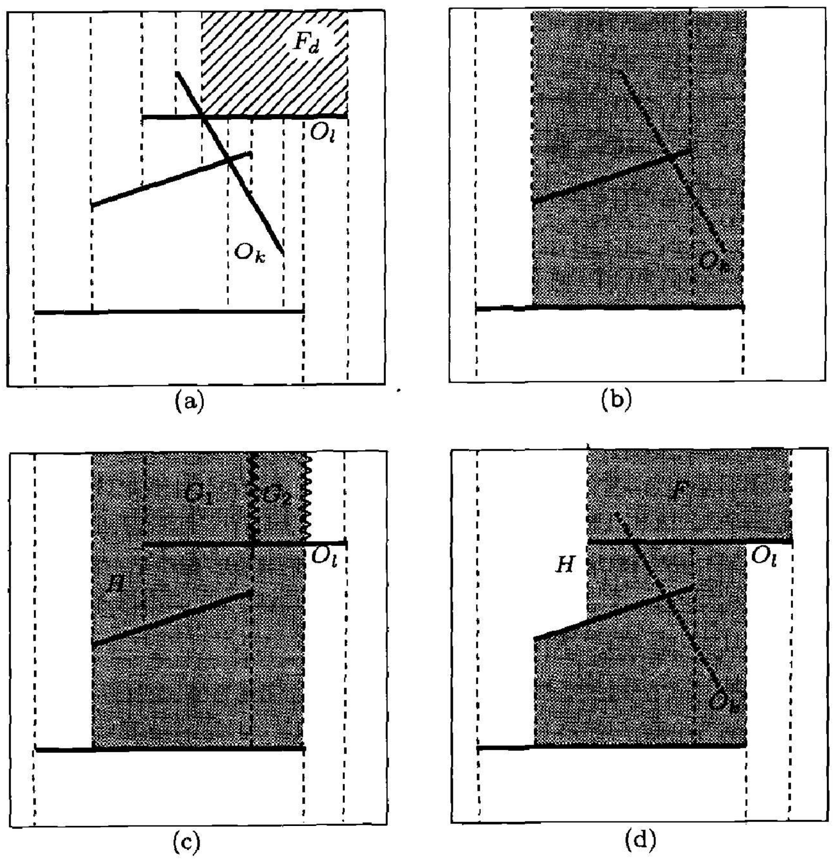


Figure 6.2. Internal and external joins:
(a) The decomposition $\mathcal{D}ec(\mathcal{S}_l)$ before deleting O_k.
(b) The decomposition $\mathcal{D}ec(\mathcal{S}'_{l-1})$, with the critical zone $\mathcal{Z}_{l-1}$ shaded.
(c) Reinserting O_l. Splitting the critical regions and joining: internal join $G = G_1 \cup G_2$, external join $F = G \cup F_d$. Region H is unhooked.
(d) The decomposition $\mathcal{D}ec(\mathcal{S}'_l)$ and the critical zone $\mathcal{Z}_l$.

list of F, the segments intersecting F_i. Then the algorithm processes the internal and the external joins, as explained below.

1. Internal joins. Every maximal set $\{G_1, \ldots, G_h\}$ of subregions, pairwise adjacent and separated by walls to be removed, must be joined together into a single region G. The algorithm creates a temporary node for G. The nodes corresponding to $G_1, G_2, \ldots, G_h$ are removed from the graph and the node corresponding to G inherits all the parents of these nodes. The conflict list of G is obtained by merging the conflict lists of $G_1, G_2, \ldots, G_h$, removing redundancies. For this, we use a procedure similar to that of subsection 5.2.2, but which need not know the order along O_l of the subregions to be joined. By scanning the conflict lists of these subregions successively, the algorithm can build for each segment O in $\mathcal{S}$ a list $L_G(O)$ of the subregions that conflict with O. A bidirectional pointer interconnects the entry in the list $L'(G_i)$ that corresponds to an object O with the entry in $L_G(O)$ corresponding to the subregion G_i. The conflict list of G can be retrieved by scanning again all the conflict lists $L'(G_i)$ of the subregions $G_1, \ldots, G_h$. This time, each segment O encountered in one of these lists is added to the conflict list of G and removed from the other conflict lists, using the information stored in $L_G(O)$.

