Optimal geometric data structures

DOI:
10.6100/IR721255

Document status and date:
Published: 01/01/2012

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Optimal Geometric Data Structures

Amirali Khosravi Dehkordi

Optimal Geometric Data Structures

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op dinsdag 10 januari 2012 om 16.00 uur

door

Amirali Khosravi Dehkordi

geboren te Teheran, Iran

Dit proefschrift is goedgekeurd door de promotor:

prof.dr. M.T. de Berg

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Khosravi Dehkordi, Amirali

Optimal Geometric Data Structures / by Amirali Khosravi Dehkordi.
Eindhoven: Technische Universiteit Eindhoven, 2011.
Proefschrift.
A catalogue record is available from the
Eindhoven University of Technology Library
ISBN: 978-90-386-3062-5

Subject headings: computational geometry / data structures / algorithms
CR Subject Classification (1998): I.3.5, E.1, F.2.2

The work in this thesis is supported by the Netherlands' Organization for Scientific Research (NWO) under project no. 639.023.301.

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

Cover Design: M. Gilaki
Printing: Eindhoven University Press

Contents

Acknowledgments

This thesis is the result of my 4 years research as a PhD on some interesting and challenging algorithmic problems in computational geometry. My PhD years in Eindhoven were full of new experiences and it is needless to say that finishing this thesis would not have been possible without the support and guidance of so many people. Due to this I owe many thanks to them.

During my PhD I had the opportunity to work with an excellent supervisor. Mark de Berg was not only a supervisor for me but also a supportive teacher and a good friend. Thank you Mark, for all our useful discussions about algorithms, politics, ... It happened so many times that I entered your office without prior notice, having a wrong idea to solve a problem and you listened patiently, even you knew that it was wrong. Thank you for all your insights, guidance and support, for your enthusiastic supervision of all aspects of my professional development, for all your great ideas to solve the problems and all the effort you put on rewriting and editing my writings. I hope one day I have the chance to work with you again.

Special thanks to my friend, co-author and the former member of algorithms group, Mohammad Ali Abam. I was lucky to have him around, at least in Europe, to help me with his advices and his amazing taste in mathematics. Thank you Mohammad for the wonderful time I had during my research visit to Dortmund and your visits to Eindhoven and for our exciting discussions about different topics. I believe without your help this thesis would not have been like what it is now.

It is difficult to overstate my gratitude to Boris Aronov. I had the opportunity to work with him closely for several times in Eindhoven and New York City. I learned a lot from the way he was approaching the problems and exploring different aspects of them. Thank you Boris for inviting me to New York City to do research with you. Your broad knowledge of algorithms and computational geometry greatly influenced me and I am deeply indebted to you.

My gratitude goes to the former and current members of TU/e algorithms group who made this stimulating atmosphere with mutual respect for me. I had tons of fun being with you guys. Special thanks to Constantinos Tsirogiannis, Herman Haverkort, Bettina Speckmann, Alexander Wolff, Dirk Gerrits, Kevin Verbeek, Peter Hachenberger, Kevin and Maike

Buchin, Anne Driemel and Atlas Cook. I would like also to thank our secretary Meivan Cheng and TU/e HR advisor Lutgart Kollenburg for their help to take all the hurdles and making all the bureaucratic processes easy for me.

I wish to thank Mohammad Ghodsi, my ex-supervisor in Sharif University of Technology who opened my eyes to the world of algorithms and computational geometry and guided me through it during my first steps. Many thanks go to my co-authors: Ignaz Rutter, Alireza Zarei, Sander Verdonschot and Vincent van der Weele. That was of my great pleasure to work with you. I must also thank the committee members of my PhD defense: Boris Aronov, Stefan Langerman, Bettina Speckmann and Gerhard Woeginger for the time they spent on reading my thesis and for their great comments on it.

During these 4 years I met so many people and had the chance to make many wonderful friends. My special gratitude goes to Mehdi Sarkeshi, Parsa Beigi, Reza Afra, Ali Etaati, Parisa and Amin Sedghiamiri , Leila and Salman Mokhlespoor, Arash and Mohammad Abtahi, Mehrnoosh Pouyanfar, Poulak Afroozeh and my old friends outside Netherland Reza Nikoobazl, Sarmad Fatemi, Amir Sadaghiani and Ali Hashemi. Thank you guys for being always there to support me and letting me feel home.

Special thanks to my managers in TomTom, Bastiaan van Hamersveld and Cyril Merique for trusting me and giving me the opportunity to follow my passion in TomTom. Hope that I can use my knowledge to bring some value to TomTom in future.

My special gratitude and love goes to Mojgan. Thank you for your understanding and caring support during the last few months of my PhD which inspired me a lot. Whether you realize or not your support, encouragement and love was invaluable to finish this thesis.

Finally, I am particularly grateful to my parents who always thought me fundamental aspects and real values in life, my sisters Parasto and Parisa, my brother Amir and my brother in law Ali Azami, for their constant encouragement and support. This thesis is dedicated to you.

Amirali Khosravi
Eindhoven, 2011

Chapter 1

Introduction

An algorithm is a clear step-by-step description of how to solve a problem. The use of algorithms is not restricted to scientific problems. Most of the time, when a person searches in a phone book for a name, he/she unconsciously uses an algorithm called *interpolation search*. Or when someone plans a trip to a specific destination, he/she usually employs, without knowing it, an algorithm for finding a shortest path. These are simple examples of algorithms that we use in our day-to-day life.

In the examples above the algorithms are executed by humans. In many cases, however, the problems and the algorithms that solve them are too complicated for this, and the algorithms need to be run on a computer. Thus, the design and analysis of algorithms is a core area within computer science. Let's have a look at some applications where efficient computer algorithms are required. For instance, consider an international airport, where many aircrafts land and take off in a day. Here, an accident may cause a tragedy. How can we schedule these flights so that no collision or other accident happens? Note that flight times may have to be changed due to delays or technical problems. Hence, the algorithm for scheduling the flights not only needs to find schedules that are safe, it also needs to be able to adapt the schedules quickly. Not only to handle this problem, but also for other scheduling problems we need to use sophisticated and fast algorithms.

The internet is another area where algorithms play a crucial role. In 1995 two PhD students at Stanford university developed an algorithm, called PageRank, to rank web pages based on their content. The success of Google is for a large part based on this clever algorithm. Google was not the first search engine, but PageRank gave the best search results at that time. There are many other challenging problems related to internet that require sophisticated algorithms. Almost all of the network routing and traffic challenges, internet auctions and advertisement issues and many more problems are solved using algorithms. Today the internet and world wide web have become the battle field for many companies, and their means to be successful are efficient algorithms.

Another area where algorithms have had a major impact is in biology [34]. For example, the enormous progress in understanding genes and their interaction would not have been possible without efficient algorithms. The list of areas where algorithms play a crucial role is almost endless. In many of the areas, spatial data (data describing objects in two-, three- or higher-dimensional space) play a crucial role. Computational geometry is the part of algorithms research dealing with spatial data. It emerged from the general algorithms area in the late 1970s [16]. Since then it has grown fast; thousands of articles have been published and hundreds of scientists form the computational geometry community nowadays. Since the subject of this thesis falls within computational geometry, the next section is devoted to a brief introduction to computational geometry. Then we give some background on the specific problems that we have considered, and in particular on the concept of instance-optimality. Finally, we give an overview of the results that we have obtained in this thesis.

1.1 Computational geometry

As mentioned above, computational geometry is a branch of algorithm design. Typically algorithms in computational geometry deal with geometric objects such as polygons, lines and points. Since we are living in a three-dimensional world, as soon as we want to model and/or simulate the real world, spatial data and geometric algorithms come into play. Many such algorithms have been developed by the computational-geometry community, but researchers from various application areas are developing geometric algorithms as well. The focus in computational geometry is often on algorithms with provably efficient worst-case behavior, while the focus in the application areas is often on algorithms with efficient implementations. Of course these approaches need not conflict, and in many cases practically efficient solutions are based on theoretically well-founded methods.

Next we give a few examples of algorithmic problems arising in various application areas and involving spatial data. We start with a problem from computer graphics. An important task in computer graphics is to compute the view of a scene, as seen by an observer located at a given view point. This involves determining which parts of the objects in the scene are visible and which are invisible; this is the so-called *hidden-surface removal* problem. One of its solutions, the so-called painter's algorithm, is related to a structure (BSP tree) we will study in this thesis, and it will be discussed in more detail later in this introduction.

Robotics is another area where geometric algorithms are required. As an example, efficient geometric algorithms are needed to find a route for a moving robot to its destination among obstacles. Much research in computational geometry has been done to solve different versions of this problem [72]—for more applications of computational geometry in robotics see, for example, the overview by Halperin *et al.* [47] or the book by Latombe [56].

Another area which can benefit from computational geometry is Geographical Information Systems (GIS) [55]. An important task of a GIS system is to store an (often huge) amount of geographical information in a suitable data structure—often called an *indexing structure*

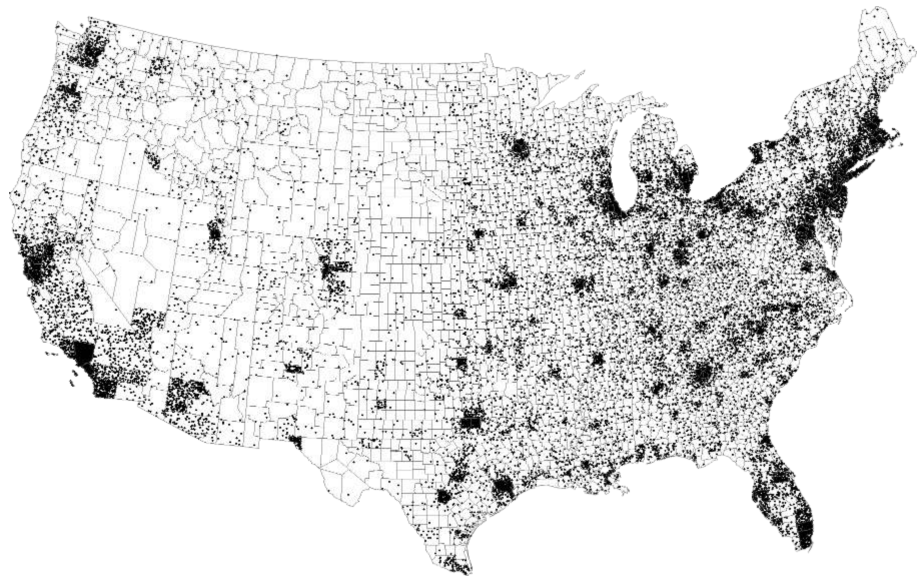


Figure 1.1 The dot density map of the U.S population in 2000, with each point representing 10000 people. Source: `http://udel.edu/~mtrainor/frec480/proj1/`

in GIS—such that certain queries can be answered efficiently. This data may consist of the borders of countries, elevation data in mountainous regions, etc. When we need to query this information, computational geometry can play an important role. As an example, suppose that we use points to show the population density in different regions. Each point represents a specific number of people living in that region. Such map is called a *dot-map*—see Fig. 1.1. To count the number of inhabitants in a specific query region we can use data structures for (weighted) range queries. We will discuss more about range searching later in this section and in Chapter 3.

Another field that has close connections to computational geometry is molecular biology. For example, one often models molecules as spheres, and this leads to many interesting geometric problems: testing for collisions during simulation of protein folding, finding offset surfaces, and more. CAD/CAM, pattern recognition and data bases are other examples of areas where geometric algorithms and data structures play a role. More information about computational geometry can be found in one of the various textbooks [16, 21, 66, 69] or handbooks [43, 70].

1.2 Spatial data structures in computational geometry

In many applications involving spatial data, the data needs to be stored in suitable data structures. The efficiency of the applications then directly depends on their underlying data structures. Many important spatial data structures, especially the ones used in practice, are based on *space partitioning* or *bounding volume hierarchies*. A space-partitioning structure for a set S of objects is an (often hierarchical) partitioning of the space into cells, such that each cell contains only a few objects. Examples are grids, octrees, and BSP trees. A bounding-volume hierarchy for a set S of objects is a tree whose leaves store the objects from S and whose interior nodes store a bounding volume for the objects stored in the node's subtree. Examples are R-trees and partition trees. We study some of these data structures in the thesis. Thus, first we introduce them briefly.

Binary space partitions

Description. In a BSP the space is recursively partitioned by hyperplanes until there is at most one object intersecting the interior of each cell in the final partitioning. Note that the splitting hyperplanes not only partition the space, they may also cut the objects into fragments. The recursive partitioning can be modeled by a tree structure, called a BSP tree. Nodes in a BSP tree correspond to regions of the original space, with the root node corrsponding to the whole space and the leaves corresponding to the cells in the final partitioning. Each internal node stores the hyperplane used to split the corresponding subspace, and each leaf stores the object fragment intersecting the corresponding cell—see Fig. 1.2. Note that the fact that the objects can be fragmented means that the number of leaves of a BSP tree can be superlinear in the number of objects it stores.

When the objects are $(d-1)$-dimensional—for example, a BSP for line segments in the plane—then it is sometimes required that the cells do not have any object in their interior. In other words, each fragment must end up being contained in a splitting hyperplane. The fragments are then stored with the splitting hyperplanes containing them, rather than at the leaves. In particular, this is the case for so-called *auto-partitions*. The performance of algorithms that use a BSP usually depends on the *size* of the BSP tree or, in other words, on the number of its leaves. Below we explain how BSPs can be used to do hidden-surface removal.

An application of BSPs. As mentioned above, the hidden-surface-removal problem is to determine the view of a set of objects as seen by an observer at a given view point. An image-space approach to this problem uses the fact that the view will be displayed on a screen and, hence, that the problem boils down to determining for every pixel on the screen which object is visible at that pixel. A popular method used in practice to solve this problem is called the *z-buffer algorithm*.

The z-buffer algorithm is based on scan converting the objects of the scene in arbitrary

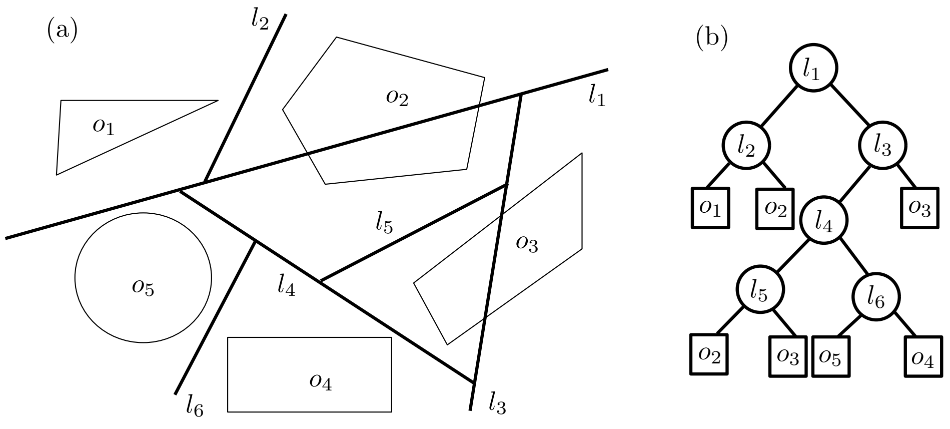


Figure 1.2 (a) A binary space partition of a set of objects. (b) The corresponding BSP tree.

order. Scan converting an object means determining which pixels the object covers in projection. We need two buffers called the *z-buffer* and the *frame buffer*. For each pixel, the frame buffer stores the intensity of the object visible at that pixel and the z-buffer stores the distance of the object visible at that pixel. When we scan convert an object we compare its distance to the observer to the value already stored in the z-buffer of a pixel at which the object is visible. If this value is smaller than the value stored in the z-buffer, we need to update the values stored in the z-buffer and frame buffer of that pixel. Note that the z-buffer algorithm has some overhead in terms of storage (namely an extra buffer in addition to the frame buffer) and computation time (because of the many depth computations and comparisons that need to be done).

As an alternative way to solve the hidden surface removal problem we can use the so-called *painter's algorithm*. This algorithm starts by sorting the objects in the scene according to their distance to the observer, so that an object that lies behind another object (as seen from the observer) comes earlier in the sorted order. After sorting the objects in this manner, they are scan-converted one by one without doing any more comparisons. Thus, the painter's algorithm always overwrites the frame buffer during scan conversion— no z-buffer is needed for comparisons of depth values.

However, computing an order for the objects is not so easy, and in some cases this order does not even exist. When we have a BSP on the objects, however, we can always generate a depth order on the object fragments stored in the BSP. This is because a fragment in another side (with respect to the observer) of a splitting plane cannot be in front of any fragment on the same side of the splitting plane. Hence, by traversing the BSP tree in a suitable order, based on the location of the observer with respect to the splitting planes, we can generate a depth order on the fragments.

Applications of BSPs is not restricted to the painter's algorithm. Binary space partitions are also used in constructive solid geometry [74], shadow generation [27], surface simplification [8] and many other application domains—see the overview by Tóth [77]. kd-trees, which are one of the well-known and widely used geometric data structure, are in fact special types of BSP trees, in which the splitting hyperplanes are required to be axis-aligned.

Previous work. In 1969 Schumacker *et al.* [71] introduced the idea of putting hyperplanes in a scene to help in constructing an order for the objects. Following this idea Fuchs *et al.* [40] proposed to use binary space partitions to solve hidden-surface-removal problem. Paterson and Yao [67] were the first to study the construction of BSPs from a theoretical point of view, and to analyze the worst-case number of fragments generated by their algorithms. They presented two algorithms, one deterministic and one randomized, for constructing BSPs for a set of segments in the plane. The worst-case (expected) size of the BSPs created (with respect to the number of fragments) by their algorithms is $O(n \log n)$. They also studied the problem in higher dimensions and proved that for any set of $(d-1)$-dimensional simplices in $\mathbb{R}^d$, for $d \geqslant 3$, a BSP of size $O(n^{d-1})$ can be constructed [67]. Paterson and Yao have proved that one can make a BSP of size $O(n\sqrt{n})$ for a set of axis-aligned boxes in $\mathbb{R}^3$ and this bound is tight in the worst case. Later, Tóth [75] showed that there exists a set of segments in $\mathbb{R}^2$ such that any BSP for this set has size $O(n \log n / \log \log n)$. The gap between this lower bound and the bound achieved by the algorithm of Paterson and Yao was open for several years, until recently Tóth gave an algorithm [80] to make a BSP of size $O(n \log n / \log \log n)$ for any set of segments in $\mathbb{R}^2$.

There are several results for special and more realistic input sets. For a set of axis-aligned segments in $\mathbb{R}^2$, Paterson and Yao proposed an algorithm that constructs a BSP of size $O(n)$ [68]. D'Amore and Franciosa [12] considered a similar problem and achieved the same result. Tóth [76] generalized these results such that for any set of segments with k different directions one can construct a BSP of size $O(kn)$. Dumitrescu, Mitchel and Sharir [33] studied the problem of making a BSP for axis-aligned segments, rectangles and hyper-rectangles in $\mathbb{R}^2$ and higher dimensions. De Berg [15] gave an algorithm which makes a BSP of size $O(n)$ for fat objects or more generally uncluttered scenes in $\mathbb{R}^d$. Agarwal *et al.* [6] studied the problem for fat rectangles in $\mathbb{R}^3$ and gave an algorithm to make a BSP of size $n \cdot 2^{O(\sqrt{\log n})}$. More results about BSPs can be found in a survey by Tóth [77].

Rectilinear r-partitions

Description. Another important class of spatial data structures are the so-called *bounding-volume hierarchies (*BVH*s)*. A bounding-volume hierarchy for a set S of objects is a tree whose leaves store the objects from S and whose interior nodes store a bounding volume for the objects stored in the node's subtree. Examples are R-trees and partition trees. Note that, in contrast to a BSP tree, the objects in a BVH are never fragmented. Hence, it always uses linear storage. An important type of BVHs is one that uses axis-aligned boxes as

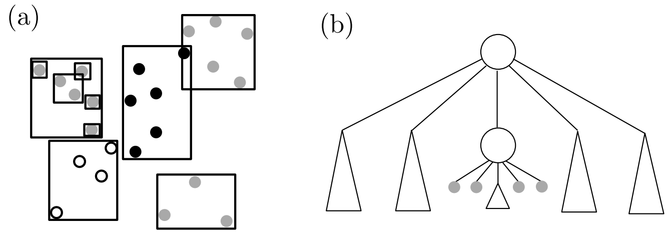


Figure 1.3 (a) A rectilinear r-partition for the set of points. (b) The box-tree corresponding to the rectilinear r-partition.

bounding volumes. Such a BVH is sometimes called a box-tree [4, 83]—see Figure 1.3.

We can use BVHs to solve one of the most fundamental problems in computational geometry, namely the *range-searching* problem. In range searching, our aim is to process a set S of n objects so that, given a query region, one can quickly find the objects lying in that query region. For an extensive overview of range searching, see the survey paper by Agarwal and Erickson [5]. When we have a BVH on the objects, we can answer a range query by traversing the tree in a top-down manner, as follows. Suppose we arrive at a node v of the tree. We then test if the bounding volume of v is completely or partially inside the query region or if it is disjoint from the query region. If the bounding volume of v is completely inside the query region, obviously all the objects inside the bounding volume of it are in the query region and we can report all the objects stored in the corresponding subtree. If the bounding volume of v is disjoint from the query region, none of its objects is in the query region and we can simply ignore its children for further processing. Finally, if the bounding volume of v is partially in the query region we repeat this process for its children.

As a concrete example, suppose that we want to answer range-searching queries on a set of points in the plane and that the query ranges are rectangles. A BVH which can be used for this is the so-called *box-tree*. A box-tree is a tree in which each leaf is associated with a bounding box of a few input objects (in this case points), and each interior node is associated with the smallest box B_v enclosing all the bounding boxes stored at the leaves of the subtree rooted at v. The basic building blocks of a box-tree are *rectilinear r-partitions*, which are defined as follows. A rectilinear r-partition of size r for a set S of n points in $\mathbb{R}^d$, is a collection $\psi(S) = \{(S_1, b_1), (S_2, b_2), \ldots, (S_r, b_r)\}$ such that the sets S_i form a partition of S and each b_i is a bounding box enclosing the points in S_i. A rectilinear r-partition is called *fine* if for each S_i we have $n/2r \leqslant |S_i| \leqslant 2n/r$. Note that the boxes b_i need not be disjoint. The definition of fine rectilinear r-partitions is similar to the definition of fine simplicial partitions by Matoušek [63]. The *stabbing number* of an axis-aligned hyperplane h in the rectilinear r-partition is the number of bounding boxes which it intersects. The stabbing number of rectilinear r-partition is the maximum stabbing number among all axis-aligned hyperplanes.

A box-tree can be used to answer a range searching query in $\mathbb{R}^2$, using the general strategy described earlier: traverse the tree in a top-down manner, only descending into subtrees if the bounding volume (rectangle in this case) stored at the root of the subtree partially overlaps the query range. It is not hard to see that if the query region is an axis-aligned half-plane then the query time will depend on the stabbing number of the rectilinear r-partition used to construct the tree. Thus we would like to compute a rectilinear r-partition for a set of points whose stabbing number is minimal.

When the BVH is stored in external memory, one usually uses B-tree [29, Chapter 18] as underlying data structure. The resulting data structure with bounding boxes as bounding volumes is then called an *R-tree* [45]. R-trees are extensively used as external memory data structures and have been studied extensively—see the book by Manolopoulos *et al.* [61]. In an R-tree all the leaves are at the same depth and each node, except the root, has a degree between t and $2t$ for a fixed parameter t. The degree of the root can be between 2 and $2t$. A common way to build R-trees is the top-down approach: we partition the set of objects S into subsets S_i, then recursively construct a subtree $\mathcal{T}_i$ for each set S_i. Thus the degree of the tree depends on the number of subsets. When a range query with a range Q is performed, one can use the same approach, described above, to recursively search in the subtrees $\mathcal{T}_i$ for which the bounding box of S_i intersects Q. It is not hard to see that the worst-case query time in an R-tree also depends on the stabbing number of the rectilinear partitions used in the construction of the R-tree. In Chapter 3 we therefore study the problem of computing rectilinear partitions with optimal stabbing number.

Previous work. There has been a lot of work on finding simplicial partitions with low stabbing numbers and on constructing partition trees [35, 25, 48, 63]. When it comes to rectilinear partitions and box-trees, there are fewer papers. For answering rectangle query ranges on a set of points, we can construct a kd-tree on the set of input points, and then convert this kd-tree to a box-tree. Then the query time is $O(n^{1-1/2d} + k)$ in $\mathbb{R}^d$ [5, 57] where k is the output size. There are some other heuristics which use kd-tree for answering rectangle queries [5, 65]. R-trees have been introduced for the first time by Guttmann [45], and to minimize the query time several heuristics have been proposed for making R-trees [37, 41, 58], but none of them studied bounds on their worst-case performance. Faloutos *et al.* [38] proposed a worst-case bound for a restricted case of a 1-dimensional R-tree. De Berg *et al.* [18] studied the problem of constructing an R-tree on a set of rectangles in $\mathbb{R}^2$ in order to report all the rectangles containing a query point, and proved worst-case bounds on the query time. Then, Agarwal *et al.* [4], gave an algorithm to build an R-tree that answers a rectangle query in $\mathbb{R}^d$ in $O((N/B)^{1-1/d} + T \log_B N)$ I/Os, where N is the number of d-dimensional hyper-rectangles stored in the R-tree, B is the block size and T is the output size. Later, Arge *et al.* [13] improved the query time to $O((N/B)^{1-1/d} + T/B)$ I/Os, using a variant of R-trees called PR-trees.

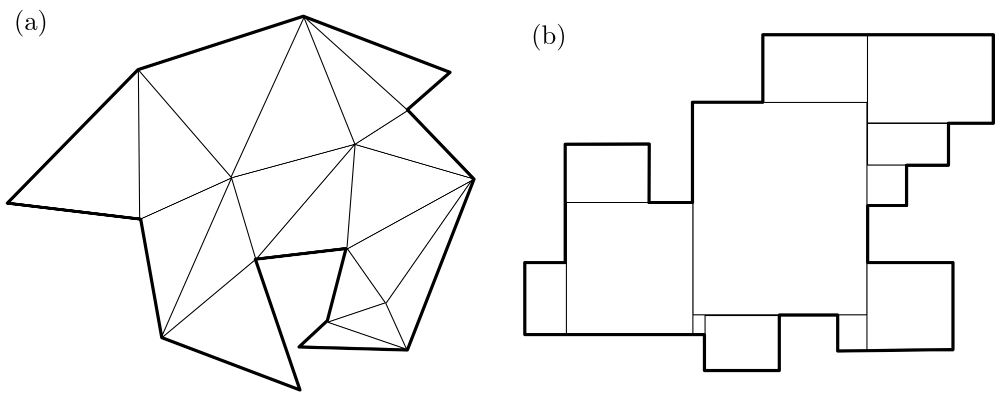


Figure 1.4 (a) A Steiner triangulation of a simple polygon. (b) A rectangular decomposition of a rectilinear polygon.

Decompositions of simple polygons.

Description. Above we discussed two important classes of geometric data structures, BSPs and BVHs. Next we turn our attention to decompositions of polygons, another problem that we will study in this thesis. Computing decompositions of simple polygons is one of the most fundamental problems in computational geometry. When the polygon at hand is arbitrary then one typically wants a decomposition into triangles, and when the polygon is rectilinear one wants a decomposition into rectangles—see Figure 1.4. Sometimes any such decomposition will do; then one can just compute an arbitrary triangulation or, for rectilinear polygons, a vertical decomposition. This can be done in linear time [23]. In other cases one would like the decomposition to have certain properties.

Suppose for example that we want to answer a ray shooting query in a simple polygon, that is, given a ray whose starting point lies inside the polygon we wish to find the first edge hit by the ray. If we have a triangulation of the polygon then, after locating the triangle containing the starting point, we can just traverse the triangles intersected by the ray to find the desired side of the polygon. The running time of the algorithm depends on the stabbing number of the triangulation, as defined next.

For a polygon P and a triangulation $\Delta(P)$ of it, the *stabbing number* of a segment s inside P is the number of triangles intersected by s. The *stabbing number* of $\Delta(P)$ is the maximum stabbing number over all segments inside P. It is easy to see that the performance of the ray shooting algorithm described above depends on the stabbing number of triangulation made for the polygon. A triangulation of a simple polygon usually only uses diagonals, that is, (non-intersecting) line segments connecting vertices of the polygons. However, one can also use additional vertices; these additional vertices are called *Steiner vertices*, and the resulting triangulation is called a *Steiner triangulation*. In fact, working with a Steiner triangulation may be necessary to ensure low stabbing number: there are simple polygons with n vertices where any non-Steiner triangulation has stabbing number $n-2$ while there are Steiner triangulations with stabbing number $O(\log n)$.

The (rectilinear) stabbing number of a rectangular decomposition of a rectilinear polygon is defined in a similar way as the stabbing number of a triangulation, except that we only consider axis-parallel segments inside the polygon. More precisely the stabbing number of an axis-parallel segment in a rectangular decomposition of a rectilinear polygon P is the number of rectangles which it intersects. The stabbing number of rectangular decomposition is the maximum stabbing number among all axis-aligned segments inside P. We will study decompositions with low stabbing number for simple polygons and rectilinear polygons in Chapter 4 of the thesis.

Previous work. There are many papers dealing with triangulations or other types of decomposition of simple polygons [52]. Hershberger and Suri [50] were the first to consider the problem of finding a Steiner triangulation with low stabbing number of a simple polygon. They proposed an algorithm that produces a Steiner triangulation with stabbing number $O(\log n)$. This is optimal in the worst case, since any Steiner triangulation of a convex polygon with n vertices has stabbing number $\Omega(\log n)$.

For the case of partitioning a rectilinear polygon into a set of rectangles De Berg and Van Kreveld [20] gave an algorithm which makes such a partition with stabbing number $O(\log n)$. This algorithm is also worst-case optimal, because any rectilinear partition of a staircase polygon of size n has stabbing number $\Omega(\log n)$. The two results just mentioned are about decompositions of simple polygons. There are also results about decompositions with low stabbing number of polygons with holes [50] and about decompositions with low stabbing number of a hypercube in d-dimensional space into a given number of boxes or other convex pieces [78, 79]. Finally, the problem of finding a triangulation with low stabbing number of a point set in $\mathbb{R}^3$ has been studied [3].

1.3 Worst-case optimality versus instance-optimality

In the previous section we discussed several geometric (data) structures. We also noted that for each of these data structures there are algorithms which construct data structures that are worst-case optimal with respect to a given characteristic such as size (BSPs) or stabbing number (rectilinear partitions, Steiner triangulations). As an example, for a set of n points and given r one can make a rectilinear r-partition with stabbing number $O(\sqrt{r})$ for the set of points. Moreover, there are point sets for which any rectilinear r-partition has stabbing number $\Omega(\sqrt{r})$. An example of such a point set is a set of n points forming a regular $\sqrt{n} \times \sqrt{n}$ grid. On the other hand, there are point sets for which we can make rectilinear r-partitions with better stabbing numbers. For example, when the points are all on a diagonal line we can make a rectilinear r-partition with stabbing number 1. Similarly when the points are in a convex position we can make a rectilinear r-partition with stabbing

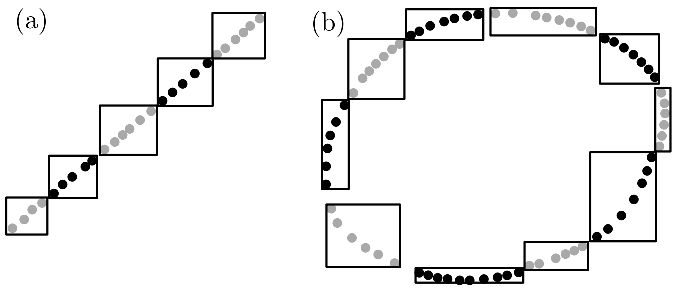


Figure 1.5 (a) A set of points on a diagonal and their rectilinear r-partition with stabbing number 1. (b) A set of points in a convex position.

number 2—see Fig. 1.5. Note that an algorithm that guarantees a worst-case optimal stabbing number may in fact produce a rectilinear r-partition with stabbing number $\Theta(\sqrt{r})$ in these two examples.

This gives the idea of looking for algorithms which make a rectilinear r-partition with minimum or close to minimum stabbing number for an input instance. We call the rectilinear r-partition with minimum stabbing number for a set of n points an *optimal rectilinear r-partition*.

The same phenomenon arises for the other two structures we discussed. First, consider the problem of computing BSPs of small size. The algorithm by Tóth [80] for computing a BSP for n segments in the plane produces a BSP of size $O(n \log n / \log \log n)$, which is optimal in the worst case, but there are also sets of segments that admit a BSP of size n. Second, consider the problem of computing decompositions with low stabbing number of simple and rectilinear polygons. As mentioned, the algorithms by Hershberger and Suri [50] and by De Berg and Van Kreveld [20], respectively, give decompositions with stabbing number $O(\log n)$ which is optimal in the worst case. However, there are polygons for which there exists rectilinear partitions or Steiner triangulations with lower stabbing numbers. The polygon which is shown in Figure 1.6 has a rectangular partition and a Steiner triangulation with $O(1)$ stabbing number.

To summarize, for all the above data structures there are algorithms that are worst-case optimal. More precisely, for each of these structures there are algorithms that produce a structure whose quality (in terms of size or stabbing number) is worst-case optimal in an asymptotic sense. However there are input instances for each of the problems for which the proposed algorithms do not do well. What we would prefer is an algorithm that is not just asymptotically optimal in the worst case, but that is optimal (or close to optimal) for the given input instance. This leads to the following general problem statement: given a class of geometric structures (BSPs, or rectilinear r-partitions, ...), and an input instance (a set of line segments, points, ...), construct a geometric structure on the given instance with minimal cost.

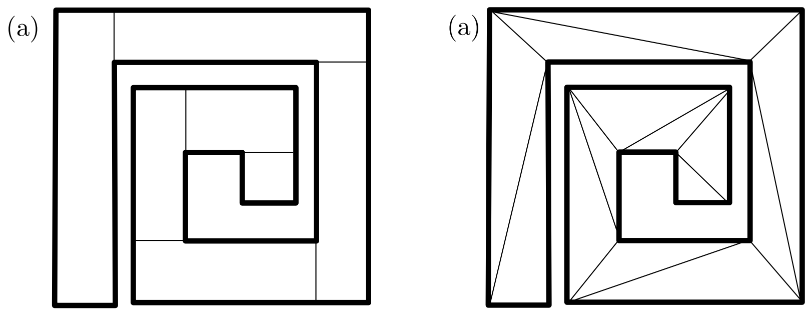


Figure 1.6 (a) The rectangular decomposition with $O(1)$ stabbing number. (b) The triangulation with $O(1)$ stabbing number.

Previous work. As just explained, we are interested in algorithms for computing geometric (data) structures that are optimal for the given instance. Observe that this way of looking at a problem is, in fact, standard practice in optimization problems. Thus the novelty in the above is not so much that we want solutions that are optimal for the given instance, but rather that we cast the problems of constructing BSPs of small size, and rectilinear r-partitions and polygon decompositions with low stabbing number, in this light.

