EXPECTED-CASE PLANAR POINT LOCATION

by

THEOCHARIS MALAMATOS

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy
in Computer Science

September 2002, Hong Kong

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

THEOCHARIS MALAMATOS

EXPECTED-CASE PLANAR POINT LOCATION

by

THEOCHARIS MALAMATOS

This is to certify that I have examined the above Ph.D. thesis
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by
the thesis examination committee have been made.

DR. SUNIL ARYA, THESIS SUPERVISOR

PROF. LIONEL M. NI, HEAD OF DEPARTMENT

Department of Computer Science

4 September 2002

ACKNOWLEDGMENTS

I am most grateful to my supervisor, Sunil Arya, for sharing his expert knowledge with me and for introducing me to many exciting problems in computational geometry. Without doubt, I have been very fortunate in working with him.

I would like to thank the members of my committee, Siu-Wing Cheng, Rudolf Fleischer, Ajay Joneja and Ramesh Hariharan for their comments, which have improved the final result. I wish to thank Mordecai J. Golin for helping me in numerous ways, especially by teaching a course on approximation algorithms.

I was privileged to collaborate with David Mount. I have benefited greatly from his insightful observations and advice.

I am indebted to Dimitris Papadias and Yiannis Manolopoulos for persuading me to come to this university.

Many thanks to my friends and colleagues here; among them are Panos, Thanasis, Spiros, Mamoulis, Meretakis, Samee, Antoine, Gerhard, Hung, Hyeon-Suk, Xuerong, Yo Sub and Yuan Ping.

I thank Arista for the good times together. Lastly, I am thankful to my sister for her support and our discussions and to my father and mother for bringing me up with love and for always being there when I need them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXPECTED-CASE PLANAR POINT LOCATION

by

THEOCHARIS MALAMATOS

Department of Computer Science

The Hong Kong University of Science and Technology

ABSTRACT

Planar point location is one of the most important search problems in computational geometry. Given a set of non-overlapping polygons in the plane, the goal is to build a data structure for finding efficiently the polygon in which a query point lies. In terms of the worst-case query time, several methods are known for solving the problem optimally. We focus instead on optimizing the expected-case query time of planar point location. The analogous problem in one dimension is that of building optimal binary search trees.

We assume that we are given a set of non-overlapping polygons and for each polygon the probability that a query point lies in it. These probabilities define a probability distribution whose entropy we denote by H. By fundamental results of information theory, the expected query time must be at least H. We develop methods for answering planar point location queries in expected query time nearly matching the entropy lower bound, using close to linear space. In particular, the expected number of comparisons for locating the query point by our methods is at most H plus lower order terms. These results hold for polygons having a constant number of sides and convex polygons with uniform query distribution in their interiors. In special cases, such as planar subdivisions consisting of axis-parallel rectangles, we reduce the space to linear.

We also present a practical method that achieves $O(H)$ expected query time, which is optimal to within a small multiplicative factor.

CHAPTER 1

INTRODUCTION

Planar point location is among the most important two-dimensional problems in computational geometry. The problem is to preprocess a polygonal subdivision of the plane into a data structure so that the polygon containing a given query point can be reported efficiently. Apart from its theoretical significance, planar point location has important applications, notably in geographical information systems. For example, the map of a country can be divided into regions of similar pollution level, and the user may ask to know the pollution level at various locations.

A plethora of solutions that achieve optimal worst-case query time and optimal space have been formulated for this problem. The subject of our thesis is, in contrast, to examine planar point location from the perspective of the expected case. We assume that we are given the probability of the query point lying in each region and our goal is to construct a data structure that will minimize the *expected* query time. We refer to this problem as expected-case planar point location.

There is plenty of motivation for studying expected-case planar point location. First, it is a generalization of the fundamental problem of computing an optimal binary search tree for one-dimensional keys. In one dimension, the analogous problem asks, given a subdivision of the real line into intervals and the probability of the query number lying in each interval, to build a data structure (specifically, a search tree) with minimum expected query time.

Second, the way we look at planar point location, that is, to provide a solution that takes advantage of the information on the query distribution is relatively new in the field of geometric search problems. Previous work focusing on expected search performance restricted itself primarily to uniform query distributions. This work applies to any query distribution.

Third, there are practical reasons. Expected-case search time tends to be, for non-uniform distributions, much smaller than worst-case search time. We can

consider, for example, a situation where the majority of queries concern only a small number of regions; one would expect (and would desire) a performance depending on the complexity of these regions and not on the complexity of the whole subdivision. Given that non-uniform query distributions appear frequently in practice, expected-case point location algorithms promise to achieve significant reductions in real response query times.

It follows from Shannon's information-theoretic results that the expected query time cannot be less than the *entropy* defined by the probability distribution of visiting each polygon of the subdivision, which we call simply the entropy of the subdivision. We denote this entropy by H. In the one-dimensional case, it is well-known that $H + 2$ expected number of comparisons can be achieved. Our thesis's central goal is to explore whether planar point location is possible in expected number of comparisons whose higher order term is exactly H, while using nearly linear space. In addition, we investigate the existence of practical planar point location methods that provide $O(H)$ expected query time, which is optimal to within a multiplicative constant.

In the next section, we give a formal definition of planar point location and some elementary introduction. In Section 1.2, we define the expected-case version of the problem, show its connection to optimal binary search trees, and review previous work. In the last section of this chapter, we present our results and the outline of the thesis.

1.1 Planar Point Location

Given a planar polygonal subdivision S formed by n non-crossing line segments, the goal is to preprocess it into a data structure so that queries of the form in which polygon of the subdivision does a query point q lie can be answered efficiently. This is known as the *planar point location* problem. As mentioned before, it is a fundamental search problem in computational geometry and has several applications. (Polygons of S are also called *cells*.)

Throughout the thesis, we assume the comparison-based model. This means that to determine the position of a query point, we compare it with respect to a number of lines. For instance, a comparison of the query point $q = (4, 2)$ with

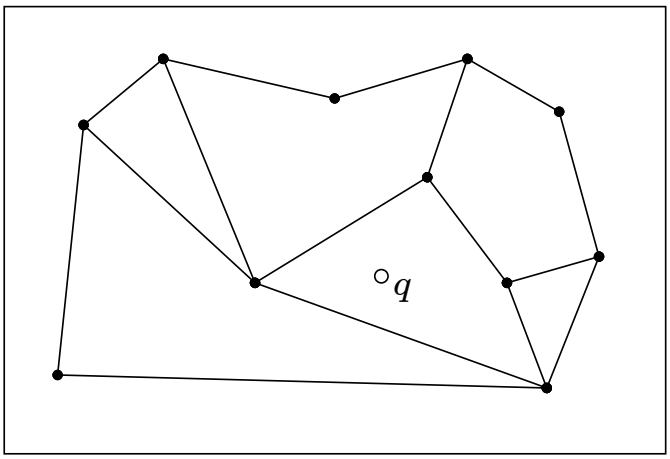


Figure 1.1 A planar subdivision and a sample query point.

the line ℓ defined by the equation $y = x - 3$, give us the information that q lies above ℓ (thus, the search can be limited among the cells in S which contain some point above the line ℓ). The comparison-based model is a standard model in the literature [32]. For simplicity, we will make the assumption that the query point does not lie on a segment of S.

A large part of the research in the problem has been on trying to minimize the worst-case query time, or in other words, to minimize the maximum number of comparisons required to locate any query point on the plane. Several methods are known that achieve optimal worst-case query time $O(\log n)$ in linear space. Moreover, it has been shown recently that the constant in the asymptotic notation of the worst-case query time can be reduced to one, which is optimal.

We provide more background on planar point location, especially for the unfamiliar reader, in Chapter 2.

1.2 Expected-Case Point Location

Expected-case planar point location, unlike worst-case planar point location, assumes that some *a priori* knowledge on the query distribution is available. We give the definition of the problem below. We will generally use the same notation as for the worst-case version.

1.2.1 Problem definition

We are given a planar polygonal subdivision S formed by n non-crossing line segments and for each cell $z \in S$, the probability p_z that a query point lies in z. We want to construct a data structure that will minimize the expected query time.

A useful observation is that, to every comparison-based method M for planar point location in S, there corresponds a decision tree T_M; thus, the expected query time, measured in terms of the number of comparisons, is equal to the weighted external path length of the decision tree T_M.

As in worst-case planar point location, we will assume that the probability that the query point lies on an edge or vertex of the subdivision is zero. We emphasize that other than the knowledge that a query point lies within cell z with probability p_z, we make no assumptions and assume no knowledge of the query distribution.

1.2.2 Connection to the one-dimensional case

This formulation for planar point location is a natural generalization of one of the best-known problems in the theory of data structures, namely that of constructing an *optimal binary search tree* for a set of keys from some totally ordered domain [17, 21]. In this problem, given an element, we search for the key that is equal to it, or if there is not such a key, we search for the "gap" between consecutive keys in which the element falls. Supposing we know the probabilities for all the keys and the gaps, an optimal binary search tree is a binary search tree that minimizes the expected cost of searching.

If we think of the keys as being real numbers then they naturally define a subdivision S_1 of the one-dimensional line into intervals. Following the geometric convention stated earlier, we assume that the probability that a query equals a key is zero. Thus the problem reduces to determining the interval between consecutive keys containing the query point. For each such interval y, let p_y denote the probability that a query point falls within this interval. In the binary search tree we build to answer such queries, each leaf corresponds to an interval

of S_1. The expected cost of searching can be expressed as

$$\sum_{y \in S_1} p_y d_y,$$

where d_y is the depth in the tree of interval y. The *entropy* of S_1, denoted H, is defined

$$\textit{entropy}(S_1) \;=\; H \;=\; \sum_{y \in S_1} p_y \log(1/p_y).$$

(Throughout all logarithms are taken base 2.) A classical result due to Shannon implies that the weighted external path length and thus the expected number of comparisons is at least as large as the entropy S_1 [21, 38].

The probability of accessing a key or an interval is referred to sometimes as the weight, thus optimal binary search trees are also called *weighted search trees*. Constructing an optimal binary search tree for n keys can be done in $O(n^2)$ time using a result by Knuth [20]. The bound can be reduced to $O(n \log n)$ [17] in the special case where the keys have access probability equal to zero. This category of optimal binary search trees is called optimal alphabetic trees. Although it is not known how to construct an optimal binary search tree or an optimal alphabetic tree in $o(n^2)$ or $o(n \log n)$ time, respectively, there are methods that run in linear time and construct a binary search tree with expected query time that is nearly optimal. These methods usually make use of simple rules (e.g., select the root so that the weight in the two subtrees is roughly balanced and then recurse) and also help to prove upper bounds on the optimal weighted path length. An example of such a method was given by Mehlhorn [24], who showed how to construct a binary search tree with expected query time at most $H + 2$. More information and references on the problem can be found in [29].

1.2.3 The entropy lower bound

By analogy with one dimension, we define the entropy of the planar subdivision S as

$$\textit{entropy}(S) \;=\; H \;=\; \sum_{z \in S} p_z \log(1/p_z).$$

where p_z is the probability of the query point lying in cell z. From the above discussion, it follows immediately that the entropy H of the subdivision is a lower bound on the expected query time of planar point location as well.

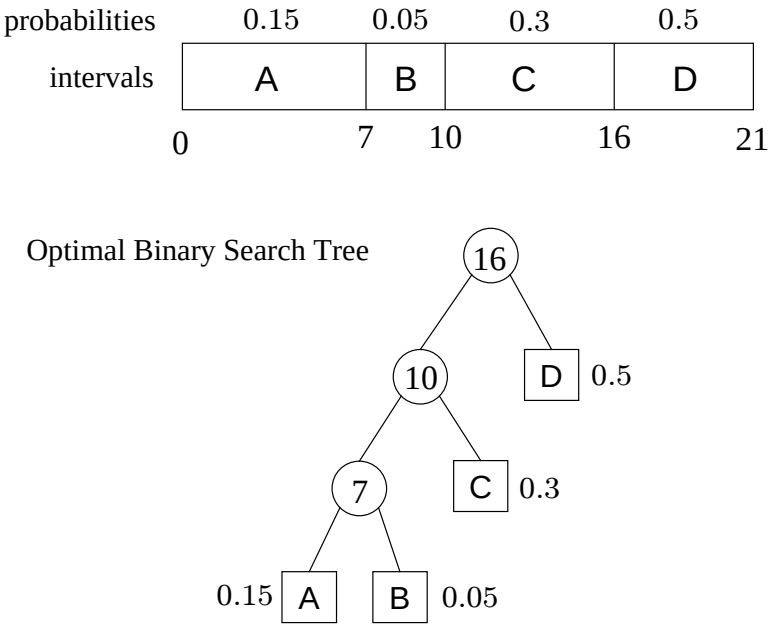


Figure 1.2 Example from one-dimension.

1.2.4 Previous work

There has been little work on expected-case planar point location, compared to the numerous results for the worst case. Goodrich, Orletsky and Ramaiyer [15] gave an adaptive point location method, which employs the splay tree by Sleator and Tarjan [39]. The data structure adjusts itself to the query distribution over the time, so that cells that receive queries more frequently are located faster. Given a number m of queries, it achieves for each cell z *amortized* query time that is at most $O(\min\{\log(m/f_z), \log(t_z+1), \log n\})$, where f_z is the frequency of accessing cell z and t_z the number of different cells that were accessed since the last request for cell z. The space occupied is linear. An advantage of the method is that no information on the query distribution is required beforehand. However, a serious drawback is that the cells in the expression above are not the cells of the initial subdivision S but the cells in the "slabbed" subdivision S (see

the slab method in Section 2.2.2). This implies that the average performance of the method in several cases may be far from optimal.

Recently, Arya et al. [3] showed that it is possible to answer point location queries in expected query time no more than $2H$, using quadratic space, and expected query time nearly $4H$ using linear space. A major limitation of these results is that they can be applied only if the coordinates of the query point are drawn from two independent probability distributions. The authors give first a solution to the problem when the query distribution is uniform, which uses a box-decomposition tree [8] of the vertices of S. For their more general result, they develop a mapping of the problem from the geometric space to the probability space and then they exploit the fact that the image of the query point in the probability space follows the uniform distribution.

Independently from the results of this thesis, Iacono [18] showed how to modify Kirkpatrick's point location method [19] to achieve expected query time bounded by $O(H)$, where H is the entropy of a triangular subdivision S of the plane. Although this method works for any query distribution and occupies linear space, it has the disadvantage that the constant factor hidden in the O-notation of the expected query time is large. As in the original method, the construction is done bottom-up in a hierarchical way starting with S. At each level a fraction of the vertices of the triangulation are removed leading to simpler triangulations until the triangulation has reached some constant complexity. Intuitively, the idea is to remove vertices that are not incident to regions of high access probability, thus ensuring that the triangles of S which are accessed frequently will be placed at the top levels of the hierarchy, where they can be reached fast.

There are also older methods, which have satisfactory average perfomance in many cases, as observed empirically. One method is the bucketing approach [13], essentially a two-dimensional hashing method. Other methods employ the jump-and-walk strategy, which starts from some chosen location and walks through the subdivision towards the query point [25]. Furthermore, several practical methods are known based on k-trees, quad-trees and R-trees [33]. In all these results, good upper bounds on the expected query time can be established analytically for uniform query distributions, or for subdivisions with uniformly distributed vertices but not in general.

1.2.5 Results and overview

To have any chance of achieving the entropy lower bound we will need to make some limiting assumptions on the complexity of the cells in the subdivision. Otherwise it will not generally be possible to bound the complexity of the search by any function of entropy alone (see also Section 3.6). This issue does not arise in the one-dimensional case, since intervals have bounded complexity. Our basic assumption is that each cell of the subdivision is bounded by a constant number of sides. For the same reasons we will also assume that the probability of the query point lying in the unbounded face of S is zero. Alternatively, we could assume that the unbounded face has bounded complexity.