Let us call *auxiliary regions* the regions obtained after all the internal joins. These regions are either subregions that needed no internal join, or regions obtained from an internal join of the subregions. An auxiliary region that does not need to undergo any external join is a region of the decomposition $\mathcal{Dec}(\mathcal{S}'_l)$. Let H be such a region. This region is new if it conflicts with O_k, unhooked otherwise. In the former case, the temporary node for H becomes permanent and the killer of H is inserted into the priority queue $\mathcal{Q}$. In the latter case, a node for H already exists in the influence graph $\mathcal{I}a(\Sigma)$. A simple query in the dictionary of unhooked nodes retrieves this node, which can then be rehooked to the parents of the auxiliary node created for H.

2. External joins. In a second phase, the algorithm performs the external joins. An auxiliary region undergoes a left join if its left wall must be removed, and a right join if its right wall must be removed, and a double left–right join if both its vertical walls must be removed. Let G be an auxiliary region undergoing a right join. For instance, this is the case for region $G = G_1 \cup G_2$ in figure 6.2. The right wall of G is on the boundary of the critical zone, since this is an external join. This wall is therefore not cut by the deleted segment O_k. When the decomposition of $\mathcal{S}$ is built incrementally according to the order in the sequence Σ, this wall appears at a certain step and is removed when O_l is inserted. Thus, among all the regions in $\mathcal{I}a(\Sigma)$, there is one region F_d created by O_l that contains the right wall of G.[3] The region F_d is necessarily destroyed or unhooked: indeed, F_d is a trapezoid in the decomposition $\mathcal{Dec}(\mathcal{S}_l)$, and has a non-empty intersection with one or more critical regions in $\mathcal{Z}_{l-1}$. As every critical region in $\mathcal{Z}_{l-1}$ is contained in the union of the trapezoids of $\mathcal{Dec}(\mathcal{S}_{l-1})$ of which O_k is a determinant, the region F_d must intersect those trapezoids. Thus at least one of the parents of F_d in the graph $\mathcal{I}a(\Sigma)$ is a destroyed node. Similarly, if the left wall of G must be removed, there is in $\mathcal{I}a(\Sigma)$ one destroyed or unhooked region F_g created by O_l that contains the left wall of G. If the join is double left–right, F_d and F_g may be distinct or identical (see figure 6.3).

Several auxiliary regions may be joined into the same permanent region (see figure 6.4). Let $\{G_1, G_2, \ldots, G_j\}$ be the sequence ordered along O_l of the auxiliary

[3]It would have been more desirable to subscript F by l and r for *left* and *right*, but this would have conflicted with the index l for O_l and created confusion. We have kept a French touch with the indices g and r for the French *gauche* and *droit*, meaning respectively left and right. (Translator's note)

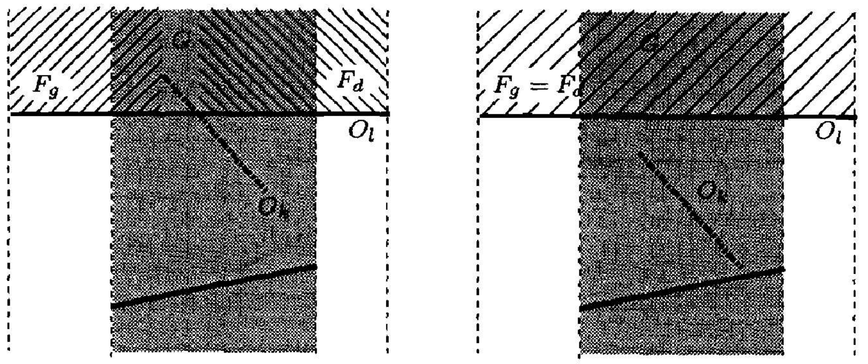


Figure 6.3. External joins: double left–right joins.