Note that some existing papers use the term ”instance-optimal algorithms” in a somewhat different meaning. For example, Afshani *et al.* [2] consider instance-optimal algorithms for convex-hull computation in $\mathbb{R}^2$ and $\mathbb{R}^3$ and some other geometric problems. Here the instance-optimality refers to the running time of the algorithm, with respect to aspects like output size and input distribution. Fagen *et al.* [36] and others [30, 31] used the term in a similar way. When we talk about (instance-)optimality, however, we refer to the quality of the computed geometric structure with respect to the given criterion. For the problems we have studied, this has not been done before, to the best of our knowledge. The only exception is the computation of BSP of small size for axis-parallel segments in the plane, where De Berg *et al.* [19] showed that one can compute an (instance-)optimal rectilinear BSP using dynamic programming. To avoid confusion with the notion of instance-optimality studied in these other papers, we will from now on usually just talk about optimal BSPs, optimal rectilinear r-partitions and so on, when we mean instance-optimal structures. When we want to talk about worst-case optimality, we will use this term explicitly. A related result achieved on optimal BSPs is that for any set of (not necessarily rectilinear) disjoint segments in the plane one can compute a so-called *perfect* BSP in $O(n^2)$ time, if it exists [17]. (A perfect BSP is a BSP in which none of the objects is cut.) If a perfect BSP does not exist, then the algorithm only reports this fact; it does not produce any BSP in this case. Thus for arbitrary sets of segments in the plane it is unknown whether one can efficiently compute an optimal BSP.

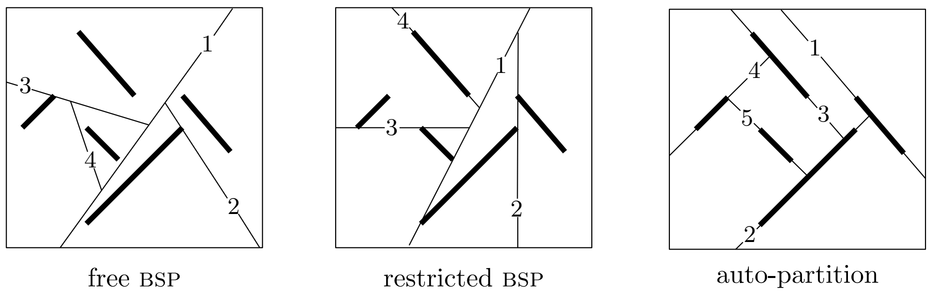


Figure 1.7 The three types of BSPs, drawn inside a bounding box of the scene. Note that, as is usually done for auto-partitions, we have continued the auto-partition until the cells are empty.

1.4 Results in this thesis

In this thesis we study the problems of finding optimal BSPs, rectilinear r-partitions with optimal stabbing number for point sets, and decompositions of simple and rectilinear polygons with optimal stabbing number. We show NP-hardness and/or give approximation algorithms for these problems. In addition, we study a problem that was not mentioned so far, namely the problem of finding optimal approximations of uncertain piecewise linear functions. Next we discuss our results in more detail.

Binary space partitions. Let S be a set of disjoint segments in the plane. Recall that a BSP is constructed by recursively splitting the space with a line, until each cell has at most one (fragment of a) segment in it. Thus, in a generic step of the algorithm we have a region R and a set $S(R)$ containing the segment fragments in R, and we wish to split R into two subregions with a line. In our search for optimal BSPs in $\mathbb{R}^2$ we considered three specific types of BSPs. The types of BSPs we consider differ in the restrictions we put on the splitting lines that we are allowed to use.

The first type of BSP we consider is one where we do not put any restrictions on the splitting lines. Such BSPs are called *Free* BSPs. The second type is called a *restricted* BSP; in this type of BSP we require that the splitting line for the region R contains (at least) two fragment endpoints in $S(R)$. The third type, which is the most restricted among those considered, is called an *auto-partition*; here we require that the splitting line for R contains one of the segments in $S(R)$. The three types of BSPs are illustrated in Figure 1.7. We first considered the problem of computing an optimal auto-partition. The reason to first study auto-partitions was that auto-partitions perform quite well in practice, even though for some input sets they produce larger BSPs than the other types. Also, since in constructing auto-partitions we are more restricted in choosing splitting lines, it might be easier to find an optimal auto-partition.

However, our attempts to develop an efficient algorithm to find optimal auto-partitions turned out to be unsuccessful. We proved that computing an optimal auto-partition is NP-hard. Interestingly, the problem of finding an optimal auto-partition seems to be more difficult than finding an optimal free BSP. As mentioned above, it is possible to decide in $O(n^2)$ time if a set n of line segments admits a perfect restricted (or free) BSP, that is, a restricted (or free) BSP that does not cut any of the input segments. However, our proof shows that determining if a set of input segments admits an auto-partition without cuts in NP-hard. Our NP-hardness proof is by reduction from a new version of 3-SAT, which we prove to be NP-complete. We call this new version PLANAR MONOTONE 3-SAT. We believe that this version of 3-SAT problem is interesting on its own and may be useful for NP-hardness proofs of other geometric and graph problems. Indeed it has been used in an NP-hardness proof for a problem related to switch graphs [53].

We also studied the relation between the numbers of cuts made by optimal free and restricted BSPs. Let us denote the number of cuts of the optimal free and restricted BSPs for a set of segments by OPT_{free} and OPT_{res}. It is obvious that $\text{OPT}_{\text{free}} \leqslant \text{OPT}_{\text{res}}$. In [28] there is an example showing that for some input sets $\text{OPT}_{\text{free}} = 2 \cdot \text{OPT}_{\text{res}}$. We show that this bound is tight: for any input set of segments we have $\text{OPT}_{\text{free}} \leqslant 2 \cdot \text{OPT}_{\text{res}}$. This result is interesting because it implies that if one can find an algorithm to make an optimal restricted BSP for an input set, the algorithm would be a 2-approximation for finding an optimal free BSP. All of the above results are presented in Chapter 2 of this thesis.

The results on binary space partitions were published as:

M. de Berg and A. Khosravi. Optimal binary space partitions in the plane. In *Proc. 16th Annual International Conference on Computing and Combinatorics (COCOON'10)*, pages 216–225, 2010.

Rectilinear r-partition. Inspired by the construction of R-trees, we studied the problem of computing optimal rectilinear r-partitions for a set of n points in the plane, both theoretically and experimentally. First, we showed that finding an optimal *rectilinear r-partition* when we have r as a parameter is NP-hard even in $\mathbb{R}^2$. Then, we gave an exact algorithm for finding an optimal *rectilinear r-partition* which has polynomial running time when r is a constant. We changed the same algorithm to be a 2-approximation with a better running time. By showing that any algorithm which only considers disjoint boxes would not give a good approximation factor we finished the theoretical studies about *rectilinear r-partitions*.

We also performed an experimental investigation of various heuristic algorithms. We tested four different heuristic methods each on four different types of point distributions. A simple variant of a kd-tree approach turned out to give the best results in our test. All the theoretical and experimental results of our research on *rectilinear r-partitions* is presented in chapter 3 of this thesis.

The results on rectilinear r-partition were published as:

M. de Berg, A. Khosravi, S. Verdonschot and V. van der Weele. On rectilinear partitions with minimum stabbing number. In *Proc. of 12th Algorithms and Data Structures Symp. (WADS 2011)*, pages 302–313, 2011.

Decompositions of simple polygons. The third problem that we consider from the instance-optimality point of view is the computation of decompositions with low stabbing number of rectilinear and simple polygons. First, we considered the problem of computing a rectangular decomposition of a rectilinear polygon with optimal stabbing number. For this problem we presented a 3-approximation algorithm. Our algorithm first partitions the rectilinear polygon P into histograms such that any segment inside P intersects at most 3 of these histograms. Our main result is an algorithm that computes a rectangular decomposition of a histogram with optimal stabbing number. This leads to a 3-approximation for the problem. Then we turned our attention to Steiner triangulations of simple polygons and proposed an $O(1)$-approximation for this problem. All of our results with respect to partitioning simple and rectilinear polygons are represented in chapter 4.

The results on decomposition of simple polygons were published as:

M. A. Abam, B. Aronov, M. de Berg and A. Khosravi. Approximation algorithms for computing partitions with minimum stabbing number of rectilinear and simple polygons. In *Proc. of 27th Annual Symp. on Computational Geometry (SoCG 2011)*, pages 407–416, 2011.

Approximating uncertain functions. In Chapters 2–4 we studied instance-optimal algorithms for computing geometric structures that are traditionally studied from a worst-case perspective. Another way to look at this is that we view the construction of these structures as an optimization problem. In Chapter 5 we also consider an optimization problem, namely how to best approximate (with respect to a given error function) a piecewise linear function $\mathrm{F}\colon \mathbb{R} \to \mathbb{R}$ by another linear function $\overline{\mathrm{F}}$ with fewer breakpoints. This is a classic problem in mathematics and computer science and it has already been studied extensively [1, 7, 9, 22, 44, 46, 51, 64, 81]. There are then two optimization problems that are considered: the min-k and the min-ε problem. In the min-k problem we are given an error ε, and our goal is to approximate F by $\overline{\mathrm{F}}$ using a minimum number of links and such that the error is at most ε. In the min-ε version of the problem we are given a number $k \geqslant 1$ and the goal is to find a piecewise linear function with at most k links which minimizes the error.

We study a new version of this problem, where the function F is not known exactly. More precisely, we are given a set of values $x_1, \ldots, x_n$—the x-coordinates of the breakpoints of the function—and for each x_i we are given a set $y_{i,1}, \ldots, y_{i,m_i}$ of potential values for $\mathrm{F}(x_i)$ together with the associated probabilities $p_{i,j}$. Thus $\mathbf{Pr}[\mathrm{F}(x_i) = y_{i,j}] = p_{i,j}$. Even though the min-$k$ and min-ε problems for piecewise linear functions have been considered thoroughly before [44, 46, 82], to the best of our knowledge these problems have not been considered for uncertain points so far. Define $m = \sum_{i=1}^{n} m_i$, that is, m is the total number of potential values over all $\mathrm{F}(x_i)$. We achieved the following two results.

First, we give an algorithm to solve the min-k problem for uncertain functions: given an uncertain function F and a maximum error ε, in $O(m)$ time we can compute a function $\overline{\mathrm{F}}$ with the minimum number of links such that *error*$(\mathrm{F}, \overline{\mathrm{F}}) \leqslant \epsilon$. Second, we show how to solve the min-ε problem: given an uncertain function F, and an integer value $1 \leqslant k \leqslant n$ and any $\delta > 0$, one can find a function $\overline{\mathrm{F}}$ of at most k links that minimizes *error*$(\mathrm{F}, \overline{\mathrm{F}})$ in $O(n^{4/3+\delta} + m \log n)$ time. The first problem is fairly easily solved by a reduction to the minimum link path problem. The second problem is more challenging, however. The details of the algorithms are presented in chapter 5 of the thesis.

The results on approximating piecewise linear functions were published as:

M. A. Abam, M. de Berg and A. Khosravi. Piecewise-linear approximations of uncertain functions. In *Proc. of 12th Algorithms and Data Structures Symp. (WADS 2011)*, ages 1–12, 2011.

Chapter 2

Binary Space Partitions for Segments in the Plane

Chapter summary. An optimal BSP for a set S of disjoint line segments in the plane is a BSP for S that produces the minimum number of cuts. We study optimal BSPs for three classes of BSPs, which differ in the splitting lines that can be used when partitioning a set of fragments in the recursive partitioning process: *free* BSPs can use any splitting line, *restricted* BSPs can only use splitting lines through pairs of fragment endpoints, and *auto-partitions* can only use splitting lines containing a fragment. We obtain the following two results:

- It is NP-hard to decide whether a given set of segments admits an auto-partition that does not make any cuts.
- An optimal restricted BSP makes at most twice as many cuts as an optimal free BSP for the same set of segments.

2.1 Introduction

Motivation. Many problems involving objects in the plane or some higher-dimensional space can be solved more efficiently if a hierarchical partitioning of the space is given. One of the most popular hierarchical partitioning schemes is the *binary space partition*, or BSP for short [16]. In a BSP the space is recursively partitioned by hyperplanes until there is at most one object intersecting the interior of each cell in the final partitioning. Note that the splitting hyperplanes not only partition the space, they may also *cut* the objects into fragments.

The recursive partitioning can be modeled by a tree structure, called a BSP *tree*. Nodes in a BSP tree correspond to regions of the original space, with the root node corresponding to the whole space and the leaves corresponding to the cells in the final partitioning. Each internal node stores the hyperplane used to split the corresponding region, and each leaf stores the object fragment intersecting the corresponding cell. When the objects are $(d-1)$-dimensional—as is the case, for example, for a BSP for line segments in the plane—then some fragments may end up being contained in a splitting hyperplane. The fragments are then stored with the splitting hyperplanes containing them, rather than at the leaves.

BSPs have been used in numerous applications. In most of these applications, the efficiency is determined by the *size* of the BSP tree, which is equal to the total number of object fragments created by the partitioning process. As a result, many algorithms have been developed that create small BSPs; Paterson and Yao [67] presented an algorithm that computes for any given set of n disjoint segments in the plane a BSP of size $O(n \log n)$. In a recent paper [80], Tóth gave an algorithm which constructs a BSP of size $\Omega(n \log n / \log \log n)$ for any set of disjoint segments in the plane, which is tight in the worst case [75]. For many other settings—axis-parallel objects, 3-dimensional objects, fat objects, etc.—algorithms have been developed that produce provably small BSPs. Paterson and Yao [68] presented an algorithm for constructing BSPs for orthogonal objects in 3-dimension. Dumitrescu, Mitchell and Sharir [33] have studied BSPs for axis-parallel segments, rectangles and hyper-rectangles in $\mathbb{R}^2$ and higher dimensions. Similarly D'Amore and Franciosa [12] considered the problem of making a BSP for axis aligned segments in $\mathbb{R}^2$. De Berg [15] gave an algorithm which constructs a BSP of linear size for fat objects, and more generally for so-called uncluttered scenes. Finally, Agarwal *et al.* [6] studied making BSPs for fat rectangles in $\mathbb{R}^3$, presenting an algorithm for constructing a BSP of size $n \cdot 2^{O(\sqrt{\log n})}$. For a more extensive overview see the survey paper by Tóth [77].

In all these algorithms, bounds are proved on the *worst-case size* of the computed BSP *over all sets of n input objects* from the class of objects being considered. Ideally, one would like to have an algorithm that computes a BSP that is *optimal for the given input*, rather than optimal in the worst-case. In other words, given an input set S, one would like to compute a BSP that is optimal (that is, has the minimum number of object fragments) for S.

For n axis-aligned segments in the plane, one can compute an optimal rectilinear BSP in $O(n^5)$ time using dynamic programming [19].(A rectilinear BSP is one in which all

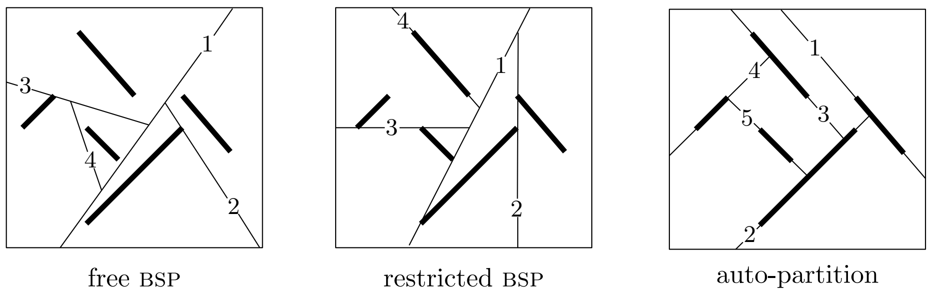


Figure 2.1 The three types of BSPs, drawn inside a bounding box of the scene. Note that, as is usually done for auto-partitions, we have continued the auto-partition until the cells are empty.

splitting hyperplanes are axis-aligned.) Another result related to optimal BSPs is that, for any set of (not necessarily rectilinear) disjoint segments in the plane, one can compute a so-called *perfect* BSP in $O(n^2)$ time, if it exists [17]. (A perfect BSP is a BSP in which none of the objects is cut.) If a perfect BSP does not exist, then the algorithm only reports this fact; it does not produce any BSP in this case. Thus for arbitrary sets of segments in the plane it is unknown whether one can efficiently compute an optimal BSP.

Problem statement and our results. In our search for optimal BSPs, we consider three types of BSPs. These types differ in the splitting lines they are allowed to use. Let S denote the set of n disjoint segments for which we want to compute a BSP, and suppose at some point in the recursive partitioning process we have to construct the subtree rooted at a node v. Let R be the region corresponding to v and let $S(R)$ be the set of segment fragments lying in the interior of R. Then the three types of BSPs can use the following splitting lines to partition R.

- *Free* BSP*s* can use any splitting line.
- *Restricted* BSP*s* must use a splitting line containing (at least) two endpoints of fragments in $S(R)$. We call such a splitting line a *restricted splitting line*.
- *Auto-partitions* must use a splitting line that contains a segment from $S(R)$.

Fig. 2.1 illustrates the three types of BSPs. Note that an auto-partition is only allowed to use splitting lines containing a fragment lying in the region to be split; it is not allowed to use a splitting line that contains a fragment lying in a different region. Also note that when a splitting line contains a fragment—such splitting lines must be used by auto-partitions, but may be used by the other types of BSPs as well—then that fragment is no longer considered in the rest of the recursive partitioning process. Hence, it will not be fragmented further.

For auto-partitions we require that the splitting process continues until each fragment is contained in a splitting line.

We use $\mathrm{OPT}_{\mathrm{free}}(S)$ to denote the minimum number of cuts in any free BSP for S. Thus the number of fragments in an optimal free BSP for S is $n + \mathrm{OPT}_{\mathrm{free}}(S)$. Similarly, we use $\mathrm{OPT}_{\mathrm{res}}(S)$ and $\mathrm{OPT}_{\mathrm{auto}}(S)$ to denote the minimum number of cuts in any restricted BSP and in any auto-partition for S. Clearly, $\mathrm{OPT}_{\mathrm{free}}(S) \leqslant \mathrm{OPT}_{\mathrm{res}}(S) \leqslant \mathrm{OPT}_{\mathrm{auto}}(S)$. It is well known that for some sets of segments $\mathrm{OPT}_{\mathrm{res}}(S) < \mathrm{OPT}_{\mathrm{auto}}(S)$; indeed, it is easy to construct an example where $\mathrm{OPT}_{\mathrm{res}}(S) = 0$ and $\mathrm{OPT}_{\mathrm{auto}}(S) = n/3$. Nevertheless, auto-partitions seem to perform well in many situations. Moreover, the collection of splitting lines to choose from in an auto-partition is smaller than for restricted or free BSPs, so computing optimal auto-partitions might be easier than computing optimal restricted or free BSPs. Unfortunately, our hope to find an efficient algorithm for computing optimal auto-partitions turned out to be unsuccessful: in Section 2.2 we prove that computing optimal auto-partitions is an NP-hard problem. In fact, it is even NP-hard to decide whether a set of segments admits a perfect auto-partition. This should be contrasted to the result mentioned above, that deciding whether a set of segments admits a perfect restricted BSP can be done in $O(n^2)$ time. (Notice that when it comes to perfect BSPs, there is no difference between restricted and free BSPs: if there is a perfect free BSP then there is also a perfect restricted BSP [17].) Hence, optimal auto-partitions seem more difficult to compute than optimal restricted or free BSPs.

Our hardness proof is based on a new 3-SAT variant, *monotone planar* 3-SAT, which we define and prove NP-complete in Section 2.2. We believe this new 3-SAT variant is interesting in its own right, and may find applications in other NP-completeness proofs. Indeed, our 3-SAT variant has already been used in a recent paper [53] to prove NP-hardness of a problem on so-called switch graphs.

We turn our attention in Section 2.3 to unrestricted and free BSPs. In particular, we study the relation between optimal free BSPs and optimal restricted BSPs. In general, free BSPs are more powerful than restricted BSPs: in his MSc thesis [28], Clairbois gave an example of a set of segments for which the optimal free BSP makes one cut while the optimal restricted BSP makes two cuts, and he also proved that $\mathrm{OPT}_{\mathrm{res}}(S) \leqslant 3 \cdot \mathrm{OPT}_{\mathrm{free}}(S)$ for any set S. In Section 2.3 we improve this result by showing that $\mathrm{OPT}_{\mathrm{res}}(S) \leqslant 2 \cdot \mathrm{OPT}_{\mathrm{free}}(S)$ for any set S.

2.2 Hardness of computing perfect auto-partitions

Recall that an auto-partition of a set S of disjoint line segments in the plane is a BSP in which, whenever a region is partitioned, the splitting line contains one of the fragments lying in that region. We call an auto-partition *perfect* if none of the input segments is cut, and we consider the following problem.

PERFECT AUTO-PARTITION

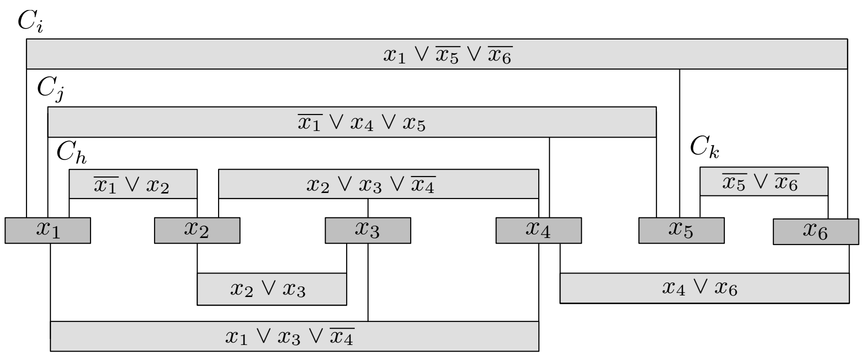

C_i
$x_1 \vee \overline{x_5} \vee \overline{x_6}$
C_j
$\overline{x_1} \vee x_4 \vee x_5$
C_h
C_k
$\overline{x_1} \vee x_2$
$x_2 \vee x_3 \vee \overline{x_4}$
$\overline{x_5} \vee \overline{x_6}$
x_1
x_2
x_3
x_4
x_5
x_6
$x_2 \vee x_3$
$x_4 \vee x_6$
$x_1 \vee x_3 \vee \overline{x_4}$

Figure 2.2 A rectilinear representation of a planar 3-SAT instance.

Input: A set S of n disjoint line segments in the plane.
Output: YES if S admits a perfect auto-partition, NO otherwise.

We will show that PERFECT AUTO-PARTITION is NP-hard. Our proof is by reduction from a special version of the satisfiability problem, which we define and prove NP-complete in the next subsection. After that we prove the hardness of PERFECT AUTO-PARTITION.

Planar monotone 3-SAT. Let $\mathcal{U} := \{x_1, \ldots, x_n\}$ be a set of n boolean literals, and let $\mathcal{C} := C_1 \wedge \cdots \wedge C_m$ be a CNF formula defined over these literals, where each clause C_i is the disjunction of at most three literals. Then 3-SAT is the problem of deciding whether such a boolean formula is satisfiable. An instance of 3-SAT is called *monotone* if each clause is monotone, that is, each clause consists only of positive literals or only of negative literals. 3-SAT is NP-complete, even when restricted to monotone instances [42].

For a given (not necessarily monotone) 3-SAT instance, consider the bipartite graph $\mathcal{G} = (\mathcal{U} \cup \mathcal{C}, \mathcal{E})$, where there is an edge $(x_i, C_j) \in \mathcal{E}$ if and only if x_i or its negation $\overline{x}_i$ is one of the literals in the clause C_j. Lichtenstein [60] has shown that 3-SAT remains NP-complete when $\mathcal{G}$ is planar. Moreover, as shown by Knuth and Raghunatan [26], one can always draw the graph $\mathcal{G}$ of a planar 3-SAT instance as in Fig. 2.2: the literals and clauses are drawn as axis-aligned rectangles with all the literal-rectangles on a horizontal line, the edges connecting the literals to the clauses are vertical segments, and the drawing is crossing-free. We call such a drawing of a planar 3-SAT instance a *rectilinear representation*. PLANAR 3-SAT remains NP-complete when a rectilinear representation is given.

Next we introduce a new version of 3-SAT, which combines the properties of monotone and planar instances. We call a clause with only positive literals a *positive clause*, a clause with only negative literals a *negative clause*, and a clause with both positive and negative literals a *mixed clause*. Thus a monotone 3-SAT instance does not have mixed clauses. Now consider a 3-SAT instance that is both planar and monotone. A *monotone rectilinear* *representation* of such an instance is a rectilinear representation where all positive clauses are drawn on the positive side of (that is, above) the literals and all negative clauses are drawn on the negative side of (that is, below) the literals. Our 3-SAT variant is defined as follows.

PLANAR MONOTONE 3-SAT
Input: A monotone rectilinear representation of a planar monotone 3-SAT instance.
Output: YES if the instance is satisfiable, NO otherwise.

PLANAR MONOTONE 3-SAT is obviously in NP. We will prove that it is NP-hard by a reduction from PLANAR 3-SAT. Let $\mathcal{C} = C_1 \wedge \cdots \wedge C_m$ be a given rectilinear representation of a planar 3-SAT instance defined over the literal set $\mathcal{U} = \{x_1, \ldots, x_n\}$. We call a literal-clause pair $(\overline{x}_i, C_j)$ *inconsistent* if $\overline{x}_i$ appears in C_j and C_j is placed on the positive side of the literals. Similarly we call (x_i, C_j) *inconsistent* if x_i appears in C_j and C_j is placed on the negative side. When a rectilinear representation does not have inconsistent literal-clause pairs, then it must be monotone. Indeed, any monotone clause must be placed on the correct side of the literals, and there cannot be any mixed clauses because any mixed clause must form an inconsistent pair with at least one of its literals.

We convert the given instance $\mathcal{C}$ step by step into an equivalent instance with a monotone planar representation, in each step reducing the number of inconsistent literal-clause pairs by one.

Let $(\overline{x}_i, C_j)$ be an inconsistent pair; inconsistent pairs involving a positive literal in a clause on the negative side can be handled similarly. We eliminate this inconsistent pair as follows.We introduce two new literals, a and b, and modify the set of clauses as follows (see Fig. 2.3).

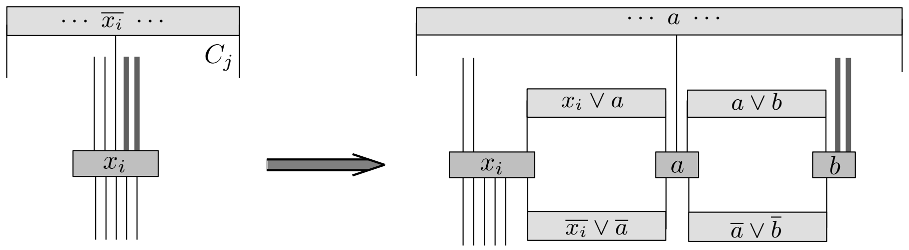
- In clause C_j, replace $\overline{x}_i$ by a.
- Introduce the following four clauses: $(x_i \vee a) \ \wedge \ (\overline{x}_i \vee \overline{a}) \ \wedge \ (a \vee b) \ \wedge \ (\overline{a} \vee \overline{b})$.
- In each clause containing x_i that is placed on the positive side of the literals and that connects to x_i to the right of C_j, replace x_i by b.

Let $\mathcal{C}'$ be the new set of clauses with the rectilinear representation obtained by the above process.

Lemma 2.1 *$\mathcal{C}$ is satisfiable if and only if $\mathcal{C}'$ is satisfiable.*

Proof. Suppose there is a truth assignment to the literals $x_1, \ldots, x_m$ that satisfies $\mathcal{C}$. Now consider $\mathcal{C}'$, which is defined over $\{x_1, \ldots, x_m\} \cup \{a, b\}$. Use the same truth assignment for $x_1, \ldots, x_m$, and set $a := \overline{x}_i$ and $b := x_i$. One easily checks that with this truth assignment to a and b, all new and modified clauses are satisfied.

Conversely, suppose there is a truth assignment to $\{x_1, \ldots, x_m\} \cup \{a, b\}$ that satisfies $\mathcal{C}'$. We claim that using the same assignment for $x_1, \ldots, x_m$ will satisfy $\mathcal{C}$. Indeed, $(x_i \vee a) \ \wedge \ (\overline{x}_i \vee \overline{a}) \ \wedge \ (a \vee b) \ \wedge \ (\overline{a} \vee \overline{b})$ implies that $x_i = \overline{a} = b$.



$\cdots\ \overline{x_i}\ \cdots$
C_j
x_i
$\cdots\ a\ \cdots$
$x_i \vee a$
$a \vee b$
x_i
a
b
$\overline{x_i} \vee \overline{a}$
$\overline{a} \vee \overline{b}$

Figure 2.3 Eliminating an inconsistent literal-clause pair.

This means that C_j (where $\overline{x}_i$ was replaced with a) and all clauses in $\mathcal{C}$ where x_i was replaced with b, are satisfied. All other clauses in $\mathcal{C}$ appear unchanged in $\mathcal{C}'$, and hence, they are also satisfied. □

Fig. 2.3 shows how this modification is reflected in the rectilinear representation. (In this example, there are two clauses for which x_i is replaced by b, namely the ones whose edges to x_i are drawn fat.) We keep the rectangle for x_i at the same location. Then we shift the vertical edges that now connect to b instead of x_i a bit to the right—because of this, we may have to slightly grow or shrink some of the clause rectangles as well—to make room for a and b and the four new clauses. This way we keep a valid rectilinear representation.

By applying the conversion described above without any specific order to each of the at most $3m$ inconsistent literal-clause pairs, we obtain a new 3-SAT instance with at most $13m$ clauses defined over a set of at most $n+6m$ literals. This new instance is satisfiable if and only if $\mathcal{C}$ is satisfiable, and it keeps the monotone representation. We get the following theorem.

Theorem 2.2 PLANAR MONOTONE 3-SAT *is* NP-*complete.*

Next we show how to reduce a planar monotone 3-SAT instance to a perfect auto-partition instance.

From planar monotone 3-SAT to perfect auto-partitions. Let $\mathcal{C} = C_1 \wedge \cdots \wedge C_m$ be a planar monotone 3-SAT instance defined over a set $\mathcal{U} = \{x_1, \ldots, x_n\}$ of literals, with a monotone rectilinear representation. We show how to construct a set S of line segments in the plane that admits a perfect auto-partition if and only if $\mathcal{C}$ is satisfiable. The idea is illustrated in Fig. 2.4.

The literal gadget. For each literal x_i there is a gadget consisting of two segments, s_i and $\overline{s}_i$. Setting x_i = TRUE corresponds to extending s_i before $\overline{s}_i$, and setting x_i = FALSE corresponds to extending $\overline{s}_i$ before s_i.

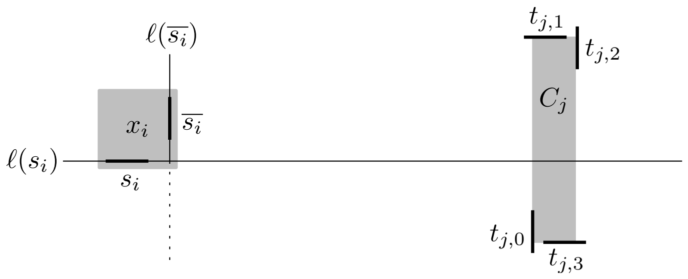


Figure 2.4 The idea behind replacing clauses and literals (C_j contains literal $\overline{x}_i$

The clause gadget. For each clause C_j there is a gadget consisting of four segments, $t_{j,0}, \ldots, t_{j,3}$. The segments in a clause form a cycle, that is, the splitting line $\ell(t_{j,k})$ cuts the segment $t_{j,(k+1) \bmod 4}$; see Fig.2.4. Note that, in the absence of other splitting lines, any auto-partition of the segments in the cycle makes at least one cut. In case a splitting line passes through a cycle, the cycle is broken and one can construct an auto-partition for the cycle with no cuts.

This means that a clause gadget, when considered in isolation, would generate at least one cut. Now suppose that the gadget for C_j is crossed by the splitting line $\ell(s_i)$ through the segment s_i in such a way that $\ell(s_i)$ separates the segments $t_{j,0}, t_{j,3}$ from $t_{j,1}, t_{j,2}$, as in Fig. 2.4. Then the cycle is broken by $\ell(s_i)$ and no cut is needed for the clause. But this does not work when $\ell(\overline{s}_i)$ is used before $\ell(s_i)$, since then $\ell(s_i)$ is blocked by $\ell(\overline{s}_i)$ before crossing C_j.

The idea is thus as follows. For each clause $(x_i \vee x_j \vee x_k)$, we want to make sure that the splitting lines $\ell(s_i)$, $\ell(s_j)$, and $\ell(s_k)$ all cross the clause gadget. Then by setting one of these literals to TRUE, the cycle is broken and no cuts are needed to create a perfect autopartition for the segments in the clause. We must be careful, though, that the splitting lines are not blocked in the wrong way—for example, it could be problematic if $\ell(\overline{s}_k)$ would block $\ell(s_i)$—and also that clause gadgets are only intersected by the splitting lines corresponding to the literals in that clause. Next we show how to overcome these problems.

Detailed construction. From now on we assume that the literals are numbered according to the monotone rectilinear representation, with x_1 being the leftmost literal and x_n being the rightmost literal.

The gadget for a literal x_i will be placed inside the unit square $[2i-2, 2i-1] \times [2n-2i, 2n-2i+1]$, as illustrated in Fig. 2.5. The segment s_i is placed with one endpoint at $(2i-2, 2n-2i)$ and the other endpoint at $(2i-\frac{3}{2}, 2n-2i+\varepsilon_i)$ for some $0 < \varepsilon_i < \frac{1}{4}$. The segment $\overline{s}_i$ is placed with one endpoint at $(2i-1, 2n-2i+1)$ and the other endpoint at $(2i-1-\overline{\varepsilon}_i, 2n-2i+\frac{1}{2})$ for some $0 < \overline{\varepsilon}_i < \frac{1}{4}$. Next we specify the slopes of the segments, which determine the values ε_i and $\overline{\varepsilon}_i$, and the placement of the clause gadgets.

The gadgets for the positive clauses will be placed to the right of the literals, in the

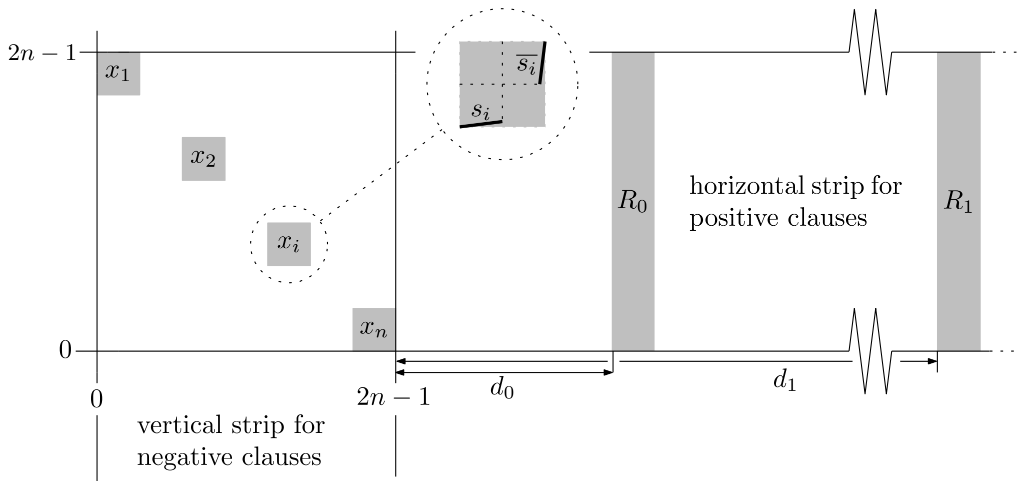


Figure 2.5 Placement of the literal gadgets and the clause gadgets (not to scale).

horizontal half-strip $[2n-1,\infty] \times [0,2n-1]$; the gadgets for the negative clauses will be placed below the literals, in the vertical half-strip $[0,2n-1] \times [-\infty,0]$. We describe how to choose the slopes of the segments s_i and to place the positive clauses; the segments $\overline{s}_i$ and the negative clauses are handled in a similar way.