We will present most of our results for cells of S that are triangles, but the extension to cells of bounded complexity is straightforward. Our results can also be extended to convex polygonal cells with an arbitrary number of sides, but to do so we add the assumption that the query distribution is uniform within each cell. Our main contribution is to show that it is possible to design a point location search structure such that the expected query time is nearly optimal, growing as $H + o(H)$. We note that entropy generally increases with n, so it is reasonable to treat H as an asymptotic quantity. (To simplify the expressions, we abuse notation by using $O(f(n))$ and $o(f(n))$ for some non-negative function $f(n)$ instead of $O(f(n)) + O(1)$ and $o(f(n)) + O(1)$, respectively.)

The results contained in this thesis are joint work with Sunil Arya and David M. Mount and have been published before in [4, 5, 6].

The thesis is organized as follows. Chapter 2 contains a detailed background on planar point location and reviews some of the worst-case methods that will prove to be useful for our results in the remainder of the thesis.

In Chapter 3, we address the question of whether the expected query time can match the entropy lower bound as is the case in one dimension. In addition to low query time, we seek to minimize the space occupied by the data structure. For planar subdivisions consisting of polygonal cells of fixed complexity, we present a method with expected query time bounded by $H + 2\sqrt{2H} + \log(\sqrt{H} + 1) + O(1)$ and $O(n2^{\sqrt{2H}} \log n/(\sqrt{H} + 1))$ space. The space in the above method is nearly linear. A different method we propose improves the above space to $O(n \log n)$

while increasing the expected query time only to $H+O(H^{2/3}+1)$. If the segments of the subdivision are further oriented along a constant number of directions, we show that the space can be reduced to $O(n)$. In addition, we establish for the expected query time a stronger lower bound than that of the entropy H. Specifically, we prove a lower bound of $H + \sqrt{H} - O(1)$. This new lower bound justifies the need for the $O(\sqrt{H})$ term in the expected query time of our first method.

In Chapter 4, we further investigate the question of whether the $H + o(H)$ expected query time can be achieved with linear space, in any subdivision of cells with bounded complexity. We come very close to this goal. In particular we show that if H is at least $\Omega(\log \log n)$, linear space is indeed possible. Without this assumption we show that the space can be reduced from $O(n \log n)$ to as low as $O(n \log^* n)$.

All the methods described above have the advantage of taking time that is optimal in terms of the constant in front of the entropy term. Nonetheless, they are quite complicated. In Chapter 5, we examine whether there is a practical method with low expected query time. We show that a simple weighted variant of the randomized incremental algorithm for point location achieves $O(H)$ expected query time with $O(n)$ space. We have run experiments demonstrating the practical efficiency of this algorithm and its superiority over standard approaches for non-uniform query distributions.

CHAPTER 2

BACKGROUND

In this chapter, we provide further background on the planar point location problem, focusing on worst-case algorithms. As we will see in later chapters, our expected-case methods borrow ideas from them. Readers having familiarity with the subject may choose to skip parts or the whole of this chapter.

2.1 Preliminaries

In planar point location, the query point is typically given by its two cartesian coordinates. The location of its position proceeds through a series of comparisons. A *comparison* is the decision of whether the query point lies above or below some oriented line of our choice. When the line is vertical the same question translates to whether the query point lies to the left or to the right of the line. Based on the result of such a comparison, a region of the plane is excluded as potential to contain the query point. After a number of appropriately chosen comparisons, the position of the query point is limited to a region that is entirely contained inside a single polygon. At that point, the label associated with that polygon can be safely reported as the answer to where the query point q lies. We will make the standard assumption that no query point lies on a segment of the subdivision or on a line of comparison.

To solve the problem for just one query point, $\Theta(n)$ comparisons would be enough and necessary in the worst case. However, given that the intention is to serve a large number of queries, it is beneficial to do some preprocessing so that afterwards the answer to any query can be computed as efficiently as possible. A quantitative measure of the efficiency is the number of comparisons required, referred to commonly as the *query time*. We note that examining the problem from the worst-case perspective assumes that no information is given during preprocessing about the location of the future queries.

Before giving more details about the planar point location problem, it is worth considering the analogous problem in one dimension. We are given a number of consecutive intervals and the goal is to report quickly, given a query number, to which of the intervals this number belongs. One trivial solution to this problem is to sort all the numbers that correspond to the endpoints of the intervals in an array and then, when some query arrives, perform a binary search with the given number on the array and locate the endpoint immediately above this number. With proper labeling, the interval containing the number can then also be identified. The query time for this problem is logarithmic in the number of intervals and by a simple information-theoretic argument this time can be proved to be optimal for any algorithm that uses only comparisons.

When comparing the various solutions for the planar point location problem the main focus is on the query time they provide. The asymptotically optimal worst-case query time is $O(\log n)$, as in the one-dimensional problem. A second important criterion is the space the data structure occupies. This should be ideally linear in the complexity of the subdivision S, that is $O(n)$. Another measure of comparison is the preprocessing time (and space) used to build the data structure. This tends to be a measure of less weight, since preprocessing is only performed once in the beginning. It is commonly in $O(n)$ or $O(n \log n)$ especially if the subdivision S is not given in some standard form or is disconnected. In recent years, there is particular concern, both theoretical and practical, on the exact number of comparisons required for planar point location. It is now considered important to report the constant hidden in the asymptotic notation of the query time of the algorithm and any lower order terms. Again for similar reasons, one is concerned with the multiplicative constant in the space, although good bounds on it are harder to obtain. Finally, the simplicity of the method is yet another criterion, which becomes highly relevant, if not critical, when practice is involved. All the above criteria apply to expected-case methods as well, although the optimal values may differ from the worst case.

Instead of the planar point location problem itself, the algorithms are sometimes stated for another closely related problem, the *vertical ray shooting* problem. In this problem, we are given n non-crossing line segments and a query point and we ask which segment will be hit first by a vertical ray starting from the query

point and going upwards (Figure 2.1). Having a solution for the vertical ray shooting problem for the segments of a subdivision S trivially gives a solution for the planar point location problem by simply labeling each segment with the region exactly below it.

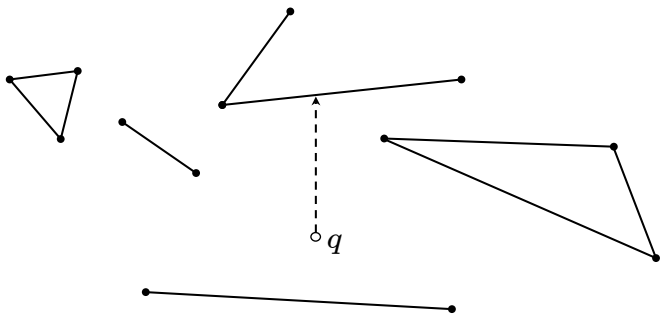


Figure 2.1 Vertical ray shooting.

When describing planar point location algorithms, some assumptions are usually tacitly made, such as that there is no vertical segment, no two endpoints of segments have the same x-coordinate, the segments are finite (i.e., no rays or lines are allowed) or that all the queries lie in a box, etc. These assumptions, introduced to prevent *degeneracies*, can be removed without much difficulty when needed [7].

Another view of the problem is to look at the subdivision S as a planar graph. In this context the terms used in place of segments and their endpoints are edges and vertices respectively. We note that by Euler's formula the number of vertices and faces (polygons) in the graph is $O(n)$. In some of the algorithms we will later present, it will be more natural to denote by n the number of vertices instead of segments. We will also occasionally find the following terminology useful. A (comparison) line is said to split the plane into two regions. Such a split is said to create new regions. The segments intersecting a (comparison) line are said to be cut by this line.

The use of comparisons for locating the query point gives rise to a *decision tree* [32]. An internal node of the decision tree represents a convex subregion of

the plane and is associated with a line. When an internal node is visited the query point is compared with this line and according to the result (positive or negative distance from the line) the appropriate link to the next node is followed. The external nodes represent regions where the location of a query point is completely determined. The root normally represents the entire plane. To answer a query, the search starts from the root and at each step a comparison is performed and the right path is followed until reaching a leaf where the result can be reported. The depth of the decision tree corresponds to the worst-case query time. On the other hand, the expected-case query time is equal to the weighted external path length of the decision tree.

A (correct) decision tree for point location is in effect a *binary space partition* (BSP) tree [7]. In a BSP for a set of non-intersecting objects, the space is recursively split into two by hyperplanes until any region contains a single object fragment. For our case, the objects are taken to be the faces of S (the segments work as well), while the space which becomes a plane is to be split by lines. There is a lot of independent interest in BSPs. The emphasis when studying them is on minimizing the total number of object fragments or, in other words, on building BSP trees of small size.

In order to reduce the space, many of the algorithms for planar point location exploit in an implicit manner the fact that different query points are compared during their search with the same sequence of lines. They use what is called *substructure sharing*. Given a subdivision S in this case, the output of the algorithm is a data structure that is not a tree but a directed acyclic graph or *DAG*. A recent result by Tóth [41] implies that substructure sharing is in fact necessary for general subdivisions with all data structures of linear size. More precisely, he proved that there exists a planar subdivision S for which any BSP tree must have size at least $\Omega(n \log n / \log \log n)$.

2.2 Related Work on Worst Case

2.2.1 Summary

It is striking that in spite of and in contrast to the uncomplicated solution for the analogous problem in one dimension, a solution for the planar point location that

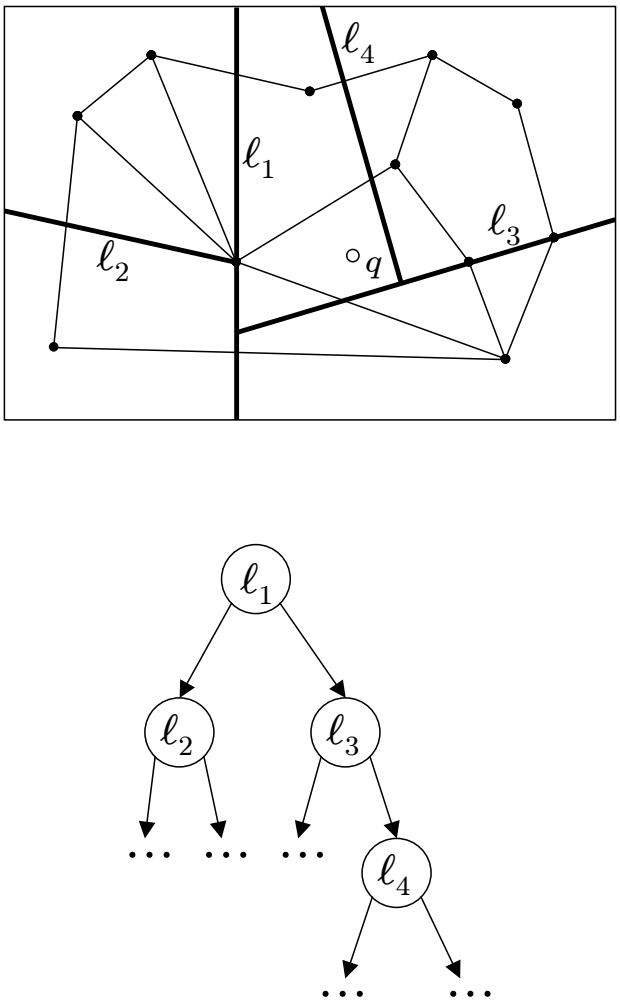


Figure 2.2 A subdivision and its BSP tree.

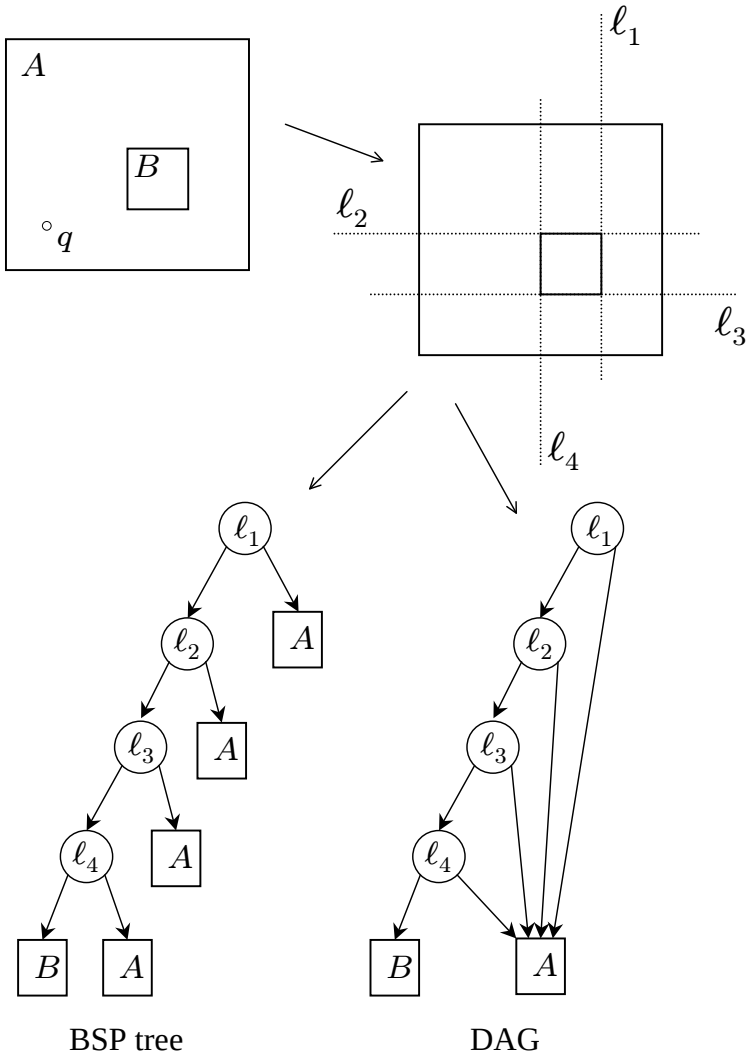


Figure 2.3 Substructure sharing. Does q lie in box A or box B? The DAG on the right leads to less space, especially if q was to be located further in smaller boxes lying inside A and outside B.

is optimal in the worst-case—that is, it has query time $O(\log n)$ and takes $O(n)$ space—is not obvious. In fact, it took some years to find one since the problem was first stated. Today, several optimal algorithms are known. They are based on a variety of ideas with each algorithm having its own beauty.

Dipton and Lipton [12] proposed the earliest algorithm for point location. It is known as the *slab* method and achieved $O(\log n)$ query time and $O(n^2)$ space. This was followed by the *chain* method of Lee and Preparata [22], which used linear space and $O(\log^2 n)$ time. Applying their planar separator theorem, Lipton and Tarjan [23] were the first to solve the problem optimally. However, their solution was very complicated and only of theoretical interest. Then came the *trapezoid* method by Preparata [30] that had $O(\log n)$ query time and $O(n \log n)$ space. Kirkpatrick [19] proposed a practical and, at the same time, optimal algorithm, based on *hierarchical* triangulations. A few years later, Edelsbrunner, Guibas and Stolfi [14], Cole [11] and Sarnak and Tarjan [34] gave optimal solutions characterized by small constants in front of the $\log n$ term of the query time. The first solution improved the chain method by using the technique of *fractional cascading* while the other two combined the slab method with the notion of *persistent* data structures. Simple randomized algorithms by Mulmuley [26, 27] and Seidel [35] appeared next. Goodrich, Orletsky and Ramaiyer [15] managed to bring the query time down to $2 \log n$ and used results on *cuttings* for reducing the space of nearly linear point location methods to linear without essentially affecting their query time. Recently, Seidel and Adamy [2, 36] showed that point location queries can be answered in $\log n + 2\sqrt{\log n} + o(\sqrt{\log n})$ time and linear space. This was the first result that showed that the query time can be as low as $\log n + o(\log n)$. The same authors also proved that the above bound is tight under the *trapezoidal search graph* model [36].