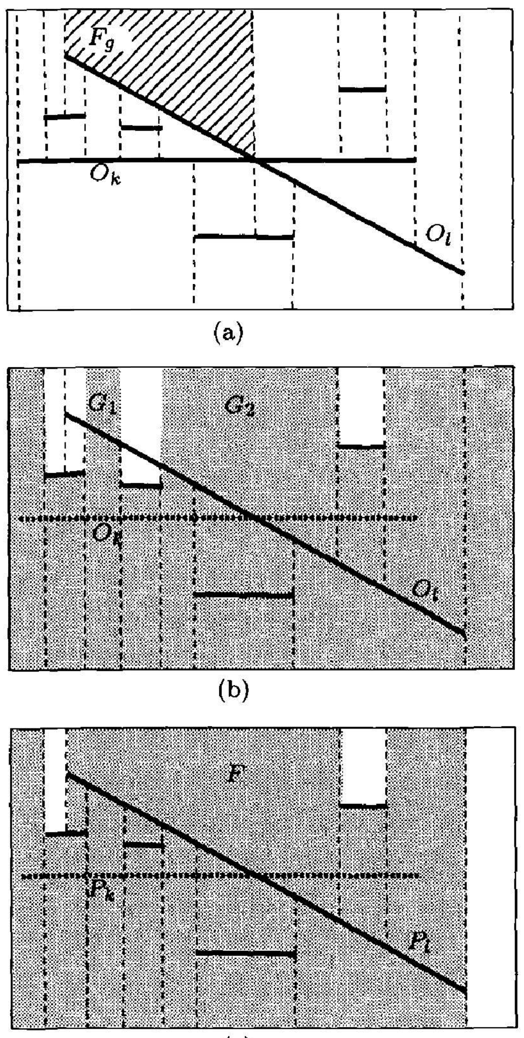
regions[4] whose left wall is contained in the same region F_g of $\mathcal{I}a(\Sigma)$ created by O_l. If $j > 1$, then the right walls of these auxiliary regions $\{G_1, G_2, \ldots, G_{j-1}\}$ are also contained in F_g and must be removed as well. If the right wall of G_j is a permanent wall (that does not have to be removed), the join results in a single trapezoid of the decomposition $\mathcal{D}ec(\mathcal{S}'_l)$ that is the same as $F_g \cup G_1 \cup \ldots \cup G_j = F_g \cup G_j$ (see figure 6.4). If the right wall of G_j also has to be removed, then we introduce the ordered sequence of auxiliary regions $\{G_j, G_{j+1}, \ldots, G_h\}$: this sequence consists of regions whose right wall must be removed and which lie in the same region F_d of $\mathcal{I}a(\Sigma)$ created by O_l. The left walls of the regions in $\{G_j, G_{j+1}, \ldots, G_h\}$ then also belong to F_d and have to be removed. The join operates on the auxiliary regions $\{G_1, \ldots, G_j, \ldots, G_h\}$ and results in a unique trapezoid in $\mathcal{D}ec(\mathcal{S}'_l)$ that is the same as $F_g \cup G_1 \cup \ldots \cup G_h \cup F_d = F_g \cup G_j \cup F_d$.

We present below the operations to be performed in the latter case of a double left–right join. The former cases can be handled in a similar manner. Suppose for now that the auxiliary regions $\{G_1, \ldots, G_j, \ldots, G_h\}$ as well as the regions F_g and F_d in the decomposition $\mathcal{I}a(\Sigma)$ that participate in the join are known to the algorithm.

If the trapezoid resulting from the join $F = F_g \cup G_j \cup F_d$ does not conflict with O_k (see figure 6.3, right), it is a trapezoid in the decomposition $\mathcal{D}ec(\mathcal{S}_l)$. Necessarily, the regions F_g, F_d, and F are the same, and the corresponding node in $\mathcal{I}a(\Sigma)$ is unhooked. It then suffices to search for this node in the dictionary of unhooked nodes, to remove the auxiliary nodes created for $G_1, G_2, \ldots, G_h$, and to rehook the node corresponding to F, with the critical nodes in the parents of $G_1, G_2, \ldots, G_h$ as the parents of F.

If the resulting trapezoid $F = F_g \cup G_j \cup F_d$ conflicts with O_k (see figure 6.3, left), then it is a new region of $\mathcal{I}a(\Sigma')$, and the regions F_g and F_d in $\mathcal{I}a(\Sigma)$

[4]We must emphasize that even though the given description of the region resulting from an external join refers to the order of the joined auxiliary regions along O_l, the algorithm does not know this order, nor does it need it.