Consider the set $\mathcal{C}^+$ of all positive clauses in our 3-SAT instance, and the way they are placed in the monotone rectilinear representation. We say that, in a given rectilinear representation, a clause C_i *directly encloses* a clause C_j if the following holds: C_i and C_j can be connected by a vertical segment that does not cross any other clause, with C_i being further away from the horizontal line on which the literals are placed. Let us call the clause directly enclosing a clause C_j the *parent* of C_j. In Fig. 2.2, for example, C_i is the parent of C_j and C_k but it is not the parent of C_h. Now let $\mathcal{G}^+ = (\mathcal{C}^+, \mathcal{E}^+)$ be the directed acyclic graph where each clause C_j has an edge to its parent (if it exists), and consider a topological order on the nodes of $\mathcal{G}^+$. We define the *rank* of a clause C_j, denoted by $\mathrm{rank}(C_j)$, to be its rank in this topological order. Clause C_j will be placed at certain distance from the literals that depends on its rank. More precisely, if $\mathrm{rank}(C_j) = k$ then C_j is placed in a $1 \times (2n+1)$ rectangle R_k at distance d_k from the line $x = 2n-1$ (see Fig. 2.5), where $d_k := 2 \cdot (2n)^{k+1}$.

Before describing how the clause gadgets are placed inside these rectangles, we define the slopes of the segments s_i. To this end we define $\mathrm{rank}(x_i)$, the rank of a literal x_i (with respect to the positive clauses), as the maximum rank of any clause it participates in. Now the slope of s_i is $\frac{1}{2 \cdot d_k}$, where $k = \mathrm{rank}(x_i)$. Recall that x_i is placed inside the unit square $[2i-2, 2i-1] \times [2n-2i, 2n-2i+1]$. We have the following lemma.

Lemma 2.3 *Let x_i be a literal, and $\ell(s_i)$ be the splitting line containing s_i.*

(i) For all x-coordinates in the interval $[2i-2, 2n-1+d_{\mathrm{rank}(x_i)}+1]$, the splitting line $\ell(s_i)$ has a y-coordinate in the range $[2n-2i, 2n-2i+1]$.

(ii) The splitting line $\ell(s_i)$ intersects all rectangles R_k with $0 \leqslant k \leqslant \mathrm{rank}(x_i)$.

(iii) The splitting line $\ell(s_i)$ does not intersect any rectangle R_k with $k > \mathrm{rank}(x_i)$.

Proof. (i) Observe that

$$(2n-1+d_{\mathrm{rank}(x_i)}+1-(2i-2)) \cdot \frac{1}{2d_{\mathrm{rank}(x_i)}} \;\leqslant\; \frac{1}{2} + \frac{2n-2i+2}{2d_{\mathrm{rank}(x_i)}} \;<\; 1,$$

since $i \geqslant 1$ and $d_{\mathrm{rank}(x_i)} \geqslant d_0 = 4n$. Hence, the increase in y-coordinate of $\ell(s_i)$ in the x-interval $[2i-2 : 2n-1+d_{\mathrm{rank}(x_i)}+1]$ is less than 1, proving (i).

(ii) immediately follows from (i).

(iii) We observe that

$$(2n-1+d_{\mathrm{rank}(x_i)+1}-(2i-2)) \cdot \frac{1}{2d_{\mathrm{rank}(x_i)}} \;\geqslant\; \frac{d_{\mathrm{rank}(x_i)+1}}{d_{\mathrm{rank}(x_i)}} \;=\; 2n,$$

so the increase in y-coordinate is at least $2n$ by the time $R_{\mathrm{rank}(x_i)+1}$ is reached. Hence, $\ell(s_i)$ passes above R_k for all $k > \mathrm{rank}(x_i)$. □

We can now place the clause gadgets. Consider a clause $C = (x_i \vee x_j \vee x_k) \in \mathcal{C}^+$, with $i < j < k$; the case where C contains only two literals is similar. By Lemma 2.3(ii), the splitting lines $\ell(x_i), \ell(x_j), \ell(x_k)$ all intersect the rectangle $R_{\mathrm{rank}(C)}$. Moreover, by Lemma 2.3(i) and since we have placed the literal gadgets one unit apart, there is a 1×1 square in $R_{\mathrm{rank}(C)}$ just above $\ell(s_i)$ that is disjoint from the supporting line of any segment. Similarly, just below $\ell(s_k)$ there is a square that is not crossed. Hence, if we place the segments forming the clause gadget as in Fig. 2.6, then the segments will not be intersected by any splitting line. Moreover, the splitting lines of segments in the clause gadget—these segments either have slope -1 or are vertical—will not intersect any other clause gadget.

This finishes the construction.

One important property of our construction is that the rectangle containing a clause gadget is only intersected by splitting lines of the literals in that clause. Another important property has to do with the blocking of splitting lines by other splitting lines. Recall that the rank of a literal is the maximum rank of any clause it participates in. We say that a splitting line $\ell(s_i)$ is *blocked* by a splitting line $\ell(s_j)$ if $\ell(s_j)$ intersects $\ell(s_i)$ between s_i and $R_{\mathrm{rank}(x_i)}$. This is dangerous, since it may prevent us from using $\ell(s_i)$ to resolve the cycle in the gadget of a clause containing x_i. The next lemma states the two key properties of our construction.

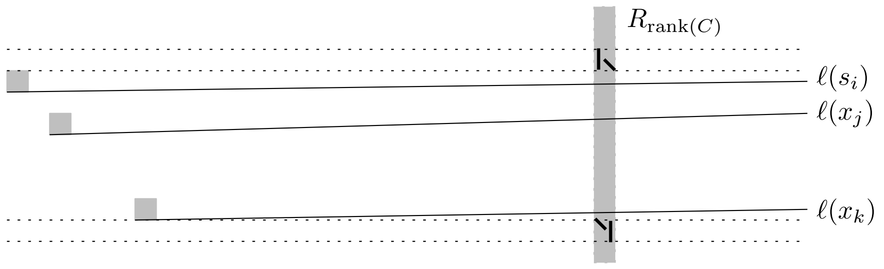


Figure 2.6 Placement of the segments forming a clause gadget.

Lemma 2.4 *The literal and clause gadgets are placed such that:*

(i) The gadget for any clause $(x_i \vee x_j \vee x_k)$ is only intersected by the splitting lines $\ell(s_i)$, $\ell(s_j)$, and $\ell(s_k)$. Similarly, the gadget for any clause $(\overline{x}_i \vee \overline{x}_j \vee \overline{x}_k)$ is only intersected by the splitting lines $\ell(\overline{s}_i)$, $\ell(\overline{s}_j)$, and $\ell(\overline{s}_k)$.

(ii) A splitting line $\ell(s_i)$ can only be blocked by a splitting line $\ell(s_j)$ or $\ell(\overline{s}_j)$ when $j \geqslant i$; the same holds for $\ell(\overline{s}_i)$.

Proof. (i) Consider a positive clause $C = (x_i \vee x_j \vee x_k)$ with $i < j < k$; the proof for positive clauses with two literals and for negative clauses is similar. The lines $\ell(s_i)$, $\ell(s_j)$, and $\ell(s_k)$ intersect the gadget for C by construction. Now consider any splitting line $\ell(s_l)$ with $l \notin \{i, j, k\}$. If $\mathrm{rank}(x_l) < \mathrm{rank}(C)$, then $\ell(s_l)$ does not intersect the gadget for C by Lemma 2.3(iii). If $\mathrm{rank}(x_l) \geqslant \mathrm{rank}(C)$ and $l < i$ or $l > k$, then $\ell(s_l)$ intersects $R_{\mathrm{rank}(C)}$ but not in between $\ell(s_i)$ and $\ell(s_k)$, by Lemma 2.3(i). Hence, in this case $\ell(s_l)$ does not intersect the clause gadget for C. The remaining case is that $\mathrm{rank}(x_l) \geqslant \mathrm{rank}(C)$ and $i < l < k$. But this is impossible, since the planarity of the embedding implies that if $i < l < k$ and $l \neq j$, then x_l can only participate in clauses enclosed by C, so $\mathrm{rank}(x_l) < \mathrm{rank}(C)$. Finally, we note that the gadget for C obviously is not intersected by any splitting line $\ell(\overline{s}_l)$, nor by any splitting line of a segment used in any other clause gadget.

(ii) Consider a splitting line $\ell(s_i)$; the proof for a splitting line $\ell(\overline{s}_i)$ is similar. If $\ell(s_i)$ is blocked by some $\ell(\overline{s}_j)$ then the diagonal placement of the literal gadgets (see Fig. 2.5) immediately implies $j \geqslant i$. Now suppose that $\ell(s_i)$ is intersected by some $\ell(s_j)$ with $j < i$. Then the slope of $\ell(s_i)$ is greater than the slope of $\ell(s_j)$. This implies that $\mathrm{rank}(x_i) < \mathrm{rank}(x_j)$. Hence, by Lemma 2.3(i) the intersection must be after $R_{\mathrm{rank}(x_i)}$, proving that $\ell(s_i)$ is not blocked by $\ell(s_j)$. □

Lemma 2.4 implies the main result of this section.

Theorem 2.5 PERFECT AUTO-PARTITION *is* NP*-complete.*

Proof. We can verify in polynomial time whether a given ordering of applying the splitting lines yields a perfect auto-partition, so PERFECT AUTO-PARTITION is in NP.

To prove that PERFECT AUTO-PARTITION is NP-hard, take an instance of PLANAR MONOTONE 3-SAT with a set $\mathcal{C}$ of m clauses defined over the literals $x_1, \ldots, x_n$. Apply the reduction described above to obtain a set S of $2n+4m$ segments forming an instance of PERFECT AUTO-PARTITION. Note that the reduction can be done such that the segments have endpoints with integer coordinates of size $O(n^{2m})$, which means the number of bits needed to describe the instance is polynomial in $n + m$. It remains to show that $\mathcal{C}$ is satisfiable if and only if S has a perfect auto-partition.

Suppose S has a perfect auto-partition. Set $x_i :=$ TRUE if s_i is extended before $\overline{s}_i$ in this perfect auto-partition, and set $x_i :=$ FALSE otherwise. Consider a clause $C \in \mathcal{C}$. Since the auto-partition is perfect, the cycle in the gadget for C must be broken. By Lemma 2.4(i) this can only be done by a splitting line corresponding to one of the literals in the clause, say x_i. But then s_i has been extended before $\overline{s}_i$ and, hence, $x_i =$ TRUE and C is true. We conclude that $\mathcal{C}$ is satisfiable.

Now consider a truth assignment to the literals that satisfies $\mathcal{C}$. A perfect auto-partition for S can be obtained as follows. We first consider s_1 and $\overline{s}_1$. When $x_1 =$ TRUE we first take the splitting line $\ell(s_1)$ and then the splitting line $\ell(\overline{s}_1)$; if $x_1 =$ FALSE then we first take $\ell(\overline{s}_1)$ and then $\ell(s_1)$. Next we treat s_2 and $\overline{s}_2$ in a similar way, then we proceed with s_3 and $\overline{s}_3$, and so on. So far we have not made any cuts. We claim that after having put all splitting lines $\ell(s_i)$ and $\ell(\overline{s}_i)$ in this manner, we can put the splitting lines containing the segments in the clause gadgets, without making any cuts. Indeed, consider the gadget for some, say, positive clause C. Because the truth assignment is satisfying, one of its literals, x_i, is TRUE. Then $\ell(s_i)$ is used before $\ell(\overline{s}_i)$. Moreover, because we treated the segments in order, $\ell(s_i)$ is used before any other splitting lines $\ell(s_j)$, $\ell(\overline{s}_j)$ with $j > i$ are used. By Lemma 2.4(ii) these are the only splitting lines that could block $\ell(s_i)$. Hence, $\ell(s_i)$ reaches the gadget for C and so we can use it to resolve the cycle and get a perfect auto-partition. □

2.3 Optimal free BSPs versus optimal restricted BSPs

Let S be a set of n disjoint line segments in the plane. In this section we will show that $\text{OPT}_{\text{res}}(S) \leqslant 2 \cdot \text{OPT}_{\text{free}}(S)$ for any set S. It follows from the lower bound of Clairbois [28] that this bound is tight.

Consider an optimal free BSP tree $\mathcal{T}$ for S, that is, a tree corresponding to an optimal free BSP of S. Let ℓ be the splitting line of the root of $\mathcal{T}$, and assume without loss of generality that ℓ is vertical. Let P_1 be the set of all segment endpoints to the left or on ℓ, and let P_2 be the set of segment endpoints to the right of ℓ. Let CH_1 and CH_2 denote the convex hulls of P_1 and P_2, respectively. We follow the same global approach as Clairbois [28]. Namely, we replace ℓ by a set L of three or four restricted splitting lines that do not intersect the

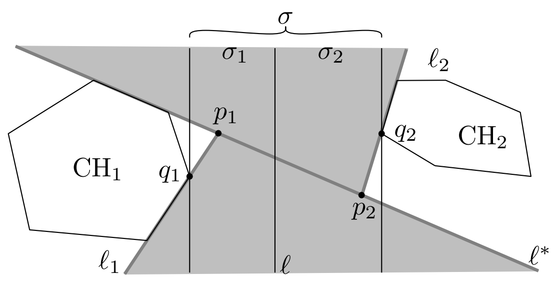


Figure 2.7 Illustration for Lemma 2.6, note that ℓ_1, ℓ_2 and ℓ^* are restricted splitting lines.

interiors of CH_1 and CH_2, and are such that CH_1 and CH_2 lie in different regions of the partition induced by L. In Fig. 2.7, for instance, we would replace ℓ by the lines ℓ^*, ℓ_1, ℓ_2 (it will be shown below how to find ℓ^*, ℓ_1, ℓ_2). The regions not containing CH_1 and CH_2—the grey regions in Fig. 2.7—do not contain endpoints, so inside them we can simply make splits along any segments intersecting the regions. After that, we recursively convert the BSPs corresponding to the two subtrees of the root to restricted BSPs. The challenge in this approach is to find a suitable set L, and this is where we will follow a different strategy than Clairbois.

Observe that the segments that used to be cut by ℓ will now be cut by one or more of the lines in L. Another potential cause for extra cuts is that existing splitting lines that used to end on ℓ may now extend further and create new cuts. This can only happen, however, when ℓ crosses the region containing CH_1 and/or the region containing CH_2 in the partition induced by L (the white regions in Fig. 2.7); if ℓ is separated from these regions by the lines in L, then the existing splitting lines will actually be shortened and thus not create extra cuts. Hence, to prove our result, we will ensure the following properties:

(I) the total number of cuts made by the lines in L is at most twice the number of cuts made by ℓ.

(II) in the partitioning induced by L, the regions containing CH_1 and CH_2 are not crossed by ℓ.

The lines in L are of three types. They are either *inner tangents* of CH_1 and CH_2, or *extensions* of edges of CH_1 or CH_2, or they pass through a vertex of CH_1 (or CH_2) and the intersection of another line in L with a segment in S.

We denote the vertex of CH_1 closest to ℓ by q_1 and we denote the vertex of CH_2 closest to ℓ by q_2 (with ties broken arbitrarily). Let σ be the strip enclosed by the lines through q_1 and q_2 parallel to ℓ, and for $i \in \{1, 2\}$ let σ_i denote the part of σ lying on the same side of ℓ as CH_i—see also Fig. 2.7. (When q_1 lies on ℓ, then σ_1 will just be a line; this does not invalidate the coming arguments.)

Lemma 2.6 *Let ℓ^* be a restricted splitting line separating CH_1 from CH_2. Suppose there are points $p_1 \in \ell^* \cap \sigma_1$ and $p_2 \in \ell^* \cap \sigma_2$ such that, for $i \in \{1,2\}$, the line ℓ_i through p_i and tangent to CH_i that separates CH_i from ℓ is a restricted splitting line (after making a cut along ℓ^*). Then we can find a set L of three partition lines satisfying conditions (I) and (II) above.*

Proof. Take ℓ^* as the first splitting line in L. Of all the points p_1 satisfying the conditions in the lemma, take the one closest to $\ell \cap \ell^*$. (If a segment $s \in S$ passes exactly through $\ell \cap \ell^*$, then $p_1 = \ell \cap \ell^*$.) The corresponding line ℓ_1 is the second splitting line in L. The third splitting line, ℓ_2, is generated similarly: of the points p_2 satisfying the conditions of the lemma, take the one closest to $\ell \cap \ell^*$ and use the corresponding line ℓ_2. Note that, when we replace ℓ by ℓ^*, the segments which were intersected by ℓ, are intersected by ℓ^* and we can use their intersection points as new segment endpoints. By construction, p_1p_2 is not intersected by any segment in S, which implies that condition (I) holds. Moreover, ℓ does not cross the regions containing CH_1 and CH_2, so condition (II) holds. $\square$

To show we can always find a set L satisfying conditions (I) and (II), we distinguish six cases. To this end we consider the two inner tangents ℓ' and ℓ'' of CH_1 and CH_2, and look at which of the points q_1 and q_2 lie on which of these lines. Cases (a)–(e) are handled by applying Lemma 2.6, case (f) needs a different approach. Next we discuss different cases in detail—see Fig. 2.8.

Case (a): Neither ℓ' nor ℓ'' contains any of q_1 and q_2. Let e_1 and e_2 be the edges of CH_1 and CH_2 incident to and below q_1 and q_2 respectively. Let $\ell(e_1)$ and $\ell(e_2)$ be the lines through these edges. Since neither ℓ' nor ℓ'' contains any of q_1 and q_2 then we have $\ell(e_1) \cap \ell(e_2) \in \sigma$. Assume without loss of generality that $\ell(e_1) \cap \ell(e_2) \in \sigma_2$. We can now apply Lemma 2.6 with $\ell^* = \ell(e_1)$, and $p_2 = \ell(e_1) \cap \ell(e_2)$, and $p_1 = q_1$.

Case (b): ℓ' contains one of q_1, q_2, and ℓ'' does not contain any of q_1, q_2. Assume without loss of generality that the inner tangent ℓ' that has CH_1 below it, contains q_2. Note that ℓ'' is above CH_2. We can now proceed as in case (a), except that we let e_1 and e_2 be the edges of CH_1 and CH_2 incident to and *above* q_1 and q_2, respectively.

Case (c): ℓ' contains q_1 and not q_2, and ℓ'' contains q_2 and not q_1. Similar to the previous cases. Note that e_1 and e_2 are the edges below q_1 and q_2.

Case (d): ℓ' contains both of q_1, q_2, and ℓ'' contains one of q_1, q_2. Apply Lemma 2.6 with $\ell^* = \ell'$, and $p_1 = q_1$, and $p_2 = q_2$.

Case (e): ℓ' contains both of q_1, q_2, and ℓ'' does not contain any of q_1, q_2. Apply Lemma 2.6 with $\ell^* = \ell'$, and $p_1 = q_1$, and $p_2 = q_2$.

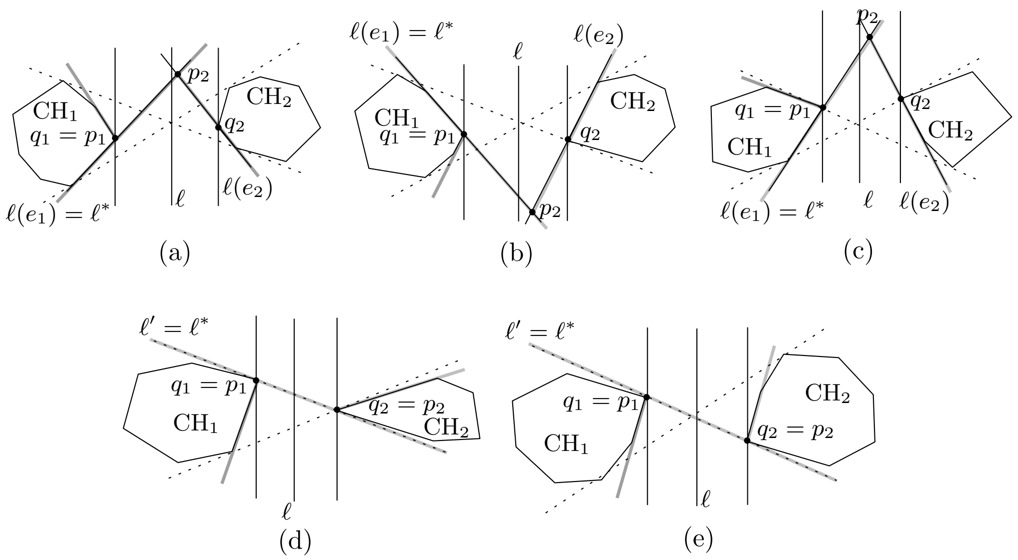


Figure 2.8 Illustrations for cases (a)–(e). ℓ' and ℓ'' are shown by dotted lines, $\ell(e_1)$ and $\ell(e_2)$ are shown by sold lines and the bold gray lines represent the lines in L.

Case (f): Both ℓ' and ℓ'' contain q_2 but not q_1. This is the most difficult case, and the only one where we need to replace ℓ with four splitting lines. Let e_1 be the edge of CH_1 incident to and above q_1 and e_1' the edge incident to and below q_1. Similarly, let e_2 be the edge incident to and above q_2 and e_2' the edge incident to and below q_2. We denote the intersection of ℓ' and $\ell(e_1)$ by o' and the intersection of ℓ'' and $\ell(e_1')$ by o''. Also, let $i' = \ell(e_2') \cap \ell(e_1')$, and $i'' = \ell(e_2) \cap \ell(e_1)$; see Fig. 2.9. We consider four subcases below.

Case (f.1): ℓ passes to the right of at least one of o' and o''. Assume without loss of generality that ℓ passes to the right of o'. Now we can apply Lemma 2.6 with $\ell^* = \ell'$, and $p_1 = o'$, and $p_2 = q_2$.

Case (f.2): ℓ passes to the left of at least one of i' or i''. Assume without loss of generality that ℓ passes to the left of i'. Now we can use Lemma 2.6 with $\ell^* = \ell(e_1')$, $p_1 = q_1$ and $p_2 = i'$.

Case (f.3): an input segment intersects $\ell(e_1)$ or $\ell(e_1')$ in a point $u \in \sigma_2$. Assume without loss of generality $u \in \ell(e_1)$. Now we can apply Lemma 2.6 with $\ell^* = \ell(e_1)$, $p_2 = u$ and $p_1 = q_1$.

Case (f.4): none of the cases (f.1)-(f.3) applies. As the first splitting line we choose ℓ'. For the second splitting line we initially set $p_1 = o'$ and draw a line $\ell(p_1)$ from p_1 passing through q_1; see Fig. 2.10(a). We move p_1 toward $\ell \cap \ell'$ while moving $\ell(p_1)$ with it, keeping

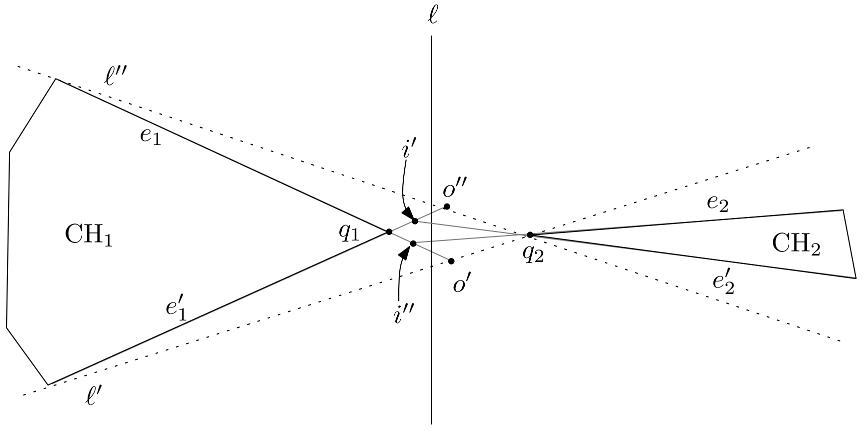


Figure 2.9 The case that i, i' are on the left and o' and o'' are on the right side of ℓ.

it tangent to CH_1, until it reaches the intersection of an input segment with ℓ'. If we reach such a point before arriving at $\ell \cap \ell'$, we take the resulting line $\ell(p_1)$ as the second splitting line; otherwise we move p_1 back to o' and use that line which is equal to $\ell(e_1)$ as the second splitting line. Note that if a line passes through o' we have $p_1 = o'$.

For the third splitting line we set $p_2 = \ell' \cap \ell$ and draw a line $\ell(p_2)$ from p_2 that is tangent to CH_2, such that CH_2 lies above it. We move p_2 toward q_2 until it reaches the intersection of a segment with ℓ' (if such an intersection does not exist at the end it will reach q_2). For the last splitting line we set $p_3 = \ell(p_1) \cap \ell$ and $p_4 = \ell' \cap \ell$ and make the segment p_3p_4, first we move p_3 toward q_1 until it reaches the intersection of an input segment with $\ell(p_1)$ or q_1. Then, we move p_4 until it reaches the intersection of an input segment with ℓ', if we cannot find such an intersection we rotate p_3p_4 to become fixed by CH_1.

It is easy to check that the resulting set, L, of splitting lines satisfies condition (II). It is important to note that p_2 can be on the left or right side of p_1 on ℓ'. When there is at least one input segment intersecting the segment made by o' and $\ell \cap \ell'$, then p_2 is on the left side of p_1 (or the same point as p_1), otherwise p_2 will be on the right side of p_1 and $\ell(p_1) = \ell(e_1)$. To show that condition (I) holds, we first consider the case where p_2 is on the left side of p_1, and then the case where p_2 is on the right side of p_1 is studied.

The segments which are intersected by L have an endpoint in CH_1 and another endpoint in CH_2. Imagine moving along such a segment from its endpoint inside CH_2 to its endpoint inside CH_1. We distinguish two types of segment, depending on whether the first splitting line in L that is crossed is ℓ' or $\ell(p_2)$

A segment of the first type, after intersecting ℓ', can intersect $\ell(p_1)$ or it can intersect p_1p_3 and then p_3p_4. In the first case it intersects two lines of L. To argue about the

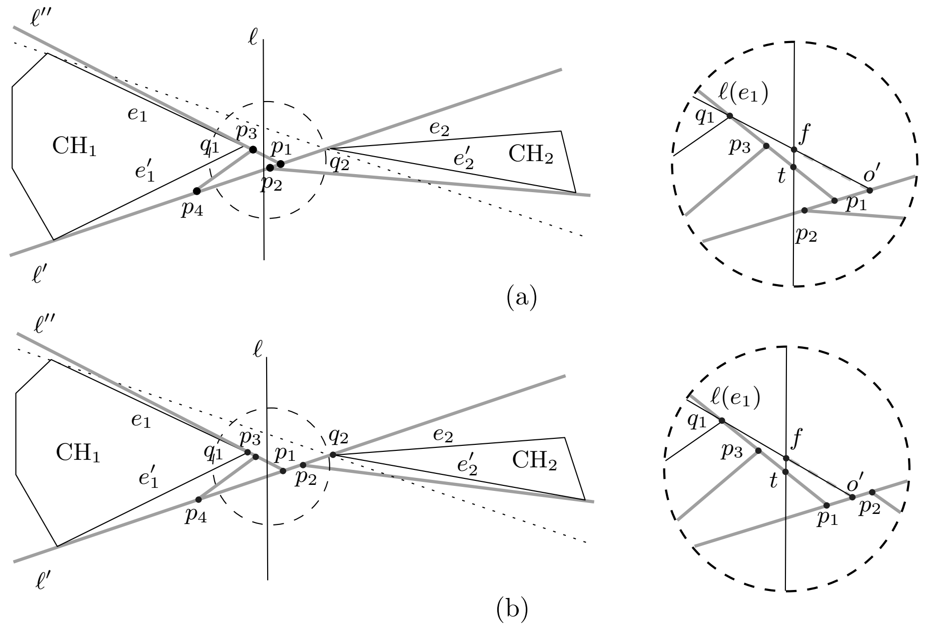


Figure 2.10 Illustration of difficult case and replacing ℓ with 4 splitting lines.

second case, denote the intersection of p_1p_3 with ℓ by t. By construction none of the input segments intersect tp_3, thus the segments of the second type can only intersect p_1t. According to case (f.3) none of the input segments intersects $\ell(e_1)$ in σ_2, thus $o'f$ (which is a part of $\ell(e_2)$ in σ_2) is not intersected by any input segment. By construction p_1o' is not intersected by any input segment. Thus, p_1t is not intersected and there cannot be any segments intersecting ℓ', p_1p_3 and p_3p_4. In conclusion, all input segments of the first type are intersected twice by the lines in L.

Now consider the segments of the second type, which first cross $\ell(p_2)$. After crossing $\ell(p_2)$, they can intersect ℓ', or they can intersect p_2p_4 (a part of ℓ') and then p_3p_4. In the first case, only two lines in L are intersected. As for the second case, by construction we know that none of the input segments intersects p_2p_4. Thus, this case in fact cannot occur. We can conclude that all input segments of the second type are intersected twice by the lines in L.

Now consider the case where p_2 is on the right side of p_1—see Fig. 2.10.(b). Again we can divide the segments into two types; the segments which first intersect ℓ', and the segments which first intersect $\ell(p_2)$. A segment of the first type can, after intersecting ℓ', intersect $\ell(p_1)$ or it can intersect first p_1p_3 and then p_3p_4. In the first case it intersects two lines of L.

To handle the second case, we denote $\ell(p_1) \cap \ell$ by f. The argument to show that none of

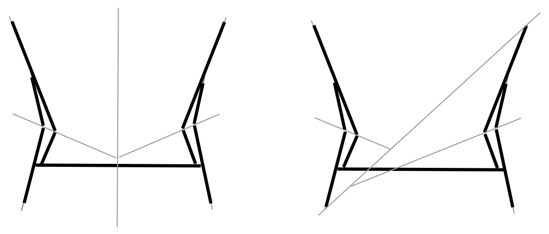


Figure 2.11 The structure in which we have $\mathrm{OPT}_{\mathrm{res}}(S) = 2 \cdot \mathrm{OPT}_{\mathrm{free}}(S)$ (the optimal free BSP on the left and optimal restricted BSP on the right), the bold segments are input segments and the grey lines are splitting lines. Note that the input segments are disjoint segments very close to each other and some of the splitting lines contain input segments.

the input segments intersects p_1p_3 is the same as the previous case. Hence all the segments which first intersect ℓ', are intersected twice by the lines in L.

The segments which first intersect $\ell(p_2)$, can then intersect p_2p_1, p_1p_3 and p_3p_4, or they can intersect p_1p_4 and p_3p_4 after intersecting $\ell(p_2)$, or they can intersect only ℓ'. In the last case there are just two lines in L intersected by the segments. By construction we know that none of the input segments intersects p_2p_4, and thus none of them intersects p_2p_1 and p_1p_4. Thus, the first two subsets are also empty and the set of input segments in this set are also intersected twice by the lines in L.

Theorem 2.7 *For any set S of disjoint segments in the plane,* $\mathrm{OPT}_{\mathrm{res}}(S) \leqslant 2 \cdot \mathrm{OPT}_{\mathrm{free}}(S)$.

A new lower-bound example. Clairbois [28] has shown a construction with a set S of 13 segments for which $\mathrm{OPT}_{\mathrm{res}}(S) = 2$ while $\mathrm{OPT}_{\mathrm{free}}(S) = 1$. That construction shows that our bound is tight. In Fig. 2.11 a simpler construction is given, which uses only 9 line segments, and for which we also have $\mathrm{OPT}_{\mathrm{res}}(S) = 2$ and $\mathrm{OPT}_{\mathrm{free}}(S) = 1$.

2.4 Conclusion

We showed that it is NP-hard to decide whether a set of segments in the plane admits a perfect auto-partition. This of course implies one cannot get an approximation algorithm for minimizing the number of cuts with a multiplicative approximation factor (unless P=NP). We do not know, however, how difficult it is to compute or approximate optimal restricted (or free) BSPs. Note that deciding whether there is a perfect restricted (or free) BSP is in fact easy, so any attempt to prove that computing optimal restricted (or free) BSPs is NP-hard should follow a different approach. We also showed that an optimal restricted BSP makes at most twice as many cuts as an optimal free BSP. Thus, when searching for an approximation algorithm for computing optimal BSPs, one could focus on the restricted BSPs—having such an algorithm would then immediately imply an approximation algorithm for free BSPs, whose approximation factor is at most twice as large.

Chapter 3

Rectilinear r-partitions

Chapter summary. Let S be a set of n points in $\mathbb{R}^d$, and let r be a parameter with $1 \leqslant r \leqslant n$. A rectilinear r-partition for S is a collection $\Psi(S) := \{(S_1, b_1), \dots, (S_t, b_t)\}$, such that the sets S_i form a partition of S, each b_i is the bounding box of S_i, and $n/2r \leqslant |S_i| \leqslant 2n/r$ for all $1 \leqslant i \leqslant t$. The (rectilinear) stabbing number of $\Psi(S)$ is the maximum number of bounding boxes in $\Psi(S)$ that are intersected by an axis-parallel hyperplane h. We study the problem of finding an optimal rectilinear r-partition—a rectilinear partition with minimum stabbing number—for a given set S. We obtain the following results.

- Computing an optimal partition is NP-hard already in $\mathbb{R}^2$.
- There are point sets such that any partition with disjoint bounding boxes has stabbing number $\Omega(r^{1-1/d})$, while the optimal partition has stabbing number 2.
- An exact algorithm to compute optimal partitions, running in polynomial time if r is a constant, and a faster 2-approximation algorithm.
- An experimental investigation of various heuristics for computing rectilinear r-partitions in $\mathbb{R}^2$.

3.1 Introduction

Motivation. Range searching is one of the most fundamental problems in computational geometry. In its basic form it can be stated as follows: preprocess a set S of objects in $\mathbb{R}^d$ into a data structure such that the objects intersecting a query range can be reported (or counted) efficiently. The range-searching problem has many variants depending, for example, on the type of objects (points, polygons, etc.), on the dimension of the underlying space (two- or higher-dimensional), and on the type of query range (boxes, simplices, etc.)—see the survey of Agarwal and Erickson [5] for an overview.

A range-searching data structure that is popular in practice is the *bounding-volume hierarchy*, or BVH for short. This is a tree in which each object from S is stored in a leaf, and each internal node stores a bounding volume of the objects in its subtree. Often the bounding volume is a *bounding box*: the smallest axis-aligned box containing the objects in the subtree. When a BVH is stored in external memory, one usually uses a B-tree [29, Chapter 18] as underlying tree structure; the resulting structure (with bounding boxes as bounding volumes) is then called an *R-tree*. R-trees are one of the most widely used external-memory data structures for spatial data, and they have been studied extensively—see for example the book by Manolopoulos *et al.* [61]. In this chapter we study a fundamental problem related to the construction of R-trees, as explained next.

One common strategy to construct R-trees is the top-down construction. Top-down construction algorithms partition S into a number of subsets S_i, and then recursively construct a subtree $\mathcal{T}_i$ for each S_i. Thus the number of subsets corresponds to the degree of the R-tree. When a range query with a range Q is performed, one has to recursively search in the subtrees $\mathcal{T}_i$ for which the bounding box of S_i (denoted by b_i) intersects Q. If $b_i \subset Q$, then all objects stored in $\mathcal{T}_i$ lie inside Q; if, however, b_i intersects ∂Q (the boundary of Q) then we do not know if the objects stored in $\mathcal{T}_i$ intersect Q. Thus the overhead of the search algorithm is determined by the bounding boxes intersecting ∂Q. If Q is a box, as is often the case, then the number of bounding boxes b_i intersecting ∂Q is bounded, up to a factor $2d$, by the maximum number of bounding boxes intersecting any axis-parallel plane. Thus we want to partition S into subsets so as to minimize the number of bounding boxes intersecting any axis-parallel plane.