2.2.2 Slab method

Dobkin and Lipton's idea [12] is as follows. First, a vertical line through each vertex of the subdivision is drawn. These vertical lines partition the plane into $O(n)$ vertical strips called *slabs*. The key observation is that in each slab the segments crossing it form a total order by the "above" relation. As a consequence, after knowing in which slab the query point lies, the problem reduces to

the one-dimensional case, where the query point is said to be greater in a comparison against a segment if and only if it lies above the segment. Here is how a query point is located. Suppose that the vertices have been stored in an array in increasing order of their x-coordinates. Then using binary search in the array with the x-coordinate of the point, the slab that contains it is located. Another binary search follows to locate the segment of the slab that lies directly above the point. It is presumed that the different segments of each slab have been sorted and stored in the necessary arrays during the preprocessing.

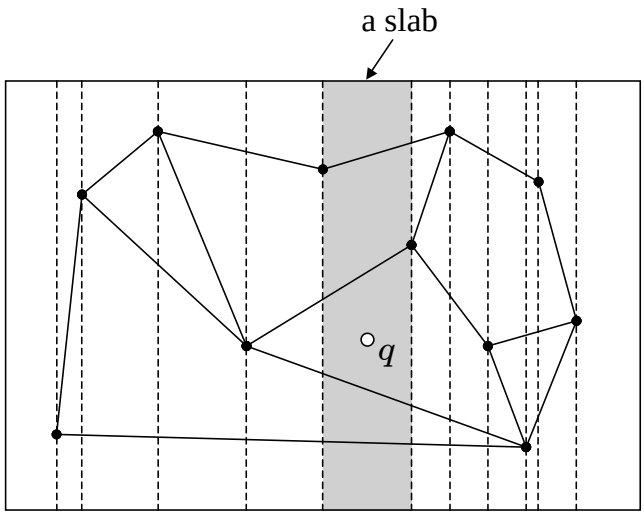


Figure 2.4 The slab method.

Since there are $O(n)$ slabs and a slab can have up to $O(n)$ crossing segments (there are n segments in total), the number of comparisons required by the two binary searches is totally not greater than $2 \log n$. (We shall omit possible additive constants in the query times.) The space consumed by the data structure is $O(n^2)$ which is tight in the worst-case. The huge space is the drawback of the slab method, which is otherwise straightforward. This method is still considered important since several other methods, including the most efficient known one, are based on it.

2.2.3 Trapezoid method

The trapezoid method by Preparata [30] is a recursive method with some ideas similar to the slab method. It has optimal query time and $O(n \log n)$ space.

The search area is considered to be bounded by a large enough trapezoid with its parallel edges aligned vertically. The algorithm selects as the first comparison the vertical line passing through the median vertex when the vertices of the subdivision are sorted according to increasing x-coordinate. This comparison creates two slabs. In each slab there are some edges that completely cross the slab, partitioning it into a number of vertically aligned trapezoids. Some of the trapezoids contain vertices *(nonempty)* while others do not *(empty)*. It is easy to search which one of the trapezoids in a slab contains the query point by comparing with the appropriate crossing edges of the trapezoids. Unlike the slab method, a balanced binary search tree with respect to the trapezoid edges is not good enough for this kind of search. As will be seen later, another more global balancing criterion is employed. If the query is found to be in an empty trapezoid, the search stops, otherwise it recurses in the located nonempty trapezoid. It is implied that the recursive construction was carried out in similar fashion on all the nonempty trapezoids during the preprocessing.

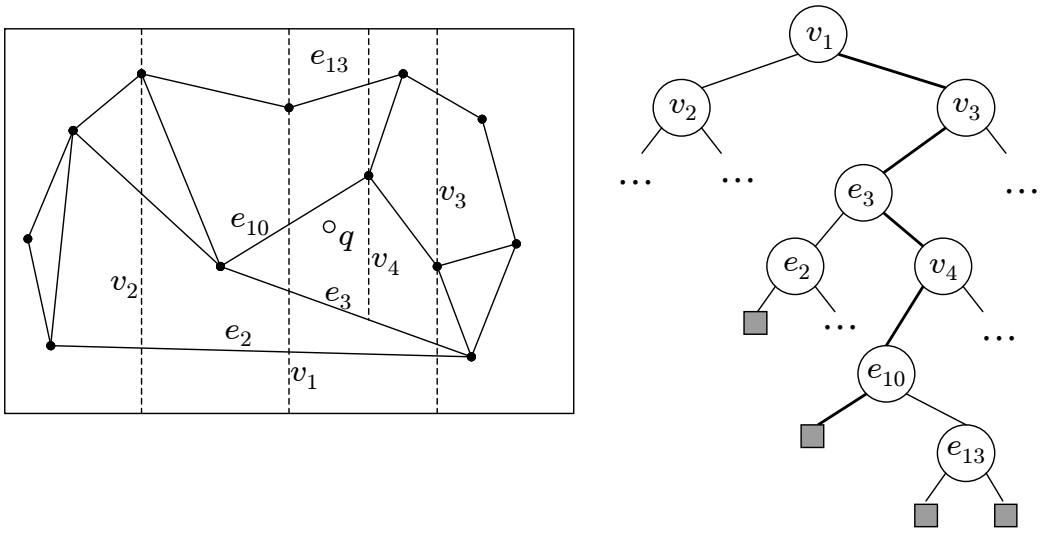


Figure 2.5 The trapezoid method. The query path is drawn in bold.

As at each recursion the number of vertices is halved, the number of recursions is bounded by $\log n$. Only the vertical comparisons are responsible for the increase in space, not the comparisons that use the trapezoid edges, which cause no fragmentation. The space can be bounded as follows. After the first cut is performed on a segment, a new cut can only happen to the part of segment delimited by one of its endpoints and the closest cut to this endpoint. Since a segment has two endpoints, this means that, at each level of recursion, only two cuts can be applied to a segment and therefore each segment is cut at most $2 \log n$ times. It follows that the total number of regions created and the total space are bounded by $O(n \log n)$.

We now consider the problem of which search tree to build to locate the trapezoid in each slab. One would want to spend few comparisons to locate the query point if it is contained in a trapezoid for which a large number of recursions are expected to follow. As an estimation of the further effort needed to locate the query point the number of endpoints in the current trapezoid is used. The search tree to build is therefore a weighted search tree [21, 24] with leaves associated with the trapezoids and where each trapezoid is assigned a weight equal to the number of vertices inside it. In the original paper, a different "manual" rebalancing approach was used. If the above approach is used a query time of at most $4 \log n$ comparisons can be shown [32]. Recently, Seidel and Adamy [36] utilized an a posteriori weighting mechanism where each nonempty trapezoid in the slab receives a weight related to the depth of the tree rooted at it. With this strategy, they were able to show a bound on the worst-case query time of $3 \log n$ comparisons.

2.2.4 Randomized incremental method

Randomization has been applied successfully to many problems to give optimal and conceptually simple algorithms [28]. Planar point location is not an exception. The randomized algorithm of this section is attributed to Mulmuley [26] and to Seidel [35].

A *trapezoidal map* of a collection of n noncrossing segments is formed by shooting a vertical ray from every endpoint of all segments both downwards and upwards. A ray stops if another segment is hit. The plane in this way is decom-

posed into vertically aligned trapezoids, some of them perhaps degenerate (i.e., they could be triangles), whose number remains $O(n)$.

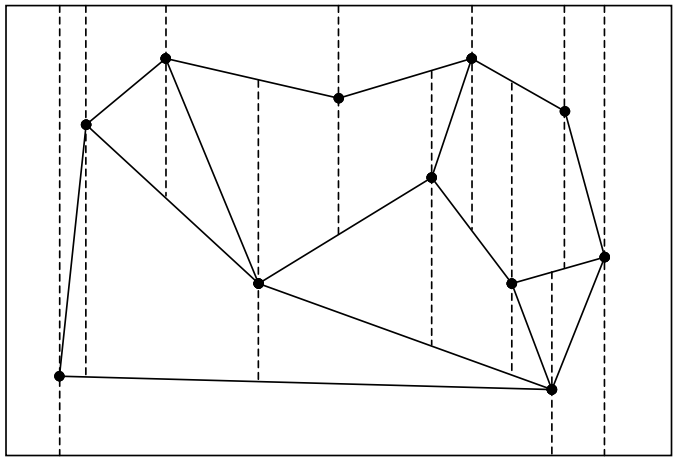


Figure 2.6 The trapezoidal map of a subdivision.

The construction of the data structure for point location is incremental. At each step, one randomly picked segment of the subdivision is inserted onto the plane and the corresponding trapezoidal map is updated. Together with the trapezoidal map, there is a related search structure which supports point location. Three types of nodes can be distinguished in this structure: *up-down* nodes where the query point is compared with the line passing through a segment, *left-right* nodes where it is compared with a vertical line passing through the endpoint of a segment and *leaf* nodes that correspond to a trapezoid in the trapezoidal map.

When a segment is inserted the following occur. First, its left endpoint is located in the current trapezoidal map with the help of the current search structure. The trapezoids the segment intersects are visited from left to right, or, as it is called, are "threaded". These old trapezoids are destroyed while new trapezoids take their place. The necessary comparison nodes are added in the search structure. Interestingly, it is possible to merge some of the old trapezoids that used to be represented in different leaves into a new single trapezoid. Consequently, the search structure is a DAG.

Both the query time and the space of this method are relatively easily analysed

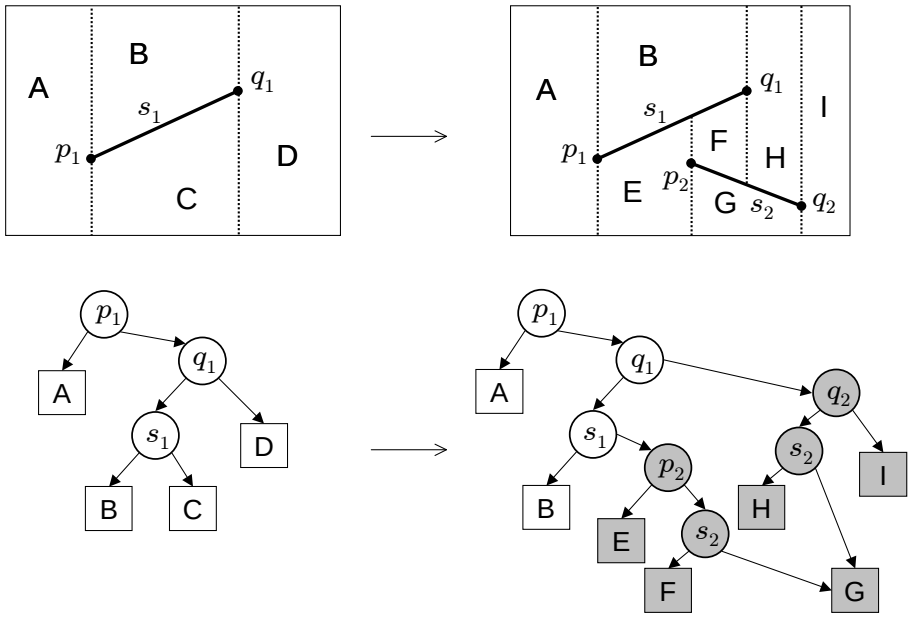


Figure 2.7 Updating the trapezoidal map and the search structure.

by backwards analysis. For the query time, any fixed query point is assumed and it is asked what is the probability of the trapezoid currently bounding the query point to have changed as a result of the insertion of the i-th segment in order. The probability of this averaged over all permutations can be shown to be $O(1/i)$. If the bounding trapezoid has changed, this means that a constant number of additional comparisons were needed to locate the query point. The i-th segment contributes then an expected $O(1/i)$ increase in the comparisons required. Summing over all steps of the algorithm, the expected query time turns out to be $O(\sum_i (1/i)) = O(\log n)$. For the space analysis, the question is how many new trapezoids are expected to be created by the insertion of the i-th segment. This number can be proven to be constant and thus the expected space is linear. More precisely, the query time is bounded by $(5 \ln 2) \log n = 3.47 \log n$ while a rough bound for the space is $12n$. It can also be proven that there is a good probabilistic guarantee that the search structure will not exceed the logarithmic depth [7].

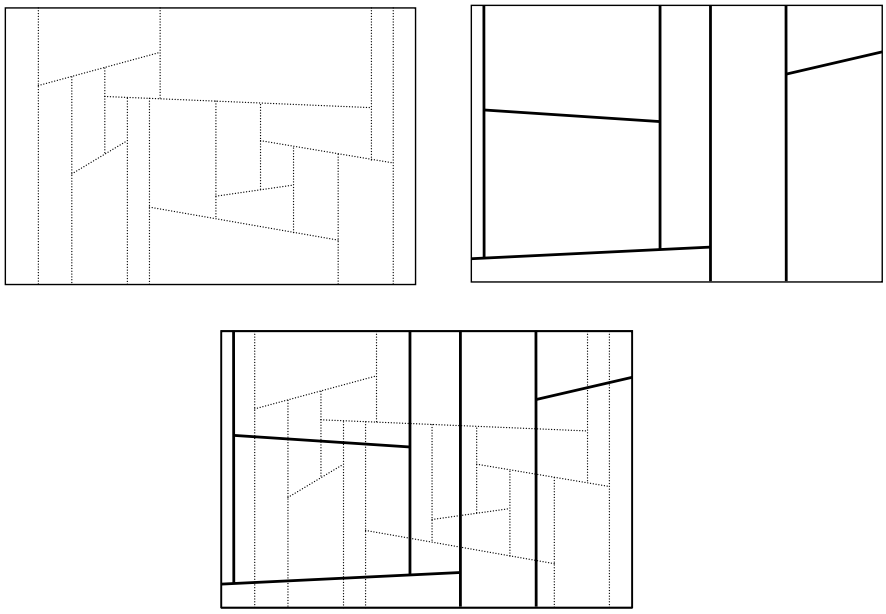


Figure 2.8 A subdivision (up-left), a cutting (up-right) and superimposing the cutting on the subdivision (below-center).

2.2.5 Space reduction by cuttings

Goodrich et al. [15] provided a general way to reduce the space of superlinear planar point location algorithms to linear without significantly harming the query time by using $(1/r)$-*cuttings* [10]. Cuttings is a technique for achieving geometric divide and conquer with applications to a broad class of problems. The result that the authors built on is the following: *For a set S of n noncrossing segments and a parameter r with $2 \leq r \leq n$, a $(1/r)$-cutting for S is a partition of the plane into $O(r)$ trapezoids so that each of those trapezoids is intersected by at most n/r segments in S.* The construction of a $(1/r)$-cutting can be performed easily using random sampling techniques in expected time $O(n \log n)$.

Their method works by first finding a $(1/r)$-cutting of the subdivision S for some suitable value of r. For this cutting, the superlinear space algorithm is used to locate the trapezoid of the cutting containing the query point. The value of r is such that the total space consumed to support the above operation is linear. The *conflict list* for a trapezoid is the set of segments that intersect it. Depending on the exact space used by the algorithm, it may be necessary to recursively apply the procedure of finding a cutting and building a search structure on the conflict

lists of the trapezoids. At the end, when the conflict list is guaranteed to be small in size, a linear-space method that is less efficient than the original algorithm in terms of query time suffices.

This technique is especially attractive because it achieves the reduction to linear space with a sacrifice in the query time that is only in $o(\log n)$, leaving thus the multiplicative constant in front of the $\log n$ term unchanged.

2.2.6 Recursive slab method

Seidel and Adamy [36] recently showed the remarkable fact that planar point location can be performed in $\log n + o(\log n)$ time. This is the best possible in terms of the multiplicative constant factor in front of the $\log n$ term.