(c)

Figure 6.4. External joins.
(a) The decomposition before deleting O_k.
(b) Reinserting O_l. The auxiliary regions.
(c) Joining auxiliary regions G_1 and G_2 into $F = F_g \cup G_1 \cup G_2$.

are destroyed. The auxiliary nodes created for $G_1, G_2, \ldots, G_h$ are removed, and replaced by a single node corresponding to F. This node is then rehooked to the parents of F_g and F_d that are not destroyed, and to all the critical parents of G_1, G_2, ..., G_h. The conflict list of F is derived from those of F_g,G_1, G_2, ..., G_h and F_d, as is the case for internal joins. Lastly, the killer of F is inserted into the priority queue $\mathcal{Q}$.

We now have to explain how to retrieve the unhooked or destroyed nodes corresponding to the regions F_g and F_d involved in the join. Let G be an auxiliary region whose left wall must be removed. The corresponding region F_g is either destroyed or unhooked, created by O_l, and the segments that support its floor and ceiling[5] respectively support the floor and ceiling of G. Any region in the decomposition of a given set of segments is identified uniquely by its floor, its ceiling, and one of its walls. Below, we show that either we can find one of the walls of F_g, or we can identify a destroyed region F'_g which is the unique sibling of F_g in $\mathcal{I}a(\Sigma)$.

- If G conflicts with O_k (as in figure 6.5a), the right wall of F_g is determined by O_k, and can be computed (by looking only at G and O_k).
- If G does not conflict with O_k, but its right wall is permanent (see figure 6.5b), then this right wall is also that of F_g.
- Lastly, if G does not conflict with O_k, and if both its walls must be removed (see figure 6.5c), then segment O_l intersects both walls of a critical region that was subsequently split into G and G'. The other subregion G' also conflicts with O_k but does not undergo any join. In $\mathcal{I}a(\Sigma)$, exactly one node F'_g has O_l for creator, is destroyed, and shares the same floor, same ceiling, and same left wall as G'. This node F'_g has only one parent, and this parent has two children, one of which is F'_g and the other the node F_g that we are looking for: indeed, the parent of F'_g corresponds to a trapezoid in the decomposition $\mathcal{D}ec(\mathcal{S}_{l-1})$ whose two walls are intersected by O_l.

In either case, the region F_g, or its sibling F'_g, is known through its creator, its floor, its ceiling, and one of its left or right walls. This information is enough to characterize it. Naturally, the same observation goes for F_d or its sibling F'_d. We can then use the dictionary $\mathcal{D}$ storing all the destroyed or unhooked nodes. This dictionary comes in two parts, $\mathcal{D}_g$ and $\mathcal{D}_d$. In the dictionary $\mathcal{D}_g$, the nodes are labeled with:

- the creator segment,
- the segment supporting the floor of the trapezoid,
- the segment supporting the ceiling of the trapezoid,
- the pair of segments whose intersection determines the right wall of the trapezoid, or the same segment repeated twice if the wall stems from the segment's endpoint.

Similarly, in its counterpart $\mathcal{D}_d$, nodes are labeled the same way, except that in the last component the right wall is replaced by the left wall. Any destroyed or unhooked node is inserted into both dictionaries $\mathcal{D}_g$ and $\mathcal{D}_d$.

[5]We recall that the *floor* and *ceiling* of a trapezoid are its two non-vertical sides.