Further background and problem statement. Let S be a set of n points in $\mathbb{R}^d$, and let r be a parameter with $1 \leqslant r \leqslant n$. A *rectilinear r-partition* for S is a collection $\Psi(S) := \{(S_1, b_1), \ldots, (S_t, b_t)\}$ such that the sets S_i form a partition of S, each b_i is the bounding box of S_i, and $n/2r \leqslant |S_i| \leqslant 2n/r$, for all $1 \leqslant i \leqslant t$. Note that even though the subsets S_i form a (disjoint) partition of S, the bounding boxes b_i need not be disjoint. The *stabbing number* of an axis-parallel hyperplane h with respect to $\Psi(S)$ is the number of boxes b_i whose relative interior intersects h, and the *(rectilinear) stabbing number* of $\Psi(S)$ is the maximum stabbing number of any axis-parallel hyperplane h. Observe that our rectilinear r-partitions are the axis-parallel counterpart of the (fine) simplicial partitions introduced by Matoušek [63].

It has been shown that there are point sets S for which any rectilinear r-partition has stabbing number $\Omega(r^{1-1/d})$ [63]; as an example when the points in S form a grid of size $n^{1/d} \times \cdots \times n^{1/d}$. Moreover, any set S admits a rectilinear r-partition with stabbing number $O(r^{1-1/d})$; such a rectilinear r-partition can be obtained by a construction similar to a kd-tree [16]. Thus from a worst-case and asymptotic point of view the problem of computing rectilinear r-partitions with low stabbing number is solved. However, any specific point set may admit a rectilinear r-partition with a much lower stabbing number than $\Theta(r^{1-1/d})$. For instance, if the points from S are all collinear on a diagonal line, then there is a rectilinear r-partition with stabbing number 1. The question now arises: given a point set S and a parameter r, can we compute a rectilinear r-partition that is *optimal for the given input set* S, rather than worst-case optimal? In other words, we want to compute a rectilinear r-partition that has the minimum stabbing number over all rectilinear r-partitions for S.

Our results. We start by a theoretical investigation of the complexity of the problem of finding optimal rectilinear r-partitions. In Section 3.2 we show that already in $\mathbb{R}^2$, finding an optimal rectilinear r-partition is NP-hard if r is considered as a parameter. In Section 3.3 we then give an exact algorithm for computing optimal rectilinear r-partitions which runs in polynomial time if r is a constant, and a 2-approximation with a better running time. We conclude our theoretical investigations by showing that algorithms only considering partitions with disjoint bounding boxes cannot have a good approximation ratio: there are point sets such that any partition with disjoint bounding boxes has stabbing number $\Omega(r^{1-1/d})$, while the optimal partition has stabbing number 2. We also perform an experimental investigation of various heuristics for computing rectilinear r-partitions with small stabbing number in $\mathbb{R}^2$. A simple variant of a kd-tree approach, which we call the *windmill kd-tree* turns out to give the best results.

3.2 Finding optimal rectilinear r-partitions is NP-hard

The exact problem we consider in this section is as follows.

OPTIMAL RECTILINEAR r-PARTITION
Input: A set S of n points in $\mathbb{R}^2$ and two parameters r and k.
Output: YES if S admits a rectilinear r-partition with respect to r with stabbing number at most k, NO otherwise.

We will show that this problem is already NP-complete for fixed values of k.

Theorem 3.1 OPTIMAL RECTILINEAR r-PARTITION *is* NP*-complete for* $k = 5$.

To prove the theorem we use a reduction from 3-SAT, which is similar to the proof by Fekete *et al.* [39] of the NP-hardness of minimizing the stabbing number of a matching on a planar point set. Let $\mathcal{U} := \{x_1, \ldots, x_m\}$ be a set of m boolean literals, and let $\mathcal{C} := C_1 \wedge \cdots \wedge C_s$ be a CNF formula defined over these literals, where each clause C_i is the disjunction of three literals. The 3-SAT problem is to decide whether such a boolean formula is satisfiable; 3-SAT is NP-hard [42]. Our reduction will be such that there is a rectilinear r-partition with stabbing number $k = 5$ for the OPTIMAL RECTILINEAR r-PARTITION instance if and only if the 3-SAT instance is satisfiable. To simplify the reduction we assume that $n = 72r$ (to make $\sqrt{2n/r}$ an integer greater than or equal to 12); however, the reduction works for any $n = \alpha \cdot r$ for an integer $\alpha \geqslant 72$. We first describe the various gadgets we need and then explain how to put them together.

The barrier gadget. A barrier gadget is a set G of $25 \cdot h^2$ points, where $h \geqslant 12$ and $h^2 = 2n/r$, arranged in a regular $5h \times 5h$ grid. To simplify the construction we fix $h = 12$. Thus a barrier gadget is simply a 60×60 grid placed in a small square. The idea is essentially that if we partition a barrier gadget and require stabbing number 5, then both the vertical and the horizontal stabbing numbers will be 5. This will prevent any other bounding boxes from crossing the vertical strip (and, similarly, the horizontal strip) whose bounding lines contain the vertical (resp. horizontal) edges of the square containing the barrier gadget. Thus the barrier gadget can be used to make sure there is no interaction between different parts of the global construction. Lemma 3.2 below makes this precise by giving a bound on the minimum stabbing number of any r-partition of a barrier gadget. In fact, we are interested in the case where G is a subset of a larger set S. In our construction we will place any barrier gadget G in such a way that the points in $S \setminus G$ lie outside the bounding box of G, so when analyzing the stabbing number of a barrier gadget we will always assume that this is the case.

Let G be a barrier gadget and $S \supset G$ be a set of n points. We define $\Psi(S \downarrow G)$, the *restriction to* G of a rectilinear r-partition $\Psi(S) = \{(S_1, b_1), \ldots, (S_t, b_t)\}$, as

$$\Psi(S \downarrow G) := \{(S_i \cap G, b_i) : 1 \leqslant i \leqslant t \text{ and } S_i \cap G \neq \emptyset\}.$$

In other words, the boxes in $\Psi(S \downarrow G)$ are the boxes from $\Psi(S)$ whose associated point set contains at least one point from the barrier. The following lemma gives a bound on the vertical and horizontal stabbing numbers of a rectilinear partition of a barrier gadget, where the vertical (horizontal) stabbing number is defined as the maximum number of boxes intersected by any vertical (horizontal) line.

Lemma 3.2 *A barrier gadget G can be covered by a set of 25 boxes with stabbing number 5. Moreover, for any rectilinear r-partition $\Psi(S)$ of stabbing number 5, the restriction $\Psi(S \downarrow G)$ has vertical as well as horizontal stabbing number 5.*

Proof. The first part of the lemma is easy: since we can put up to $h^2 \leqslant 2n/r$ points in a box, we can cover all the points from G using 25 square boxes each containing a subgrid of h^2 points.

To prove the second part of the lemma, consider a rectilinear r-partition $\Psi(S)$ of stabbing number 5. Consider the set B containing the boxes in $\Psi(S \downarrow G)$. Each box $b_i \in B$ contains at least one points from G, and together they contain at least $25 \cdot h^2$ points from G. Let L_V be a set of $5(h-1)+6$ vertical lines and L_H be a set of $5(h-1)+6$ horizontal lines that make the grid for the barrier gadget. The horizontal and vertical lines together separate all the points in G—see Fig. 3.1. Let $L = L_V \cup L_H$. Define $\lambda(b_i)$ to be the number of lines from L that intersect b_i, and define

$$\lambda^*(b_i) := \frac{\lambda(b_i)}{|S_i \cap G|}.$$

For example, if b_i is a box whose associated set $S_i \cap G$ is a $h \times h$ subgrid, then $\lambda(b_i) = 2h-2$ and $\lambda^*(b_i) = (2h-2)/h^2$.

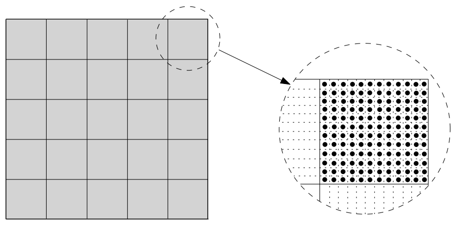


Figure 3.1 One of the subsquares of the barrier gadget with 144 points in it.

It is not difficult to verify that $(2h-2)/h^2$ is the minimum possible cost of any box $b_i \in B$. Indeed, consider an $a \times b$ box b_i. Then $|S_i \cap G| \leqslant a \cdot b$. Let us consider the case when $|S_i \cap G| = a \cdot b$ which gives the maximum value for all the $a \times b$ boxes. We have $\lambda(b_i) = a + b - 2$ and the minimum value for $\lambda(b_i)$ is when we choose a and b such that $|b-a| \leqslant 1$. Thus for a $a \times b$ box, when $|b-a| \leqslant 1$ we have the maximum value for $\lambda^*(b_i)$. If we consider all the boxes b_i in which $|b-a| \leqslant 1$, for larger values of a and b the value of $\lambda^*(b_i)$ would be smaller. This is because by increasing a and b the growth of h^2 is more than $2h-2$. Then the minimum value is when we have the largest box.

Thus the total cost of all boxes is

$$\sum_{b_i \in B} \lambda(b_i) \;=\; \sum_{b_i \in B} \left\{ \lambda^*(b_i) \cdot |S_i \cap G| \right\} \;\geqslant\; \frac{2h-2}{h^2} \cdot \sum_{b_i} |S_i \cap G| \;\geqslant\; \frac{2h-2}{h^2} \cdot 25 \cdot h^2 \;=\; 50h - 5.$$

Define $\lambda(\ell)$, for a line $\ell \in L$, as the number of boxes in $\Psi(S \downarrow G)$ that intersect ℓ. Observe that

$$\sum_{\ell \in L} \lambda(\ell) = \sum_{b_i \in B} \lambda(b_i).$$

Now assume for a contradiction that, say, the vertical stabbing number of $\Psi(S \downarrow G)$ is 4 or less. Then

$$\sum_{\ell \in L_H} \lambda(\ell) \;=\; \sum_{\ell \in L} \lambda(\ell) - \sum_{\ell \in L_V} \lambda(\ell) \;\geqslant\; 50h - 5 - 4 \cdot |L_V| \;=\; 30h - 9,$$

This implies that $\sum_{\ell \in L_H} \lambda(\ell) \geqslant 30h - 9$ and for $h \geqslant 12$ this implies that the average number of boxes intersected by each horizontal line is greater than 5. Thus the lemma holds. □

The literal gadget. Fig. 3.2 shows the literal gadget. The three subsets in the left part of the construction, and the three subsets in the right part, each contain $n/2r = 36$ points. Because of the barrier gadgets, the points from one subset cannot be combined with other points and must be put together into one rectangle in the partition. The six subsets in the middle part of the construction each contain $4n/r = 288$ points. To make sure the stabbing number does not exceed 5, these subsets can be grouped in two different ways. One grouping corresponds to setting the literal to true, the other grouping to false—see Fig. 3.2. Note that the gadget defines two vertical *slabs*. If the literal is set to true then the left slab has stabbing number 2 and the right slab has stabbing number 4, otherwise the opposite is the case.

The clause gadget. A clause gadget consists of three subsets of $4n/r = 288$ points, arranged as shown in Fig. 3.3(a), and placed in the left or right slab of the corresponding literals: a positive literal is placed in the left slab, a negative lateral in the right slab. If the stabbing number of the slab is already 4, which is the case when the literal evaluates to false, then the subset of $4n/r$ points in the clause gadget must be grouped into two “vertical” rectangles. Hence, not all literals in a clause can evaluate to false if the stabbing number is to be at most 5.

The global structure. The global construction is shown in Fig. 3.3(b). There are literal gadgets, clause gadgets, and barrier gadgets. The literal gadgets are placed diagonally and the clause gadgets are placed below the literals. We also place barriers separating the clause gadgets from each other. Finally, the gadgets for occurrences of the same literal in different clauses should be placed such that they are not stabbed by a common vertical line. This concludes our sketch of the construction which proves Theorem 3.3.

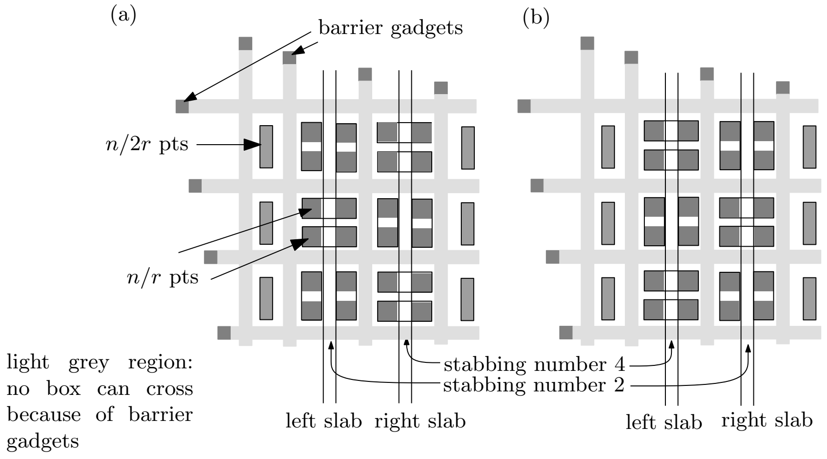


Figure 3.2 The literal gadget. (a) True setting. (b) False setting.

Theorem 3.3 OPTIMAL RECTILINEAR r-PARTITION *is* NP*-complete for* $k = 5$.

Proof. We can verify in polynomial time whether for a given set of boxes B we can make a rectilinear r-partition with stabbing number of 5 or not, so OPTIMAL RECTILINEAR r-PARTITION is in NP.

To prove that OPTIMAL RECTILINEAR r-PARTITION is NP-hard, take an instance of 3-SAT with a set $\mathcal{C}$ of m clauses defined over the literals $x_1, \ldots, x_n$. Apply the reduction described above to obtain a set of n points forming an instance of OPTIMAL RECTILINEAR r-PARTITION. As we described above we can set the values of k and r so that the relations for making a barrier gadget holds.

Suppose S has a rectilinear r-partition with stabbing number 5. Based on the construction this is only possible if for each clause gadget, at least 4 sets of n/r points (one set of $4n/r$ points) have paired horizontally. When this set of points is placed in the right slab of a literal we set its value to

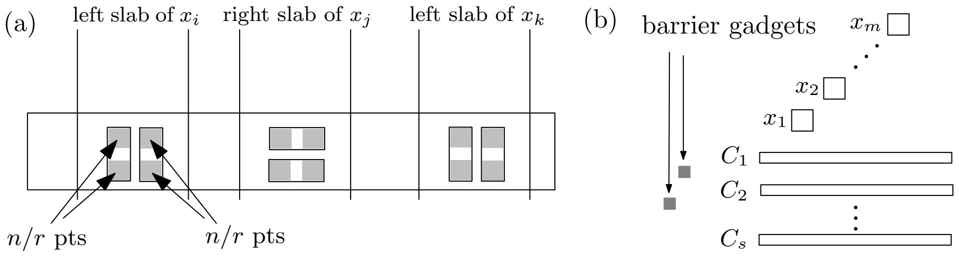


Figure 3.3 (a) A clause gadget for $(x_i \vee \overline{x_j} \vee x_k)$, and one possible grouping of the points. (b) The global structure.

false and when it is placed in the left slab of a literal we set its value to true. Since in each clause there exists 4 sets of n/r points which are paired horizontally. We conclude in each clause there is a true literal. Thus, $\mathcal{C}$ is satisfiable.

Now consider a truth assignment to the literals that satisfies $\mathcal{C}$. A rectilinear r-partition for S with stabbing number of 5 can be obtained as follows. For each barrier gadget it has already been shown that how we can make a rectilinear r-partition with stabbing number 5. Based on the values of literals we make the rectilinear r-partition partitions for every literal gadget as shown in Figure 3.2. For each literal gadget, we pair the sets of points in its slab with stabbing number 2 horizontally and its slab with slabbing number 4 vertically. Since every clause has a true literal, and based on the descriptions for literal and barrier gadgets the vertical and horizontal stabbing numbers of the rectilinear r-partition made is 5. $\square$

3.3 Polynomial time algorithms for constant r

In the previous section we showed that OPTIMAL RECTILINEAR r-PARTITION is NP-hard even in the plane when r is considered part of the input. Now we give a simple algorithm to show that the problem in $\mathbb{R}^d$ can be solved in polynomial time for fixed r in fixed dimension d. Our algorithm works as follows.

1. Let C be the set of all boxes defined by at most $2d$ points in S. Note that $|C| = O(n^{2d})$.
2. For each t with $r/2 \leqslant t \leqslant 2r$, proceed as follows. Consider all $O(n^{2dt})$ possible subsets $B \subset C$ with $|B| = t$. Check whether B induces a valid solution, that is, whether we can assign the points in S to the boxes in B such that (i) each point is assigned to a box containing it, and (ii) each box is assigned between $n/2r$ and $2n/r$ points. How this is done will be explained later.
3. Over all sets B that induce a valid solution, take the set with the smallest stabbing number. Each box in it is the bounding box of the points assigned to it, so we can report these boxes as the partition.

To implement Step 2 we construct a flow network with node set $\{v_{\text{source}}, v_{\text{sink}}\} \cup S \cup B$. The source node v_{source} has an arc of capacity 1 to each point $p \in S$, each $p \in S$ has an arc of capacity 1 to every $b_j \in B$ that contains p, and each $b_j \in B$ has an arc of capacity $2n/r$ to the sink node v_{sink}. The arcs from the boxes to the sink also have (besides the upper bound of $2n/r$ on the flow) a lower bound of $n/2r$ on the flow. The set B induces a valid rectilinear r-partition if and only if there is an integer flow of n units from v_{source} to v_{sink}. Such a flow problem can be solved in $O(\min(V^{3/2}, E^{1/2})E \log(V^2/E + 2) \log c)$ time [10], where V is the number of vertices in the network, E is the number of arcs, and c is the maximum capacity of any arc. We have $V = O(n)$, $E = O(nr)$, and $c = 2n/r$. Since we have to check $O(n^{4dr})$ subsets B, the running time is $O(n^{4dr} \cdot (nr)^{3/2} \log^2(n/r + 2))$ and is polynomial (assuming r and d as constants). We obtain the following result.

Theorem 3.4 *Let S be a set of n points in $\mathbb{R}^d$, and r a constant. Then we can compute a rectilinear r-partition with optimal stabbing number in time $O(n^{4dr+3/2} \log^2 n)$.*

We can significantly improve the running time if we are satisfied with a 2-approximation. The trick is to place a collection H_i of $3r$ hyperplanes orthogonal to the x_i-axis (the ith-axis) such that there are at most $n/3r$ points from S between any two consecutive hyperplanes in H_i. Instead of finding $O(n^{2d})$ boxes in the first step of the algorithm, we now find $O(r^{2d})$ boxes defined by the hyperplanes in $H := H_1 \cup \cdots \cup H_d$. Then we have $|C| = O(r^{2d})$. We apply the Step 2 of the algorithm and find for $r/2 \leqslant t \leqslant 2r$ all the $O(r^{2dt})$ subsets $B \subset C$. We check for each subset whether it is a valid solution, and take the best valid solution.

Theorem 3.5 *Let S be a set of n points, and r a constant. Then we can compute a rectilinear r-partition with stabbing number at most $2 \cdot OPT$, where OPT is the minimum stabbing number of any rectilinear partition for S, in time $O(n^{3/2} \log^2 n)$.*

Proof. Let $\Psi := \{(S_1, b_1), \ldots, (S_t, b_t)\}$ be an optimal rectilinear r-partition for S, and let OPT denote the stabbing number of Ψ. Expand every b_j in all directions until each facet of b_j is contained in a hyperplane from H. Let $\overline{b}_j$ denote the expanded box, and let $\overline{\Psi} := \{(S_1, \overline{b}_1), \ldots, (S_t, \overline{b}_t)\}$. The set $\{\overline{b}_1, \ldots, \overline{b}_t\}$ is one of the subsets B considered in Step 2, and it induces a valid solution. Hence, the stabbing number of the reported partition is at most the stabbing number of $\overline{\Psi}$.

Now consider any axis-parallel hyperplane h. Assume without loss of generality that h is orthogonal to the x_1-axis and that h lies in between hyperplanes $h_i, h_{i+1} \in H$. Let $\overline{b}_j$ be a box intersecting h. Note that b_j must intersect h_i or h_{i+1} (or both), otherwise b_j contains too few points. Hence, the h intersects at most $2 \cdot OPT$ boxes $\overline{b}_j$. □

3.4 Arbitrary versus disjoint rectilinear r-partitions

Since computing optimal rectilinear r-partitions is NP-hard, one should look at approximation algorithms. It may be easier to develop an approximation algorithm considering only rectilinear r-partitions with disjoint bounding boxes. The next theorem shows that in $\mathbb{R}^2$ such an approach will not give a good approximation ratio.

Theorem 3.6 *Assume that $32 \leqslant r \leqslant 4 \cdot \sqrt{n}$. Then there is a set S of n points in $\mathbb{R}^2$ whose optimal rectilinear r-partition has stabbing number 2, while any rectilinear r-partition with disjoint bounding boxes has stabbing number $\Omega(\sqrt{r})$.*

Proof. Let $\mathcal{G}$ be a $\sqrt{r/8} \times \sqrt{r/8}$ grid in $\mathbb{R}^2$. (For simplicity assume that $\sqrt{r/8}$ is an integer. Since $32 \leqslant r$ we have $\sqrt{r/8} \geqslant 2$.) We put each grid point in S and call them *black points*. We call the lines forming the grid $\mathcal{G}$ *black lines*. Note that there are $r/8$ black points. Fig. 3.4 shows an example with $r = 128$. Next we refine the grid using $2(\sqrt{r/8} - 1)$ additional axis-parallel *grey lines*. At each of the new grid points that is not fully defined by gray lines—the grey dots in the figure—we put a tiny cluster of $2n/r$ points, which we also put in S. If the cluster lies on one or more black lines, then all points from the cluster lie in the intersection of those lines, as shown in Fig. 3.4.

So far we used $(2r/8 - 2 \cdot \sqrt{r/8}) \cdot 2n/r + r/8$ points. Since $r \leqslant 4 \cdot \sqrt{n}$, the number of points which we used so far is less than n. The remaining points can be placed far enough from the construction

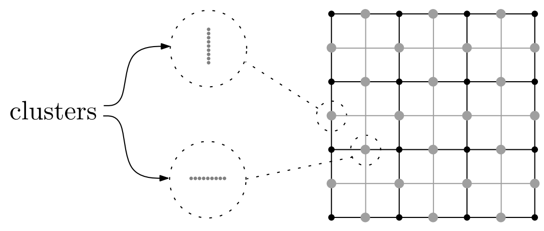


Figure 3.4 Every rectilinear r-partition with disjoint bounding boxes has stabbing number $\Omega(\sqrt{r})$ while there exists a partition with stabbing number 2.

(not influencing the coming argument.) Next, we rotate the whole construction slightly so that no two points have the same coordinate in any dimension. This rotated set is our final point set S.

To obtain a rectilinear r-partition with stabbing number 2, we make each of the clusters into a separate subset S_i, and put the black points into one separate subset; the latter is allowed since $r/8 \leqslant 2n/r$. (If $r/8 < n/2r$ we can use some of the remaining points or the points of gray dots to fill up the subset.)

If the clusters are small enough, then the rotation we have applied to the point set guarantees that no axis-parallel line can intersect two clusters at the same time. Any line intersects at most one of the clusters and the rectangle containing the black points, and the stabbing number of this rectilinear r-partition is 2.

We claim that any disjoint rectilinear r-partition for S has stabbing number $\Omega(\sqrt{r/8})$. To see this, observe that no subset S_i in a disjoint rectilinear r-partition can contain two black points. Indeed, the bounding box of any two black points contains at least one full cluster and, hence, together with the black points would have too many points. We conclude that each black point is assigned to a different bounding box. Let B be the collection of these bounding boxes. Now duplicate each of the black lines, and move the two duplicates of each black line slightly apart. This makes a set H of $O(\sqrt{r/8})$ axis-parallel lines such that each bounding box in B intersects at least one line from H.

Then the total number of intersections between the boxes in B and the lines in H is $\Omega(r)$, implying that there is a line in H with stabbing number $\Omega(\sqrt{r/8})$. □

The same argument holds for $\mathbb{R}^d$. Next, we prove the above theorem for $\mathbb{R}^d$.

Theorem 3.7 *Let d be a constant, and assume $d \cdot 2^{d+2} \leqslant r \leqslant 2d^{d/2} \cdot \sqrt{n}$. Then there is a set S of n points in $\mathbb{R}^d$ whose optimal rectilinear r-partition has stabbing number 2, while any rectilinear r-partition with disjoint bounding boxes has stabbing number $\Omega(r^{1-1/d})$.*

Proof. Let $\mathcal{G}$ be a $(r/2d^d)^{1/d} \times \ldots \times (r/2d^d)^{1/d}$ grid in $\mathbb{R}^d$. (We assume for simplicity that $(r/2d^d)^{1/d}$ is an integer; note that since $d \cdot 2^{d+2} \leqslant r$ we have $(r/2d^d)^{1/d} \geqslant 2$.) We put each grid point in S. We call these points *black points*, and we call the hyperplanes forming the grid $\mathcal{G}$ *black hyperplanes*. Note that there are $r/2d^d$ black points. Fig. 3.4 shows an example for $d = 2$ with

$r = 128$. Next we refine the grid using $d((r/2d^d)^{1/d} - 1)$ additional axis-parallel *grey hyperplanes*. At each of the new grid points that is not fully defined by gray hyperplanes—the grey dots in the figure—we put a tiny cluster of $2n/r$ points, which we also put in S. If the cluster lies on one or more black hyperplanes, then all points from the cluster lie in the intersection of those hyperplanes.

So far we used

$$((2((r/2d^d)^{1/d} - 1))^d - ((r/2d^d)^{1/d} - 1)^d - ((r/2d^d)^{1/d})) \cdot 2n/r$$

points; in gray dots. Since we have r $\leqslant 2d^{d/2} \cdot \sqrt{n}$ it is easy to show that this number is less than $(r/2 - 1) \cdot 2n/r$. Moreover $(r/2d^d)$ (the number of black points) is less than $2n/r$. Thus the total number of points we have put so far is less than n. The remaining points can be placed far enough from the construction, not influencing the coming argument.) Next, we rotate the whole construction slightly so that no two points have the same coordinate in any dimension. This rotated set is our final point set S.

To obtain a rectilinear r-partition with stabbing number 2, we make each of the clusters into a separate subset S_i, and we put the black points into one separate subset; the latter is allowed since $r/d^2 \leqslant 2n/r$. (If $r < n/2r$ we can use some of the remaining points or the points of gray dots to fill up the subset.) If the clusters are small enough, then the rotation we have applied to the point set guarantees that no axis-parallel hyperplane can intersect two clusters at the same time. Hence, the stabbing number of this rectilinear r-partition is 2.

We claim that any disjoint rectilinear r-partition for S has stabbing number $\Omega(r^{1-1/d})$. To see this, observe that no subset S_i in a disjoint rectilinear r-partition can contain two black points. Indeed, the bounding box of any two black points contains at least one full cluster and, hence, together with the black points would be too many points. We conclude that each black point is assigned to a different bounding box. Let B be the collection of these bounding boxes. Now consider a set H of $O(r^{1/d})$ axis-parallel hyperplanes such that each bounding box in B intersects at least one hyperplane from H. (Such a set can be found by duplicating each of the black hyperplanes, and moving the two duplicates of each black hyperplane slightly apart.) Then the total number of intersections between the boxes in B and the hyperplanes in H is r, which implies that there is a hyperplane in H with stabbing number $\Omega(r^{1-1/d})$. □

3.5 Experimental results

In the previous sections we studied the complexity of finding an optimal rectilinear r-partition of a given point set. For arbitrary r the problem is NP-hard, and for constant r the exact algorithm was polynomial but still very slow. Hence, we now turn our attention to heuristics. In the initial experiments on which we report below, the focus is on comparing the various heuristics and investigating the stabbing numbers they achieve as a function of r, for fixed n.

Data sets. We tested our heuristics on four types of point sets–see Fig. 3.5. The *Uniform* data set picks the points uniformly at random from the unit square. For the *Dense* data set we take a *Uniform* data set and square all y-coordinates, so the density increases near the bottom. For the *Line Clusters* data set we first generated a few line segments, whose endpoints are chosen uniformly at random in the unit square. To generate a point in P, we pick one of the line segments randomly, select a position along the line segment at random and add some Gaussian noise. The *Point Clusters* data

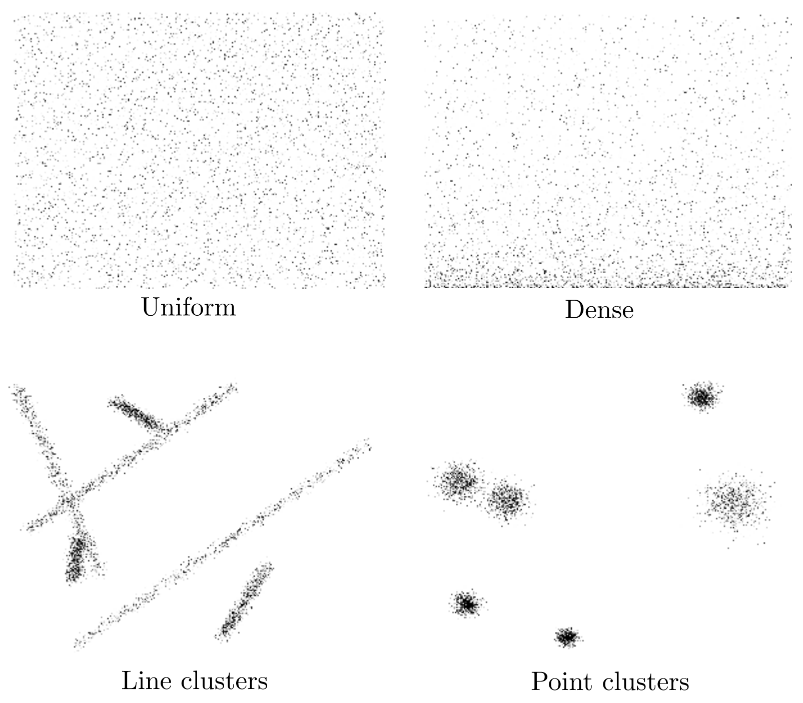


Figure 3.5 The different types of input sets.

set is similar, except that it clusters around points instead of line segments. All sample sets contain $n = 50,000$ points and the reported stabbing numbers are averages over 20 samples.

Next we describe our heuristics. Let P denote the set of points in $\mathbb{R}^2$ for which we want to find a rectilinear r-partition with low stabbing number.

The windmill kd-tree. A natural heuristic is to use a kd-tree [16]: partition the point set P recursively into equal-sized subsets, alternatingly using vertical and horizontal splitting lines, until the number of points in each region drops below $2n/r$. For each region R of the kd-tree subdivision, put the pair $(R \cap P, b_R)$ into the rectilinear r-partition, where b_R is the bounding box of $R \cap P$. Note that this method runs in $O(n \log n)$ time, and gives a stabbing number $O(\sqrt{r})$, which is worst-case optimal. The *windmill kd-tree* is a version of kd-tree in which, for two of the four nodes of depth 2, the splitting line has the same orientation as the splitting line at their parents. This is done in such a way that the subdivision induced by the nodes at level 2 has stabbing number 3 rather than 4–see Fig. 3.6. It turns out that the windmill kd-tree is always at least as good as the regular kd-tree, and often performs significantly better. The results for the uniform Data set are shown in Fig. 3.7 for r ranging from 100 to 2,500 with step size 100. The figure shows that, depending on the value of r, the stabbing number of the kd-tree and the windmill kd-tree are either the same, or the windmill has 25% lower stabbing number. The switch between these two cases occurs exactly at the values of r

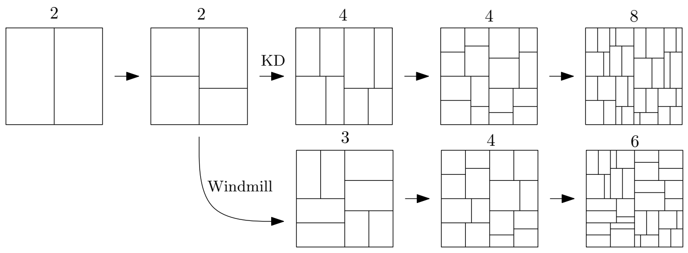


Figure 3.6 A kd-tree, a windmill kd-tree, and their stabbing numbers.

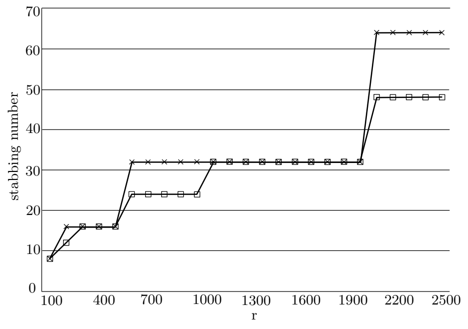


Figure 3.7 Comparison of the kd-tree and the windmill kd-tree.

where the depth switches from even to odd (or vice versa), which is as expected when looking at Fig. 3.6. In the remainder we only compare the windmill kd-tree to the other methods, and ignore the regular kd-tree.

The greedy method. We first compute a set $\mathcal{B}$ of candidate boxes such that $n/2r \leqslant |b_i|$ for each box $b_i \in \mathcal{B}$. Each box has a certain *cost* associated to it. Among these boxes consider the boxes b_j such that $|b_j| \leqslant 2n/r$. We then take the cheapest box b_j, put the pair $(b_j, P \cap b_j)$ into the rectilinear r-partition, and remove the points in b_j from P . Finally, boxes that now contain too few points are discarded from $\mathcal{B}$, the costs of the remaining boxes are updated, and the process is repeated. The method ends when the number of points drops below $2n/r$; these points are then put into a single box (which we allow to contain fewer than $n/2r$ points if needed).

This method can be implemented in various ways, depending on how the set $\mathcal{B}$ and the cost of a box are defined. In our implementation we took m vertical lines with (roughly) $n/(m-1)$ points in between any two consecutive lines, and m horizontal lines in a similar manner. $\mathcal{B}$ then consists of all $O(m^4)$ boxes that can be constructed by taking two vertical and two horizontal lines from these lines. In our experiments we used $m = 50$, because this was the largest value that gave reasonable computation times. The cost of a box b_i is defined as follows. We say that a point $p \in P$ is *in conflict with* b_i if $p \notin b_i$ and the horizontal or the vertical line through p intersects b_i. Let C_i be the set

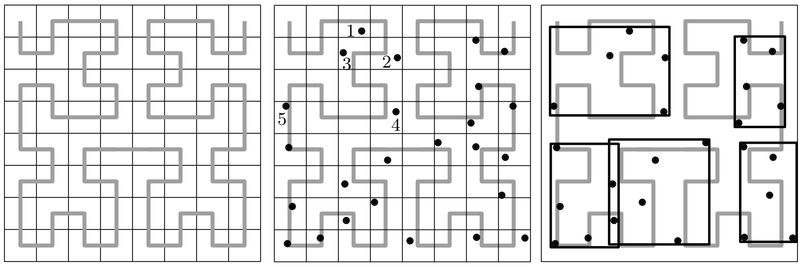


Figure 3.8 A Hilbert curve and its use to generate a rectilinear r-partition.

of points in conflict with b_i. Then the cost of b_i is $|C_i|/|b_i \cap P|$. The idea is that we prefer boxes containing many points and in conflict with few points.