The method is similar both to the slab method and the trapezoid method. What is new about this method is that, in contrast to the trapezoid method that makes only one vertical split, builds a weighted search tree for the trapezoid segments and then recurses, and, unlike the slab method that makes all the vertical splits at once and has no recursion, this method selects to do something in the middle. Specifically, it creates 2^t vertical slabs that split equally the endpoints of the segments, builds the relevant weighted search tree for each slab and then recurses. The parameter t depends on the number of endpoints inside the current trapezoid. Intuitively, its value is chosen so as to balance the number of recursions and the number of comparisons spent on vertical cuts, with the aim of reducing the total depth of the data structure. Its value is selected to be approximately $t = t(n) = 2^{2\sqrt{\log n}}$. Another new feature of the method is that, instead of assigning the weights in the weighted search trees for the trapezoids *a priori*, as is typically done in the trapezoid method based on the number of endpoints they contain, the assignment of weights here is done *a posteriori*; first the recursion in all the trapezoids of a slab is performed and then the weight for each trapezoid of that slab is computed. The weight w_i assigned to a trapezoid is 2^{d_i}, where d_i is the depth of the search structure constructed for the trapezoid. The intuition is that the weight is selected so as to anticipate the comparisons that follow. The empty trapezoids receive a weight of one. The accurate analysis of the algorithm is a little technical. The final bound for the query time is $\log n + 2\sqrt{\log n} + \frac{1}{2}\log\log n + 2$, while the space is $O(n2^{2\sqrt{\log n}})$.

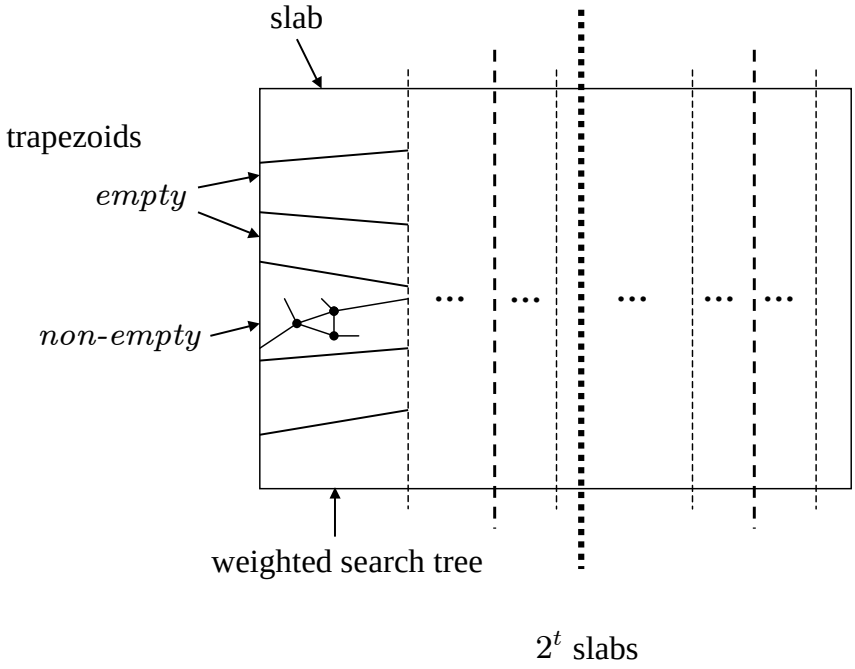


Figure 2.9 The recursive slab method.

Using a two-level $(1/r)$-cuttings construction in a similar way to that proposed by Goodrich et al. [15], the space of the method can be brought down to linear with just an $O(\log^{\frac{1}{4}} n)$ additive increase in the query time. In experiments [1], this method did not yield the expected improvement in query time. This is due to the fact that the second order term, for usual values of n, is comparable to the first. Also, the use of cuttings increased the hidden constant for the space, which could go up to $16n$.

2.2.7 Discussion

Finding an optimal algorithm for planar point location that is as simple as one-dimensional binary search seems to be elusive. The divide and conquer paradigm is directly or indirectly behind all the methods. Ideally, one would want with one comparison to exclude half of the faces of the subdivision. The difficulty is that any comparison is prone to introduce new regions, thus increasing the complexity of the subdivision on which searching is performed.

We attempt a relaxed characterization of the comparison-based methods that are known for planar point location. There exist methods that try to define a total

ordering of the various regions related either to their topology or their geometry or both. Other methods try to search hierarchically with a clear two-dimensional flavor. The query point is located progressively in finer and finer subdivisions. Most of the known linear-spaced methods follow this latter approach. The best method known in terms of the exact number of comparisons, the recursive slab with cuttings, is an intriguing combination of these two types of strategies.

The interested reader can find additional material on planar point location and its variations and extensions in [7, 16, 28, 31, 32, 40].

CHAPTER 3

OPTIMIZING THE EXPECTED QUERY TIME

In this chapter we show that it is possible to design a point location search structure such that the expected query time nearly matches the entropy lower bound. We present two methods that differ with respect to the space needed. For the higher-space structure the lower order term grows as $O(\sqrt{H})$, and in the lower-space structure it grows as $O(H^{2/3})$. The space used by our data structures is $O(n \log n)$ for triangles (cells of bounded complexity), $O(n)$ for axis-parallel rectangles (*iso-oriented* cells of bounded complexity, where by iso-oriented we mean that the boundary of the cells consists of line segments oriented along a constant number of directions) and $O(n \log n)$ for convex polygons with uniform query distribution. Our best results on query time involve a data structure whose size is $O(n^{1+\epsilon})$. The space is bounded by a function of H, which is presented in detail in Theorem 3.1. Our results are summarized in Table 3.1.

Along with the above upper bounds, we present an almost tight lower bound on the expected query time, which is $H + \sqrt{H} - O(1)$. It is well known that $H \leq \log n + O(1)$. For comparison, we mention that it has been shown in [36] that the worst-case query time of planar point location confined to its first two largest terms, in both the upper and the lower bound, is $\log n + 2\sqrt{\log n}$.

3.1 Intuition

Our two point location data structures are based on many of the same methods used in the construction of worst-case efficient point location structures. However, establishing efficient expected query time involves considerably different techniques from worst-case query time. In this section we give the intuition behind the methods, which we needed to develop for this task.

Subdivision Type	Space	Expected Query Time
Triangles (bounded complexity)	$O(n^{1+\epsilon})$ $O(n \log n)$	$H + 2\sqrt{2H} + \log(\sqrt{H}+1) + O(1)$ $H + O(H^{2/3}+1)$
Axis-parallel rectangles (iso-oriented, bounded complexity)	$O(n)$	$H + O(H^{2/3}+1)$
Convex polygons with uniform interior distribution	$O(n^{1+\epsilon})$ $O(n \log n)$	$H + 2\sqrt{2H} + \log(\sqrt{H}+1) + O(1)$ $H + O(H^{2/3}+1)$

Table 3.1 Summary of results in this chapter.

Let us focus for now on the case when the cells of the given subdivision S are triangles. Recall that we assume no knowledge of the distribution of query points within each triangle. As mentioned before, the use of comparisons naturally defines a binary space partition (BSP) tree [7], in which each node of the tree represents a convex polygonal region of the plane. Suppose that we construct a binary space partition (BSP) tree for S, and answer point location queries by simply descending the tree to find the leaf containing q. What properties would suffice to ensure that this tree can be used to answer point location queries efficiently?

To answer this question, notice that the expected query time is given by the weighted external path length [21], where the weight of a leaf is the probability that the query point lies in the region associated with the leaf; since the entropy of the set of leaves is a lower bound on the weighted path length [21], this suggests that the following two properties are desirable in the BSP tree:

Property 1: The entropy of the leaves should be as small as possible. Since the entropy of the leaves cannot be less than the entropy of S, we would like the entropy of the leaves to be close to the entropy of S.

Property 2: The depth of a leaf should be close to $\log(1/p)$, where p is the probability associated with the leaf.

The second property helps to ensure that the expected query time is close to the entropy of the leaves. We elaborate on these properties.

3.1.1 Property one

The first property suggests that we should try to minimize the number of leaves generated by each cell in S. To see this let f_z denote the number of leaves generated by a cell $z \in S$. By elementary calculus, the entropy of the leaves is maximized when each of the f_z leaves generated by cell z has the same probability (i.e., p_z/f_z). The entropy of the leaves is therefore at most

$$\sum_{z \in S} p_z \log(f_z/p_z) \;=\; H + \sum_{z \in S} p_z \log(f_z). \tag{3.1}$$

Observe that the bound in Eq. (3.1) is at most $H + \log(f_{\max})$, where $f_{\max}$ is the maximum number of fragments generated from any cell. This implies that if we could construct a BSP tree that partitions each cell in S into at most a constant number of fragments, then the entropy of the leaves would exceed the entropy of S by at most an additive constant. Unfortunately, a recent result by Tóth [41] implies that constructing such a tree is not possible for every planar subdivision.

A key insight is to observe that a much weaker requirement (instead of partitioning each cell of S into a constant number of fragments) both ensures that the entropy of the leaves is small and leads to BSP trees that are easy to construct. To be precise, it suffices to construct a BSP tree that splits each cell of S with associated probability p into at most $O(\log(1/p) + 1)$ fragments. Using Eq. (3.1) and simple calculations, it can be shown that the entropy of the leaves of this tree is at most $H + \log(H + 1)$. We show that such a BSP tree can be constructed using a modification of the well-known trapezoid method of Preparata [30].

3.1.2 Property two

The probability that the query point lies in the region associated with a leaf is not known exactly, since we are not given the query distribution within a cell. Thus, to satisfy Property 2, our strategy is to construct a tree in which the depth of any leaf generated from a cell $z \in S$ is close to $\log(1/p_z)$. In this chapter we present two different ways of achieving this property. The first approach is given in Section 3.2 and is based on a modification of the methods given by Preparata [30] and Seidel and Adamy [36]. The second approach is given in

Section 3.3 and is more space-efficient. It is based on first constructing a BSP tree satisfying Property 1 and then rearranging the leaves of this tree using the centroid decomposition technique and applying certain other transformations. As a footnote, we remark that while property two is useful, unlike property one, it is not a necessary condition for matching the entropy bound.

3.2 High-Space Method

In this section we prove the following theorem. For simplicity we present the result for the case of triangular subdivisions, but we will show later that it may be generalized to subdivisions in which the cells have constant combinatorial complexity.

Theorem 3.1 *Given an n-vertex triangular subdivision S, together with probabilities p_z that a query point lies within each cell z, we can build a data structure that answers point location queries in expected query time*

$$H + 2\sqrt{2H} + \log(\sqrt{H} + 1) + O(1).$$

The space for the data structure is

$$O(n2^{\sqrt{2H}} \log n/(\sqrt{H} + 1)).$$

Other than the probabilities p_z we assume no knowledge of the query probability distribution. Recall that H is bounded by $\log n + O(1)$, and hence the space used is at most $O(n2^{\sqrt{2\log n}}\sqrt{\log n})$, which is $O(n^{1+\epsilon})$ for any $\epsilon > 0$.

As mentioned earlier, our data structure is based on constructing a BSP tree T for S. Before building the tree we construct a discrete probability distribution, called the *pseudo-probability*, as follows. Let m denote the total number of triangles in S. We note that m is $\Theta(n)$. For each triangle $z \in S$, we assign a pseudo-probability of $\hat{p}_z/6$ to each of its three vertices, where $\hat{p}_z = \max(p_z, 1/m)$. If a vertex is incident to several triangles, then the pseudo-probability assigned to it is the sum of the contribution from each incident triangle. It is easy to see that the total pseudo-probability of all the vertices is at most one. If the total pseudo-probability is strictly less than one, we increase the pseudo-probability of

one or more vertices arbitrarily, so that the total pseudo-probability becomes one. As we will see in the analysis, the use of pseudo-probability instead of the true probability is crucial to limiting the fragmentation of cells of small probability. This helps to reduce the space used by the search structure.

The tree T is built recursively in a top-down fashion. In each stage of the recursion, we do the following. Suppose that we are working on the subdivision contained within a trapezoid u. Let p'_u denote the sum of the pseudo-probabilities of the vertices in its interior. Initially u is the entire space and p'_u is 1. We split u into two vertical *slabs* such that the pseudo-probability of the vertices in the interior of each slab is at most $p'_u/2$. We repeat this for t levels, where $t \geq 1$ is a suitable parameter (to be fixed later), each time ensuring that the pseudo-probability of the resulting slabs is halved. This partitioning can be represented in a natural way by a balanced tree having 2^t leaves, representing the 2^t vertical slabs. Each slab is further partitioned into trapezoids by the segments of the subdivision that completely cross it. Following Seidel and Adamy [36], we build a weighted search tree [24] for each slab, where the weights of the trapezoids are assigned as follows:

(a) If there are no segments intersecting the trapezoid (*empty* trapezoid), its weight is $\hat{p}_z/(h2^t)$, where z is the triangle in S that generates the trapezoid, and h is a suitable parameter (to be fixed later).

(b) For the remaining trapezoids (*non-empty* trapezoids), the weight is the pseudo-probability of all the vertices in their interior.

Finally, we recurse on the non-empty trapezoids.

We now analyze the space and expected query time as a function of the parameters t and h. For the purpose of analysis, it is convenient to view the partitioning scheme as a multi-way tree as follows. Suppose that in a stage of the recursion, trapezoid u is split into 2^t vertical slabs, each of which is partitioned into smaller trapezoids. Then in the multi-way tree, there is a node representing trapezoid u, which is made the parent of the nodes representing these smaller trapezoids. Let T' denote this multi-way tree.

For a node x of T, let $\mathit{region}(x)$ denote the region associated with x, p_x denote the probability of the query point lying in this region (more precisely,

the probability of visiting x during point location), and p'_x denote the pseudo-probability of all the vertices in its interior. The following lemma bounds the number of leaves in T generated by any triangle $z \in S$.

Lemma 3.1 *The number of leaves in the BSP tree T generated by triangle z in S is at most*

$$3 \cdot 2^t \left(\frac{1}{t} \log \frac{1}{\hat{p}_z} + 4 \right).$$

Proof. We start by bounding the number of internal nodes of T' that overlap the interior of z. Since any internal node of T' that overlaps the interior of z must contain one of the vertices of z, it follows that there are at most three such internal nodes at any level of T'.

Also, since the pseudo-probability of a node decreases by a factor of at least 2^t as we descend one level in T', the pseudo-probability of a node at level i of T' is at most $1/2^{t(i-1)}$. Recall that the pseudo-probability of the vertices of triangle z is at least $\hat{p}_z/6$. It follows that if a node containing a vertex of z is at level i, then

$$\frac{1}{2^{t(i-1)}} \geq \frac{\hat{p}_z}{6}.$$

Simplifying this gives

$$i \leq \frac{1}{t} \left(\log \frac{1}{\hat{p}_z} + \log 6 \right) + 1 \leq \frac{1}{t} \log \frac{1}{\hat{p}_z} + 4.$$

Thus, the number of internal nodes of T' that overlap z is at most $3((1/t) \cdot \log(1/\hat{p}_z) + 4)$. Since any node of T' can have at most 2^t children that overlap the interior of z, the bound given in the lemma follows. □

Using this lemma, it is easy to bound the size of the tree.

Lemma 3.2 *The total number of nodes in the BSP tree T is at most $O(n2^t (\log(n)/t + 1))$.*

Proof. By Lemma 3.1, a triangle $z \in S$ yields at most $3 \cdot 2^t(\log(m)/t + 4)$ leaves in T (since $\hat{p}_z \geq 1/m$). Thus the total number of leaves, and hence the total

number of nodes, in T is at most $O(m2^t(\log(m)/t+1))$. Since $m = O(n)$, the result follows. □

In Lemma 3.4, we bound the depth of a leaf generated from a triangle $z \in S$. To this end, we need the following technical result.

Lemma 3.3 *Let x be an internal node in the multi-way tree T'. Let s be any of the 2^t vertical slabs into which region(x) is partitioned. Then the total weight of the weighted search tree corresponding to s is at most $(p'_x/2^t)(1+6/h)$.*

Proof. Recall that the segments in S partition s into empty and non-empty trapezoids. Since the pseudo-probability associated with s is at most $p'_x/2^t$, the total weight of all the non-empty trapezoids is at most $p'_x/2^t$. We will show that the total weight of the empty trapezoids is at most $6p'_x/(2^t h)$, which will complete the proof.

Let $\mathcal{G}_x$ denote the set of triangles in S that overlap *region*(x). The construction implies that the triangles in $\mathcal{G}_x$ have at least one vertex in *region*(x). Since p'_x is the sum of the pseudo-probability of all the vertices in *region*(x), it follows that $p'_x \geq \sum_{z\in\mathcal{G}_x} \hat{p}_z/6$.