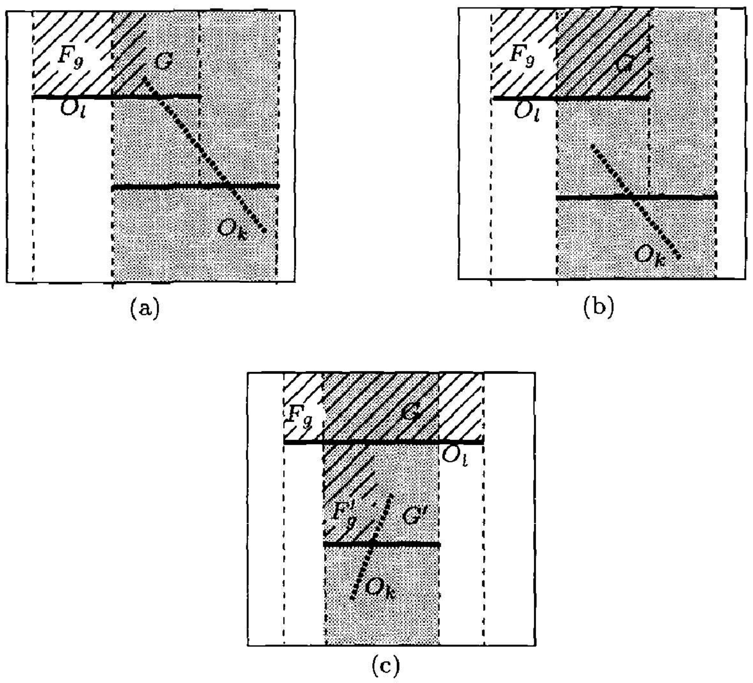


Figure 6.5. External joins:
(a) G conflicts with O_k.
(b) the right wall of G is permanent.
(c) Double left–right join.

Analysis of the algorithm

To analyze this algorithm, we first check that it does satisfy the update conditions 6.3.5. The first condition is satisfied, since the augmented influence graph has the same nodes and arcs as the influence graph built by the on-line algorithm of subsection 5.3.2, which itself satisfies the update condition 5.3.3. Therefore, we need only look at deletions.

1. Number of operations on the dictionaries. Each deletion involves a two-sided dictionary $\mathcal{D}$ of destroyed or unhooked nodes, as well as a dictionary $\mathcal{D}'_l$, for each reinserted segment O_l, of walls in the critical zone intersected by O_l. A destroyed or unhooked node is inserted and queried at most once in $\mathcal{D}$. A critical region in $\mathcal{Z}_{l-1}$ has at most two walls which must be inserted into $\mathcal{D}'_l$, and this region will not be a critical region any more after the reinsertion of O_l. The number of operations on all dictionaries $\mathcal{D}'_l$ is thus at most proportional to the total number of critical regions encountered in the rebuilding phase. Any critical region is either killed or new. The total number of operations is thus at most proportional to the number of nodes that are killed, destroyed, unhooked, or new.

2. Conflict lists of new nodes. The conflict list of a new node is obtained by scanning the conflict lists of the auxiliary or destroyed regions of which it is the union. Similarly, the conflict list of an auxiliary region is obtained by traversing the conflict lists of the subregions of which it is the union, and the conflict lists of those subregions are themselves obtained by consulting the conflict lists of the critical regions cut by the reinserted object. During the rebuilding process, each killed or new region appears at most once as a critical region which conflicts with the reinserted object. Moreover, each subregion is involved in at most one internal join, and each auxiliary or destroyed region in at most one external join. From this, we derive that the conflict lists of the new nodes can be computed in time proportional to the total size of the conflict lists of the nodes that are killed, destroyed, or new.

3. Other operations. Apart from managing the priority queue $\mathcal{Q}$, querying the dictionaries, and setting up the conflict lists of the new nodes, all the remaining operations are elementary. Their number is at most proportional to the number of new or destroyed nodes, and to the number of incident arcs in the augmented influence graph.

If $\mathcal{S}$ is a set of n segments, with a intersecting pairs, the mathematical expectation $f_0(l, \mathcal{S})$ of the number of regions defined and without conflict over a random l-sample of $\mathcal{S}$ is $O(l + a\frac{l^2}{n^2})$, as given by lemma 5.2.4. Thus,

$$O\left(\frac{1}{n}\sum_{l=1}^{n}\frac{f_0(l,\mathcal{S})}{l}\right) = O\left(1+\frac{a}{n}\right),$$

$$O\left(\sum_{l=1}^{n}\frac{f_0(l,\mathcal{S})}{l^2}\right) = O\left(\log n+\frac{a}{n}\right).$$

We can now use theorem 6.3.6 to state the following theorem, which summarizes the results so far:

Theorem 6.4.1 *Under the assumptions of dynamic randomized analyses, an augmented influence graph can be used to maintain the vertical decomposition of a set of segments with the following performances. Let $\mathcal{S}$ be the current set of segments, n be the size of $\mathcal{S}$, and a be the number of intersecting pairs in $\mathcal{S}$.*

- *The expected storage required by the algorithm is*

$$O(n \log n + a).$$

- *Inserting the n-th segment takes an average time*

$$O(\log n + \frac{a}{n}).$$

- *Deleting a segment takes an average time*

$$O\left(\log n + (1 + \frac{a}{n})(t + t')\right),$$

where the parameters t and t' stand respectively for the complexities of the operations on dictionaries and priority queues.

Therefore, if we use perfect dynamic hashing together with stratified tree, the expected cost of a deletion is

$$O\left(\log n + \left(1 + \frac{a}{n}\right) \log\log n\right).$$

If we use balanced binary trees, it remains

$$O\left(\log n + \left(1 + \frac{a}{n}\right) \log n\right).$$

For the preceding algorithm, we have merely applied the general principles of the augmented influence graph to the case of computing the vertical decomposition of a set of segments. In fact, in this specific case, we may derive a simpler algorithm, yet one that uses less storage. This algorithm does not need to keep the conflict lists and maintains a non-augmented influence graph. It is outlined in exercises 6.1, 6.2 and 6.3, and its performances are summarized in the following theorem:

Theorem 6.4.2 *Under the assumptions of dynamic randomized analyses, an influence graph can be used to maintain the vertical decomposition of a set of segments with the following performances. Let S be the current set of segments, n be the size of S, and a be the number of intersecting pairs in S.*

- *The expected storage required by the algorithm is*

$$O(n + a).$$

- *Inserting a segment takes an average time of*

$$O(\log n + \frac{a}{n}).$$

- *Deleting a segment takes an average time of*

$$O\left((1 + \frac{a}{n})(t + t')\right),$$

where the parameters t and t' stand respectively for the complexity of the operations on dictionaries and priority queues.

Therefore, the expected cost of a deletion is $O\left((1 + \frac{a}{n}) \log\log n\right)$ if we use perfect dynamic hashing coupled with stratified trees. It remains $O\left((1 + \frac{a}{n}) \log n\right)$ if we use balanced binary trees.

6.5 Exercises

Exercise 6.1 (Dynamic decomposition) Let us maintain dynamically the decomposition of a set of segments using an influence graph. Show that the creator of a new trapezoid, or a trapezoid unhooked during a deletion, is also the creator of at least one destroyed trapezoid.

Hint: The proof of this fact relies on the two additional properties possessed by the influence graph of a decomposition:

1. The influence domain of an internal node is contained in the union of the influence domains of its children.
2. If an object O_k is the determinant of an internal node, it necessarily is a determinant of at least one child of this node.

Let O_l be a segment creating a new trapezoid, or a trapezoid unhooked during the deletion of O_k. As in the entire chapter, the segments are indexed by their chronological rank and $\mathcal{S}_i$ stands for the set of the first i segments (in chronological order). To prove our assertion, we investigate the addition of O_k and successively O_l to the decomposition $\mathcal{D}ec(\mathcal{S}'_{l-1})$. The decompositions obtained are then successively $\mathcal{D}ec(\mathcal{S}_{l-1})$ and $\mathcal{D}ec(\mathcal{S}_l)$. It can be shown that there is a region H in $\mathcal{D}ec(\mathcal{S}_l)$ determined by a set that contains both O_k and O_l.