The Hilbert curve. A commonly used approach to construct R-trees is to use a space-filling curve such as a Hilbert curve [49]—see Fig. 3.8(a). We can also use a Hilbert curve to compute a rectilinear r-partition: first, sort the given points according to their position on the Hilbert curve, and then generate the subsets in the rectilinear r-partition by taking consecutive subsets along the Hilbert curve. Since the lowest stabbing number is usually achieved by using as few rectangles as possible, we do this in a greedy manner: put the first $2n/r$ points in the first subset, the next $2n/r$ rectangles in the second subset, and so on—see Fig. 3.8(b).

K-Means. The final method we tested was to compute r clusters using K-means—in particular, we used K-Means++ [14]—and then take the clusters as the subsets in the rectilinear r-partition. Some of the resulting clusters may contain too many or too few points. We solved this by shifting points into neighboring clusters.

3.5.1 Results of the comparisons

Figs. 3.9.(a)-(d) shows the results of our experiments. The clear conclusion is that the windmill kd-tree outperforms all other methods on all data sets. The Hilbert-curve approach always comes in second, except for the *Dense* data set. Note that the windmill and the greedy method give the same results for the *Uniform* data set and the *Dense* data set—which is easily explained, since the rectilinear r-partition computed by these methods only depends on the ranks (their position in the sorted order) of the coordinates, and not on their actual values—while the other two methods perform worse on the *Dense* data set: apparently they do not adapt well to changing density. The windmill and the Hilbert-curve method not only gave the best results, they were also the fastest. Indeed, both methods could easily deal with large data sets. (On inputs with $n = 10,000,000$ and $r = 500$ they only took a few minutes.)

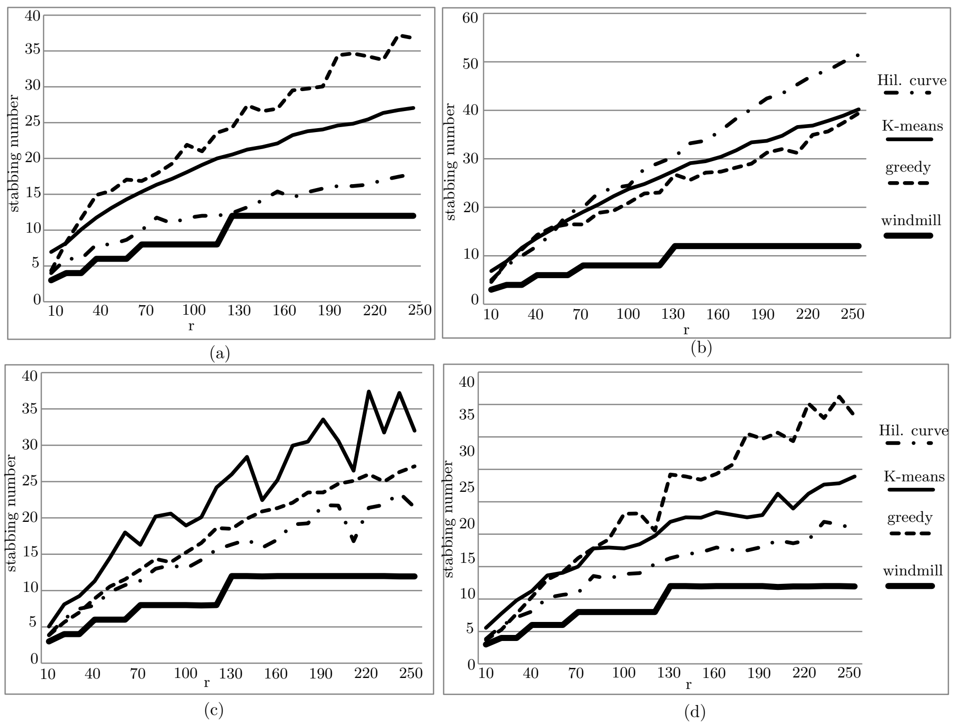


Figure 3.9 The results of the comparison of methods on (a) *uniform* (b) *dense* (c) *line clusters*, and (d) *point cluster* point sets

3.6 Conclusion

We studied the problem of finding optimal rectilinear r-partitions of point sets. On the theoretical side, we proved that the problem is NP-hard when r is part of the input, although it can be solved in polynomial time for constant r. The exact and approximation algorithms for constant r are still unpractically slow, however, so it would be interesting to design faster exact algorithms (or perhaps a practically efficient PTAS).

We also tested several heuristics and concluded that our so-called windmill kd-tree performs the best among them. This immediately leads to the question whether the windmill approach could also lead to R-trees that are practically efficient. This is, in fact, unclear. What we have tried to optimize is the maximum stabbing number of any axis-parallel line. When querying with a rectangular region, however, we are interested in the number of regions intersected by the boundary of the region. First of all, the boundary does not consist of full lines, but of line segments that in practice are possibly small compared to the data set. Secondly, the boundary of the rectangle consists of horizontal and vertical segments. Now, what the windmill does (as compared to a regular kd-tree) is to balance the horizontal and vertical stabbing number, so that the maximum is minimized. The sum of the horizontal and vertical stabbing number in the subdivision does not change, however. So it might be that the windmill approach is good to minimize the worst-case query time for long and skinny queries. This would require further investigation. It would also be interesting to find rectilinear r-partitions whose (maximum or average) stabbing number is optimal with respect to a given set of query boxes, or try to minimize the full partition tree, instead of just one level.

Chapter 4

Decompositions of Rectilinear and Simple Polygons

Chapter summary. Let P be a rectilinear simple polygon. The stabbing number of a partition of P into rectangles is the maximum number of rectangles stabbed by any axis-parallel line segment inside P. We present a 3-approximation algorithm for the problem of finding a partition with minimum stabbing number. It is based on an algorithm that finds an optimal partition for histograms.

We also study Steiner triangulations of a simple (non-rectilinear) polygon P. Here the stabbing number is defined as the maximum number of triangles that can be stabbed by any line segment inside P. We give an $O(1)$-approximation algorithm for the problem of computing a Steiner triangulation with minimum stabbing number.

4.1 Introduction

Background and problem statement Computing decompositions of simple polygons is one of the most fundamental problems in computational geometry. When the polygon at hand is arbitrary then one typically wants a decomposition into triangles, and when the polygon is rectilinear one wants a decomposition into rectangles. Sometimes any such decomposition will do; then one can just compute an arbitrary triangulation or, for rectilinear polygons, a vertical or horizontal decomposition. This can be done in linear time [23]. In other cases one would like the decomposition to have certain additional properties. The property we are interested in, is that the so-called stabbing number—see below for a definition—of the decomposition is low.

Let P be a simple polygon with n vertices, and consider a decomposition of P into triangles. The vertices of the triangles need not all be vertices of P: it is allowed to introduce additional vertices, on the boundary of P and/or in its interior. However, we require the decomposition to be *conforming*: two triangles can only intersect in a common vertex or in a common edge. In other words, no triangle edge can end in the interior of another triangle edge. Such a decomposition is called a *Steiner triangulation*. The triangle vertices that are not vertices of P are called *Steiner vertices*. Note that we allow Steiner vertices on the boundary of P. The *stabbing number* of a segment s with respect to a Steiner triangulation T is the number of triangles from T intersecting s, and the stabbing number of T is the maximum stabbing number of any segment s that lies in the interior of P. Having a Steiner triangulation with low stabbing number is useful for ray shooting in P: after locating the starting point of the ray in the triangulation, one can answer a ray-shooting query by walking through the triangulation until the boundary of P is reached. The time for the walk is bounded by the stabbing number of the triangulation. Hershberger and Suri [50] showed that any simple polygon has a Steiner triangulation with stabbing number $O(\log n)$. This bound is asymptotically tight in the worst case, because any Steiner triangulation of a convex polygon with n vertices has stabbing number $\Omega(\log n)$ [50].

The above concepts can also be studied for rectilinear polygons. We call a decomposition of a simple rectilinear polygon P into rectangles a *rectangular partition*. A rectangular partition need not be conforming: a rectangle edge may end in the interior of another rectangle edge. The stabbing number of a segment s with respect to a rectangular partition R is the number of rectangles intersected by s, and the *(rectilinear) stabbing number* of R is the maximum stabbing number of any axis-parallel segment s in the interior of P. De Berg and Van Kreveld [20] showed that any rectilinear polygon admits a rectangular partition with stabbing number $O(\log n)$. Again, this bound is asymptotically tight in the worst case: any rectangular partition of a staircase polygon with n vertices has stabbing number $\Omega(\log n)$ [20].

The algorithms of Hershberger and Suri [50] and of De Berg and Van Kreveld [20] guarantee partitions with stabbing number $O(\log n)$, which is tight in the worst case. However, some polygons admit a partition with a much smaller stabbing number than $\Theta(\log n)$. Indeed, it is easy to see that some simple polygons admit a Steiner triangulation with stabbing number $O(1)$ and, similarly, that some rectilinear polygons admit a rectangular partition with stabbing number $O(1)$—as an example see Fig. 4.1.

This leads to the topic of our paper: given a polygon P, we wish to compute an *optimal* partition of P, that is, a partition whose stabbing number is minimum over all such partitions of P.

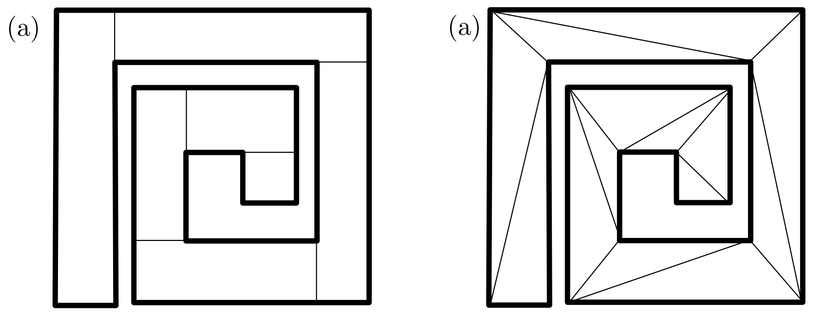


Figure 4.1 (a) The rectangular decomposition with $O(1)$ stabbing number. (b) The triangulation with $O(1)$ stabbing number.

Our results. We present approximation algorithms for this problem, for Steiner triangulations of simple polygons and for rectangular partitions of rectilinear polygons. (We remark that these problems are not known to be NP-complete.) Our main result is a 3-approximation algorithm for the rectilinear case. It is based on an algorithm that computes an optimal rectangular partition for histograms. We also give a linear-time $O(1)$-approximation algorithm for computing a Steiner triangulation of a simple polygon.

Related work. Chazelle *et al.* [24] studied the stabbing number of convex decompositions of polytopes. Later, Tóth [79] proved that any subdivision of d-dimensional space for $d \geqslant 2$ into n convex cells has stabbing number $\Omega((\log n / \log\log n)^{1/(d-1)})$. He also showed that any partition of d-dimensional space ($d \geqslant 2$) into n axis-aligned boxes has rectilinear stabbing number $\Omega(\log^{1/(d-1)} n)$, and presented a partitioning scheme achieving this bound [78]. Considering triangulations of point sets, Agarwal, Aronov and Suri [3] proved that one can triangulate n points in $\mathbb{R}^3$ (using Steiner points) with stabbing number $O(\sqrt{n} \cdot \log n)$.

4.2 Rectangular partitions

Let P be a rectilinear simple polygon with n vertices. We denote the interior of P by $\text{int}(P)$ and its boundary by ∂P. In the remainder of this section, whenever we speak of partitions and stabbing numbers, we always mean rectangular partitions and rectilinear stabbing numbers. We denote the stabbing number of a partition R by $\sigma(R)$. The *horizontal stabbing number* of R, denoted $\sigma_{\text{hor}}(R)$, is defined as the maximum stabbing number of any horizontal segment $s \subset \text{int}(P)$. The *vertical stabbing number*, denoted $\sigma_{\text{vert}}(R)$, is defined similarly. Note that $\sigma(R) = \max(\sigma_{\text{hor}}(R), \sigma_{\text{vert}}(R))$.

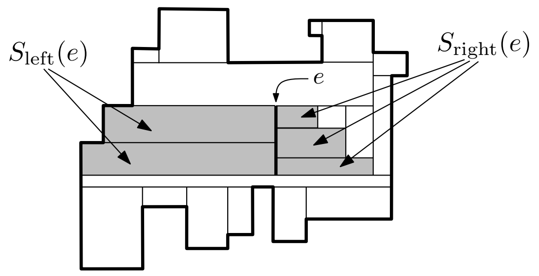


Figure 4.2 A maximal edge that is not anchored, and the set of rectangles bordering it.

4.2.1 The structure of optimal partitions

We start by proving some structural properties of optimal partitions. An optimal partition of P, is the partition with minimum stabbing number among all partitions of P. Consider a partition R of P. The partition is induced by a set $E(R)$ of *maximal edges*. A maximal edge is a segment of maximal length that is a part of the union of one or more rectangle edges in the interior of ∂P. A maximal edge is *anchored* if at least one of its endpoints is a vertex of P. We first show that we can restrict our attention to anchored edges when computing optimal partitions.

Lemma 4.1 *Any rectilinear simple polygon P has an optimal partition R_{opt} in which all maximal edges are anchored.*

Proof. Over all optimal partitions of P, let R_{opt} be one with the minimum number of non-anchored edges and, among those partitions, one with the minimum number of rectangles. Suppose for a contradiction that R_{opt} has a maximal edge e that is not anchored, and assume without loss of generality that e is vertical. We denote the set of rectangles bordering e on the left by $S_{\mathrm{left}}(e)$, and the set of rectangles bordering e on the right by $S_{\mathrm{right}}(e)$; see Figure 4.2. Assume without loss of generality that $|S_{\mathrm{left}}(e)| \leqslant |S_{\mathrm{right}}(e)|$.

Now imagine moving e horizontally to the right until either (i) e hits a vertex of ∂P (and thus becomes anchored), or (ii) one of the rectangles in $S_{\mathrm{right}}(e)$ disappears. Let R denote the resulting partition. We claim that $\sigma(R) \leqslant \sigma(R_{\mathrm{opt}})$. Since R either has fewer non-anchored edges than R_{opt} or the same number of non-anchored edges and fewer rectangles, this contradicts the choice of R_{opt} and thus proves the lemma.

To prove the claim, we observe that the horizontal movement of e clearly does not increase the horizontal stabbing number. The vertical stabbing number does not increase either, since any maximal vertical segment still stabs the same set of rectangles or it stabs the rectangles from $S_{\mathrm{left}}(e)$ instead of those from $S_{\mathrm{right}}(e)$, and $|S_{\mathrm{left}}(e)| \leqslant |S_{\mathrm{right}}(e)|$. The claim follows. □

A *rectilinear binary space partition*, or BSP for short, of a rectilinear polygon P is a rectangular partitioning obtained by the following recursive process. First, P is cut into two subpolygons with an axis-parallel segment inside P, and then the two subpolygons are partitioned recursively in the same way. A BSP is *anchored* if each of its cuts is anchored. One may think that any rectilinear polygon admits an optimal partition that is an anchored BSP. Unfortunately this is not the case: Fig. 4.3 shows an example of a polygon that has a non-BSP partition with stabbing number 3 and for which any BSP has stabbing number at least 4. However, for so-called histograms we can show that there is always an optimal partition that is an anchored BSP. In fact, we show that any anchored partition of a histogram is a BSP. A *(vertical) histogram* is a rectilinear polygon H that has a horizontal edge seeing every point $q \in \text{int}(H)$. Here we say that an edge e *sees* a point $q \in H$ if there is an axis-parallel segment s connecting e to q that is completely lying inside $\text{int}(H)$ except possibly for its endpoints. We call the horizontal edge that sees all points in the histogram H the *base* of the histogram and denote it by $\text{base}(H)$. A horizontal histogram is defined similarly: it has a vertical base that can see any point in the interior of the polygon.

Lemma 4.2 *Any anchored partition of a histogram is a* BSP.

Proof. Let R be an anchored partition of a histogram H. We will prove the statement by induction on n, the number of vertices of H. For $n = 4$ the statement is trivially true, so assume $n > 4$.

If $\text{base}(H)$ has only a single rectangle $r \in R$ adjacent to it, then $H \setminus r$ consists of one or more subhistograms, which are separated from r by horizontal BSP cuts. The partitionings inside these subhistograms induced by R are anchored and, hence, they are BSPs by induction. If there is more than one rectangle adjacent to $\text{base}(H)$, then there is a maximal edge ending in the interior of $\text{base}(H)$ with a vertex of P as its other endpoint. This is a BSP cut, which partitions H into two subhistograms. The partitions inside these subhistograms induced by R are BSPs by induction. □

4.2.2 A 3-approximation algorithm

Next we present our 3-approximation algorithm for the problem of finding an optimal rectangular partition of a given rectilinear polygon P. The algorithm constructs the partition in two steps. In the first step we split P into a set of histograms such that any axis-parallel segment inside P stabs at most three histograms. This can be done in $O(n)$ time [59]—see also [20]. The approach is as follows, we pick an arbitrary edge e of P, and make the histogram $H(e)$ inside P having e as its

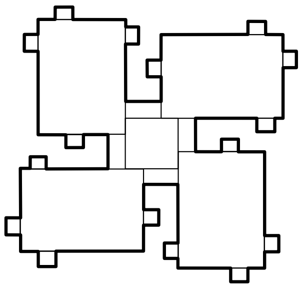


Figure 4.3 A rectilinear polygon for which the optimal rectangular partition is not a BSP.

base. The histogram $H(e)$ has a number of windows to the remaining part of P, we repeat the process using each of these windows as the base of a new histogram. In the second step we compute an optimal rectangular partition for each resulting histogram H. By proving that this can be done in polynomial time, we have the following theorem.

Theorem 4.3 *Let P be a rectilinear simple polygon with n vertices. Then we can compute a rectangular partition of P with stabbing number at most $3 \cdot OPT$ in polynomial time, where OPT is the minimum stabbing number of any rectangular partition of P.*

Proof. Consider a partition of P into a set of histograms using the above approach. Let us denote the set of histograms in the partition by $H = \{H_1, \ldots, H_i\}$. Let OPT_j denote the stabbing number of an optimal partition of a histogram $H_j \in H$. We claim that $OPT_j \leqslant OPT$ for each $H_j \in H$. Indeed, let R_{opt} denote an optimal partition of P. A property of the partitioning into histograms [59, 20] is that it only adds segments that "completely cut through P", that is, segments both of whose endpoints are on ∂P. This implies that any rectangle $r \in R_{\text{opt}}$ is cut into rectangles by the partition (if it is cut at all)—no other shape arise. This means that the part of R_{opt} inside H_j is a valid rectangular partition of H_j. This implies $OPT_j \leqslant OPT$. Any axis-aligned segment $s \in \text{int}(P)$ intersects at most three histograms of H, and we have $OPT_j \leqslant OPT$ for each histogram. Thus, finding OPT_j in polynomial time for each H_j results in a partition of P with stabbing number at most $3 \cdot OPT$. □

4.2.3 An optimal algorithm for histograms

Let H be a histogram. With a slight abuse of notation, we use n to denote the number of vertices of H. We assume without loss of generality that H is a vertical histogram lying above its base. By Lemmas 4.1 and 4.2, the histogram H admits an optimal partition that is an anchored BSP. We will need the following additional properties.

Lemma 4.4 *There is an optimal partition R_{opt} for H that is an anchored* BSP *and such that, for every rectangle $r \in R_{\text{opt}}$, we have*

(i) the bottom edge of r is contained in either the top edge of a single rectangle $r' \in R_{opt}$ or in $\text{base}(H)$, and

(ii) the top edge of r contains an edge of H.

Proof. By Lemmas 4.1 and 4.2 there is an optimal partition R_{opt} that is an anchored BSP. We claim that R_{opt} already has property (i) and that we can convert it into a partition having property (ii) as well without increasing its stabbing number.

Let e_{bot} be the bottom edge of some rectangle r in R_{opt}. Because H is a histogram, e_{bot} is either contained in $\text{base}(H)$ or in the union of some top edges of other rectangles in the partition. If the former is the case then r has property (i), so assume e_{bot} is contained in the union of some top edges of other rectangles. If e_{bot} overlapped with more than a single top edge, then there would be a vertical edge e in the partition whose top endpoint lies on e_{bot}. But because H lies above its base, the maximal edge containing e cannot have a vertex of H as bottom endpoint, so e would not be anchored. Thus e_{bot} must be contained in the top edge of a single rectangle, proving that r has property (i).

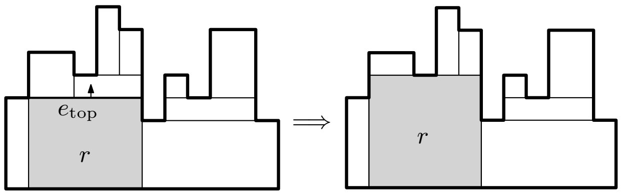


Figure 4.4 Illustration for the proof of Lemma 4.4.

Next we convert $R_{\rm opt}$ as follows. For each rectangle r that does not yet have property (ii) we push its top edge $e_{\rm top}$ upward until it hits ∂H; see Figure 4.4. After doing this, the edge has property (ii). Because of property (i), there are no rectangles whose bottom edge overlaps only partially with $e_{\rm top}$ and so the partition remains a partition into rectangles. The stabbing number is not increased by this operation, so after pushing up all edges we have an optimal partition with properties (i) and (ii).

Note that the partition is still an anchored BSP after pushing all edges upward. Indeed, any vertical maximal edge is anchored at its top vertex, so it remains anchored (unless it completely disappears because of the pushing operations). Every horizontal maximal edge must also be anchored otherwise it would have been pushed up further. Thus the partition is anchored and by Lemma 4.2 we conclude that it is still a BSP. □

In the sequel we only consider partitions with properties (i) and (ii) from Lemma 4.4 (but not all partitions are anchored).

Reduction to a decision problem Our algorithm will do a binary search for the smallest value k such that H admits a partition with stabbing number k. Since there is always a partition with stabbing number at most $2\log_2 n$ [20], the binary search needs $O(\log\log n)$ steps. It remains to describe our decision algorithm, called HISTOGRAMPARTITION(H,k), which decides whether H has a partition with stabbing number at most k.

Canonical chords Define a *chord* of H to be a maximal horizontal segment contained in the interior of H except for its endpoints. A chord s partitions H into two parts. The part above s is a histogram, which we denote by $H(s)$. Note that any partition R of H induces a partition of $H(s)$; this partition is denoted by $R(s)$. Now consider a partition of H obtained by adding a chord from each vertex of H for which this is possible; this partition is sometimes called the *horizontal decomposition* of H. We call the resulting set of chords the *canonical chords* of H—see Figure 4.5(a). It will be convenient to also treat $\mathrm{base}(H)$ as a canonical chord.

The basic idea behind the algorithm is to treat the canonical chords from top to bottom. Now consider a canonical chord s_i with, say, two canonical chords s_j and s_ℓ immediately above it. Here we say that s_i is *immediately above* s_j, if we can connect s_i to s_j with a vertical segment that does not cross any other canonical chord. One may hope that, if we have optimal partitions for $H(s_j)$ and $H(s_\ell)$, then we can somehow "extend" these to an optimal partition for $H(s_i)$. Unfortunately this is not the case, since an optimal partition need not be composed of optimal subpartitions. The next idea is to

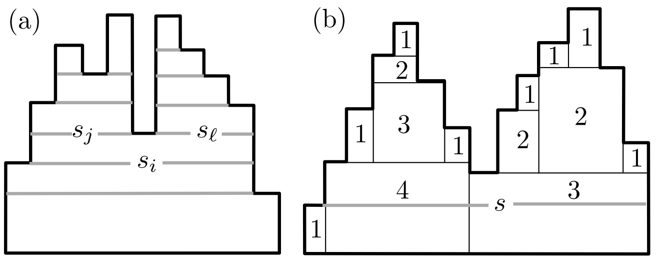


Figure 4.5 (a) Partitioning a histogram using canonical chords. (b) A partition with a unimodal labeling. The label sequence of the chord s is $4, 3$ and the label sequence of the base is $1, 4, 3$.

compute all possible partitions for $H(s_i)$. These can be obtained by considering all combinations of a possible partition for $H(s_j)$ and a possible partition for $H(s_\ell)$. Implementing this idea naively would lead to an exponential-time algorithm, however. Next we show how to compute a subset of all possible partitions that has only polynomial size and is still guaranteed to contain an optimal partition.

Labeled partitions and label sequences We first introduce some notation and terminology. Let R be any partition of H (satisfying the properties (i) and (ii) in Lemma 4.4). We say that a rectangle $r \in R$ is *on top of* a rectangle $r' \in R$ if the bottom edge of r is contained in the top edge of r'. When the bottom edge of r is contained in $\text{base}(H)$ then we say that r is on top of the base. A *labeling* of R assigns a positive integer label $\lambda(r)$ to each rectangle $r \in R$. We say that a labeled partition is *valid* (with respect to the stabbing number k we are aiming for) if it satisfies the following conditions:

- if r is on top of r' then $\lambda(r) < \lambda(r')$;
- the vertical stabbing number of R equals the maximum label of any rectangle $r \in R$;
- the stabbing number of R is at most k.

Observe that the first two conditions together imply that the stabbing number of R is equal to the maximum label assigned to any rectangle on top of $\text{base}(H)$. Also note that any partition with stabbing number k has a valid labeling: for example, one can set $\lambda(r)$ to be equal to the maximum number of rectangles that can be stabbed by a vertical segment whose lower endpoint lies inside r. We will use the labelings to decide which partitions can be disregarded and which ones we need to keep.

Consider a chord s of H. We define the *label sequence of s* with respect to a labeled partition R as the sequence of labels of the rectangles crossed by s, ordered from left to right; here we say that s crosses a rectangle r if s intersects $\text{int}(r)$ or the bottom edge of r. We denote this sequence by $\Sigma(s, R)$; see Figure 4.5(b) for an example. We say that a label sequence is *valid* if it consists of at most k labels and the maximum label is at most k. Note that a labeled partition is valid if and only if the label sequence of each of its canonical chords is valid. A label sequence $\lambda_1, \ldots, \lambda_t$ is called *unimodal* if there is an index i such that $\lambda_1 \leqslant \cdots \leqslant \lambda_i$ and $\lambda_i \geqslant \cdots \geqslant \lambda_t$. A labeling of a partition is unimodal if the label sequence of every chord is unimodal. A given label sequence can be made

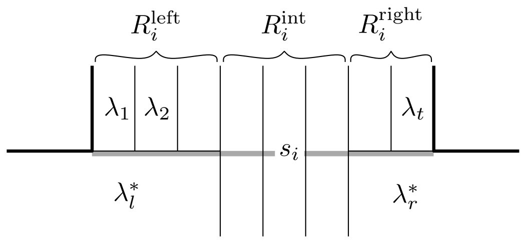


Figure 4.6 Illustration for the proof of Lemma 4.5.

unimodal using the following simple procedure.

```
MAKEUNIMODAL($\Sigma$)
Let $\Sigma = \lambda_1, \ldots, \lambda_t$, and let $\lambda_{i^*}$ be a maximum label in the sequence. For each $i < i^*$
set $\lambda_i := \max_{j \leqslant i} \lambda_j$, and for each $i > i^*$ set $\lambda_i := \max_{j \geqslant i} \lambda_j$.
```

The next lemma states that we can make the label sequences of all canonical chords unimodal, and still keep a valid sequence.

Lemma 4.5 *Any anchored partition of H of stabbing number at most k admits a valid unimodal labeling.*

Proof. Let R be a partition of H. We first create a valid labeling for R as explained earlier: we set the label of a rectangle r to be equal to the maximum number of rectangles that can be stabbed by a vertical segment whose lower endpoint lies inside r. Next we turn this labeling into a valid unimodal labeling using the following process. Let $s_1, \ldots, s_m$ be the set of all canonical chords, sorted from bottom to top (that is, by increasing y-coordinates) and breaking ties arbitrarily. For each chord s_i in order, we apply MAKEUNIMODAL to the label sequence $\Sigma(s_i, R)$. We claim that this process satisfies the following invariant: after handling s_i (i) the labeling is still valid, and (ii) the label sequence of any chord $s_{i'}$ with $i' < i$ is unimodal.

To prove that the invariant is maintained, consider a chord s_i. Let $r_1, \ldots, r_t$ be the rectangles ending on or properly intersecting s_i. We denote the current label of a rectangle r_ℓ by λ_ℓ, and the label after applying MAKEUNIMODAL to s_i by $\overline{\lambda}_\ell$. Let $\Sigma_i := \lambda_1, \ldots, \lambda_t$. Let R_i^{int} be the set of rectangles properly intersecting s_i. The rectangles in R_i^{int} are consecutive, because the partition is anchored. Let R_i^{left} and R_i^{right} be the sets of rectangles to the left and right of R_i^{int}, respectively—see Figure 4.6. Let Σ_i^{int} be the subsequence of Σ_i consisting of the labels of the rectangles in R_i^{int}. Note that Σ_i^{int} is already unimodal before s_i is handled, because these rectangles were handled in a previous step.

We claim that the labels in Σ_i^{int} are not modified when we make Σ_i unimodal. To prove this claim, note that if the label of some $r_\ell \in R_i^{\text{int}}$ has been changed, then there must be labels $\lambda_{\ell'}$ and $\lambda_{\ell''}$, with $\ell' < \ell < \ell''$, such that $\lambda_\ell < \min(\lambda_{\ell'}, \lambda_{\ell''})$. We cannot have $\lambda_{\ell'}, \lambda_{\ell''} \in \Sigma_i^{\text{int}}$, because that would mean that Σ_i^{int} was not unimodal before s_i was handled. Let λ_l^* and λ_r^* be the labels of the rectangles below the rectangles in R_i^{left} and R_i^{right}, respectively. It follows from the definition of labeling that for any rectangle $r_j \in R_i^{\text{left}}$ we have $\lambda_j < \lambda_l^*$. Similarly, for any rectangle $r_k \in R_i^{\text{right}}$ we have $\lambda_k < \lambda_r^*$. Since the labeling below s_i was unimodal before handling s_i, we have $\lambda_l^* \leqslant \lambda_\ell$ or

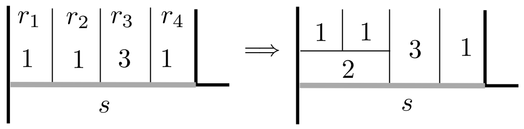


Figure 4.7 Merging two rectangles.

$\lambda_r^* \leqslant \lambda_\ell$, or both. If both inequalities hold, then the claim holds since at least one of (the rectangles corresponding to) $\lambda_{\ell'}$ or $\lambda_{\ell''}$ belongs to one of R_i^{left} or R_i^{right} and is smaller than λ_ℓ. Otherwise, assume without loss of generality that $\lambda_l^* \leqslant \lambda_\ell$ and $\lambda_\ell \leqslant \lambda_r^*$. If $\lambda_{\ell'} \in R_i^{\text{left}}$ then since $\lambda_{\ell'} < \lambda_l^*$ we have $\lambda_{\ell'} < \lambda_l$ and the claim holds. When $\lambda_{\ell'}$ is not in R_i^{left}, it should be in R_i^{int}. Since $\lambda_\ell \leqslant \lambda_r^*$, and from the fact that the labeling below s_i before modification was unimodal, we can conclude that $\lambda_{\ell'} \leqslant \lambda_\ell$ and the claim holds.

Since the labels in Σ_i^{int} are not modified, we clearly have property (ii): the label sequence of any chord $s_{i'}$ with $i' < i$ is still unimodal. We also have property (i). Indeed, if a rectangle $r_\ell \in R_i^{\text{left}}$ gets a new label, then there is a label $\lambda_{\ell'} \in \Sigma_i^{\text{left}}$ to the left of it that is at least as large as the new label $\overline{\lambda}_\ell$. But this implies that the label of the rectangle below all rectangles in R_i^{left} is larger than this label. Hence, the modification of the label of r_ℓ does not make the labeling invalid. A similar argument shows that the modification of the labels of rectangles in R_i^{right} does not make the labeling invalid. □

Dominated and non-dominated sequences Next we explain how the labelings help us decide which partitions can safely be discarded. Consider an algorithm that handles the chords from top to bottom, and suppose that the algorithm reaches a chord s. Let R_1 and R_2 be two labeled partitions of $H(s)$. Suppose that $\Sigma(s, R_1)$ is a subsequence of $\Sigma(s, R_2)$. Then there is no need to keep R_2: both partitions have stabbing number at most k so far, and it is easy to see that if we can complete R_2 to a partition with stabbing number k of the full histogram H then we can do so with R_1 as well. As another example in which we can ignore one of the two partitions, suppose $\Sigma(s, R_1) = 1, 1, 3, 1$ and $\Sigma(s, R_2) = 2, 3, 1$, and let $r_1, \ldots, r_4$ be the four rectangles in R_1 reaching the chord s. Then we could have *merged* r_1 and r_2 just before reaching s, that is, we could have terminated r_1 and r_2 and start a new rectangle with label "2"—see Figure 4.7. The new subsequence is then 2,3,1. This is a subsequence of $\Sigma(s, R_2)$—in fact, it happens to be equal to $\Sigma(s, R_2)$— so we can disregard R_2.

We now make this idea formal. We say that $\Sigma(s, R_1)$ *dominates* $\Sigma(s, R_2)$ if we can obtain a sequence $\Sigma(s, R_1')$ that is a subsequence of $\Sigma(s, R_2)$ by applying the following *merging operation* zero or more times to $\Sigma(s, R_1)$:

- Replace a subsequence $\lambda_i, \ldots, \lambda_j$ of $\Sigma(s, R_1)$ by the single label "max$(\lambda_i, \ldots, \lambda_j)$ + 1". Note that we can have $i = j$; in this case the operation just adds 1 to the label λ_i.

Intuitively, if a label sequence dominates another label sequence, then the first sequence has postponed some merging operations that we can still do later on. Thus there is no need to maintain partitions with dominated label sequences. (Note that postponing a merging operation implies that the resulting partition may not be anchored anymore. This is not a problem; it just means that in the algorithm presented below, we cannot restrict ourselves to anchored partitions.) The next lemma gives a bound on the number of label sequences that we need to maintain in the worst case.

Lemma 4.6 *Let $\mathcal{S}$ be any collection of valid unimodal sequences such that no sequence in $\mathcal{S}$ dominates any other sequence in $\mathcal{S}$. Then $|\mathcal{S}| = O(2^{3k/2}/\sqrt{k})$.*

Proof. For a unimodal sequence Σ, let $\mathrm{inc}(\Sigma)$ denote the largest (not necessarily strictly) increasing subsequence of Σ. Here by a subsequence we mean a continues subsequence. Since Σ is unimodal, $\mathrm{inc}(\Sigma)$ is the prefix of Σ ending at the rightmost occurrence of the maximum value in the sequence. Similarly, let $\mathrm{dec}(\Sigma)$ denote the largest decreasing subsequence of Σ. Define $\mathcal{S}_{\mathrm{inc}} \subset \mathcal{S}$ to be the set of sequences for which $|\mathrm{inc}(\Sigma)|$, the length of $\mathrm{inc}(\Sigma)$, is at most $|\mathrm{dec}(\Sigma)|$. We now bound the number of sequences in $\mathcal{S}_{\mathrm{inc}}$; the other sequences (for which the reverse holds) can be counted in the same way.