Next observe that a triangle in $\mathcal{G}_x$ can generate at most one empty trapezoid in slab s. Recall that the weight of an empty trapezoid generated by triangle $z \in S$ is $\hat{p}_z/(2^t h)$. Thus the total weight of the empty trapezoids in s is at most $\sum_{z\in\mathcal{G}_x} \hat{p}_z/(2^t h)$. By the bound on p'_x from the previous paragraph, this is at most $6p'_x/(2^t h)$.

□

Lemma 3.4 *Let x be a leaf in the BSP tree T generated by triangle $z \in S$. Then the depth of x in T is at most*

$$\log\frac{1}{\hat{p}_z} + t + \left(2 + O\left(\frac{1}{h}\right)\right)\left(\frac{1}{t}\log\frac{1}{\hat{p}_z} + 1\right) + \log(h+1) + O(1).$$

Proof. Let $P = x_1, x_2, \ldots, x_l$ be the path from the root to the leaf $x = x_l$ in the multi-way tree T'. Let s denote the vertical slab in *region*(x_i) that contains the

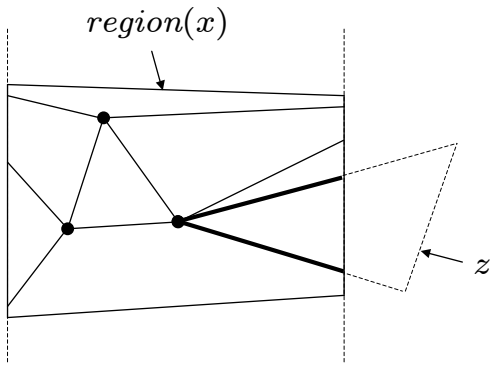


Figure 3.1 Lemmas 3.1 and 3.3.

trapezoid associated with x_{i+1}, and let y denote the node in T corresponding to s. To prove the lemma, we will separately bound the length of the paths in T from x_i to y and from y to x_{i+1}. By construction, the length of the path in T from x_i to y is t.

To bound the length of the path in T from y to x_{i+1}, recall that x_{i+1} is a leaf in the weighted search tree for slab s. By standard results on weighted search trees [24], the length of the path in T from y to x_{i+1} is at most $\log(W/w) + 2$, where W is the weight of all the trapezoids in slab s, and w is the weight of the trapezoid associated with x_{i+1}. By Lemma 3.3, $W \leq (p'_{x_i}/2^t)(1 + 6/h)$. We now consider two cases: (i) $1 \leq i \leq l-2$ and (ii) $i = l-1$. In the first case, x_{i+1} is a non-empty trapezoid, so its weight w is the same as its pseudo-probability $p'_{x_{i+1}}$. Thus, the length of the path in T from y to x_{i+1} is at most

$$\begin{aligned} & \log\left[\frac{(p'_{x_i}/2^t)(1+6/h)}{p'_{x_{i+1}}}\right] + 2 \\ = \;& (\log p'_{x_i} - \log p'_{x_{i+1}}) - t + \log\left(1 + \tfrac{6}{h}\right) + 2. \end{aligned}$$

In the second case, $x_{i+1} = x_l$ is an empty trapezoid, so its weight w is $\hat{p}_z/(2^t h)$. Thus, the length of the path in T from y to x_l is at most

$$\begin{aligned} & \log\left[\frac{(p'_{x_{l-1}}/2^t)(1+6/h)}{\hat{p}_z/(2^t h)}\right] + 2 \\ = \;& (\log p'_{x_{l-1}} - \log \hat{p}_z) + \log(h+6) + 2. \end{aligned}$$

By using the above claim to bound the lengths of the paths in T between adjacent pairs of vertices in P, and summing and cancelling the telescoping probability terms, it is easy to see that the depth of x in T is at most

$$\log \frac{1}{\hat{p}_z} + t + \left(2 + \log\left(1 + \frac{6}{h}\right)\right)(l-2) + \log(h+1) + O(1). \tag{3.2}$$

Since x_{l-1} must contain a vertex of triangle z (which generates leaf x), it follows that $p'_{x_{l-1}} \geq \hat{p}_z/6$. Also, since the pseudo-probability of a node at level i is at most $1/2^{t(i-1)}$, so $p'_{x_{l-1}} \leq 1/2^{t(l-2)}$. Thus,

$$l - 2 \leq \frac{1}{t} \log \frac{1}{p'_{x_{l-1}}} \leq \frac{1}{t}\left(\log \frac{1}{\hat{p}_z} + \log 6\right).$$

Substituting this value of l in Eq. (3.2), and using the fact that $\log(1 + 6/h) = O(1/h)$, after some simplification, we get the bound on the depth of x given in the statement of the lemma.

□

We can now bound the expected query time.

Lemma 3.5 *The expected query time using the BSP tree T is at most*

$$H + t + \left(2 + O\left(\frac{1}{h}\right)\right)\left(\frac{H}{t} + 1\right) + \log(h+1) + O(1).$$

Proof. For any triangle $z \in S$, let $\mathcal{L}_z$ denote the set of leaves generated by z. The expected query time is given by

$$\sum_{z \in S} \sum_{x \in \mathcal{L}_z} p_x d_x,$$

where d_x denotes the depth of x. Applying Lemma 3.4, this sum is at most

$$\sum_{z \in S} \sum_{x \in \mathcal{L}_z} p_x \left[\log \frac{1}{\hat{p}_z} + t + \left(2 + O\left(\frac{1}{h}\right)\right)\left(\frac{1}{t} \log \frac{1}{\hat{p}_z} + 1\right) + \log(h+1) + O(1)\right].$$

Noting that $\hat{p}_z \geq p_z$ and simplifying we easily obtain the bound given in the statement of the lemma. □

In order to get the best bound on the expected query time, we choose $t = \lceil\sqrt{2H}\,\rceil$ and $h = \lceil\sqrt{H}\,\rceil$ in Lemma 3.5. This yields an expected query time of at most $H + 2\sqrt{2H} + \log(\sqrt{H}+1) + O(1)$. Using Lemma 3.2 and noting that t is at most $O(\sqrt{\log n})$, we obtain a bound on the space of $O(n2^{\sqrt{2H}} \log n/(\sqrt{H}+1))$. This completes the proof of Theorem 3.1.

Worst-case performance

Setting $t = \lceil\sqrt{2H}\,\rceil + c$, where c is a positive integer, and $h = \lceil\sqrt{H}\,\rceil$, it is easy to see from Lemma 3.4 that the worst-case query time is $(1 + O(1/c)) \log n$. (Recall that for a triangle z, $\hat{p}_z \geq 1/m$.) Simultaneously, the bound on expected performance given by Theorem 3.1 also holds.

3.3 Low-Space Method

In this section we prove the following theorem:

Theorem 3.2 *Let S be an n-vertex planar subdivision. Assume that the query distribution within each cell is unknown.*

(i) If S consists of triangles, we can build a data structure of $O(n \log n)$ space that provides expected query time of $H + O(H^{2/3} + 1)$.

(ii) If S consists of iso-oriented cells of bounded complexity, we can build a data structure of $O(n)$ space that provides expected query time of $H + O(H^{2/3} + 1)$.

Note that the space used in these solutions is better than the space used in Theorem 3.1, while the expected query time is just a little worse. The improvement in space is achieved by following a two-step approach. In the first step we construct a BSP tree T for the given subdivision S, that partitions each cell of S into *few* fragments (constant number of fragments if S consists of iso-oriented cells of bounded complexity and $O(\log(1/p) + 1)$ fragments, where p is the probability associated with a cell, if S consists of triangles). However, we do not care about the depth of the leaves, so a high probability leaf may be very deep in the tree T. Thus, T may not provide good expected query time if directly used for

point location. In the second step, we correct this by rearranging the leaves of the tree and applying certain other transformations. The goal of these transformations is to ensure that, if a leaf of the BSP tree is generated from a cell of S with probability p, then we can reach it using close to $\log(1/p)$ comparisons.

3.3.1 BSP tree transformation

The following lemma is crucial to the proof of Theorem 3.2. It is related to the second of the two steps mentioned above. Define a *simple* BSP tree to be a BSP tree with the property that the region associated with any node of the tree is a polygon with at most a constant number of sides.

Lemma 3.6 *Let T be a simple BSP tree with N nodes. We are given a weight w_x for each leaf x of T such that the total weight of all the leaves is at most one. Then for any positive integer $\alpha > 1$ (we call α the compression parameter), we can build a search structure that allows us to do the following: Given a query point q we can determine the leaf x of T that contains q using at most*

$$\left[1 + O\left(\frac{1}{\sqrt{\alpha}}\right)\right] \log \frac{1}{w_x} + O(\alpha)$$

comparisons. The space for the search structure is $O(N)$.

Phase-one transformation

We devote the remainder of this section to proving this lemma. We show how to build the desired search structure in two steps. In the first step we build a *centroid decomposition tree* [9] of tree T. Define the weight of a tree to be the total weight of the leaves in the tree, and the weight of a node to be the total weight of the leaves in the subtree rooted at the node. Define a *centroid edge* in a binary tree of weight W to be an edge whose removal partitions the tree into two subtrees of weight at most $2W/3$. It follows from standard results that a binary tree must have such an edge or it must have a leaf with weight more than $2W/3$ (in the latter case, we will abuse terminology and refer to the edge connecting the leaf to the rest of the tree as the centroid edge). Moreover, by taking centroids, the nodes of T can be recursively restructured into a binary tree T' with the following properties:

(a) T' has the same leaves as T.

(b) Let y be any internal node of T' and let w_y denote its weight. Either both children of y have weight at most $2w_y/3$ or one child of y is a leaf of weight more than $2w_y/3$.

(c) Any node v of T' is associated with a polygon $P(v)$ having at most c sides, where c is a constant. All leaves in the left subtree of v are contained within $P(v)$ and all leaves in the right subtree of v are contained outside $P(v)$. (This property follows from the fact that the regions associated with the nodes of T are the separators $P(v)$ for the nodes of T'. Since T is a simple BSP tree, $P(v)$ has constant complexity.)

By property (c), we can use T' to do point location by a simple descent in the tree starting from the root. If a query point lies in leaf x, then the number of comparisons needed to locate it is at most $c \cdot d_x$, where d_x denotes the depth of x. By property (b), the depth of leaf x is at most $\log_{3/2}(1/w_x) + 1$. Thus the number of comparisons needed is at most $c(\log_{3/2}(1/w_x) + 1)$.

Phase-two transformation

In the remainder of the proof, we will show how to reduce the multiplicative factor of $\log(1/w_x)$ from $(c/\log(3/2))$ to close to 1. To this end, we transform the tree T' to a partition tree T'' using the following recursive procedure. Let α be the positive integer specified in the statement of the lemma. We create a root node v'' for the tree T''; the region associated with v'' is the same as the region associated with the root of T'. If T' consists of a single leaf, then there is nothing else to be done. Otherwise, we construct a set M of nodes of T' as follows. Initially M consists of only the root of T'. In each iteration, we remove the node u from M that has the largest weight among all the nodes in M that are internal nodes of T'. We then insert the two children of u into M. We continue in this manner until we have accumulated 2^α nodes in M, or all the nodes in M are leaves in T'. It is clear that the set M consists of disjoint descendants of the root of T', and all the leaves in T' are contained in the associated subtrees. Let $d = |M|$, and let $T'_1, T'_2, \ldots, T'_d$ be the subtrees of T' rooted at the nodes in M. We recursively

transform trees $T'_1, T'_2, \ldots, T'_d$ into $T''_1, T''_2, \ldots, T''_d$, respectively. Finally, we make the roots of the trees $T''_i, 1 \leq i \leq d$, children of node v''.

This completes the description of the construction. We view T'' as a compressed form of the tree T', which represents the same hierarchical subdivision. Clearly, any node v'' in T'' has a corresponding node v' in T'; the associated regions and the set of leaves in the subtrees rooted at the nodes is the same. In order to achieve the desired speed-up in locating a query point, our strategy is to employ T'' instead of T' for point location.

Suppose that the query point q lies in the region associated with a node $v \in T''$. It is easy to see that we can determine the child of v which contains the query point q by doing point location in a planar subdivision of complexity at most $c \cdot d_v$, where d_v denotes the number of children of v. As part of the preprocessing we build the worst-case planar point location data structure given by Seidel and Adamy [36] for each node of T''. For a node v, this data structure uses $O(d_v)$ space and allows us to determine the child containing q in $\log(d_v) + O(\sqrt{\log(d_v)})$ comparisons. Since $d_v \leq 2^\alpha$, the number of comparisons is bounded by $\alpha + O(\sqrt{\alpha})$,

Analysis

We now bound the space and query time. Observe that the space used by the point location data structures for all the internal nodes of T'' is $O(s)$, where s is the number of nodes in T''. Further, s is no more than the number of nodes in T. Hence the total space used is $O(N)$. The following lemma is crucial to bounding the query time.

Lemma 3.7 *Let v'' be a child of u'' in T'', and let $w_{v''}$ and $w_{u''}$ denote the weight of the subtrees rooted at nodes v'' and u'', respectively. Then if v'' is an internal node of T'', $w_{v''}$ is at most $(1/2^{\alpha-1})w_{u''}$.*

Proof. Let u' and v' be the nodes in T' corresponding to nodes u'' and v'' in T''. Obviously $w_{u'} = w_{u''}$, $w_{v'} = w_{v''}$, and v' is an internal node. Recall that during the construction of T'' we determine the children of u'' by incrementally growing a set M of nodes of T'. Initially M consists only of u'; the final set M, which we denote it by M_f, contains the descendants of u' corresponding to the children

of u''. By construction $|M_f| = 2^\alpha$ since v' is an internal node (because otherwise M_f should contain only leaf nodes).

For the sake of contradiction, suppose that $w_{v'} > (1/2^{\alpha-1})w_{u'}$. It is easy to see that, as we grow the set M, M must always contain either v' or some ancestor of v' (lying on the path from u' to v'). Thus, there is always a node in M that is an internal node of T' and whose weight is $\geq w_{v'} > (1/2^{\alpha-1})w_{u'}$. Recall that at each step of the construction, we remove a node x from M that has the largest weight among all nodes in M that are internal nodes of T' and then insert the two children of x into M. It follows that any node x removed from M must have weight more than $(1/2^{\alpha-1})w_{u'}$.

We claim that the average weight of a node in M_f exceeds $(1/2^\alpha)w_{u'}$. To prove this we divide the nodes in M_f into two categories. The first category consists of nodes whose siblings (in T') are also present in M_f, and the second category consists of nodes whose siblings are not present in M_f. We will show the following: (i) for any node y_1 in the first category, the sum of the weight of node y_1 and its sibling is more than $(1/2^{\alpha-1})w_{u'}$, and (ii) the weight of any node y_1 in the second category is more than $(1/2^\alpha)w_{u'}$. Clearly, (i) and (ii) together imply the desired claim.

Let y_2 and y denote the sibling and parent in T', respectively, of node y_1. To prove (i), note that y must have been removed from M and so by our earlier observation the weight of y must exceed $(1/2^{\alpha-1})w_{u'}$. Since the sum of the weight of y_1 and y_2 equals the weight of their parent y, this completes the proof of (i).

To prove (ii), observe that since y_2 is not present in M_f, it must have been removed from M, and so its weight must exceed $(1/2^{\alpha-1})w_{u'}$. Noting that y_2 cannot be a leaf, and using property (b) of T', it follows that the weight of y_1 is at least $1/2$ the weight of y_2. Thus, the weight of y_1 is more than $(1/2^\alpha)w_{u'}$.

Since there are 2^α nodes in M_f and their average weight exceeds $(1/2^\alpha)w_{u'}$, hence the total weight of the nodes in M_f exceeds $w_{u'}$, which is a contradiction (the weight of the nodes in M_f should be exactly $w_{u'}$, since these nodes are disjoint descendants of u' covering the same region as u').