Exercise 6.2 (Dynamic decomposition) Let us assume that we use an influence graph to dynamically maintain the decomposition of a set $\mathcal{S}$ of segments. Here, we consider the deletion of a segment O_k. We still use the notation of section 6.4. In particular, the segments are indexed by their chronological rank. Let O_l be the segment to be reinserted during the deletion of O_k. Show that for any critical region F in the critical zone $\mathcal{Z}_{l-1}$ that conflicts with O_l, there is at least one destroyed region H, created by O_l, that intersects F, and satisfies at least one of the following conditions:

1. F contains a wall of H that stems from an endpoint of O_l, or an intersection point on O_l, and that butts against O_k (see figure 6.6a),
2. F contains a wall of H that stems from an endpoint of O_k, or an intersection point on O_k, and that butts against O_l (see figure 6.6b),
3. F contains a wall of H that stems from the intersection $O_l \cap O_k$ (see figure 6.6c),
4. F is bounded by two walls, one stemming from a point on O_k, the other stemming from a point on O_l, and both walls are contained within F (see figure 6.6d),
5. O_l and O_k support the floor and ceiling of H, both of which intersect F (see figure 6.6e).

Exercise 6.3 (Dynamic decomposition) The aim of this exercise is to show how we may dynamically maintain the decomposition of a set $\mathcal{S}$ of segments using a simple influence graph, without the conflict lists.

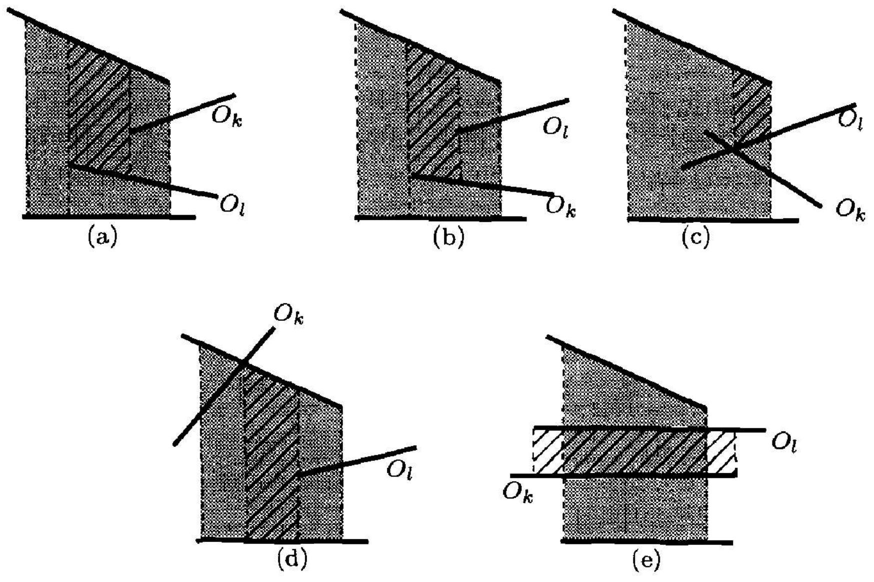


Figure 6.6. Detecting conflicts in critical zone. Region F is shaded, and region H within is emphasized.

The segments that must be reinserted during a deletion are the creators of destroyed regions (see exercise 6.1) and can be detected during the locating phase.

Let O_l be one of the segments to be reinserted during the deletion of O_k. To retrieve all the critical regions that conflict with O_l, the algorithm considers in turn the destroyed regions H with creator O_l, and selects the critical regions F related to H by one of the five cases described in exercise 6.2.

For this, the deletion algorithm maintains an augmented dictionary $\mathcal{A}$, storing the sequence ordered along O_k of critical regions intersected by O_k. Let H be one of the destroyed regions, created by O_l. If H has a wall that stems from a point on O_l and butts against O_k, or a wall stemming from $O_l \cap O_k$, this wall is located in the structure $\mathcal{A}$, and the critical region containing this wall is retrieved. If H has two walls stemming from a point on O_k and from a point on O_l, the region containing the wall stemming from the point on O_k is searched for in $\mathcal{A}$, and it is selected if it also contains the wall of H stemming from the point on O_l. Lastly, if O_l and O_k support the floor and ceiling of H, the right wall of H is searched for in $\mathcal{A}$, and any critical region that intersects the floor and the ceiling of H is selected.

1. The selected region obviously conflicts with O_l. As shown in exercise 6.2, any critical region that conflicts with H is selected. Show that such a region can be selected at most 16 times.

To speed up the locating phase, the algorithm maintains the lists of nodes killed by each object stored in the structure. To perform the deletion, the algorithm proceeds along the following lines.