Consider two different unimodal sequences Σ, Σ' such that $\mathrm{inc}(\Sigma) = \mathrm{inc}(\Sigma')$. We claim that either Σ dominates Σ' or vice versa. Indeed, traverse $\Sigma \setminus \mathrm{inc}(\Sigma)$ and $\Sigma' \setminus \mathrm{inc}(\Sigma')$ simultaneously from left to right until the first position where they differ. Suppose the label of Σ is smaller than the label of Σ' at this position (or Σ has ended). Then it is easy to see that Σ dominates Σ'. We conclude that $\mathcal{S}_{\mathrm{inc}}$ does not contain two sequences Σ, Σ' with $\mathrm{inc}(\Sigma) = \mathrm{inc}(\Sigma')$. Hence, the number of label sequences in $\mathcal{S}_{\mathrm{inc}}$ is bounded by the number of different (non-strictly) increasing subsequences in $\mathcal{S}_{\mathrm{inc}}$.

We conclude that the number of label sequences in $\mathcal{S}_{\mathrm{inc}}$ is bounded by the number of different (non-strictly) increasing sequences of length at most k and consisting of the integers $1, \ldots, k$. This is equivalent to the number of increasing sequences of length exactly k and consisting of integers $0, \ldots, k$. This, in turn, is equivalent to the number of ways in which one can place k balls into $k+1$ labeled bins. Now we note that the maximum label of any two sequences in $\mathcal{S}_{\mathrm{inc}}$ is the same—otherwise the sequence with the smaller maximum label would dominate the other sequence. Moreover, the number of times the maximum label, M, occurs can differ by at most one. Since, otherwise the sequence with fewer maximum labels dominates the other sequence. Suppose that M occurs x or $x+1$ times in any sequence. We now only consider the sequences where M occurs x times; to obtain the final bound we just have to multiply by two. Then, in terms of the balls and bins metaphor, we only have to look at sequences that all put exactly the same number of balls into one of the bins. But that means we can ignore these balls (and this bin). Since we have $|\mathrm{inc}(\Sigma)| \leqslant |\mathrm{dec}(\Sigma)|$ for each of the subsequences in $\mathcal{S}_{\mathrm{inc}}$, this means that we have to consider at most $\lfloor k/2 \rfloor$ balls. The number of ways in which one can put $\lfloor k/2 \rfloor$ into k bins is $\binom{\lfloor \frac{3k-3}{2} \rfloor}{k} = O(2^{3k/2}/\sqrt{k})$. $\square$

The algorithm We can now describe our decision algorithm.

```
HISTOGRAMPARTITION($H, k$)
  1. Compute the set of canonical chords of $H$ and sort the chords by decreasing $y$-coordinate.
  2. For each chord $s$ in order, compute a collection $\mathcal{R}(s)$ of labeled partitions of $H(s)$, as follows.
       (i)  If $H(s)$ is a rectangle then set $\mathcal{R}(s) := \{H(s)\}$.
       (ii) Otherwise $s$ has one or more chords $s_1, \ldots, s_t$ immediately above it—see Figure 4.8.
            We compute all valid unimodal partitions of $H(s)$ that can be obtained from any com-
            bination of partitions in $\mathcal{R}(s_1), \ldots, \mathcal{R}(s_t)$ and whose label sequence is not dominated
```

by the label sequence of any other such partition. (How this is done will be explained below.) Let $\mathcal{R}(s)$ be the set of all computed partitions. If $\mathcal{R}(s)$ is empty, then report that no partition with stabbing number k exists for H, and exit.

3. Return any partition in $\mathcal{R}(\text{base}(H))$.

Next we explain how Step 2(ii) is performed. We assume that $t > 1$, that is, that s has several chords immediately above it; the case $t = 1$ can be handled in a similar (but much simpler) way. In the sequel, we identify each partition with its label sequence and only talk about label sequences. Note that the operations we perform on the label sequences can be easily converted into the corresponding operations on the partitions. For every pair of labeled partitions $R_1 \in \mathcal{R}(s_1), R_t \in \mathcal{R}(s_t)$ we proceed as follows.

(a) For each $1 < i < t$, consider the set $\mathcal{R}(s_i)$. Note that the label sequences in $\mathcal{R}(s_i)$ all have the same maximum value, M_i. This is true because a label sequence dominates any label sequence with larger maximum value. (The number of times the maximum label occurs can differ by at most one.) We pick an arbitrary label sequence $\Sigma_i \in \mathcal{R}(s_i)$ for which M_i occurs the minimum number of times.

(b) We now have, besides the partitions $\Sigma_1 \in \mathcal{R}(s_1)$ and $\Sigma_t \in \mathcal{R}(s_t)$, picked a partition Σ_i from each $\mathcal{R}(s_i)$ with $1 < i < t$. Let $\overline{\Sigma}$ be the label sequence obtained by concatenating the sequences Σ_i in order, inserting a label "1" for any horizontal histogram edge incident to a chord s_i, as illustrated in Figure 4.8. The labels "1" correspond to new rectangles that we can start, whose top edge is the given histogram edge. We then make $\overline{\Sigma}$ unimodal. This is done using a variant of the procedure MAKEUNIMODAL explained earlier: the difference is that if we give several consecutive labels the same value, then we merge them into a single new label—see Figure 4.8.

(c) If the number of labels in $\overline{\Sigma}$ is at most k, then we put $\overline{\Sigma}$ into $\mathcal{R}(s)$. Otherwise $\overline{\Sigma}$ is invalid because it contains too many labels, and we have to merge some rectangles to obtain a shorter sequence. This is done as follows. Suppose that $\overline{\Sigma}$ contains $k + x$ labels $\lambda_1, \ldots, \lambda_{k+x}$. Then we have to get rid of x labels by merging. Let $x_{\text{left}}, x_{\text{right}}$ be integers such that $x_{\text{left}} + x_{\text{right}} = x + 2$ and both $x_{\text{left}}, x_{\text{right}}$ are non-zero, or $x_{\text{left}} + x_{\text{right}} = x + 1$ and one of $x_{\text{left}}, x_{\text{right}}$ is zero. We merge x_{left} labels from the left into one new label, and x_{right} labels from the right into one, as in Figure 4.8. In other words, on the left side we replace $\lambda_1, \ldots, \lambda_{x_{\text{left}}}$ by a single new label $\lambda_{x_{\text{left}}} + 1$ (and similarly on the right). If there are some labels immediately to the right of $\lambda_{x_{\text{left}}}$ with the same value as $\lambda_{x_{\text{left}}}$, then we include them into the merging process. (We can do this for free, since it reduces the number of labels, without increasing the value of the new label.) If this merging process yields a new label whose value is more than the previous maximum label value, then we simply merge the entire sequence into a single new label. If the value of this label is $k + 1$, then we discard the sequence.

After having applied the above steps to every pair $R_1 \in \mathcal{R}(s_1), R_t \in \mathcal{R}(s_t)$, we remove from $\mathcal{R}(s)$ all partitions with a label sequence that is dominated by the sequence of some other partition. How this is done, is explained below. The next lemma shows the correctness of the decision algorithm.

Lemma 4.7 HISTOGRAMPARTITION(H, k) *returns a partition of H with stabbing number at most k if it exists.*

Proof. We will prove that the algorithm maintains the following invariant. Let R^* be an anchored partition of H of stabbing number at most k with a valid unimodal labeling.

Invariant: After handling the chord s, the set $\mathcal{R}(s)$ contains at least one label sequence Σ that dominates $\Sigma(s, R^*)$.

Let $\Sigma^* := \Sigma(s, R^*)$ and $\Sigma_i^* := \Sigma(s_i, R^*)$. Consider the handling of s in Step 2. If we are in case (i) then the invariant obviously holds, so now suppose we are in case (ii). Let $s_1, \ldots, s_t$ be the chords immediately above s. By the invariant, each set $\mathcal{R}(s_i)$ contains a label sequence Σ_i dominating Σ_i^*. We argue that this implies that we can generate a label sequence Σ from $\Sigma_1, \ldots, \Sigma_t$ that dominates Σ^*, and that our algorithm actually finds such a label sequence.

The former statement is easy to see. By definition we can apply some merging operations to each Σ_i to turn it into a subsequence of Σ_i^*—in fact, we may also have to increase some labels, but this does not change the argument—and then we can simply "copy" the operations that turn $\Sigma_1^*, \ldots, \Sigma_t^*$ into Σ^*, thus turning $\Sigma_1, \ldots, \Sigma_t$ into a sequence dominating Σ^*. To argue that our algorithm actually finds a dominating sequence Σ—that is, that Step 2(ii) is implemented correctly—we have to argue a bit more carefully.

Suppose that in Step (b) we replace several labels by a single label λ. When these labels are between two other labels with value at least λ, we cannot avoid increasing their values to at least λ, and the best we can do is to replace all of them by a single label λ. Consider the sequence $\overline{\Sigma}$ made in step (b), that was generated from a pair Σ_1, Σ_t such that Σ_1 dominates Σ_1^* and Σ_t dominates Σ_t^*. Recall that for making $\overline{\Sigma}$ we picked, for $1 < i < t$, label sequences $\Sigma_i \in \mathcal{R}(s_i)$ for which the label with maximum value occurs the minimum number of times. Together with the fact that Σ_1 dominates Σ_1^* and Σ_t dominates Σ_t^*, and that, as just explained, we replace labels in an optimal way, this implies that $\overline{\Sigma}$ dominates Σ^*.

Now suppose that the number of labels in $\overline{\Sigma}$ is more than k. If the maximum label value in Σ^* is higher than the maximum label value in $\overline{\Sigma}$, then Step (c) will clearly result in a sequence dominating Σ^*. Otherwise the maximum label values in Σ^* and $\overline{\Sigma}$ are the same. Consider a sequence of replacement operations that turns $\overline{\Sigma}$ into a subsequence of Σ^*; such a sequence of operations exists because $\overline{\Sigma}$ dominates Σ^*. Let j be such that $\lambda_j \in \overline{\Sigma}$ has the maximum value. Let M_{left} (and M_{right})

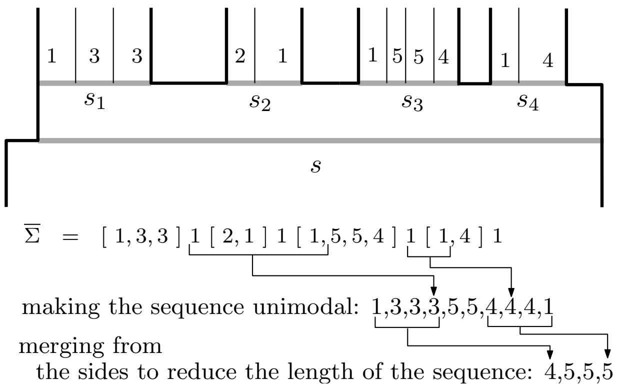

1 3 3 2 1 1 5 5 4 1 4
s_1
s_2
s_3
s_4
s
$\overline{\Sigma}$ = [1,3,3] 1 [2,1] 1 [1,5,5,4] 1 [1,4] 1
making the sequence unimodal: 1,3,3,3,5,5,4,4,4,1
merging from
the sides to reduce the length of the sequence: 4,5,5,5

Figure 4.8 A chord s, the chords immediately above it, and the label sequence $\overline{\Sigma}$ defined by the label sequences of the chords.

$$\begin{array}{llll}
\text{(a)} & \Sigma_i = \langle 1,2,4,4,1,1\rangle & \Sigma_j = \langle 1,3,4,4,2\rangle & \\
 & \Sigma_i^1 = \langle 1,2\rangle & \Sigma_j^1 = \langle 1,3\rangle & \\
 & \Sigma_i^2 = \langle 1,1\rangle & \Sigma_j^2 = \langle 2\rangle & \\
\text{(b)} & \Sigma_i = \langle 1,2,4,4,2,1\rangle & \Sigma_j = \langle 1,1,4,4,4,3\rangle & \\
 & \Sigma_i^1 = \langle 1,2\rangle & \Sigma_j^{1,1} = \langle 1,1,4\rangle & \Sigma_j^{2,1} = \langle 3\rangle \\
 & \Sigma_i^2 = \langle 2,1\rangle & \Sigma_j^{1,2} = \langle 1,1\rangle & \Sigma_j^{2,2} = \langle 4,3\rangle
\end{array}$$

Figure 4.9 (a) Two sequences Σ_i and Σ_j in which $n_i = n_j$ and the four subsequences made from them. (b) Two sequences Σ_i and Σ_j in which $n_i = n_j + 1$ and the six subsequences made from them.

be the maximum value of any label to the left (resp. right) of λ_j that is involved in a replacement operation. Then simply replacing all x_{left} labels to the left of λ_j whose value is at most M_{left} by a single label $M_{\text{left}} + 1$, and replacing all x_{right} labels to the right of λ_j whose value is at most M_{right} by a single label $M_{\text{right}} + 1$, leads to a sequence that dominates Σ^* and has at most k labels. Step (c) must consider some combination of merging x'_{left} labels from the left and x'_{right} labels from the right with $x'_{\text{left}} \leqslant x_{\text{left}}$ and $x_{\text{right}} \leqslant x_{\text{right}}$, and this will then result in a sequence dominating Σ^*. □

The following lemma explains an approach for removing all partitions with a label sequence that is dominated by another partition in the set.

Lemma 4.8 *Let $\mathcal{R}$ denote a set of unimodal sequences, and let n be the total number of elements of the sequences in $\mathcal{R}$. Then we can remove all sequences from $\mathcal{R}$ that are dominated by another sequence in $\mathcal{R}$ in time $O(n \log n)$.*

Proof. Let M_i denote the maximum label in a sequence Σ_i, and let n_i be the number of times it occurs in Σ_i. Define $M_{\min} := \min\{M_i : \Sigma_i \in \mathcal{R}\}$ and $n_{\min} := \min\{n_i : \Sigma_i \in \mathcal{R}\}$. As mentioned above, the non-dominated sequences in $\mathcal{R}$ all have $M_i = M_{\min}$ and $n_i \leqslant n_{\min} + 1$. Thus, first we remove all the sequences with $M_i > M_{\min}$ and the sequences with $n_i > n_{\min} + 1$. This can be done in $O(n)$ time.

We partition the set of remaining sequences into two subsets: a subset $\mathcal{R}(n_{\min})$ containing all Σ_i with $n_i = n_{\min}$, and a subset $\mathcal{R}(n_{\min} + 1)$ containing all Σ_i with $n_i = n_{\min} + 1$. Note that a sequence in $\mathcal{R}(n_{\min} + 1)$ can never dominate a sequence in $\mathcal{R}(n_{\min})$. Hence, our task is now to (i) remove all sequences from $\mathcal{R}(n_{\min})$ that are dominated by another sequence in $\mathcal{R}(n_{\min})$, and (ii) remove all sequences from $\mathcal{R}(n_{\min} + 1)$ that are dominated by another sequence in $\mathcal{R}(n_{\min} + 1)$, and (iii) remove all sequences from $\mathcal{R}(n_{\min} + 1)$ that are dominated by a sequence in $\mathcal{R}(n_{\min})$.

Task (i) is performed as follows. Each sequence $\Sigma_i \in \mathcal{R}(n_{\min})$ is divided into two subsequences by removing all labels of maximum value from it. We denote the resulting sequences by Σ_i^1 and Σ_i^2—see Figure 4.9(a). Now consider two sequences $\Sigma_i, \Sigma_j \in \mathcal{R}(n_{\min})$. The crucial observation is that Σ_i dominates Σ_j if and only if Σ_i^1 dominates Σ_j^1 and Σ_i^2 dominates Σ_j^2. Let $\mathcal{R}^1(n_{\min}) := \{\Sigma_i^1 : \Sigma_i \in \mathcal{R}(n_{\min})\}$ and $\mathcal{R}^2(n_{\min}) := \{\Sigma_i^2 : \Sigma_i \in \mathcal{R}(n_{\min})\}$, where we make sure all sequences in $\mathcal{R}^1(n_{\min})$ and $\mathcal{R}^1(n_{\min})$ have length exactly k by adding extra zeros; for $\mathcal{R}^1(n_{\min})$ these are added at the beginning of the sequence and for and $\mathcal{R}^1(n_{\min})$ they are added at the end. Note that all sequences in $\mathcal{R}^1(n_{\min})$ are non-decreasing, and all sequences in $\mathcal{R}^2(n_{\min})$ are non-increasing. For two sequences $\Sigma_i^1, \Sigma_j^1 \in \mathcal{R}^1(n_{\min})$ we write $\Sigma_i^1 \prec \Sigma_j^1$ if Σ_i^1 dominates Σ_j^1. Since the sequence in $\mathcal{R}^1(n_{\min})$ are non-decreasing this is equivalent to the following. Let $\Sigma_i^1 = \lambda_1, \ldots, \lambda_k$ and $\Sigma_j^1 = \lambda_1', \ldots, \lambda_k'$. We have $\Sigma_i^1 \prec \Sigma_j^1$ if there is an index m such that $\lambda_m < \lambda_m'$ and for all $m < l \leqslant k$ we have $\lambda_l = \lambda_l'$. Note that $\prec$ defines a total order on $\mathcal{R}^1(n_{\min})$.

For two sequences $\Sigma_i^2, \Sigma_j^2 \in \mathcal{R}^2(n_{\min})$ we also write $\Sigma_i^2 \prec \Sigma_j^2$ if Σ_i^2 dominates Σ_j^2. Again, $\prec$ defines a total order on $\mathcal{R}^2(n_{\min})$. We now remove the dominated sequences from $\mathcal{R}(n_{\min})$ as follows.

Sort $\mathcal{R}^1(n_{\min})$ according to $\prec$, and sort $\mathcal{R}^2(n_{\min})$ according to $\prec$. Using RadixSort this can be done in $O(nk) = O(n \log n)$ time [29]. Now suppose that the first sequence in $\mathcal{R}^1(n_{\min})$ is Σ_i^1. Find the corresponding sequence Σ_i^2 in $\mathcal{R}^2(n_{\min})$; by maintaining cross-pointers this can be done in $O(1)$ time. Remove all the sequences Σ_j^2 from $\mathcal{R}^2(n_{\min})$ such that $\Sigma_i^2 \prec \Sigma_j^2$, and remove the corresponding sequences from Σ_j^1 from $\mathcal{R}^1(n_{\min})$. Since for the removed sequences we have $\Sigma_i^1 \prec \Sigma_j^1$ and $\Sigma_i^2 \prec \Sigma_j^2$, we also have that Σ_i dominates Σ_j. Hence, we remove Σ_j from $\mathcal{R}(n_{\min})$. Now remove Σ_i^1 and Σ_i^2 from $\mathcal{R}^1(n_{\min})$ and $\mathcal{R}^2(n_{\min})$, and repeat the process: take the next sequence from $\mathcal{R}^1(n_{\min})$, locate its corresponding sequence in in $\mathcal{R}^2(n_{\min})$, and remove the dominated sequences, and so on. The whole process, including the sorting, takes $O(n \log n)$ time, and it removes all dominated sequences from $\mathcal{R}(n_{\min})$.

Task (ii), removing all sequences from $\mathcal{R}(n_{\min}+1)$ that are dominated by another sequence in $\mathcal{R}(n_{\min}+1)$, can be done in the same way, so it remains to perform task (iii). This is done as follows. Again, we divide each sequence $\Sigma_i \in \mathcal{R}(n_{\min})$ into two subsequences by removing all labels of maximum value from it. Now consider the sequences $\Sigma_j \in \mathcal{R}(n_{\min}+1)$. We also remove $n_{\min}$ labels of maximum value from them, thus dividing them into two. However, we can do this in two ways, either adding the remaining label of maximum value to the left sequence or two the right sequence. Thus we can obtain four different sequences, which we denote by $\Sigma_j^{1,1}$, $\Sigma_j^{1,2}$, $\Sigma_j^{2,1}$ and $\Sigma_j^{2,2}$—see Figure 4.9(b). We now have: $\Sigma_i \in \mathcal{R}(n_{\min})$ dominates $\Sigma_j \in \mathcal{R}(n_{\min}+1)$ if and only if either Σ_i^1 dominates $\Sigma_j^{1,1}$ and Σ_i^2 dominates $\Sigma_j^{2,1}$, or Σ_i^1 dominates $\Sigma_j^{1,2}$ and Σ_i^2 dominates $\Sigma_j^{2,2}$. Thus filtering out the sequences from $\mathcal{R}(n_{\min}+1)$ that are dominated by a sequence from $\mathcal{R}(n_{\min})$ can be done in $O(n \log n)$, using a similar strategy as before. □

The next lemma explains the running time of the algorithm. It follows from the above lemmas and the fact that $k \leqslant 2\log_2 n$.

Lemma 4.9 *Algorithm* HISTOGRAMPARTITION *runs in* $O(n^7 \log n \log\log n)$ *time.*

Proof. By Lemma 4.6 the number of sequences in each $R(s_i)$ is at most $O(2^{3k/2}/\sqrt{k})$. The fact that $k \leqslant 2\log_2 n$ then implies $|R(s_i)| = O(n^3/\sqrt{\log n})$. Thus there are $O(n^6/\log n)$ different pairs of $R_1 \in \mathcal{R}(s_1)$ and $R_t \in \mathcal{R}(s_t)$ in Step 2.

Step (a) takes time linear in the total length of all subsequences, which is $O(n^4/\sqrt{\log n})$, but we can re-use this result for each combination R_1, R_t. The concatenated sequence with which we start in Step (b) has length at most n. We need to do $O(k)$ different combinations of merging-from-the-left/merging-from-the-right as follows. Let us denote the number of times the maximum value occurs for the resulting sequence $\overline{\Sigma} = \{\lambda_1, \ldots, \lambda_h\}$ by m. If $m \geqslant k$ we simply merge the entire sequence. Now suppose that $m < k$, and the labels $\lambda_i, \ldots, \lambda_j$ all have the maximum value. We need to find two labels λ_s $(s < i)$ and λ_h $(h > j)$ such that $h - s + 1 = k$. When there is more than one label with the same value as λ_s pick the rightmost one. Similarly when there is more than one label with the same value as λ_h pick the leftmost one. Merge all the labels to the left of and including λ_s, and merge all the labels to the right of and including λ_h. Repeat this for $O(k)$ different possible values of h and s.

Thus the total running time needed for a single pair $R_1 \in \mathcal{R}(s_1)$ and $R_t \in \mathcal{R}(s_t)$ is $O(k) = O(\log n)$. Multiplying by the number of pairs, the time becomes $O(n^6)$.

We have now generated $O(n^6)$ sequences with length at most k, from which we have to select the non-dominated ones. Using the approach described in Lemma 4.8 we can find all dominating sequences in $O(n^6 \log n)$. This is the time that we spend for each chord s_i. We have at most n chords and we need to test $\log\log n$ different values for k. This makes the total running time to be $O(n^7 \log n \log\log n)$. □

4.3 Approximating optimal Steiner triangulations

In this section we give a $O(1)$-approximation algorithm for the problem of finding a Steiner triangulation with minimum stabbing number of a given n-vertex polygon.

We say that a point $p \in P$ *sees* a point $q \in P$ if the line segment pq lies completely in P, and we say that a point p sees a segment s—or, equivalently, that the segment s sees p—if p sees at least one point of s. The *weak visibility polygon* of a segment s in P, denoted by $\text{VP}(s)$, is the simple polygon consisting of all points of P that see s. The number of vertices of $\text{VP}(s)$ is denoted by $|\text{VP}(s)|$. Let $OPT(P)$ denote the minimum stabbing number of any Steiner triangulation of P, and let $V_{\max}$ be the maximum complexity of any weak visibility polygon $\text{VP}(s)$, where s is either an edge of P or a chord of P (that is, a segment whose endpoints lie on ∂P and that otherwise lies in $\text{int}(P)$). We have the following lemma.

Lemma 4.10 $OPT(P) \geqslant \log(V_{\max} - 2) - \log\log(V_{\max} - 2)$.

Proof. Let s be a segment such that $|\text{VP}(s)| = V_{\max}$ and let T_{opt} be an optimal Steiner triangulation of P. We may assume without loss of generality that s does not pass through vertices of triangles. We denote the set of triangles crossed by s from right to left by $\mathcal{D}(s) := \{\delta_1, \delta_2, \ldots, \delta_k\}$; see Fig. 4.10(a). If $k > \log(V_{\max} - 2)$ we are done, so suppose that $k \leqslant \log(V_{\max} - 2)$. The proof is based on constructing a rooted binary tree $\mathcal{B}$ whose nodes correspond to edges of the triangles in T_{opt}. We show that $\mathcal{B}$ has at least $(V_{\max} - 2)/k$ leaves; the height of $\mathcal{B}$ is thus at least $\log((V_{\max} - 2)/k) = \log(V_{\max} - 2) - \log k \leqslant \log(V_{\max} - 2) - \log\log(V_{\max} - 2)$.

By construction, the height of $\mathcal{B}$ will be equal to the number of triangles stabbed by some segment in P, implying the claim of the lemma. The details are as follows.

For each edge e of the visibility polygon $\text{VP}(s)$, except for the two edges where s touches ∂P, we define a directed segment $\text{witness}(e)$ inside P that connects s to e. Thus $\text{witness}(e)$ is a witness of the fact that e is visible from s. Note that we have at least $V_{\max} - 2$ witness edges. We partition the set of witness edges into k subsets, as follows. For each triangle δ_i, let $\text{exit}(\delta_i)$ denote the

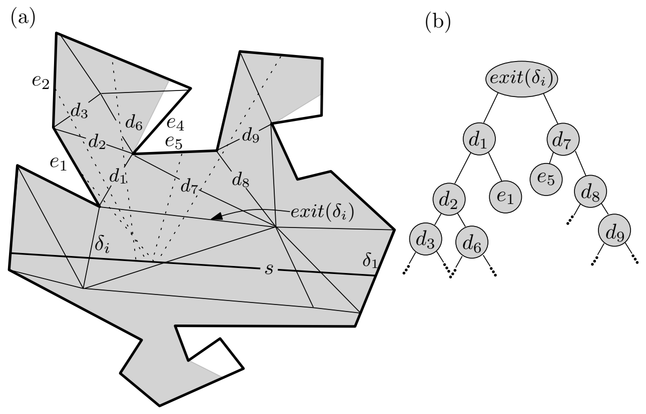


Figure 4.10 (a) The weak visibility polygon of s. (b) A part of the binary tree $\mathcal{B}$.

edge of δ_i that does not intersect s. We put each witness $w := \text{witness}(e)$ into a subset associated with the first exit edge that it intersects. The first exit edge is the first edge which we encounter, when we traverse w from s to e. Let W_i denote the set of witnesses associated with $\text{exit}(\delta_i)$ and let $\text{exit}(\delta_{i^*})$ be the exit edge with the maximum number of witnesses associated with it. Note that $|W_{i^*}| \geqslant (V_{\max} - 2)/k$.

We now construct the tree $\mathcal{B}$, whose nodes will correspond to edges of triangles in T_{opt}, iteratively as follows. The root of $\mathcal{B}$ is the edge $\text{exit}(\delta_{i^*})$. Consider the witnesses $w \in W_{i^*}$ in arbitrary order. For each $w := \text{witness}(e)$, we expand the tree as follows. Traverse w from s to e, visiting the triangle edges intersected by w in order. At the same time, traverse the current tree $\mathcal{B}$ starting at the root in such a way that the node visited in $\mathcal{B}$ corresponds to the triangle edge being crossed by w. At some point this may no longer be possible: then we step from an edge e' to an edge e'' while the node ν of $\mathcal{B}$ that we are in (and which corresponds to e') does not have a child corresponding to e''. In this case we create a new child μ , which corresponds to e''—see Fig. 4.10(b). When we step from e'' to e''', we create a child for μ which corresponds to e''' and so on. Thus, we create a path hanging from ν that corresponds to the edges intersected after e'. It is important to note that an edge can appear multiple times in $\mathcal{B}$. When we enter a triangle through a given edge, we can obviously only leave it through two other edges. (Here we use the assumption that we have a Steiner triangulation and not an arbitrary decomposition into triangles, that is, that there are no vertices in the interior of any triangle edge.) Hence $\mathcal{B}$ is a binary tree. Moreover, the path from the root of $\mathcal{B}$ to any leaf corresponds to the sequence of triangle edges intersected by some witness. Hence, the height of $\mathcal{B}$ is indeed a lower bound on the stabbing number of T_{opt}. □

Next we give an algorithm that computes a Steiner triangulation for P with stabbing number $O(\log V_{\max})$. By Lemma 4.10 this is an $O(1)$-approximation for finding an optimal Steiner triangulation of a simple polygon P. The algorithm consists of two stages.

In the first stage we recursively compute a decomposition of P into weak visibility polygons, in a standard way as follows. In a generic step of the algorithm we get a subpolygon $P' \subset P$ and a designated edge e' of P'. Initially $P' = P$ and e' is an arbitrary edge of P. We then compute $\text{VP}(e')$. Note that $P' \setminus \text{VP}(e')$ consists of several subpolygons, each separated from $\text{VP}(e')$ by a

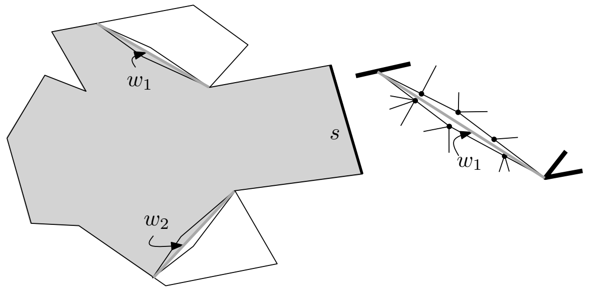


Figure 4.11 We need to replace each of the windows by a diamond-like polygon, and then triangulate them separately.

chord which is called the *window* of the subpolygon. We recurse on each subpolygon with its window as designated edge. This weak-visibility-polygon decomposition can be found in $O(n)$ time in total [73]. Note that any line segment $s \subset P$ intersects at most three of the weak visibility polygons.

The method of Hershberger and Suri [50] makes a Steiner triangulation with stabbing number $O(\log n)$ for a polygon with n vertices. Using their method in the second stage we compute a Steiner triangulation with stabbing number $O(\log V_{\max})$ of each weak visibility polygon VP. This produces a decomposition of each weak visible polygon into triangles with stabbing number $O(\log V_{\max})$. Based on Lemma 4.10 we have $OPT = \Omega(\log V_{\max})$, so this is a $O(1)$-factor approximation.

However, there is one problem: the decomposition is not necessarily a Steiner triangulation, because the Steiner triangulation of some polygon VP may introduce Steiner vertices on a window that are not used on the other side of the window. Hence, we adapt the algorithm as follows. Before applying the method of Hershberger and Suri to the visibility polygons, we first replace each window by a thin diamond, as shown in Figure 4.11. The decomposition of the visibility polygons can now add Steiner vertices on the edges of the diamonds, so the diamonds must be further decomposed.

Consider a diamond Δ, and let n_Δ be the number of Steiner vertices on the boundary of Δ. We move each vertex slightly outwards, so Δ becomes a convex polygon (with $n_\Delta + 4$ vertices) none of whose edges is collinear. Since we now have a convex polygon, we can easily compute a non-Steiner triangulation (in linear time) whose stabbing number is $O(\log n_\Delta)$. Note that any segment inside P can intersect at most three visibility polygons and at most two diamonds. Since the stabbing number of each visibility polygon is $O(\log V_{\max})$, each edge of the diamond has $O(\log V_{\max})$ vertices on it. This implies that $n_\Delta = O(\log V_{\max})$ for any diamond Δ. We get the following theorem.

Theorem 4.11 *Let P be a simple polygon with n vertices. Then we can compute a Steiner triangulation of P with stabbing number $O(OPT)$ in $O(n)$ time, where OPT is the minimum stabbing number of any Steiner triangulation of P.*

4.4 Concluding remarks

We have studied the problem of finding a decomposition with minimum stabbing number for a simple polygon. We gave a 3-approximation algorithm for the rectilinear version of the problem (which was based on an optimal algorithm for histograms) and we gave an $O(1)$-approximation algorithm for the non-rectilinear case. We have not been able to construct an exact polynomial-time algorithm for either problems, but the problems are not known to be NP-complete either. Establishing the computational complexity of the problem is thus the first open problem. Another interesting open problem is to study the case of polygons with holes.

Chapter 5

Piecewise-Linear Approximations of Uncertain Functions

Chapter summary. We study the problem of approximating a function $\mathrm{F}\colon \mathbb{R} \to \mathbb{R}$ by a piecewise-linear function $\overline{\mathrm{F}}$ when the values of F at $\{x_1, \ldots, x_n\}$ are given by a discrete probability distribution. Thus, for each x_i we are given a discrete set $y_{i,1}, \ldots, y_{i,m_i}$ of possible function values with associated probabilities $p_{i,j}$ such that $\mathbf{Pr}[\mathrm{F}(x_i) = y_{i,j}] = p_{i,j}$. We define the error of $\overline{\mathrm{F}}$ as *error*$(\mathrm{F}, \overline{\mathrm{F}}) = \max_{i=1}^{n} \mathbf{E}[|\mathrm{F}(x_i) - \overline{\mathrm{F}}(x_i)|]$. Let $m = \sum_{i=1}^{n} m_i$ be the total number of potential values over all $\mathrm{F}(x_i)$. We obtain the following two results: (i) an $O(m)$ time algorithm that, given F and a maximum error ϵ, computes a function $\overline{\mathrm{F}}$ with the minimum number of links such that *error*$(\mathrm{F}, \overline{\mathrm{F}}) \leqslant \epsilon$; (ii) an $O(n^{4/3+\delta} + m \log n)$ time algorithm that, given F, an integer value $1 \leqslant k \leqslant n$ and any $\delta > 0$, computes a function $\overline{\mathrm{F}}$ of at most k links that minimizes *error*$(\mathrm{F}, \overline{\mathrm{F}})$.

5.1 Introduction

Motivation and problem statement. Fitting a function to a given finite set of points sampled from an unknown function $\mathrm{F}\colon \mathbb{R} \to \mathbb{R}$ is a basic problem in mathematics. Typically one is given a class of "simple" functions—linear functions, piecewise linear functions, quadratic functions, etcetera—and the goal is to find a function $\overline{\mathrm{F}}$ from that class that fits the sample points best. One way to measure how well $\overline{\mathrm{F}}$ fits the sample points is the *uniform metric*, defined as follows. Suppose that F is sampled at $x_1, \ldots, x_n$, with $x_1 < \cdots < x_n$. Then the error of $\overline{\mathrm{F}}$ according to the uniform metric is $\max |\mathrm{F}(x_i) - \overline{\mathrm{F}}(x_i)|$. This measure is also known as the l_∞ or the Chebychev error measure.

The problem of finding the best approximation $\overline{\mathrm{F}}$ under the uniform metric has been studied from an algorithmic point of view, in particular for the case where the allowed functions are piecewise linear. There are then two optimization problems that can be considered: the min-k and the min-ε problem. In the min-k problem one is given a maximum error $\epsilon \geqslant 0$ and the goal is to find piecewise-linear function $\overline{\mathrm{F}}$ with error at most ε that minimizes the number of links. In the min-ϵ problem one is given a number $k \geqslant 1$ and the goal is to find a piecewise-linear function with at most k links that minimizes the error.

The min-k problem was solved in $O(n)$ time by Hakimi and Schmeichel [46]. They also gave an $O(n^2 \log n)$ algorithm for solving the min-ε problem. This was later improved to $O(n^2)$ by Wang *et al.* [82]. Goodrich [44] then managed to obtain an $O(n \log n)$ algorithm.