□

We can now bound the query time as follows. Suppose that the query point q lies in a leaf x of T''. Let $P = x_1, x_2, \ldots, x_l$ be the path from the root to the leaf $x = x_l$ in T''. It follows from Lemma 3.7 that the weight of an internal node at level i is at most $(1/2^{\alpha-1})^{i-1}$. Thus $w_{x_{l-1}} \leq (1/2^{\alpha-1})^{l-2}$. Since $w_x = w_{x_l} \leq w_{x_{l-1}}$, it follows that $w_x \leq (1/2^{\alpha-1})^{l-2}$, which yields

$$l - 1 \leq \frac{1}{\alpha - 1} \log \frac{1}{w_x} + 1.$$

Recall that the number of comparisons needed at each of the $l-1$ internal nodes is bounded by $\alpha + O(\sqrt{\alpha})$. Thus, the number of comparisons needed to locate q is at most

$$(\alpha + O(\sqrt{\alpha})) \left(\frac{1}{\alpha - 1} \log \frac{1}{w_x} + 1 \right) \leq \left(1 + O\left(\frac{1}{\sqrt{\alpha}} \right) \right) \log \frac{1}{w_x} + O(\alpha).$$

This completes the proof of Lemma 3.6.

3.3.2 Triangles

First we build a BSP tree T for S as in Section 3.2. The construction is carried out with the parameter t set to one. There is only one major difference from the construction given earlier. We do not need to build weighted search trees for the slabs; instead, any search tree will suffice (note this means that the parameter h is irrelevant). In the second step, we assign a weight to each leaf of T as follows. If a triangle $z \in S$ generates f leaves, then we assign a weight of $\hat{p}_z/(2f)$ to each of these leaves. Recall that $\hat{p}_z = \max(p_z, 1/m)$, where m is the number of triangles in S. Finally we build the search structure T'' (using compression parameter α) corresponding to T, as described in Lemma 3.6.

Note that T is a simple BSP tree since the region associated with each node is a trapezoid. By Lemma 3.2, the number of nodes in T is $O(n \log n)$. Setting t to one in Lemma 3.1 implies the following lemma.

Lemma 3.8 *The number of leaves in the BSP tree T generated by triangle $z \in S$ is $O(\log(1/\hat{p}_z) + 1)$.*

We now analyze the time for answering queries using T''.

Lemma 3.9 *Let q be a query point contained in a triangle $z \in S$. Then using T'' the number of comparisons needed to determine the leaf containing q is at most*

$$\left[1 + O\left(\frac{1}{\sqrt{\alpha}}\right)\right] \cdot \left[\log \frac{1}{\hat{p}_z} + \log\left(\log \frac{1}{\hat{p}_z} + 1\right)\right] + O(\alpha).$$

Proof. By Lemma 3.8, triangle z generates at most $c(\log(1/\hat{p}_z) + 1)$ leaves, where c is a constant. Thus, the weight assigned to each of these leaves is at least $\hat{p}_z/(2c(\log(1/\hat{p}_z) + 1))$. The lemma now follows by applying Lemma 3.6. □

Lemma 3.10 *The expected query time using the tree T'' is at most*

$$\left[1 + O\left(1/\sqrt{\alpha}\right)\right](H + \log(H + 1)) + O(\alpha).$$

Proof. For a triangle $z \in S$, let $\mathcal{L}_z$ denote the set of leaves in T generated from z. Using Lemma 3.9, it follows that the expected query time is at most

$$\begin{aligned} &\sum_{z \in S} \sum_{x \in \mathcal{L}_z} p_x \left(\left[1 + O\left(\frac{1}{\sqrt{\alpha}}\right)\right]\left[\log \frac{1}{\hat{p}_z} + \log\left(\log \frac{1}{\hat{p}_z} + 1\right)\right] + O(\alpha)\right) \\ &\leq \left[1 + O\left(\frac{1}{\sqrt{\alpha}}\right)\right] \sum_{z \in S} \left[p_z \log \frac{1}{p_z} + p_z \log\left(\log \frac{1}{p_z} + 1\right)\right] + O(\alpha). \quad (3.3) \end{aligned}$$

We now bound the last term in the summation as follows.

$$\sum_z p_z \log\left(\log \frac{1}{p_z} + 1\right) \leq \sum_z \log\left(\log \frac{1}{p_z} + 1\right)^{p_z} \leq \log\left(\prod_z \log \frac{1}{p_z} + 1\right)^{p_z}.$$

Using the fact that the geometric mean can be no more than the arithmetic mean, we obtain

$$\sum_z p_z \log\left(\log \frac{1}{p_z} + 1\right) \leq \log\left(\sum_z p_z \left(\log \frac{1}{p_z} + 1\right)\right) \leq \log(H + 1).$$

The lemma now follows by substituting this in Eq. (3.3) and simplifying. □

For the best bound, we set $\alpha = \left\lceil H^{2/3} \right\rceil$ in Lemma 3.10. It follows that the expected query time is at most $H + O(H^{2/3}) + O(1)$. Since the space used by T is $O(n \log n)$, the space used by T'' is also $O(n \log n)$. This completes the proof of Theorem 3.2(i).

Worst-case performance

Setting $\alpha = \lceil H^{2/3} \rceil + c$, where c is a positive integer, it is easy to see from Lemma 3.9 that the worst-case query time is $(1+O(1/\sqrt{c}))\log n$. Simultaneously, the bound on expected performance given by Theorem 3.2(i) also holds.

Cells of bounded complexity

We observe that the results given in Theorems 3.1 and 3.2(i) can be generalized to polygons with bounded complexity. The search structures are built as follows. We triangulate each polygon $z \in S$, and assign a probability of p_z/c to the resulting triangles, where c is the number of triangles in z. We then build the point location structures as in Sections 3.2 and 3.3.2. The straightforward proofs are omitted. Intuitively, the theorems hold because the entropy of the triangulation differs from the entropy of S by at most an additive constant.

3.3.3 Iso-oriented cells of bounded complexity

Theorem 3.2(ii) is based on the fact that we can construct an $O(n)$ size BSP tree, when S consists of segments oriented along a constant number of directions (e.g., axis-parallel rectangles). This important result was proved very recently by Tóth [42]. We state below the main theorem slightly modified to suit our purposes.

Theorem 3.3 (Tóth [42]) *Let S be an n-vertex planar subdivision consisting of line segments oriented along a constant number of directions. Then we can construct a simple BSP tree that partitions each line segment in S into at most $O(1)$ fragments. Further, if the cells of the subdivision have bounded complexity, then each cell is partitioned into at most $O(1)$ fragments.*

We construct the desired search structure in two steps. In the first step we construct the BSP tree T described in Theorem 3.3. We assign a weight to each leaf of T as follows. For a cell $z \in S$, define $\hat{p}_z = \max(p_z, 1/m)$, where m denotes the number of cells in S. If a cell z generates f leaves, then we assign a weight of $\hat{p}_z/(2f)$ to each of these leaves. In the second step, we build the search

structure T'' (using compression parameter α) corresponding to T, as described in Lemma 3.6. We answer point location queries by descending T'' to find the leaf containing the query point. We now analyze the query time.

Lemma 3.11 *Let q be a query point contained in a cell $z \in S$. Then using T'' the number of comparisons needed to determine the leaf containing q is at most $[1 + O(1/\sqrt{\alpha})] \log(1/\hat{p}_z) + O(\alpha)$.*

Proof. By Theorem 3.3, cell z generates at most $O(1)$ leaves, and so the weight w_x assigned to each of the leaves is at least $\Omega(\hat{p}_z)$. The lemma now follows by applying Lemma 3.6. □

Lemma 3.12 *The expected query time using the tree T'' is at most*

$$\left[1 + O\left(1/\sqrt{\alpha}\right)\right] H + O(\alpha).$$

Proof. For a cell $z \in S$, let $\mathcal{L}_z$ denote the set of leaves in T generated from z. Using Lemma 3.11, it follows that the expected query time is at most

$$\sum_{z \in S} \sum_{x \in \mathcal{L}_z} p_x \left(\left[1 + O\left(\frac{1}{\sqrt{\alpha}} \right) \right] \log \frac{1}{\hat{p}_z} + O(\alpha) \right).$$

Noting that $\hat{p}_z \geq p_z$, the lemma follows after some simplification. □

For the best bound, we set $\alpha = \left\lceil H^{2/3} \right\rceil$ in Lemma 3.12. It follows that the expected query time is at most $H + O(H^{2/3}) + O(1)$. The space used by T'' is $O(n)$. This completes the proof of Theorem 3.2(ii).

3.4 Convex Polygons with Uniform Interior Distribution

The main result of this section is the following.

Lemma 3.13 *Let S be an n-vertex planar subdivision consisting of convex polygons; the query distribution within each polygon is assumed to be uniform. Then we can triangulate each polygon such that the entropy of the resulting set of triangles exceeds the entropy of S by at most an additive constant.*

It readily follows from the above lemma that the bounds on space and expected query time given in Theorems 3.1 and 3.2(ii) also apply to any subdivision S consisting of convex polygons, assuming that the query distribution is uniform within each polygon. More generally, this lemma implies that any similar bounds that hold for triangular subdivisions can be extended to convex subdivisions assuming uniform interior distribution.

The proof of Lemma 3.13 relies on the following observation.

Lemma 3.14 *Given a convex polygon P with n vertices, there exist three vertices such that the area of the triangle defined by these vertices is at least $1/4$ the area of P.*

Proof. Let v_1 and v_2 denote the pair of vertices of P that realize the diameter of P, and let v_3 denote the vertex that is farthest from the line $\overline{v_1v_2}$. We claim that the area of the triangle defined by v_1, v_2, and v_3 is at least $1/4$ the area of P. Without loss of generality, let v_3 lie above $\overline{v_1v_2}$. Let v_4 denote the vertex farthest from $\overline{v_1v_2}$ among vertices that are below it. Let R denote the rectangle defined by the two lines parallel to $\overline{v_1v_2}$, passing through v_3 and v_4, respectively, and by the two lines perpendicular to $\overline{v_1v_2}$, and passing through v_1 and v_2, respectively. It is an easy geometric exercise to show that P is completely contained within R, and the area of the triangle defined by v_1, v_2, and v_3 is at least $1/4$ the area of R. This implies the lemma. □

Proof. (of Lemma 3.13) We triangulate each convex polygon in S as follows. Let z denote any convex polygon. By Lemma 3.14, we can find a triangle in z whose area is at least $1/4$ the area of z. We insert this triangle into the triangulation. This partitions the remainder of z into at most three convex polygons, which we triangulate recursively.

We bound the entropy of this triangulation. Let $\mathcal{T}_z$ denote the set of triangles in the triangulation of z, constructed by the above procedure. We claim that

$$\mathit{entropy}(\mathcal{T}_z) \leq p_z \log \frac{1}{p_z} + 8p_z, \tag{3.4}$$

where $\mathit{entropy}(\mathcal{T}_z)$ is the quantity $\sum_{x \in \mathcal{T}_z} p_x \log(1/p_x)$. The proof of this claim is by induction on the number of sides of z. For the induction basis case, z has

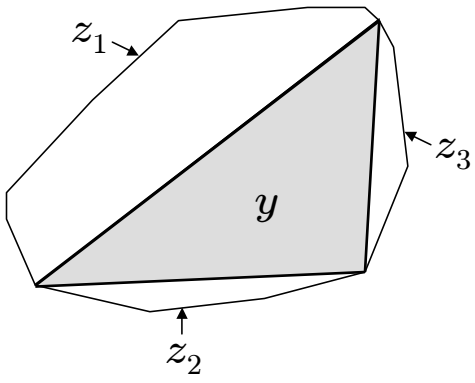


Figure 3.2 Triangulating a convex cell.

three sides, and the claim is trivially true. Suppose that the claim holds for any convex polygon with at most i sides, for some $i \geq 3$. We will show the claim for any convex polygon z with $i+1$ sides.

Let y denote the first triangle added to the triangulation of z. Since the area of y is at least $1/4$ the area of z and the query distribution within z is uniform, so $p_y \geq p_z/4$. Note that $z - y$ consists of (at most) three convex polygons; denote them by z_1, z_2, and z_3 (see Figure 3.2). By the induction hypothesis, $\mathit{entropy}(\mathcal{T}_{z_i}) \leq p_{z_i} \log(1/p_{z_i}) + 8p_{z_i}$, for $1 \leq i \leq 3$. Thus $\mathit{entropy}(\mathcal{T}_z)$ can be written as

$$\begin{aligned} & p_y \log \frac{1}{p_y} + \sum_{i=1}^{3} \mathit{entropy}(\mathcal{T}_{z_i}) \\ \leq \;& p_y \log \frac{1}{p_y} + \sum_{i=1}^{3} \left(p_{z_i} \log \frac{1}{p_{z_i}} + 8p_{z_i} \right) \\ = \;& \left(p_y \log \frac{1}{p_y} + \sum_{i=1}^{3} p_{z_i} \log \frac{1}{p_{z_i}} \right) + 8 \sum_{i=1}^{3} p_{z_i}. \end{aligned} \tag{3.5}$$

Obviously $p_y + \sum_{i=1}^{3} p_{z_i} = p_z$. Since $p_y \geq p_z/4$, it follows that $\sum_{i=1}^{3} p_{z_i} \leq 3p_z/4$. Also, by elementary calculus, the maximum value of

$$\left(p_y \log \frac{1}{p_y} + \sum_{i=1}^{3} p_{z_i} \log \frac{1}{p_{z_i}} \right)$$

subject to the constraint that $p_y + \sum_{i=1}^{3} p_{z_i} = p_z$ occurs when $p_y = p_{z_1} = p_{z_2} = p_{z_3} = p_z/4$, and is given by $p_z \log(4/p_z)$. Using these bounds in Eq. (3.5) we

obtain

$$entropy(\mathcal{T}_z) \leq p_z \log \frac{4}{p_z} + 8\left(\frac{3}{4}p_z\right) = p_z \log \frac{1}{p_z} + 8p_z,$$

which completes the proof by induction.

Summing both sides of Eq. (3.4) over all the polygons in S, it follows that the entropy of triangulation exceeds the entropy of S by at most 8.

□

Finally, we note that by combining this lemma with our point location methods, we can obtain nearly optimal expected query time for the *nearest neighbor* problem [7] assuming that the query distribution is uniform. This is obvious from the fact that the Voronoi diagram of sites in the plane consists of convex polygons.

3.5 A Stronger Lower Bound

In this section we will prove a lower bound of $H + \sqrt{H} - O(1)$ on the expected number of comparisons for planar point location. This demonstrates that the entropy lower bound cannot be reached exactly and that our upper bound of Theorem 3.1 is optimal on the second order term as well (to within a constant factor). For our lower bound argument, we assume that we have no information on how the queries are distributed in the *interior* of the cells and that the data structure is a *trapezoidal search graph*.

Seidel and Adamy [36] introduced the model of trapezoidal search graphs in their lower bound on the worst-case query time of planar point location. A data structure is considered to be a trapezoidal search graph if it is a DAG and it satisfies certain conditions. We describe these conditions in terms of the BSP tree that corresponds to the data structure. For any node v in the BSP tree, let r_v denote the region associated with it. If the node v is internal, a comparison is performed either (a) with a line that is vertical and passes through some vertex inside the region r_v or (b) with a line passing through a segment of S that completely crosses r_v. Thus for each node v of the tree, r_v is a vertically aligned trapezoid, which justifies the name of the model.

Almost all proposed data structures are equivalent to some trapezoidal search graph. Some examples are the randomized incremental method, the brute force method, and the recursive slab method. A negative example is the method by Kirkpatrick, which compares the query point with lines that do not pass through segments of S, so it is not covered by this model.