Locating. The algorithm traverses the influence graph starting on the nodes killed by O_k, and visits the destroyed or unhooked nodes. During this traversal, the algorithm sets up a dictionary $\mathcal{D}$ that stores the destroyed and unhooked nodes, and a list $\mathcal{L}$ of the creators of the destroyed nodes.

Rebuilding. The list $\mathcal{L}$ is sorted by chronological order, for instance by storing the elements in a priority queue, and extracting them in order. The redundant elements are extracted only once. The dictionary $\mathcal{A}$ initially stores the regions killed by O_k.

The objects of $\mathcal{L}$ are processed in chronological order. For each object O_l, the critical regions that conflict with O_l are selected as explained above. The remaining operations are identical to those in the algorithm of section 6.4. The conflict lists of the new regions do not have to be computed. On the other hand, the dictionary $\mathcal{A}$ must be updated.

2. Show that the performances of this algorithm are those given by theorem 6.4.2.

Exercise 6.4 (Lazy dynamic algorithms) In this exercise, we propose a *lazy* method to dynamically maintain the decomposition of a set of segments. For simplicity, let us assume that the segments do not intersect. The algorithm maintains an influence graph in the following lazy fashion:

1. The graph is a mere influence graph, no conflict lists are needed.
2. During an insertion, the nodes corresponding to the new trapezoids are hooked to the nodes corresponding to the killed trapezoids as in the algorithms described in subsection 5.3.2 and section 6.4.
3. During a deletion, the nodes corresponding to the new trapezoids are hooked to leaves of the graph that correspond to destroyed trapezoids. More precisely, a node corresponding to a new trapezoid is hooked to leaves of the graph that correspond to the destroyed nodes that have a non-empty intersection with the new trapezoid. No node is removed from the graph.
4. The algorithm keeps the age of the current graph in a counter, meaning the total number of operations (insertions and deletions) performed on this graph. Each time the number of segments effectively present falls below half the number stored in this counter, the algorithm builds the influence graph anew by inserting the segments effectively present into a brand new structure.

1. Show that when $O(n)$ segments are stored in the structure, the expected cost of an insertion or a location query is still $O(\log n)$.

2. The cost of the periodic recasting of the graph is shared among all the deletions. Show that the amortized complexity of a deletion is still $O(\log n)$ on the average. (Recall that the segments do not intersect, by assumption.)

Hint: It will be noted that the number of children of a node in the influence graph is not bounded any more. The analysis must then have recourse to *biregions* (see exercise 5.7) to estimate the expected complexity of the locating phases.

6.6 Bibliographical notes

The approach discussed in this chapter consists in forgetting deleted objects altogether, and restoring the structure to the exact state which it would have been in, had this object never been inserted. The first algorithm following this approach is that of Devillers, Meiser, and Teillaud [81] which maintains the Delaunay triangulation of a set of points in the plane. The algorithm by Clarkson, Mehlhorn, and Seidel [70] uses the same approach to maintain the convex hull of a set of points in any dimension. The method was then abstracted by Dobrindt and Yvinec [86]. A similar approach is also discussed by Mulmuley [176], whose book is the most comprehensive reference on this topic.

There is another way to dynamize randomized incremental algorithms. This approach, developed by Schwarzkopf [198, 199], can be labeled as *lazy*. As outlined in exercise 6.4, it consists in not removing from the structure the elements that should disappear upon deletions. These elements are marked as destroyed, but remain physically present, and still serve for all subsequent locating phases. Naturally, the structure may only grow. When deletions outnumber insertions, the number of objects still present in the structure is less than half the number of objects still stored, and the algorithm completely rebuilds the structure from scratch, by inserting one by one the objects that were not previously removed.

Finally, we shall only touch the topic of randomized or derandomized dynamic structures which efficiently handle repetitive queries on a given set of objects, while allowing objects to be inserted into or deleted from this set. These structures embody the dynamic version of randomized divide-and-conquer structures, discussed in the notes of the previous chapter. These dynamic versions can be found in the works by Mulmuley [175], Mulmuley and Sen [178], Matoušek and Schwarzkopf [153, 156], Agarwal, Eppstein, and Matoušek [3] and Agarwal and Matoušek [4].