In this chapter we also study the problem of approximating a sampled function by a piecewise-linear function, but we do this in the setting where the function values $\mathrm{F}(x_i)$ at the sample points are not known exactly. Instead we have a discrete probability distribution for each $\mathrm{F}(x_i)$, that is, we have a discrete set $y_{i,1}, \ldots, y_{i,m_i}$ of possible values with associated probabilities $p_{i,j}$ such that $\mathbf{Pr}[\mathrm{F}(x_i) = y_{i,j}] = p_{i,j}$. We call such a function an *uncertain function*. The goal is now to find a piecewise-linear function $\overline{\mathrm{F}}$ with at most k links that minimizes the expected error (the min-ϵ problem) or a piecewise-linear function $\overline{\mathrm{F}}$ with error at most ε that minimizes the number of links (the min-k problem).

There are several possibilities to define the expected error. We use the uniform metric and define our error measure in the following natural way.

$$\mathit{error}(\mathrm{F}, \overline{\mathrm{F}}) = \max \left\{ \mathbf{E}[|\mathrm{F}(x_i) - \overline{\mathrm{F}}(x_i)|] : 1 \leqslant i \leqslant n \right\}.$$

This error is not equal to $\mathit{error}_2(\mathrm{F}, \overline{\mathrm{F}}) = \max\{|\mathbf{E}[\mathrm{F}(x_i)] - \overline{\mathrm{F}}(x_i)| : 1 \leqslant i \leqslant n\}$. Indeed, to minimize $|\mathbf{E}[\mathrm{F}(x_i)] - \overline{\mathrm{F}}(x_i)|$ one should take $\overline{\mathrm{F}}(x_i) = \mathbf{E}[\mathrm{F}(x_i)]$ leading to an error of zero at x_i. Hence, we feel that $\mathit{error}(\mathrm{F}, \overline{\mathrm{F}})$ is more appropriate than $\mathit{error}_2(\mathrm{F}, \overline{\mathrm{F}})$. (Note that approximating F under error measure error_2 boils down to approximating the function $G\colon \mathbb{R} \to \mathbb{R}$ with $G(x_i) = \mathbf{E}[\mathrm{F}(x_i)]$.)

Related work. The problem of approximating a sampled function can be seen as a special case of line simplification. The line-simplification problem is to approximate a given polygonal curve $P = p_1, p_2, \ldots, p_n$ by a simpler polygonal curve $Q = q_1, q_2, \ldots, q_k$. The problem comes in many flavors, depending on the restrictions that are put on the approximation Q, and on the error measure $\mathit{error}(P, Q)$ that defines the quality of the approximation. A typical restriction is that the sequence of vertices of Q be a subsequence of the vertices of P, with $q_1 = p_1$ and $q_k = p_n$; the unrestricted version, where the vertices $q_1, q_2, \ldots, q_k$ can be chosen arbitrarily, has been studied as well. (In this chapter we do not restrict the locations of the breakpoints of our piecewise-linear function.) Typical error measures are the Hausdorff distance and the Fréchet distance [11].

The line-simplification has been studied extensively. The oldest and probably best-known algorithm for line simplification is the so-called Douglas-Peucker algorithm [32], dating back to 1973. This algorithm achieves good results in practice, but it is not guaranteed to give optimal results. Over the past 20 years or so, algorithms giving optimal results have been developed for many line-simplification variants [1, 9, 7, 22, 44, 46, 51, 64, 81]. Although both function approximation and line-simplification are well-studied problems, and there has been ample research on uncertain data in other contexts, the problem we study has, to the best of our knowledge, not been studied so far.

Our results. We start by studying the min-k problem. As it turns out, this problem is fairly easily reduced to the problem of computing a minimum-link path that stabs a set of vertical segments. The latter problem can be solved in linear time [46], leading to an algorithm for the min-k problem running in $O(m)$ time, where $m = \sum_{i=1}^{n} m_i$. We then turn our attention to the much more challenging min-ε problem, where we present an algorithm that, for any fixed $\delta > 0$, runs in $O(n^{4/3+\delta} + m \log n)$ time. Our algorithm uses similar ideas as the algorithm of Goodrich [44], but it requires several new ingredients to adapt it to the case of uncertain functions. For the important special case $k = 1$—here we wish to find the best linear function to approximate F—we obtain a faster algorithm for the min-ε problem, running in $O(m \log m)$ time.

5.2 The min-k problem

We start by studying the properties of *error*$(\mathrm{F}, \overline{\mathrm{F}})$. First we define an error function $\mathcal{E}_i(y)$ for every value x_i:

$$\mathcal{E}_i(y) = \mathbf{E}[|\mathrm{F}(x_i) - y|].$$

Observe that $\mathcal{E}_i(\overline{\mathrm{F}}(x_i))$ is the expected error of $\overline{\mathrm{F}}$ at x_i, and *error*$(\mathrm{F}, \overline{\mathrm{F}}) = \max_{i=1}^{n} \mathcal{E}_i(\overline{\mathrm{F}}(x_i))$. The following lemma shows what $\mathcal{E}_i(y)$ looks like. For simplicity we assume $y_{i,1} \leqslant \cdots \leqslant y_{i,m_i}$ for $1 \leqslant i \leqslant n$.

Lemma 5.1 *For any i, the function $\mathcal{E}_i(y)$ is a convex piecewise-linear function with $m_i + 1$ links.*

Proof. To simplify the presentation, define $y_{i,0} = -\infty$ and $y_{i,m_i+1} = +\infty$, and we set $p_{i,0} = p_{i,m_i+1} = 0$. Now fix some j with $0 \leqslant j \leqslant m_i$, and consider the y-interval $[y_{i,j}, y_{i,j+1}]$. Within this interval we have

$$\begin{aligned}
\mathcal{E}_i(y) &= \mathbf{E}[|\mathrm{F}(x_i) - y|] \\
&= \textstyle\sum_{\ell=1}^{j} p_{i,\ell}(y - y_{i,\ell}) + \sum_{\ell=j+1}^{m_i} p_{i,\ell}(y_{i,\ell} - y) \\
&= \left(\textstyle\sum_{\ell=1}^{j} p_{i,\ell} - \sum_{\ell=j+1}^{m_i} p_{i,\ell}\right) \cdot y \; - \; \left(\textstyle\sum_{\ell=1}^{j} p_{i,\ell} y_{i,\ell} - \sum_{\ell=j+1}^{m_i} p_{i,\ell} y_{i,\ell}\right) \\
&= a_j y + b_j,
\end{aligned}$$

where

$$a_j = \sum_{\ell=1}^{j} p_{i,\ell} - \sum_{\ell=j+1}^{m_i} p_{i,\ell} \quad \text{and} \quad b_j = -\left(\sum_{\ell=1}^{j} p_{i,\ell} y_{i,\ell} - \sum_{\ell=j+1}^{m_i} p_{i,\ell} y_{i,\ell}\right).$$

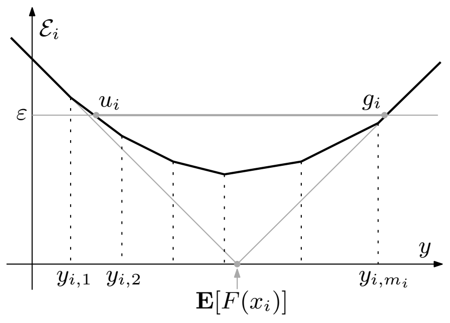


Figure 5.1 The function $\mathcal{E}_i(y)$.

Hence, $\mathcal{E}_i(y)$ is a piecewise-linear function with $m_i + 1$ links. Moreover, since $a_0 = -1 \leqslant a_1 \leqslant \cdots \leqslant a_{m_i} = 1$, it indeed is convex. It is not difficult to see that the first and last links of $\mathcal{E}_i$, when extended, meet exactly in the point $(\mathbf{E}[\mathrm{F}(x_i)], 0)$; see Fig. 5.1. (Actually these two links, when extended, form the graph of the function $g(y) = |\mathbf{E}[\mathrm{F}(x_i)] - y|$, so they correspond to the error at x_i as given by *error*$_2$.) $\square$

Now consider the min-k problem, and let ε be the given bound on the maximum error. Because $\mathcal{E}_i(y)$ is a convex function, there exists an interval $[u_i, g_i]$ such that $\mathcal{E}_i(y) \leqslant \varepsilon$ if and only if $y \in [u_i, g_i]$; see Fig. 5.1. The interval $[u_i, g_i]$ can be computed in $O(m_i)$ time. When there exists an i such that $[u_i, g_i]$ is empty, we report that there is no $\overline{\mathrm{F}}$ approximating F within error ε. Otherwise, the problem is reduced to finding a function $\overline{\mathrm{F}}$ with a minimum number of links stabbing every vertical segment $x_i \times [u_i, g_i]$. This problem can be solved in linear time [46], leading to the following theorem.

Theorem 5.2 *Let* $\mathrm{F}: \mathbb{R} \to \mathbb{R}$ *be an uncertain function whose values are given at* n *points* $\{x_1, \ldots, x_n\}$ *and let* m *be the total number of possible values at these points. For a given* ε*, a piecewise-linear function* $\overline{\mathrm{F}}$ *with a minimum number of links such that error*$(\mathrm{F}, \overline{\mathrm{F}}) \leqslant \varepsilon$ *can be computed in* $O(m)$ *time.*

Remark 5.3 Above we computed the intervals $[u_i, g_i]$ in $O(m_i)$ time in a brute-force manner. However, we can also compute the values u_i and g_i in $O(\log m_i)$ time using binary search. Then, after computing the functions $\mathcal{E}_i$ in $O(m)$ time in total, we can construct the segments $x_i \times [u_i, g_i]$ in $O(\sum_i \log m_i) = O(n \log m)$ time. Thus, after $O(m)$ preprocessing, the min-k problem can be solved in $O(n \log m)$ time. This can be used to speed up the algorithm from the next section when $n = o(m/\log m)$. For simplicity we do not consider this improvement in the next section.

5.3 The min-ε problem

We now turn our attention to the min-ε problem. Let k be the maximum number of links we are allowed to use in our approximation. For any $\varepsilon > 0$, define $\mathcal{K}(\varepsilon)$ to be the minimum number of links of any approximation $\overline{\mathrm{F}}$ such that *error*$(\mathrm{F}, \overline{\mathrm{F}}) \leqslant \varepsilon$. Note that $\mathcal{K}(\varepsilon)$ can be computed with the algorithm from the previous section. Clearly, if $\varepsilon_1 < \varepsilon_2$ then $\mathcal{K}(\varepsilon_1) \geqslant \mathcal{K}(\varepsilon_2)$. Hence, $\mathcal{K}(\varepsilon)$ is a non-increasing function of ε and $\mathcal{K}(\varepsilon) \leqslant n$. Our goal is now to find the smallest ε such that $\mathcal{K}(\varepsilon) \leqslant k$. Let's call this value ε^*. Because $\mathcal{K}(\varepsilon)$ is non-increasing, the idea is to use a binary search to find ε^*, with the algorithm from the previous section as decision procedure. Doing this in an efficient manner is not so easy, however. We will proceed in several phases, zooming in further and further to the value ε^*, as explained next.

Our algorithm maintains an *active interval* $\mathcal{I}$ containing ε^*. Initially $\mathcal{I} = [0, \infty)$. A basic subroutine is to refine $\mathcal{I}$ on the basis of a set $\mathcal{S}$ of ε-values, whose output is the smallest interval containing ε^* whose endpoints come from the set $\mathcal{S} \cup \{\text{endpoint of } \mathcal{I}\}$. The subroutine *ShrinkActiveInterval* runs in $O(|\mathcal{S}| \log |\mathcal{S}| + m \log |\mathcal{S}|)$ time.

> *ShrinkActiveInterval*($\mathcal{S}, \mathcal{I}$)
> Sort $\mathcal{S}$ to get a sorted list $\varepsilon_1 < \varepsilon_2 < \cdots < \varepsilon_h$. Add values $\varepsilon_0 = -\infty$ and $\varepsilon_{h+1} = \infty$. Do a binary search over $\varepsilon_0, \ldots, \varepsilon_{h+1}$ to find an interval $[\varepsilon_j, \varepsilon_{j+1}]$ containing ε^*; here the basic test during the binary search—namely whether $\varepsilon^* \leqslant \varepsilon_l$, for some ε_l—is equivalent to testing whether $\mathcal{K}(\varepsilon_l) \leqslant k$; this can be done in $O(m)$ time with the algorithm from the previous section. Finally, return $\mathcal{I} \cap [\varepsilon_j, \varepsilon_{j+1}]$.

The first phase. In the previous section we have seen that each error function $\mathcal{E}_i$ is a convex piecewise-linear function with m_i breakpoints. Let $E_i = \{\mathcal{E}_i(y_{i,j}) : 1 \leqslant j \leqslant m_i\}$ denote the set of error-values of the breakpoints of $\mathcal{E}_i$, and let $E = E_1 \cup \cdots \cup E_n$. The first phase of our algorithm is to call *ShrinkActiveInterval*(E, $[0, \infty)$) to find two consecutive values $\varepsilon_j, \varepsilon_{j+1} \in E$ such that $\varepsilon_j \leqslant \varepsilon^* \leqslant \varepsilon_{j+1}$. Since $|E| = m$, this takes $O(m \log m)$ time.

Recall that for a given ε, the approximation $\overline{\text{F}}$ has to intersect the segment $x_i \times [u_i, g_i]$ in order for the error to be at most ε at x_i. Now imagine increasing ε from ε_j to ε_{j+1}. Then the values u_i and g_i change continuously. In fact, since $\mathcal{E}_i$ is a convex piecewise-linear function and ε_j and ε_{j+1} are consecutive values from E, the values u_i and g_i change linearly, that is, we have

$$u_i(\varepsilon) = a_i\varepsilon + b_i \quad \text{and} \quad g_i(\varepsilon) = c_i\varepsilon + d_i$$

for constants a_i, b_i, c_i, d_i that can be computed from $\mathcal{E}_i$. As ε increases, u_i decreases and g_i increases—thus $a_i < 0$ and $c_i > 0$—and so the vertical segment $x_i \times [u_i, g_i]$ is growing. After the first phase we have $\mathcal{I} = [\varepsilon_j, \varepsilon_{j+1}]$ and the task is to find the smallest $\varepsilon \in [\varepsilon_j, \varepsilon_{j+1}]$ such that there exists a k-link path stabbing all the segments.

Intermezzo: the case $k = 1$. We first consider the second phase for the special but important case where $k = 1$. This case can be considered as the problem of finding a regression line for uncertain points except that our error is not the squared distance. Thus we want to approximate the uncertain function F by a single line $\ell\colon y = ax + b$ that minimizes the error. The line ℓ stabs a segment $x_i \times [u_i, g_i]$ if ℓ is above (x_i, u_i) and below (x_i, g_i). In other words, we need $ax_i + b \geqslant a_i\varepsilon + b_i$ and $ax_i + b \leqslant c_i\varepsilon + d_i$. Hence, the case $k = 1$ can be handled by solving the following linear program with variables a, b, ε:

$$\begin{array}{lll}
\text{Minimize} & \varepsilon & \\
\text{Subject to} & x_i a + b - a_i\varepsilon \geqslant b_i & \text{for all } 1 \leqslant i \leqslant n \\
 & x_i a + b - c_i\varepsilon \leqslant d_i & \text{for all } 1 \leqslant i \leqslant n
\end{array}$$

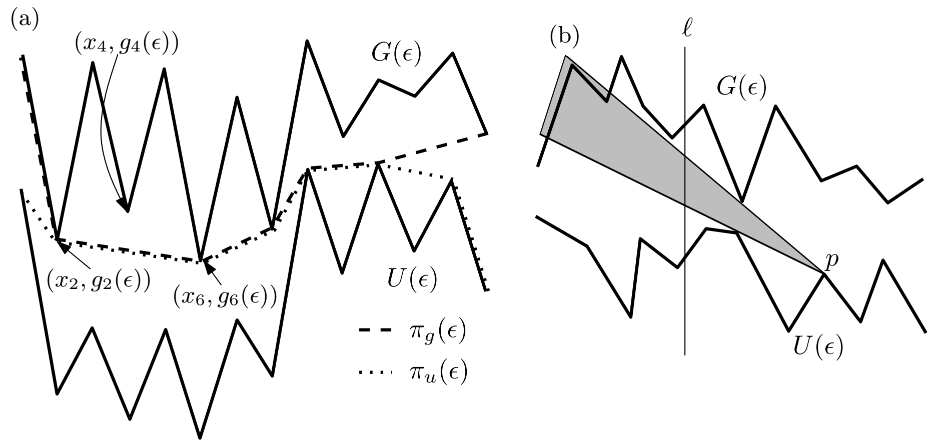


Figure 5.2 (a) The paths $\pi_u(\varepsilon)$ and $\pi_g(\varepsilon)$. (b) The visibility cone of p.

Since a 3-dimensional linear program can be solved in linear expected time [16], we get the following theorem.

Theorem 5.4 *The line ℓ minimizing error(F, ℓ) can be computed in $O(m \log m)$ expected time.*

The second phase. As mentioned earlier, our algorithm uses ideas from the algorithm that Goodrich [44] developed for the case of certain[1] functions. Next we sketch his algorithm and explain the difficulties in applying it to our problem.

For a certain function F, the error functions $\mathcal{E}_i(y)$ are cones whose bounding lines have slope -1 and $+1$, respectively. This implies that, using the notation from above, we have $u_i(0) = g_i(0)$, and $a_i = -1$, and $c_i = 1$ for all i. For a given ε, we define $U(\varepsilon)$ to be the polygonal chain $(x_1, u_1(\varepsilon)), \dots, (x_n, u_n(\varepsilon))$ and $G(\varepsilon)$ to be the polygonal chain $(x_1, g_1(\varepsilon)), \dots, (x_n, g_n(\varepsilon))$. (Note that: the minimum-link path of error at most ε stabbing all segments $x_i \times [u_i, g_i]$ does not have to stay within the region bounded by $U(\varepsilon)$ and $G(\varepsilon)$.) Let $\pi_u(\varepsilon)$ be the Euclidean shortest path from $(x_1, u_1(\varepsilon))$ to $(x_n, u_n(\varepsilon))$ that is below $G(\varepsilon)$ and above $U(\varepsilon)$—see Fig. 5.2(a) for an illustration. Similarly, let $\pi_g(\varepsilon)$ be the Euclidean shortest path from $(x_1, g_1(\varepsilon))$ to $(x_n, g_n(\varepsilon))$ that is below $G(\varepsilon)$ and above $U(\varepsilon)$. The paths $\pi_u(\varepsilon)$ and $\pi_g(\varepsilon)$ together form a so-called *hourglass*, which we denote by $H(\varepsilon)$. An edge $e \in H(\varepsilon)$ is called an *inflection edge* if one of its endpoints lies on $U(\varepsilon)$ and the other one lies on $G(\varepsilon)$. Goodrich [44] showed that there is a minimum-link function $\overline{\mathrm{F}}$ with error ε such that each inflection edge is contained in a link of $\overline{\mathrm{F}}$ and in between two links containing inflection edges the function is convex or concave. (In other words, the "zig-zags" of $\overline{\mathrm{F}}$ occur exactly at the inflection edges.)

Goodrich then proceeds by computing all values of ε at which the set of inflection edges changes; these are called *geodesic-critical* values of ε. Note that the moments at which an inflection edge

[1]We use the term *certain function* for a function with exactly one possible value per sample point, that is, when no uncertainty is involved.

changes are exactly the moments at which the hourglass $H(\varepsilon)$ changes. The number of geodesic-critical values is $O(n)$ and they can be found in $O(n \log n)$ time [44]. After finding these geodesic-critical values, a binary search is applied to find two consecutive critical values $\varepsilon_j, \varepsilon_{j+1}$ such that $\varepsilon_j \leqslant \varepsilon^* \leqslant \varepsilon_{j+1}$. Then the inflection edges that $\overline{\mathrm{F}}$ must contain are known, and from this $\overline{\mathrm{F}}$ can be computed using parametric search.

The main difference between our setting and the setting of Goodrich is that the points $(x_i, u_i(\varepsilon))$—and, similarly, the points $(x_i, g_i(\varepsilon))$—do not move at the same speed. As a result the basic lemma underlying the efficiency of the approach, namely that there are only $O(n)$ geodesic-critical values of ε, no longer holds. The following lemma shows that the number of such values can actually be quadratic.

Lemma 5.5 *There is an instance of n vertical segments $x_i \times [u_i(\varepsilon), g_i(\varepsilon)]$ where the $(x_i, u_i(\varepsilon))$'s and $(x_i, g_i(\varepsilon))$'s are moving at constant (but different) velocities such that the number of geodesic-critical events is $\Omega(n^2)$.*

Proof. The example is presented in Fig. 5.3.(a). First we explain the locations of the points and then describe their velocities. First put the two points $(x_{n/2}, u_{n/2}(\varepsilon)) = (n/2, b)$ and $(x_{n/2}, g_{n/2}(\varepsilon)) = (n/2, b+c)$ for some b and c greater than 0. Then place two points $(x_n, u_n(\varepsilon)) = (n, b+c-n/2)$ and $(x_n, g_n(\varepsilon)) = (n, b+n/2)$. Put two other points $(x_{n-1}, u_{n-1}(\varepsilon)) = (n-1, b+c-n/2+1)$ and $(x_{n-1}, g_{n-1}(\varepsilon)) = (n-1, b+n/2-1)$. This means that the line ℓ_1 which passes through $(x_n, u_n(\varepsilon))$ and $(x_{n-1}, u_{n-1}(\varepsilon))$ goes through $(x_{n/2}, g_{n/2}(\varepsilon))$ as well. Similarly $(x_{n/2}, u_{n/2}(\varepsilon))$ is on the line ℓ_2 passing through $(x_n, g_n(\varepsilon))$ and $(x_{n-1}, g_{n-1}(\varepsilon))$. (The lines are shown as dashed lines in Fig. 5.3).

Next place all the points $(x_{n/2+1}, u_{n/2+1}(\varepsilon)), \ldots, (x_{n-2}, u_{n-2}(\varepsilon))$ such that the set of segments $(x_i, u_i(\varepsilon))(x_{i+1}, u_{i+1}(\varepsilon))$ for, $n/2 \leqslant i < n$, forms a convex chain below the line ℓ_1. Similarly put $(x_{n/2+1}, g_{n/2+1}(\varepsilon)), \ldots, (x_{n-2}, g_{n-2}(\varepsilon))$ such that the set of segments $(x_i, g_i(\varepsilon))(x_{i+1}, g_{i+1}(\varepsilon))$ for $n/2 \leqslant i < n$ forms a convex chain above ℓ_2.

As the remaining points, put the points $(x_{n/2-i}, u_{n/2-i}(\varepsilon)) = (n/2-i, b+c+i+c\sum_{j=1}^{i-1} j + i\sum_{j=1}^{i-1} j - \varepsilon)$, and the points $(x_{n/2-i}, g_{n/2-i}(\varepsilon)) = (n/2-i, u_{n/2-i}(\varepsilon)+1+\varepsilon)$ for odd values of i ($i \leqslant n/2$).

Then put the points $(x_{n/2-i}, g_{n/2-i}(\varepsilon)) = (n/2-i, b-i-c\sum_{j=1}^{i-1} j - i\sum_{j=1}^{i-1} j + \varepsilon)$ and the points $(x_{n/2-i}, u_{n/2-i}(\varepsilon)) = (n/2-i, g_{n/2-i}(\varepsilon)-1-\varepsilon)$ for even values of i ($i \leqslant n/2$). The overall structure for $\varepsilon = 0$ is shown in the Fig. 5.3.a. The Euclidean shortest path $\pi_u(0)$ is shown as the bold gray line. We show for this set of points $\pi_u(\varepsilon)$ changes $\Omega(n^2)$ as the value of ε increases.

Now consider the point $(x_{n/2-1}, u_{n/2-1}(\varepsilon))$, with $\varepsilon = c+2$ this point reaches ℓ_2—see Fig. 5.3.b. It is easy to see that for $\varepsilon = c+2$ the value of $u_{n/2-2}(\varepsilon)$ is $b-c-2$ and the point is on ℓ_2. At $\varepsilon = 0$ the shortest path $\pi_u(0)$ passes through $(x_{n/2-1}, u_{n/2-1}(\varepsilon))$. When ε increases from 0 to $c+2$, the points $(x_{n-1}, u(\varepsilon)), \ldots, (x_{n/2}, u(\varepsilon))$ are added to $\pi_u(\varepsilon)$ one by one. Thus, there would be $n/2-1$ changes to $\pi_u(\varepsilon)$. At $\varepsilon = c+2$ the point $(x_{n/2-2}, u_{n/2-2}(\varepsilon))$ reaches ℓ_2 and at this time $\pi_u(\varepsilon)$ passes through it. For $\varepsilon = 2c+6$ the point $(x_{n/2-2}, u_{n/2-2}(\varepsilon))$ reaches ℓ_1—see Fig. 5.3.b. At the same time $(x_{n/2-3}, u_{n/2-3}(\varepsilon))$ reaches ℓ_1. By increasing ε from $c+2$ to $2c+6$ again $\pi_u(\varepsilon)$ changes $n/2-1$ times, since the points $(x_{n-1}, u(\varepsilon)), \ldots, (x_{n/2}, u(\varepsilon))$ are removed from it one by one.

Similarly it can be shown that while each of the points $(x_{n/2-i}, u_{n/2-i}(\varepsilon))$ for odd values of i and $(x_{n/2-i}, g_{n/2-i}(\varepsilon))$ for even values of i are on the $\pi_u(\varepsilon)$, the shortest path $\pi_u(\varepsilon)$ changes $n/2-1$

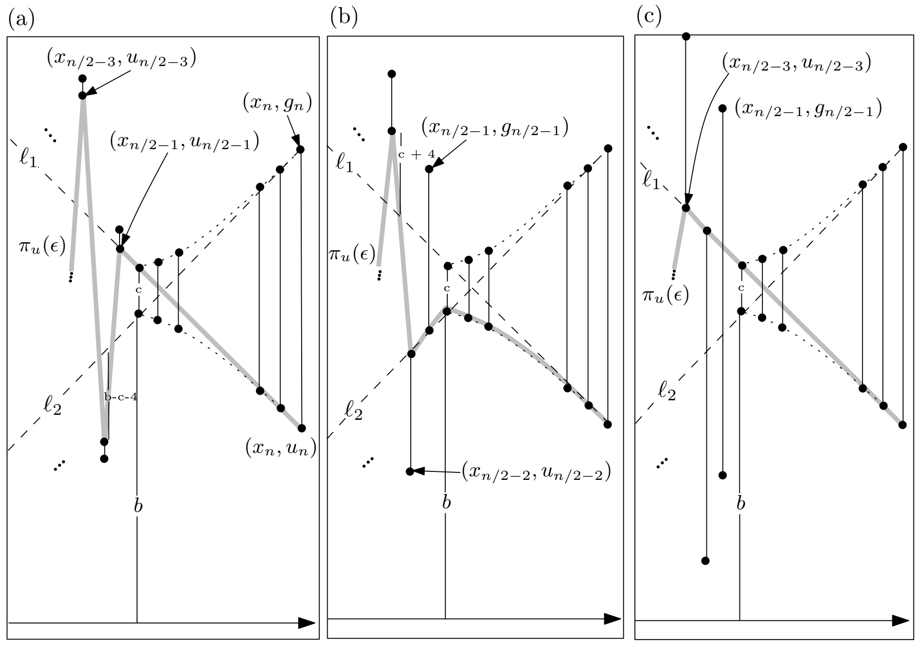


Figure 5.3 The lower bound example. (a) The moment at which the first critical event happens. (b) At this time we already had $n/4$ critical events. (c) The second point has already made $n/4$ critical events.

times. This means that $\pi_u(\varepsilon)$ changes $\Omega(n^2)$ and the lemma holds. □

The fact that the number of geodesic-critical values of ε is $\Omega(n^2)$ is not the only problem we face. The other problem is that detecting these events becomes more difficult in our setting. When all points on $U(\varepsilon)$ and on $G(\varepsilon)$ move with the same speed, then these events occur only when two consecutive edges of $\pi_u(\varepsilon)$ become collinear or when two consecutive edges of $\pi_g(\varepsilon)$ become collinear. When the points have different speeds, however, this is no longer the case. In Fig. 5.2, for example, the hourglass $H(\varepsilon)$ can change when $(x_2, g_2(\varepsilon))$, $(x_4, g_4(\varepsilon))$ and $(x_6, g_6(\varepsilon))$ become collinear (which could happen when $(x_4, g_4(\varepsilon))$ moves up relatively slowly).

Below we show how to overcome these two hurdles and obtain an algorithm with subquadratic running time.

We start with a useful observation. Let $\Psi(\varepsilon)$ be the simple polygon whose boundary consists of the chains $G(\varepsilon)$ and $U(\varepsilon)$, and the vertical segments $x_1 \times [u_1(\varepsilon), g_1(\varepsilon)]$ and $x_n \times [u_n(\varepsilon), g_n(\varepsilon])$. Let $\mathcal{G}(\varepsilon)$ be the visibility graph of $\Psi(\varepsilon)$, that is, $\mathcal{G}(\varepsilon)$ is the graph whose nodes are the vertices of $\Psi(\varepsilon)$ and where two nodes are connected by an edge if the corresponding vertices can see each other inside $\Psi(\varepsilon)$. As ε increases and the vertices of $\Psi(\varepsilon)$ move, $\mathcal{G}(\varepsilon)$ can change combinatorially, that is, edges may appear or disappear.

Lemma 5.6 *The visibility graph $\mathcal{G}(\varepsilon)$ changes $O(n^2)$ times as ε increases from 0 to ∞. Moreover, if $\mathcal{G}(\varepsilon)$ does not change when ε is restricted to some interval $[\varepsilon_1, \varepsilon_2]$, then $H(\varepsilon)$ does not change either when ε is in this interval.*

Proof. First we observe that any three vertices of $\Psi(\varepsilon)$ become collinear at most once, because each vertex moves with constant velocity and has constant x-coordinate. Indeed, three points p, q, r are collinear when $(p_y - q_y)/(p_x - q_x) = (p_y - r_y)/(p_x - r_x)$, and when p_x, q_x, r_x are constant and p_y, q_y, r_y are linear functions of ε, then this equation has one solution (or possibly infinitely many solutions, which means the points are always collinear).

Next we show that every edge e of $\mathcal{G}$, once it disappears, cannot re-appear. Define $\tilde{u}_i = (x_i, u_i)$ and $\tilde{g}_i = (x_i, g_i)$. Assume that $e = (\tilde{u}_i, \tilde{u}_j)$ for some i, j; the case where one or both of the endpoints of e are on $G(\varepsilon)$ is similar. None of the vertices of $G(\varepsilon)$ can stop $\tilde{u}_i$ and $\tilde{u}_j$ from seeing each other, since all $\tilde{u}_i$'s move down and all $\tilde{g}_i$'s move up. Hence, the only reason for $\tilde{u}_i$ and $\tilde{u}_j$ to become invisible to each other is that some vertex $\tilde{u}_l$, with $i < l < j$, crosses e. For $\tilde{u}_i$ and $\tilde{u}_j$ to become visible again, $\tilde{u}_l$ would have to cross e again, but this is impossible since $\tilde{u}_i, \tilde{u}_j, \tilde{u}_j$ can become collinear at most once. It follows that each edge can appear and disappear at most once, and since there are $O(n^2)$ edges in total, $\mathcal{G}$ changes $O(n^2)$ times.

The second part of the lemma immediately follows from the fact that the shortest paths $\pi_u(\varepsilon)$ and $\pi_g(\varepsilon)$ cannot "jump" as ε changes continuously, because shortest paths in a simple polygon are unique. Hence, these shortest paths—and, consequently, $H(\varepsilon)$—can only change when $\mathcal{G}(\varepsilon)$ changes. $\square$

Computing all combinatorial changes of $\mathcal{G}$ still results in an algorithm with running time $\Omega(n^2)$. Next we show that it suffices to compute $O(n^{4/3+\delta})$ combinatorial changes of $\mathcal{G}$ in order to find an interval $[\varepsilon_1, \varepsilon_2]$ such that $\varepsilon^* \in [\varepsilon_1, \varepsilon_2]$ and $\mathcal{G}$ does not change in this interval. (Recall that ε^* denotes the minimum error that can be achieved with k links, and that we thus wish to find).

Obtaining stable visibility cones. Let $\mathcal{I}$ be the active interval resulting from the first phase of our algorithm. We now describe an approach to quickly find a subset of the events where $\mathcal{G}$ changes, which we can use to further shrink $\mathcal{I}$.

Let ℓ be a vertical line splitting the set of vertices of Ψ into two (roughly) equal-sized subsets. We will concentrate on the visibility edges whose endpoints lie on the different sides of ℓ; the approach will be applied recursively to deal with the visibility edges lying completely to the right or completely to the left of ℓ. For a vertex p of $\Psi(\varepsilon)$ we define $\sigma(p, \varepsilon)$, the *visibility cone* of p in $\Psi(\varepsilon)$, as the cone with apex p that contains all rays emanating from p that cross a point on ℓ that is visible from p within $\Psi(\varepsilon)$. A crucial observation is that for a vertex p to the left of ℓ and a vertex q to the right of ℓ, we have that (p, q) is an edge of $\mathcal{G}(\varepsilon)$ if and only if $p \in \sigma(q, \varepsilon)$ and $q \in \sigma(p, \varepsilon)$.

As the vertices of Ψ move, $\sigma(p, \varepsilon)$ changes continuously but its combinatorial description (the vertices defining its sides) changes at discrete times. Notice that the bottom side of $\sigma(p, \varepsilon)$ passes through a vertex of the lower boundary of $\Psi(\varepsilon)$ lying to the same side of ℓ as p. More precisely, if $U(p, \ell)$ denotes the set of vertices on the lower boundary of $\Psi(\varepsilon)$ that lie between ℓ and the vertical line through p, then the lower side of $\sigma(p, \varepsilon)$ is tangent to the upper hull of $U(p, \ell)$—see Fig. 5.2(b). Similarly, if $G(p, \ell)$ denotes the set of vertices on the upper boundary of $\Psi(\varepsilon)$ that lie between ℓ and the vertical line through p, then the upper side of $\sigma(p, \varepsilon)$ is tangent to the lower hull of $G(p, \ell)$. The following lemma shows how many times the lower hull of a set of points changes when all of them move vertically upwards; by symmetry, the lemma also applies to the number of changes to the upper hull of points moving downwards.

Lemma 5.7 *Suppose n points in the plane move vertically upward, each with its own constant velocity. Then the number of combinatorial changes to the lower hull is $O(n)$. Furthermore, all event times at which the lower hull changes can be computed in $O(n \log^3 n)$ time.*

Proof. Let $\{p_1, \dots, p_n\}$ be the set of points moving vertically upwards with constant velocities, ordered from left to right. Since the points move with constant velocities, any three points become collinear at most once. As in the proof of Lemma 5.6, this implies that once a point disappears from the lower hull, it cannot re-appear. Hence, the number of changes to the lower hull is $O(n)$.