Seidel and Adamy [36] showed that for an example subdivision the worst-case query time of any trapezoidal search graph is at least

$$\log n + 2\sqrt{\log n} - (1/2)\log\log n - O(1).$$

In our analysis, we use a similar example and extend their ideas to lower-bound the expected query time. For an integer $n \geq 1$, the subdivision S_n is defined as depicted in Figure 3.3. Here n denotes the number of rows in S_n (Figure 3.4). The number of segments is bounded by $O(n)$. There are $2n$ horizontally aligned trapezoidal cells in S_n. (In the example of [36], the diagonal segments were missing, as the authors viewed the problem as vertical ray shooting.) We assume that each trapezoidal cell has equal probability, that is the query point lies in each cell with probability $1/(2n)$. We note then that $H = \log n + 1$.

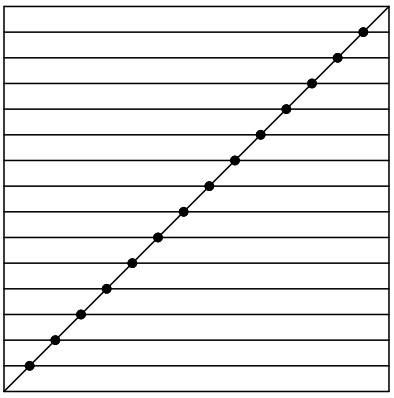


Figure 3.3 The subdivision S_n.

Before presenting our proof, we point out a distinction between worst-case and expected-case analysis. For the worst case, it suffices to show that the depth of some leaf in the tree is large. However, this does not say much about the weighted external path length, which is the relevant quantity for expected query time. To establish a good lower bound on this quantity, we need to show the

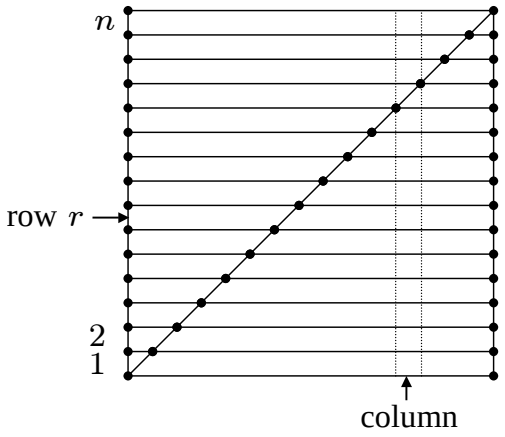


Figure 3.4 Rows and columns in S_n.

existence of leaves deep in the tree that have a large total weight.

Our strategy will be to look at the BSP tree that corresponds to the trapezoidal search graph of a method and adjust the query distribution so that the tree behaves badly. Roughly speaking, if the method makes many vertical splits, then we evenly distribute the query distribution among the many fragments produced by the vertical splits. Thus the entropy of the leaves of the tree increases and so does the expected query time. On the other hand, if the method makes few vertical comparisons then we concentrate the query distribution close to the diagonal of S_n, which puts a high cost on any recursions performed, in parallel to the worst-case argument. We now state the theorem.

Theorem 3.4 *Consider any planar point location method under the trapezoidal graph model that is built for subdivision $S = S_n$, where the probability of the query point lying in each cell in the subdivision is $1/(2n)$. Then there is a query distribution such that the expected query time is at least* $\log n + \sqrt{\log n + 1} - 1$.

Proof. The proof is by induction on n. Let $T(n)$ be the expected query time. (The query distribution will become clear in the proof.) For $n = 1$, $T(1) \geq 0$, and the induction basis holds.

Let $\mathcal{T}$ be the BSP tree constructed (perhaps implicitly) by the method. Let $\mathcal{T}'$ be the subtree of $\mathcal{T}$ consisting of all nodes v that can be reached from the root

of $\mathcal{T}$ using only vertical comparisons. The leaves of $\mathcal{T}'$ partition the bounding box of S_n into a number of vertical slabs. Let X denote the set of these slabs. Note that the number of slabs in X, which is also the number of leaves in $\mathcal{T}'$, is at least 2 since $n > 1$. We define $t = \log |X|$.

We distinguish two cases based on the value of t.

Case 1: $(t \geq 2\sqrt{\log n + 1} - 1)$

We observe that at least one of the two trapezoidal cells in each row of S_n is fragmented by $\mathcal{T}'$ into at least $(2^t/2)$ pieces. We denote this trapezoid by m_r and the other trapezoid by l_r, where $1 \leq r \leq n$ is the row of the two trapezoids. We define the set $L = \{l_r\}$ and the set M' which contains the fragments of m_r created by BSP $\mathcal{T}'$, for all $1 \leq r \leq n$. Clearly,

$$T(n) \geq H_{M'} + H_L, \tag{3.6}$$

where $H_{M'}$ and H_L denote the entropy defined by the probabilities of visiting the trapezoids in M' and L, respectively. (This follows from the fact that the entropy of the leaves of $\mathcal{T}$ cannot be less than the entropy of $M' \cup L$.)

We now give a lower bound on H_L and $H_{M'}$. By definition,

$$H_L = \sum_{r=1}^{n} (1/(2n)) \log(2n) = (1/2)(\log n + 1). \tag{3.7}$$

For H'_M, since the query distribution within the cells of S_n is in our control, we can ensure that each of the fragments of m_r generated by $\mathcal{T}'$ has equal probability of being visited. Then,

$$\begin{aligned} H_{M'} &\geq \sum_{r=1}^{n} (1/(2n)) \log(2n \cdot 2^{t-1}) \\ &= (1/2)(\log n + t). \end{aligned} \tag{3.8}$$

Using Eqs. (3.7) and (3.8) in Eq. (3.6), we get

$$T(n) \geq (1/2)(\log n + t) + (1/2)(\log n + 1) > \log n + t/2.$$

Substituting for t, we obtain

$$T(n) \geq \log n + \sqrt{\log n + 1} - 1/2 \geq \log n + \sqrt{\log n + 1} - 1,$$

as desired.

Case 2: $(t < 2\sqrt{\log n + 1} - 1)$

Let n_s denote the number of columns in a slab $s \in X$ (Figure 3.4). Note that $\sum_{s \in X} n_s = n$. Let p_s denote the probability of the query point lying in s. Let $H_{\mathcal{T}'}$ denote the entropy of the leaves in $\mathcal{T}'$. By linearity of expectation, the expected query time using $\mathcal{T}$ satisfies

$$T(n) \geq H_{\mathcal{T}'} + \sum_{s \in X} p_s \cdot ST(s),$$

where $ST(s)$ is the expected query time of locating the query point given that it lies in s. Note that $H_{\mathcal{T}'} = \sum_{s \in X} p_s \log(1/p_s)$ and thus

$$T(n) \geq \sum_{s \in X} p_s(\log(1/p_s) + ST(s)). \tag{3.9}$$

Recall that each trapezoid v in S_n has equal probability $1/(2n)$ of being visited. Let s_v be the slab in X in which the diagonal segment of v lies. We select the query distribution with the property that if the query point lies in v then it also lies in $v \cap s_v$. We can easily check that for any slab $s \in X$, p_s equals n_s/n.

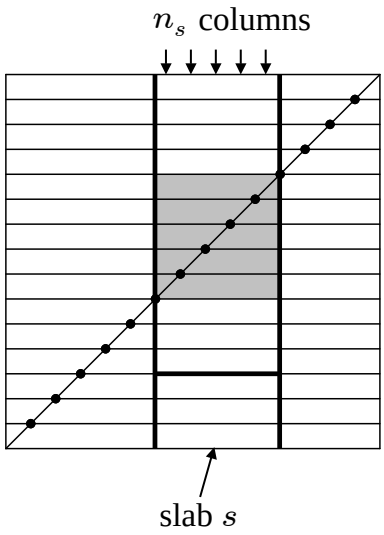


Figure 3.5 Slab s contains subdivision S_5 (colored in grey).

For a slab $s \in X$ with $1 < n_s < n$, we claim that $ST(s) \geq 1 + T(n_s)$. By definition of $\mathcal{T}'$, the next comparison after reaching slab s must be a comparison

with a horizontal segment crossing s. Let s' be the region that the method has confined the query point after this comparison. We recall that the defined query distribution forces the query point to lie in a region $S_k \subseteq s$, where $k = n_s$. Thus it holds that $S_k \subseteq s'$ and the expected number of comparisons remaining cannot be less than the expected number of comparisons required for searching in a subdivision S_k. This proves our claim. By induction, we get that $ST(s) \geq 1 + \log n_s + \sqrt{\log n_s + 1} - 1$. Also, for $n_s = 1$, it holds that $ST(s) \geq 1 \geq 1 + \log 1 + \sqrt{\log 1 + 1} - 1$. Thus, in Eq. (3.9), we can bound $T(n)$ as follows:

$$T(n) \geq \sum_{s \in X} p_s \left(\log(1/p_s) + 1 + \log n_s + \sqrt{\log n_s + 1} - 1 \right).$$

Since $p_s = n_s/n$, we have

$$\begin{aligned} T(n) &\geq \log n + \sum_{s \in X} p_s \left(\sqrt{\log n_s + 1} \right) \\ &\geq \log n + (1/n) \sum_{s \in X} n_s \sqrt{\log n_s + 1}. \end{aligned} \tag{3.10}$$

We note that the function $f(x) = x\sqrt{\log x + 1}$ is convex in the interval $[1, +\infty)$, since its second derivative $f''(x) = \frac{1}{(2 \ln 2) x \sqrt{\log x + 1}} \left(1 - \frac{1}{(2 \ln 2)(\log x + 1)} \right)$ is positive. Thus, by Jensen's inequality,

$$\sum_{s \in X} n_s \sqrt{\log n_s + 1} \geq |X| \left(\frac{1}{|X|} \sum_{s \in X} n_s \right) \sqrt{\log \left(\frac{1}{|X|} \sum_{s \in X} n_s \right) + 1}.$$

Given that $\sum_{s \in X} n_s = n$ and $|X| = 2^t$, this simplifies to

$$\sum_{s \in X} n_s \sqrt{\log n_s + 1} \geq n \sqrt{\log n - t + 1}.$$

Using this fact in Eq. (3.10), we get

$$T(n) \geq \log n + \sqrt{\log n - t + 1}.$$

To show the induction for this case as well, it suffices to prove the inequality $\sqrt{\log n - t + 1} \geq \sqrt{\log n + 1} - 1$. Indeed, since $t < 2\sqrt{\log n + 1} - 1$ and $n > 1$, we have

$$\begin{aligned} \sqrt{\log n - t + 1} &> \sqrt{\log n - \left(2\sqrt{\log n + 1} - 1 \right) + 1} \\ &= \sqrt{\log n + 1} - 1, \end{aligned}$$

which completes the proof. □

3.6 Open Problems

Cells of arbitrary complexity

It would be nice to remove the assumption that cells have bounded complexity and extend our results for planar point location to any planar polygonal subdivision. However, the problem in this general case becomes quite different. We illustrate this by an example.

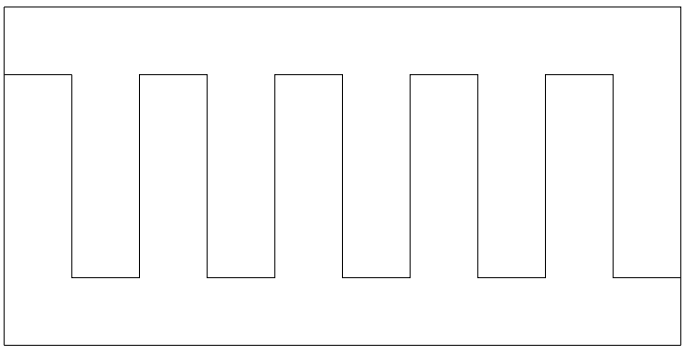


Figure 3.6 A subdivision with two cells of $\Omega(n)$ complexity.

Consider the planar subdivision of Figure 3.6. The entropy of the subdivision is clearly bounded by a constant since there are only two possible cells in which the query point can lie. Assuming that the query distribution is nonzero only close to the uppermost segment and to the lowermost segment, it is easy to construct a search structure with expected query time bounded by a constant. However, if we assume that the query distribution in the interior of the box enclosing the two cells is uniform, then it is not hard to see that any search structure must have expected query time at least $\Omega(\log n)$.

From this example, we draw two conclusions for expected-case point location in general subdivisions: (a) the entropy H is not necessarily a good lower bound, and (b) providing efficient expected-case query time requires information about the query distribution in the interiors of the cells (that is, knowing

only the probability p_z associated with each cell is not enough). Thus, an open problem is to develop data structures that achieve asymptotically the best possible expected-case query time, assuming that we have complete knowledge of the query distribution. We remark here that the non-convexity of the cells in the example is not really causing the difficulty of the problem, as there are also simple examples of convex subdivisions for which the same conclusions hold.

Second order term in expected query time

Between our upper bound of Theorem 3.1 and the lower bound of Theorem 3.4, there is a difference in the second order term of the expected query time by a $2\sqrt{2}$ factor. We would like to reduce this factor by providing stronger upper and lower bounds. Another interesting problem is to investigate the expected query time when the algorithm has information about the query distribution in the interior of each triangle. Note that in this case our lower bound argument does not hold. Can the $H+\Omega(\sqrt{H})$ expected query time still be justified or can the upper bound be improved to $H + O(1)$?

CHAPTER 4

SPACE REDUCTION BY ENTROPY-PRESERVING CUTTINGS

In this chapter we present a data structure that uses $O(n \log^* n)$ space and whose expected-case query time is $H + O(H^{2/3} + 1)$. Our methods are loosely based on an idea of Goodrich, Orletsky and Ramaiyer [15] who showed how to use $(1/r)$-cuttings to reduce the space requirements, in the context of worst-case planar point location. Given a subdivision S with n edges, and a parameter $r \geq 1$, a $(1/r)$-*cutting* is a partition of the plane into $O(r)$ trapezoids such that the interior of each trapezoid is intersected by at most n/r edges of S. If numeric weights are assigned to the edges, then this can be generalized to a *weighted* $(1/r)$-*cutting*, where now the total weight of edges intersecting any trapezoid is at most W/r, where W is the total weight of all the edges. However, the use of cuttings may refine the subdivision in a way that significantly increases its entropy, and this increases the expected query time. An important contribution we make is the notion of an *entropy-preserving cutting*, which additionally ensures that the entropy of the subdivision is increased by at most an additive constant.

To simplify the presentation, we will assume in this chapter that the given subdivision S is the trapezoidal decomposition of a set X of segments. We will also assume that we have complete information on the query distribution within each trapezoid. In particular, we assume that we can compute the probability that the query point lies within the intersection of a vertical slab of S with a trapezoid. We note that the results of the previous chapter required only knowing the probability of the query point lying in each cell of S.

4.1 Entropy-Preserving Cuttings

The main result of this section is to show the existence of entropy-preserving cuttings. This is given in the following lemma.

Lemma 4.1 *Let S be a trapezoidal decomposition of a set X of segments, with the query distribution specified. Let n denote the total number of segments in X. For any $r \geq 1$, we can partition the plane into $O(r)$ trapezoids satisfying the following properties. Let $\mathcal{C}$ denote the subdivision consisting of the $O(r)$ trapezoids, S' denote the subdivision formed by superimposing $\mathcal{C}$ on the given subdivision S, and τ denote any trapezoid in $\mathcal{C}$.*

(i) Either τ is a subset of some trapezoid in S, or the probability of the query point lying in τ is at most $1/r$.

(ii) The number of segments intersecting the interior of τ is at most n/r.

(iii) The entropy increase is bounded by a constant, i.e., $H_{S'} \leq H_S + O(1)$.

Proof. Recall that a trapezoid $z \in S$ is defined by at most four segments of X; the segments that support its top and bottom boundaries, the segment whose end point defines its right boundary, and the segment whose end point defines its left boundary. We first assign weights to the segments of X as follows. Each trapezoid $z \in S$ contributes a weight of $p_z/4$ to each of its defining segments. For each segment $x \in X$, let w'_x be the sum of the weight contribution from all the trapezoids incident to it. Finally, we set its weight $w_x = (w'_x + 1/n)/2$, and compute a standard weighted $(1/r')$-cutting for X. (Here r' is a function of r to be specified later in the proof.) We will assume that each trapezoid in this cutting is bounded from above and below by a segment in X. For this purpose we can use the cuttings in [10]. Let $\mathcal{C}'$ denote this cutting. We define the weight of any trapezoid in $\mathcal{C}'$ to be the total weight of all the segments intersecting its interior. By standard results on cuttings, $\mathcal{C}'$ has $O(r')$ trapezoids and the weight of each trapezoid is at most $O(1/r')$.