To compute all event times, we construct a balanced binary tree $\mathcal{T}$ storing the points $\{p_1, \dots, p_n\}$ in its leaves in an ordered manner based on their x-coordinates. At each node ν of $\mathcal{T}$, we maintain a kinetic data structure to track $\text{LH}(\nu)$, the lower hull of the points stored in the subtree rooted at ν. Let ν_r and ν_l be the right and left child of node ν in $\mathcal{T}$. Then $\text{LH}(\nu)$ is formed by portions of $\text{LH}(\nu_r)$ and $\text{LH}(\nu_l)$ and the common tangent of $\text{LH}(\nu_r)$ and $\text{LH}(\nu_l)$. This implies that in order to track all changes to the lower hull of the whole point set, it suffices to track for each node ν the changes to common tangent of $\text{LH}(\nu_r)$ and $\text{LH}(\nu_l)$. We thus maintain for each node ν a certificate that can be used to detect when the tangent changes. This certificate involves at most six points: the two points determining the current tangent and the at most four points (two on $\text{LH}(\nu_r)$ and two on $\text{LH}(\nu_l)$) adjacent to these points. The failure times of the $O(n)$ certificates are put into an event queue. When a certificate fails we update the corresponding tangent, and we update the failure time of the certificate (which means updating the event queue). A change at ν may propagate upwards in the tree—that is, it may trigger at most $O(\log n)$ changes in ascendants of ν. Hence, handling a certificate failure takes $O(\log^2 n)$ time. Since the number of changes at each node ν is the number of points stored at the subtree rooted at ν, we handle at most $O(n \log n)$ events in total. Each event takes $O(\log^2 n)$ time, and so we can compute all events in $O(n \log^3 n)$ time. □

Our goal is to shrink the active interval $\mathcal{I}$ to a smaller interval such that the visibility cones of the points are stable, that is, do not change combinatorially. Doing this for each p individually will still be too slow, however. We therefore proceed as follows.

Recall that we split Ψ into two with a vertical line ℓ. Let $\mathcal{R}$ be the set of vertices to the right of ℓ. We show how to shrink $\mathcal{I}$ so that the cones of the points $p \in \mathcal{R}$ are stable. Below we only consider the top sides of the cones, which pass through a vertex of $G(\varepsilon)$; the bottom sides can be handled similarly.

We construct a binary tree $\mathcal{T}_{G,\mathcal{R}}$ on the points in $G(\varepsilon) \cap \mathcal{R}$, based on their x-coordinates. For a node ν, let $P(\nu)$ denote its canonical subset, that is, $P(\nu)$ denotes the set of points in the subtree of ν. Using Lemma 5.7 we compute all event times—that is, values of ε—at which the lower hull $\text{LH}(P(\nu))$ changes, for each node ν. This takes $O(n \log^4 n)$ time in total and gives us a set $\mathcal{S}$ of $O(n \log n)$ event times. We then call procedure *ShrinkActiveInterval*($\mathcal{S}$, $\mathcal{I}$) to further shrink $\mathcal{I}$, taking $O(n \log^2 n + m \log n)$ time. In the new active interval, none of the maintained lower hulls changes combinatorially. This does not mean, however, that the top sides of the cones are stable. For that we need some more work.

Recall that the top side of $\sigma(p, \varepsilon)$ is given by the tangent of p to $\text{LH}(G(p, \ell))$, where $G(p, \ell)$ is the set of points on the $G(\varepsilon)$ in between p and ℓ (with respect to their x-coordinates). The set $G(p, \ell)$ can be formed from $O(\log n)$ canonical subsets in $\mathcal{T}_{G,\mathcal{R}}$. Each canonical subset $P(\nu)$ gives a candidate tangent for p, namely the tangent from p to $\text{LH}(P(\nu))$. Even though the lower hulls, $\text{LH}(P(\nu))$, are stable, the tangents from p to the lower hulls are not. Next we describe how to shrink the active interval, so that these tangents become stable, and we have $O(\log n)$ stable candidates.

Consider a canonical subset $P(\nu)$ and let $p_1, \ldots, p_h$ be the vertices of $\text{LH}(P(\nu))$, ordered from left to right. An important observation is that, as ε increases and the points move, the tangent from p to $\text{LH}(P(\nu))$ steps from vertex to vertex along $p_1, \ldots, p_h$, either always going forward or always going backwards. (This is true because p can become collinear with any lower-hull edge at most once.) We can therefore proceed as follows. For each point p and each of its canonical subsets, we compute in constant time at what time p and the middle edge of the lower hull of the canonical subset become collinear. Finding the middle edge can be done using binary search, if we store the lower hulls $\text{LH}(P(\nu))$ as a balanced tree. Since we have n points and each of them is associated with $O(\log n)$ canonical subsets, in total we have $O(n \log n)$ event times. We put these into a set $\mathcal{S}$ and call *ShrinkActiveInterval*($\mathcal{S}, \mathcal{I}$). In the new active interval the number of vertices of each lower hull to which p can be tangent has halved. We keep on shrinking $\mathcal{I}$ recursively, until we are left with an interval $\mathcal{I}$ such that, for each p and any canonical subset relevant for p, the tangent from p to the lower hull is stable. In total this takes $O(n \log^2 n + m \log n)$ time.

Note that within $\mathcal{I}$ we now have $O(\log n)$ stable candidate tangent lines for each p. We then compute all $O(\log^2 n)$ times at which the candidate tangent lines swap (in their circular order around p), collect all $O(n \log^2 n)$ event times, and call *ShrinkActiveInterval* once more, taking $O(n \log^3 n + m \log n)$ time.

After this, we are left with an interval $\mathcal{I}$ such that the top side of the cone of each $p \in \mathcal{R}$ is stable. In a similar way we can make sure that the bottom sides are stable, and that the top and bottom sides of the points to the left of ℓ are stable. We get the following lemma.

Lemma 5.8 *In $O(n \log^3 n + m \log n)$ time we can shrink the active interval $\mathcal{I}$ so that in the new active interval all the cones defined with respect to ℓ are stable.*

We denote the set of edges of $\mathcal{G}$ crossing ℓ by $E(\ell)$. Next we describe a randomization algorithm to shrink the active interval $\mathcal{I}$ such that in the new active interval, the edges of $E(\ell)$ are stable. After this, we recurse on the part of Ψ to the left of ℓ and on the part to the right, so that the whole visibility graph becomes stable.

As already mentioned, (p, q) is an edge of $E(\ell)$ if and only if $p \in \sigma(q, \varepsilon)$ and $q \in \sigma(p, \varepsilon)$. Thus $E(\ell)$ changes when a point p becomes collinear with a side of $\sigma(q, \varepsilon)$ for some q. Without loss of generality we assume $p \in \mathcal{R}$ and $q \in \mathcal{L}$. Thus we have a set $\mathcal{H}$ of at most n half-lines originating from points in $\mathcal{L}$, where each half-line is specified by two points of $\mathcal{L}$, and a set of $n/2$ points from $\mathcal{R}$, and we want to compute the event times at which a point of $\mathcal{R}$ and a half-line of $\mathcal{H}$ become collinear. Again, explicitly enumerating all these event times takes $\Omega(n^2)$ time, so we have to proceed more carefully. To this end we use a variant of *random halving* [62], which can be made deterministic using the expander approach [54]. We start with a primitive tool which is used in our algorithm.

Lemma 5.9 *For any ε_1 and ε_2, and any $\delta > 0$, we can preprocess $\mathcal{H}$ and $\mathcal{R}$ in $O(n^{4/3+\delta})$ time into a data structure that allows the following: count in $O(n^{4/3+\delta})$ time all events (that is, all the times at which a half-line in $\mathcal{H}$ and a point in $\mathcal{R}$ become collinear) lying in $[\varepsilon_1, \varepsilon_2]$, and select in $O(\log n)$ time one of these events uniformly at random.*

Proof. (Sketch of proof) Consider a half-line $l \in \mathcal{H}$ and a point $p \in \mathcal{R}$. They become collinear at a time in $[\varepsilon_1, \varepsilon_2]$ if and only if either $p(\varepsilon_1)$ is below $l(\varepsilon_1)$ and $p(\varepsilon_2)$ is above $l(\varepsilon_2)$, or $p(\varepsilon_1)$ is above $l(\varepsilon_1)$ and $p(\varepsilon_2)$ is below $l(\varepsilon_2)$. Therefore we need a data structure to report for all $l \in \mathcal{H}$ the points of $\mathcal{R}$ lying to a given side of $l(\varepsilon_1)$ or $l(\varepsilon_2)$. We construct a multilevel partition tree over points of $\mathcal{R}$ at times ε_1 and ε_2 (one level dealing with ε_1, the other dealing with ε_2), each of whose nodes is associated with a canonical subset of $\mathcal{R}$. The total size of all canonical subsets is $O(n^{4/3+\delta})$. For a query line l, the query procedure selects $O(n^{1/3+\delta})$ pairwise disjoint canonical subsets whose union consists of exactly those points of $\mathcal{R}$ at different sides of l at times ε_1 and ε_2. Based on this we create a set of pairs $\{(A_i, B_i)\}$ with $\sum(|A_i| + |B_i|) = O(n^{4/3+\delta})$ where A_i is a subset of $\mathcal{R}$ and B_i is the subset of $\mathcal{H}$ and each $p \in A_i$ and $l \in B_i$ become collinear at some time in $[\varepsilon_1, \varepsilon_2]$. First we select a pair (A_i, B_i) at random, where the probability of selecting (A_i, B_i) is proportional to $|A_i| * |B_i|$. Then we select an element uniformly at random from A_i and an element uniformly at random from B_i. $\square$

Based on Lemma 5.9 we proceed as follows. Let $\mathcal{I} = [\varepsilon_1, \varepsilon_2]$ be the current active interval. Let N be the number of event times in $\mathcal{I}$; we can determine N using Lemma 5.9. Then select an event time ε_3 uniformly at random from the event times in $\mathcal{I}$, again using Lemma 5.9, and we either shrink $\mathcal{I}$ to $[\varepsilon_1, \varepsilon_3]$ or to $[\varepsilon_3, \varepsilon_2]$ in $O(m)$ time. We recursively continue shrinking $\mathcal{I}$ until the set of events inside $\mathcal{I}$ is $O(n^{4/3})$; the expected number of rounds is $O(\log n)$. Once $N = O(n^{4/3})$ we list all event times, and do a regular binary search.

After this we are left with an active interval $\mathcal{I}$ such that the set $A(\ell)$ of visibility edges crossing ℓ is stable. We recurse on both halves of Ψ to get the whole visibility graph stable. Putting everything together we get the following result.

Lemma 5.10 *For any $\delta > 0$ in $O(n^{4/3+\delta} + m \log m)$ expected time we can compute an active interval $\mathcal{I}$ containing ε^* where the visibility graph $\mathcal{G}(\varepsilon)$ is stable.*

As observed before, the fact that the visibility graph is stable during the active interval $\mathcal{I} = [\varepsilon_1, \varepsilon_2]$ implies that the set of inflection edges is stable. This means we can compute all inflection edges during this interval by computing in $O(n)$ time the shortest paths $\pi_u(\varepsilon_1)$ and $\pi_g(\varepsilon_1)$. Once this has been done, we can proceed in exactly the same way as Goodrich [44] to find ε^*. Given ε^* we can find a k-link path of error ε^* by solving the min-k problem for ε^*. This leads to our main result.

Theorem 5.11 *Let $\mathrm{F}\colon \mathbb{R} \to \mathbb{R}$ be an uncertain function whose values are given at n points $\{x_1, \dots, x_n\}$ and let m be the total number of possible values at these points. For a given k, and any $\delta > 0$, we can compute in $O(n^{4/3+\delta} + m \log n)$ time a piecewise-linear function $\overline{\mathrm{F}}$ with at most k links that minimizes error$(\mathrm{F}, \overline{\mathrm{F}}) \leqslant \varepsilon$.*

Chapter 6

Concluding Remarks

In this thesis we studied the design of algorithms for making optimal geometric (data) structures. In particular, we studied three structures from this perspective: BSP trees (a widely used space-partitioning structure), rectilinear r-partitions (which can be used to design R-trees), and polygon decompositions (both for arbitrary simple polygons and for rectilinear simple polygons). We considered the optimality of these structures with respect to size (for BSPs) and stabbing number (for rectilinear r-partitions and polygon decompositions). All of these data structures have been studied extensively before and several worst-case optimal algorithms have been proposed for them. However, the existing algorithms did not guarantee solutions that are optimal for the given input. Hence, we decided to study them from a different point of view, namely instance-optimality. In other words, we view the construction of these structures as an optimization problem. The main question we tried to answer in this thesis was the following: given a specific geometric structure (like BSP, rectilinear r-partition, $\dots$) and a cost function (like size, crossing number, $\dots$), is it possible to find an algorithm that, given an input instance, constructs a data structure which is optimal or close to optimal with respect to that input instance? In other words, instead of worst-case optimal data structures which have been proposed for all of these data structures, we tried to find instance-optimal data structures. Below we summarize our contributions in this domain, and discuss some directions for further research.

In Chapter 2 of the thesis, we started our research on finding optimal geometric data structures by considering BSPs. We showed that finding an optimal auto-partition is NP-hard, but it is still open if finding an optimal free or restricted BSP is NP-hard or not. Our NP-hardness proof also showed that even to find out if a set of segments admits an auto-partition without cuts is NP-hard. This is not true for free and restricted BSPs, where we can decide in $O(n^2)$ if a BSP without cuts exists for a set of segments or not. This implies that to prove NP-hardness of finding optimal free or restricted BSPs we may need to follow another approach. The main question that we tried to answer is still open: is it possible to approximate an optimal auto-partition for a set of line segments in the plane? The same question can be considered for other types of BSPs. We also studied the relation between the number of cuts in an optimal free BSP and an optimal restricted BSP for a set of line segments in the plane. Our study shows that the number of cuts in an optimal restricted BSP is at most twice the number of cuts in an optimal free BSP. Thus, when searching for an approximation to compute optimal BSPs one can focus on finding optimal restricted BSPs and this gives a 2-approximation for making free BSPs. It might be interesting to find a relation between the number of cuts in an optimal auto-partition to the number of cuts in an optimal free or restricted BSP.

It is known that computing an optimal rectilinear BSP for a set of axis-parallel segments in the plane can be done using dynamic programming in $O(n^5)$ time [19]. An interesting question is whether there are other restricted input sets for which one can compute an optimal BSP in polynomial time. For example, can we efficiently compute an optimal auto-partition for a set of c-oriented line segments? and what is the complexity of computing an optimal BSP for a set of disks. Another direction of further research is to consider a different optimization criterion. For example, what is the complexity of computing a BSP of minimum depth? Again, for the axis-parallel case this can be done in $O(n^5)$ time [19], but for the general case this problem has not been studied so far. Considering BSPs of low stabbing number would be another natural problem. Finally, computing optimal BSPs can of course also be studied in 3- and higher-dimensional space.

After studying BSPs, we turned our attention to rectilinear r-partitions in Chapter 3 of the thesis. Again the main goal was to come up with an instance-optimal algorithm. In this case the criterion that we wanted to optimize was the (rectilinear) stabbing number of the partition. We proved that the problem for a set of n points is NP-hard when r is considered as part of the input. Our proof shows that finding a constant approximation factor less than 6/5 is also impossible unless P = NP. We also proposed an exact algorithm with polynomial running time for the case when r is a constant. we gave a faster 2-approximation algorithm for constant r, but unfortunately even this algorithm is too slow to be of practical use. Thus, it would be interesting to find a faster approximation algorithm. Another interesting question is whether a PTAS exists for constant r. (As mentioned above, for general r the problem cannot be approximated with 6/5 unless P=NP.) We showed that considering rectilinear r-partitions with only disjoint rectangles does not give a good approximation factor, so the disjoint rectilinear r-partitions should not be considered for searching an approximation or PTAS. We also performed an experimental investigation on the problem of finding optimal rectilinear r-partitions. In particular, we tested four different heuristics. The result turned out to be that a new kd-tree variant, which we called the windmill kd-tree gives the best results. However, the fact that this windmill kd-tree results in good R-trees in practice is still unclear. The windmill kd-tree tries to minimize the stabbing number of axis-aligned lines. In practice, however, the query regions are not full lines but rectangles. The boundary of these rectangles is formed by horizontal and vertical segments, and the length of these segments is short compared to the input set. So it might be that the windmill kd-tree works well for long and skinny rectangles, but its performance for a set of fat query regions is unclear and needs more investigation. Another open and interesting problem is to try to find optimal rectilinear r-partitions with respect to a set of given query regions. Then, we can try to minimize the maximum or average stabbing number. We can also try to minimize the stabbing number of a complete R-tree, instead of one level, but it seems to be more difficult. The rectilinear r-partitions are a special case of simplicial partitions. Nothing is known about the complexity of finding an optimal simplicial partition. Obviously, finding the optimal simplicial partition or an approximation for it, is an interesting topic for future research.

Next, we considered the problem of partitioning a simple polygon. We studied rectangular partitions for rectilinear polygons and Steiner triangulations for simple polygons. Here the criterion to be optimized was the stabbing number of the resulting partition. We proposed a 3-approximation for finding optimal rectangular partitions for rectilinear polygons. Our algorithm is based on finding an optimal partition for a histogram. Then we gave an $O(1)$-approximation for Steiner triangulation of a simple polygon. We did not manage to find a polynomial-time exact algorithm for the problem, but we could not find an NP-hardness proof either. Thus, the first open problem would be to find out the complexity of these problems. Our approach to find $O(1)$-approximation for optimal Steiner triangulation is by partitioning the simple polygon into a set of edge visible polygons. After that we then find good Steiner triangulations for each of these edge visible polygons. Thus, an open problem is to try to find optimal or close to optimal Steiner triangulations for these edge visible polygons. Another important open problem in this area is solving both of these problems for the case of polygons with holes. Proving NP-hardness of the problem in the presence of holes could be easier than the case of simple polygons. As mentioned above, we have given a polynomial-time optimal algorithm for the case of histograms. Another direction of research could be to try and extend the class of polygons for which we have a polynomial-time solution, for example to x-monotone rectilinear polygons or orthoconvex rectilinear polygons.

Finally, we considered another type of optimization problems: how to best approximate a function F with another function with fewer links. This problem has been considered in several papers before. However, we were (as far as we know) the first to study the problem in uncertain model where we are given a set of values $x_1, \ldots, x_n$—the x-coordinates of the breakpoints of the function—and for each x_i we are given a set $y_{i,1}, \ldots, y_{i,m_i}$ of potential values for $\mathrm{F}(x_i)$ together with the associated probabilities $p_{i,j}$. We solved both min-k and min-ε problems for this uncertain model. An interesting question is the following: given an $\varepsilon > 0$ and an uncertain function F defined over a set $X = \{x_1, \ldots, x_n\}$, whether it is possible or not to find a subset $Q \subset X$ of size depending on ε such that for any line ℓ, *error*(F, ℓ) is at most $(1 + \varepsilon)$ times error *error*(F, ℓ) over set Q. In other words an important open problem is to find out whether exists a core-set of X or not. Another interesting direction of further research is to solve same problem for the case where the approximation function is not required to be piecewise linear but some (low-degree) polynomial.

References

[1] M. A. Abam, M. de Berg, P. Hachenberger, and A. Zarei. Streaming algorithms for line simplification. In *Proc. of the 23rd annual Symposium on Computational Geometry*, pages 175–183, 2007.

[2] P. Afshani, B. Jérémy, and T. M. Chan. Instance-optimal geometric algorithms. In *Proc. of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 129–138, 2009.

[3] P. K. Agarwal, B. Aronov, and S. Suri. Stabbing triangulations by lines in 3d. In *Proc. 11th Symposium on Computational Geometry*, pages 267–276, 1995.

[4] P. K. Agarwal, M. de Berg, J. Gudmundsson, M. Hammar, and H. J. Haverkort. Box-trees and r-trees with near-optimal query time. In *Proc. of the 17th Annual Symposium on Computational Geometry*, pages 124–133, 2001.

[5] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry*, pages 1–56, 1997.

[6] P. K. Agarwal, E. F. Grove, T. M. Murali, and J. S. Vitter. Binary space partitions for fat rectangles. *SIAM Journal of Computing*, 29(5):1422–1448, 2000.

[7] P. K. Agarwal, S. Har-Peled, N. H. Mustafa, and Y. Wang. Near-linear time approximation algorithms for curve simplification. In *Proc. of the 10th Annual European Symposium on Algorithms*, pages 29–41, 2002.

[8] P. K. Agarwal and S. Suri. Surface approximation and geometric partitions. *SIAM Journal of Computing*, 27:1016–1035, 1998.

[9] P. K. Agarwal and K. R. Varadarajan. Efficient algorithms for approximating polygonal chains. *B. Chazelle, J. Goodman, and R. Pollack (eds.), Discrete & Computational Geometry*, 23:273–291, 2000.

[10] P. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1st edition, 1993.

[11] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal on Computational Geometry and Applications*, 5:75–91, 1995.

[12] F. d' Amore and P. G. Franciosa. On the optimal binary plane partition for sets of isothetic rectangles. *Information Processing Letters*, 44:255–259, 1992.

[13] L. Arge, M. de Berg, H. J. Haverkort, and K. Yi. The priority r-tree: a practically efficient and worst-case optimal r-tree. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 347–358, 2004.

[14] D. Arthur and S. Vassilvitskii. K-means++: the advantages of careful seeding. In *Proc. of the 18th Annual ACM-SIAM Symp. of Discrete Algorithms*, pages 1027–1035, 2007.

[15] M. de Berg. Linear size binary space partitions for uncluttered scenes. *Algorithmica*, 28(3):353–366, 2000.

[16] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer Verlag, Berlin, 3rd edition, 2008.

[17] M. de Berg, M. M. de Groot, and M. H. Overmars. Perfect binary space partitions. *Computational Geometry: Theory and Applications*, 7(1-2):81–91, 1997.

[18] M. de Berg, J. Gudmundsson, M. Hammer, and M. Overmars. On r-trees with low stabbing number. In *Proc. of the 8th Annual European Symposium on Algorithms*, pages 167–178, 2000.

[19] M. de Berg, E. Mumford, and B. Speckmann. Optimal BSPs and rectilinear cartograms. In *Proc. of the 14th Annual ACM international Symposium on Advances in Geographic Information Systems*, pages 19–26, 2006.

[20] M. de Berg and M. van Kreveld. Rectilinear decompositions with low stabbing number. *Information Processing Letters*, 52(4):215–221, 1994.

[21] J. D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, Cambridge, 1st edition, 1998.

[22] W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments. In *Proc. 3rd Annual Symposium on Algorithms and Computing*, pages 378–387, 1992.

[23] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Computational Geometry*, 6(5):485–524, 1991.

[24] B. Chazelle, H. Edelsbrunner, and L. J. Guibas. The complexity of cutting complexes. *Discrete Computational Geometry*, 4(2):139–181, 1989.

[25] B. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Algorithmica*, 8:407–429, 1992.

[26] B. Chazelle and E. Welzl. Quasi-optimal range searching in spaces of finite vc-dimension. *Discrete Computational Geometry*, 4(5):467–489, 1989.

[27] N. Chin and S. Feiner. Near real-time shadow generation using bsp trees". *Computer Graphics*, 23:99–106, 1989.

[28] X. Clairbois. *On Optimal Binary Space Partitions*. Msc. thesis, Department of Mathematics and Computing Science, TU Eindhoven, Eindhoven, Netherlands, 2006.

[29] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. S. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[30] E. D. Demaine, D. Harmon, J. Iacono, D. Kane, and M. Pătraşcu. The geometry of binary search trees. In *Proc. of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 496–505, 2009.

[31] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *Proc. of the 11th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 743–752, 2000.

[32] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[33] A. Dumitrescu, J. S. B. Mitchell, and M. Sharir. Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles. In *Proc. of the 17th Symposium on Computational Geometry*, pages 141–150, 2001.

[34] H. Edelsbrunner. Biological applications of computational topology. In J. E. Goodman and J. OŔourke, editors, *Handbook of Discrete and Computational Geometry*, pages 1395–1412. CRC Press LLC, Boca Raton, FL, 2004.

[35] H. Edelsbrunner and E. Welzl. Halfplanar range search in linear space and $o(n^{0.695})$ query time. *Information Processing Letter*, 23:289–293, 1986.

[36] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '01)*, pages 102–113, 2001.

[37] C. Faloutos and I. Kamel. Packed r-trees using fractals. In *Report CS-TR-2009, University of Maryland, College Park*, 1992.

[38] C. Faloutsos, T. Sellis, and N. Roussopoulos. Analysis of object oriented spatial access methods. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, pages 426–439, 1987.

[39] S. P. Fekete, M. E. Lübbecke, and H. Meijer. Minimizing the stabbing number of matchings, trees, and triangulations. In *Proc. of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 437–446, 2004.

[40] H. Fuchs, Z. M. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Computer Graphics*, 14:124–133, 1980.

[41] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30:170–231, 1998.

[42] M. R. Garey and D. S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., New York, 1st edition, 1979.

[43] J. E. Goodman and J. OŔourke. *Handbook of Discrete and Computational Geometry*. CRC Press LLC, Boca Raton, FL, 2nd edition, 2004.

[44] M. T. Goodrich. Efficient piecewise-linear function approximation using the uniform metric. *Discrete & Computational Geometry.*, 14:445–462, 1995.

[45] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. of SIGMOD'84 Annual Meeting, Boston, Massachusetts*, pages 47–57, 1984.

[46] S. L. Hakimi and E. F. Schmeichel. Fitting polygonal functions to a set of points in the plane. *Graph. Models Image Process.*, 52:132–136, 1991.

[47] D. Halperin, L. E. Kavraki, and J. C. Latombe. Robotics. In J. E. Goodman and J. OŔourke, editors, *Handbook of Discrete and Computational Geometry*, pages 1065–1093. CRC Press LLC, Boca Raton, FL, 2004.

[48] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.

[49] H. Haverkort and F. van Walderveen. Four-dimensional hilbert curves for R-trees. In *Proc. Workshop on Algorithms Engineering and Experiments (ALANEX'09)*, pages 63–73, 2009.

[50] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: shoot a ray, take a walk. *Journal of Algorithms*, 18(3):403–431, 1995.

[51] H. Imai and M. Iri. Polygonal approximations of a curve-formulations and algorithms. *G. T. Toussaint Ed., Computational Morphology.*, pages 71–86, 1988.

[52] M. Kail. Polygon decomposition. In J. R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 491–518. Elsevier Science B.V., 2000.

[53] B. Katz, I. Rutter, and G. Woeginger. An algorithmic study of switch graphs. In Christophe Paul and Michel Habib, editors, *Graph-Theoretic Concepts in Computer Science*, pages 226–237. Springer-Verlag, 2010.

[54] M. J. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM Journal of Computing*, 26:1384–1408, 1997.

[55] M. van Kreveld. Geographic information systems. In J. E. Goodman and J. OŔourke, editors, *Handbook of Discrete and Computational Geometry*, pages 1293–1314. CRC Press LLC, Boca Raton, FL, 2004.

[56] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1st edition, 1991.

[57] U. Lauther. 4-dimensional binary search trees as a means to speed up associative searches in design rule verification of integrated circuits. *Journal of Design Automation and Fault-Tolerant Computing*, 2(3):241–241, 1978.

[58] S. T. Leutenegger, J. M. Edgington, and M. A. Lopez. Str: A simple and efficient algorithm for r-tree packing. In *Institute for Computer Applications in Science and Engineering (ICASE)*, 1997.

[59] C. Levcopoulos. *Heuristics for Minimum Decompositions of Polygons*. Ph.D. thesis, Linkoping University, Linkoping, Sweden, 1987.

[60] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal of Computing*, 11(2):329–343, 1982.

[61] Y. Manolopoulos, A. Nanopoulos, Y. Theodoridis, and A. Papadopoulos. *R-trees: theory and applications*. Springer, 2005.

[62] J. Matoušek. Randomized optimal algorithm for slope selection. *Information Processing Lett.*, 36(3):183–187, 1991.

[63] J. Matoušek and G. Tarantello. Efficient partition trees. *Discrete Computational Geometry*, 8(3):315–334, 1992.

[64] A. Melkman and J. O'Rourke. On polygonal chain approximation. *G. T. Toussaint (ed.), Computational Morphology*, pages 87–95, 1998.

[65] J. Nievergelt and P. Widmayer. Spatial data structures: concepts and design choices. *M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer (eds.), Algorithmic Foundations of Geographic Information Systems*, 1340:153–198, 1997.

[66] J. OŔourke. *Computational Geometry in C (Second Edition)*. Cambridge University Press, New York, 2nd edition, 1998.

[67] M. S. Paterson and F. F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Computational Geometry*, 5(5):485–503, 1990.

[68] M. S. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. In *Proc. of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 100–106, 1990.

[69] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer Verlag, New York, 1st edition, 1985.

[70] J. R. Sack and J. Urrutia. *Handbook of Computational Geometry*. Elsevier Science, Amsterdam, the Netherlands, 1st edition, 2000.

[71] R. A. Schumacker, R. Brand, M. Gilliland, and W. Sharp. Study for applying computer-generated images to visual simulation. Technical report, 1969.

[72] M. Sharir. Algorithmic motion planning. In J. E. Goodman and J. OŔourke, editors, *Handbook of Discrete and Computational Geometry*, pages 735–753. CRC Press LLC, Boca Raton, FL, 2004.

[73] S. Suri. A linear time algorithm for minimum link paths inside a simple polygon. *Comput. Vision Graph. Image Process.*, 35(1):99–110, 1986.

[74] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. *Computer Graphics*, 21:153–162, 1987.

[75] C. D. Tóth. A note on binary plane partitions. In *Proc. of the 17th Annual Symposium on Computational Geometry*, pages 151–156, 2001.

[76] C. D. Tóth. Binary space partition for line segments with a limited number of directions. *SIAM Journal of Computing*, 32:307–325, 2003.

[77] C. D. Tóth. Binary space partitions: recent developments. *Combinatorial and Computational Geometry*, 52:525–552, 2005.

[78] C. D. Tóth. Axis-aligned subdivisions with low stabbing numbers. *SIAM Journal of Discrete Mathematics*, 22(3):1187–1204, 2008.

[79] C. D. Tóth. Stabbing numbers of convex subdivisions. *Periodica Mathematica Hungarica*, 57(1):217–225, 2008.

[80] C. D. Tóth. Binary plane partitions for disjoint line segments. In *Proc. 25th Symposium on Computational Geometry*, pages 71–79, 2009.

[81] G. T. Toussaint. On the complexity of approximating polygonal curves in the plane. In *Proc. International Symposium on Robotics and Automation.*, 1985.

[82] D. P. Wang, D. P. Huang, H. S. Chao, and R. C. T. Lee. Plane sweep algorithms for polygonal approximation problems with applications. In *Proc. 4th International Symposium on Algorithms and Computing*, 1993.

[83] Y. Zhou and S. Suri. Analysis of a bounding box heuristic for object intersection. In *Proc. 10th Annual Symposium on Discrete Algorithms*, pages 830–839, 1999.

Summary

Optimal geometric data structures

Spatial data structures form a core ingredient of many geometric algorithms, both in theory and in practice. Many of these data structures, especially the ones used in practice, are based on partitioning the underlying space (examples are binary space partitions and decompositions of polygons) or partitioning the set of objects (examples are bounding-volume hierarchies).

The efficiency of such data structures—and, hence, of the algorithms that use them—depends on certain characteristics of the partitioning. For example the performance of many algorithms that use binary space partitions (BSPs) depends on the *size* of the BSPs. Similarly, the performance of answering range queries using bounding-volume hierarchies (BVHs) depends on the so-called *crossing number* that can be associated with the partitioning on which the BVH is based. Much research has been done on the problem of computing partitioning whose characteristics are good *in the worst case*. In this thesis, we studied the problem from a different point of view, namely instance-optimality. In particular, we considered the following question: given a class of geometric partitioning structures—like BSPs, simplicial partition, polygon triangulations, ...—and a cost function—like size or crossing number—can we design an algorithm that computes a structure whose cost is optimal or close to optimal *for any input instance* (rather than only worst-case optimal). We studied the problem of finding optimal data structures for some of the most important spatial data structures.

As an example having a set of n points and an input parameter r, It has been proved that there are input sets for which any simplicial partition has crossing number $\Omega(\sqrt{r})$. It has also been shown that for any set of n input points and the parameter r one can make a simplicial partition with stabbing number $O(\sqrt{r})$. However, there are input point sets for which one can make simplicial partitions with lower stabbing number. As an example when the points are on a diagonal, one can always make a simplicial partition with stabbing number 1.

We started our research by studying BSPs for line segments in the plane, where the cost function is the size of the BSP. A popular type of BSPs for line segments are the so-called *auto-partitions*. We proved that finding an optimal auto-partition is NP-hard. In fact, finding out if a set of input segments admits an auto-partition without any cuts is already NP-hard. We also studied the relation between two other types of BSPs, called *free* and *restricted* BSPs, and showed that the number of cuts of an optimal restricted BSP for a set of segments in $\mathbb{R}^2$ is at most twice the number of cuts of an optimal free BSP for that set. The details are being represented in Chapter 2 of the thesis.

Then we turned our attention to the so-called *rectilinear r-partitions* for planar point sets, with the *crossing number* as cost function. A rectilinear r-partition of a point set P is a partition of P into r subsets, each having roughly $|P|/r$ points. The crossing number of the partition is defined using the bounding boxes of the subsets; in particular, it is the maximum number of bounding boxes that can be intersected by any horizontal or vertical line. We performed some theoretical as well as experimental studies on rectilinear r-partitions. On the theoretical side, we proved that computing a rectilinear r-partition with optimal stabbing number for a given set of points and parameter r is NP-hard. We also proposed an exact algorithm for finding optimal rectilinear r-partitions whose running time is polynomial when r is a constant, and a faster 2-approximation algorithm. Our last theoretical result showed that considering only partitions whose bounding boxes are disjoint is not sufficient for finding optimal rectilinear r-partitions. On the experimental side, we performed a comparison between four different heuristics for constructing rectilinear r-partitions. The so-called *windmill kd-tree* gave the best results. Chapter 3 of the thesis describes all the details of our research on rectilinear r-partition.

We studied another spatial data structure in Chapter 4 of the thesis. Decomposition of the interior of polygons is one of the fundamental problems in computational geometry. In case of a simple polygon one usually wants to make a Steiner triangulation of it, and when we have a rectilinear polygon at hand, one typically wants to make a rectilinear decomposition for it. Due to this reason there are algorithms which make Steiner triangulations and rectangular decompositions with low stabbing number. These algorithms are worst-case optimal. However, similar to the two previous data structures, there are polygons for which one can make decompositions with lower stabbing numbers. In Chapter 4 we proposed a 3-approximation for finding an optimal rectangular decomposition of a rectilinear polygon. We also proposed an $O(1)$-approximation for finding optimal Steiner triangulation of a simple polygon.

Finally, in Chapter 5 of the thesis, we considered another optimization problem, namely how to approximate a piecewise-linear function $\mathrm{F} : \mathbb{R} \rightarrow \mathbb{R}$ with another piecewise-linear function with fewer pieces. Here one can distinguish two versions of the problem. The first one is called the min-k problem; the goal is then to approximate the function within a given error ε such that the resulting function has the minimum number of links. The second one is called the min-ε problem; here the goal is to find an approximation with at most k links (for a given k) such that the error is minimized. These problems have already been studied before. Our contribution is to consider the problem for so-called *uncertain functions*, where the value of the input function F at its vertices is given as a discrete set of different values, each with an associated probability. We show how to compute an approximation that minimizes the expected error.

Curriculum Vitae

Amirali Khosravi was born on the 31st of October 1980 in Tehran, Iran. He graduated from the Tabriz branch of the National School for Development of Exceptional Talents in Mathematics and Physics in 1998. He received his Bachelor degree in computer engineering from Amirkabir University of Technology in 2003. Then, he got his Master degree in computer engineering from Sharif University of Technology in 2005. From Nov. 2007 to Sep. 2011 he was a Ph.D. candidate in the Algorithms Group of Department of Mathematic and Computer Science of the Technische Universiteit Eindhoven (TU/e).

Titles in the IPA Dissertation Series since 2005