We modify the cutting $\mathcal{C}'$ by applying the following procedure on each trapezoid $z \in S$. If z contains a contiguous sequence of trapezoids of $\mathcal{C}'$, all of which are contained within z, then we replace each such maximal subsequence by one trapezoid. It is easy to see that after this modification, we still have a weighted $(1/r')$-cutting. (This is because the number of trapezoids in $\mathcal{C}'$ can only decrease and the weight of the merged trapezoids is zero.) Let $\mathcal{C}$ denote this new cutting. We claim that $\mathcal{C}$ satisfies the three properties given in the statement of the lemma.

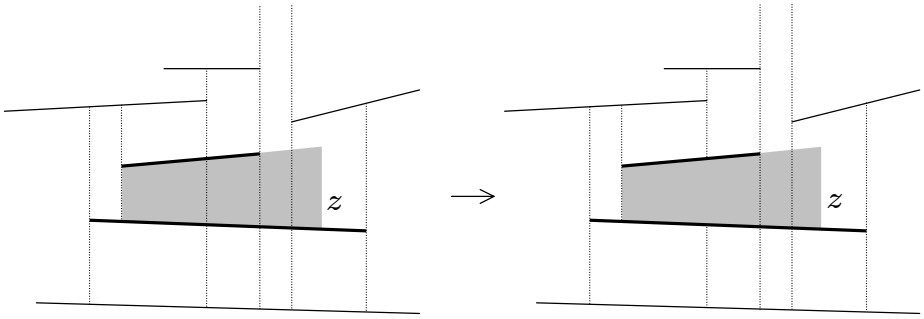


Figure 4.1 Cutting $\mathcal{C}'$ modified to cutting $\mathcal{C}$. Trapezoid z is shaded grey. Segments of the cutting are shown with solid lines (some of them are shown in bold for clarity).

Let τ denote any trapezoid in $\mathcal{C}$. Note that the weight of a segment x is at least a constant times the probability of the trapezoids of S that are incident to it. From this and the definition of a weighted cutting, it is easy to see that if there are one or more segments that intersect the interior of τ, then the total probability of the trapezoids of S that have a non-empty intersection with τ is $O(1/r')$; this implies that the probability of the query point lying in τ is $O(1/r')$. Otherwise τ must be completely contained within a trapezoid in S.

Also, since the weight of each segment is at least $1/(2n)$ and the weight of τ is $O(1/r')$, it follows that there can be at most $O(n/r')$ segments of S intersecting the interior of τ. Choosing r' to be a suitable large constant times r proves (i) and (ii).

Let S' denote the subdivision formed by superimposing $\mathcal{C}$ on S. Note that the cells of S' are trapezoids; we use the term *fragments* to refer to these cells. By definition of entropy and elementary calculus, it can be easily seen that the entropy of S' exceeds the entropy of S by at most $\sum_{z \in S} p_z \log f_z$ where f_z is the number of fragments in z. This quantity is no more than $\sum_{z \in S} p_z f_z$. In the remainder of the proof, we will show that $\sum_{z \in S} p_z f_z$ is bounded by a constant, which will prove (iii).

We distinguish between two kinds of fragments in z. Suppose that a fragment is the intersection of z with a trapezoid $\tau \in \mathcal{C}$; if τ is contained within z, we call it a type-1 fragment, otherwise it is a type-2 fragment. Let f_z^1 and f_z^2 denote the

number of fragments in z of type 1 and type 2, respectively. In view of how $\mathcal{C}'$ is modified to obtain $\mathcal{C}$, it is clear that there must be a fragment of type 2 between any two fragments of type 1, and so $f_z^1 \leq f_z^2 + 1$. Thus

$$\begin{aligned} \sum_{z \in S} p_z f_z &= \sum_{z \in S} p_z (f_z^1 + f_z^2) \\ &\leq \sum_{z \in S} p_z (2 f_z^2 + 1). \end{aligned} \tag{4.1}$$

To analyze this sum let $\mathcal{C}_1$ denote the set of trapezoids in $\mathcal{C}$ that are not completely contained within a cell of S. Observe that $\sum_{z \in S} p_z f_z^2$ is the same as $\sum_{\tau \in \mathcal{C}_1} \sum_{z \in S} p_z \delta(z, \tau)$, where $\delta(z, \tau)$ is 1 if z overlaps the interior of τ and is 0 otherwise. By our earlier reasoning, for any cell $\tau \in \mathcal{C}_1$, $\sum_{z \in S} p_z \delta(z, \tau)$ (the total probability of the trapezoids of S intersecting τ) is $1/r$. Since the number of trapezoids in $\mathcal{C}$, and hence in $\mathcal{C}_1$, is $O(r)$, it follows that $\sum_{z \in S} p_z f_z^2$ is $O(1)$. Substituting in Eq. (4.1), we have $\sum_{z \in S} p_z f_z = O(1)$, which is the desired claim. □

4.2 Search Structure

In this section we present an overview of a generic point location data structure. Specifics will be provided later. Given a subdivision S, we build a multi-way partition tree T. The point location queries are answered by a simple descent in this partition tree. Since the degree of a node in the tree can be quite large, we need to maintain an auxiliary point location search structure at each internal node, which is used to determine efficiently the child that contains the query point. For this purpose we will employ two different search structures in this paper. One structure uses $O(n \log n)$ space and achieves $H + O(H^{2/3} + 1)$ expected time [4] and the other uses $O(n)$ space and achieves $O(H)$ expected time [6] (in the thesis, they are presented in Chapters 3 and 5, respectively).

Each node u of T is associated with a trapezoid τ_u and a subdivision S_u restricted to this trapezoid. The root of T is associated with the entire plane and the given subdivision S. If the subdivision for u consists of a single cell, then u is a leaf. Otherwise u is an internal node. Let I denote the set of internal nodes.

There are two different types of internal nodes, denoted I' and I''. For a node $u \in I'$, if the subdivision S_u consists of m cells, we create m children each of which is a leaf representing one of these cells. For a node $u \in I''$, we construct an entropy-preserving $(1/r)$-cutting as described in Lemma 4.1 (the choice of r will be specified later). The node u has $O(r)$ children representing each of the trapezoids in the cutting. The associated subdivision for the children of u is the subdivision S_u restricted to the corresponding trapezoid. Note that these children are leaves if the corresponding subdivision has just one cell, otherwise they are internal nodes.

The following lemma proved in Knuth [21] will be useful in our analysis of the expected query time. Its proof uses induction from the bottom to the top of the tree.

Consider any multi-way tree T in which probabilities have been assigned to the leaves. For a node $u \in T$ define p_u to be the probability of visiting node u during the search (i.e., p_u is the total probability of the leaves descended from u). Define H_u, the entropy of an internal node u, to be the entropy of the probability distribution induced by the children of u. For example, if u has three children and the probabilities of visiting these children from node u are p_1, p_2, and p_3, respectively (note that $p_1 + p_2 + p_3 = 1$), then $H_u = p_1 \log(1/p_1) + p_2 \log(1/p_2) + p_3 \log(1/p_3)$.

Lemma 4.2 (Knuth) *The sum of $p_u H_u$ over all internal nodes u of a tree equals the entropy of the probability distribution on the leaves.*

4.3 Analysis of Expected-Case Query Time

Let S be the given subdivision and let T be any multi-way tree for S constructed as described in Section 4.2. Let I denote the set of internal nodes of T. Let H_L denote the entropy of the leaves of T. Let P_I denote the total probability of all the internal nodes of T, i.e., $P_I = \sum_{u \in I} p_u$.

The following lemma shows that the increase in the entropy of the leaves of T over the entropy of S is no more than a constant times the total probability of all its internal nodes. This bound holds irrespective of the sizes of the cuttings used for the nodes of T.

Lemma 4.3 *Let $I'' \subseteq I$ be the internal nodes of T for which an entropy-preserving cutting is constructed. Then*

$$H_L \leq H_S + O\left(\sum_{u \in I''} p_u\right) \leq H_S + O(P_I).$$

Proof. The second inequality in the lemma is trivial. We prove the first inequality by induction on the size of the subtree. For any node u, let L_u denote the leaves in the subtree rooted at u, and I''_u denote its internal nodes (a subset of I''). For the basis case, consider a tree with one leaf v. Then $H_{S_v} = H_{L_v} = 0$ and the claim holds trivially.

For the induction hypothesis, assume that the claim holds for all subtrees of size less than k. Let T be a subtree with k nodes. Let v denote the root of T with children v_i, $1 \leq i \leq d$. Let p_i denote the probability of visiting the ith child from node v. We consider two cases. If v does not have an associated cutting, then its children are leaves and represent the cells in S_v. Thus $H_{L_v} = H_{S_v}$ and since I''_v is empty we are done. Now suppose that v has an associated cutting. By the induction hypothesis, for $1 \leq i \leq d$,

$$H_{L_{v_i}} \leq H_{S_{v_i}} + O\left(\sum_{u \in I''_{v_i}} p'_u\right), \tag{4.2}$$

where p'_u denotes the probability of visiting u from node v_i. Applying Lemma 4.2, it is easy to see that

$$H_{L_v} = \sum_{i=1}^{d} \left(p_i \log \frac{1}{p_i} + p_i H_{L_{v_i}}\right) \tag{4.3}$$

and

$$H_{S'_v} = \sum_{i=1}^{d} \left(p_i \log \frac{1}{p_i} + p_i H_{S_{v_i}}\right), \tag{4.4}$$

where S'_v denotes the subdivision formed by superimposing the cutting for v on S_v. (Recall that the root u has d children and that $p_u = 1$.) Using Eqs. (4.2), (4.3), and (4.4), we obtain

$$H_{L_v} \leq \sum_{i=1}^{d} \left(p_i \log \frac{1}{p_i} + p_i \left[H_{S_{v_i}} + O\left(\sum_{u \in I''_{v_i}} p'_u\right)\right]\right) = H_{S'_v} + O\left(\sum_{u \in I''_v \setminus v} p_u\right),$$

where p_u denotes the probability of visiting u from node v.

Lemma 4.1 implies that $H_{S'_v} \leq H_{S_v} + O(1)$. Thus

$$H_{L_v} \leq H_{S_v} + O(1) + O\left(\sum_{u \in I''_v \setminus v} p_u\right) = H_{S_v} + O\left(\sum_{u \in I''_v} p_u\right),$$

since $p_v = 1$ (because it is the root of the subtree under consideration). This completes the proof by induction. □

In the next lemma we establish a general bound on the expected query time using tree T. Note this bound holds irrespective of the sizes of the cuttings used for the nodes of T.

Lemma 4.4 *Let I_1 and I_2 denote the set of internal nodes of T that are associated with point location search structures (to determine which child to visit) that guarantee expected query times of $H_u + O(H_u^{2/3} + 1)$ and $O(H_u)$, respectively. Here H_u denotes the entropy of the corresponding node u. Then the expected query time using T is at most*

$$H_S + O\left(P_I\left(H_S^{2/3} + 1\right)\right) + O\left(\sum_{u \in I_2} p_u H_u\right).$$

Proof. By linearity of expectation, the expected query time is at most

$$\begin{aligned}
&\sum_{u \in I_1} p_u \left[H_u + O\left(H_u^{2/3} + 1\right)\right] + \sum_{u \in I_2} p_u [O(H_u)] \\
&\quad = \sum_{u \in I_1} p_u H_u + O\left(\sum_{u \in I_1} p_u \left[H_u^{2/3} + 1\right]\right) \\
&\qquad + O\left(\sum_{u \in I_2} p_u H_u\right).
\end{aligned}$$

We can simplify the first term as $\sum_{u \in I_1} p_u H_u \leq \sum_{u \in I} p_u H_u = H_L$ by using Lemma 4.2. By Lemma 4.3, this is at most $H_S + O(P_I)$.

The second term can be written as

$$\sum_{u \in I_1} p_u \left[H_u^{2/3} + 1\right] \leq \sum_{u \in I} p_u \left[H_u^{2/3} + 1\right]$$

$$
\begin{aligned}
&= P_I \left(\sum_{u \in I} \frac{p_u}{P_I} \left[H_u^{2/3} + 1 \right] \right) \\
&\leq P_I \left(\left[\sum_{u \in I} \frac{p_u}{P_I} H_u \right]^{2/3} + 1 \right).
\end{aligned}
$$

Using Lemmas 4.2 and 4.3, this is at most $P_I^{1/3}(H_S + O(P_I))^{2/3} + P_I$, or at most $P_I^{1/3} H_S^{2/3} + O(P_I)$. Putting it all together, we get that the expected query time is at most $H_S + O(P_I(H_S^{2/3} + 1)) + O\left(\sum_{u \in I_2} p_u H_u\right)$. □

The following lemma shows that if the entropy is not very small, then we can provide expected query time of $H_S + o(H_S)$ using linear space. For a node u in T, let d_u denote its degree (i.e., number of its children).

Lemma 4.5 *If $H_S \geq \log\log n$, then we can construct a search structure that answers point location queries in expected time $H_S + O(H_S^{2/3} + 1)$ using $O(n)$ space.*

Proof. We build a 4-level search tree T as follows. We construct an entropy-preserving cutting for the root (level 1) using $r = n/\log n$, and for the internal nodes at level 2 using $r = \log n/\log\log n$. For any internal node u at level 3, we do not compute a cutting, instead it has a child corresponding to each cell in S_u. For any internal node u at level 1 (root) and level 2, we use the $O(d_u \log d_u)$ space auxiliary search structure that provides expected query time of $H_u + O(H_u^{2/3} + 1)$. For any internal node u at level 3, we use the $O(d_u)$ space search structure that provides expected query time $O(H_u)$.

Applying Lemma 4.4 it follows that the expected query time is

$$
H_S + O(P_I(H_S^{2/3} + 1)) + O\left(\sum_{u \in I_2} p_u H_u \right).
$$

The sum of the probabilities of the nodes at any one level is at most one. Thus, P_I is at most 3, and $\sum_{u \in I_2} p_u \leq 1$. Thus the expected query time is at most $H_S + O(H_S^{2/3} + 1) + O(\max_{u \in I_2} H_u)$. By Lemma 4.1, it follows that the number of segments intersecting the interior of a trapezoid associated with an internal

node u at level 3 is at most $\log\log n$, and so H_u can be at most $O(\log\log\log n) = O(\log(H_S))$. Thus the expected query time is bounded by $H_S + O(H_S^{2/3} + 1)$.

It is easy to check that the total space used by auxiliary search structures at each of the levels 1, 2, and 3, is $O(n)$. Thus the total space used is also $O(n)$. □

Let $\log^{(i)} n$ denote the function obtained by iterating the log function i times, i.e., $\log^{(0)} n = n$ and $\log^{(i)} n = \log(\log^{(i-1)} n)$ for $i > 0$. The proof of the above lemma can be easily generalized under the condition that $H_S \geq \log^{(c)} n$, where c is any constant, and the same bounds continue to hold. However if the entropy is extremely low, it is not clear how to achieve linear space. But as the following lemma shows we can still achieve a significant space reduction.

Lemma 4.6 *We can construct a search structure that answers point location queries in expected time $H_S + O(H_S^{2/3} + 1)$ using $O(n2^{O(\log^* n)})$ space.*

Proof. By Lemma 4.5, the theorem is obviously true if $H_S \geq \log\log n$. So let us assume that $H_S < \log\log n$. We build a search tree T for the subdivision S as follows. We construct entropy-preserving $(1/r)$-cuttings for all internal nodes of T, where the parameter r depends on the level i as (given by the equation) $r = \log^{(i-1)} n / \log^{(i)} n$. The point location structure used for each internal node u uses $O(d_u \log d_u)$ space and ensures expected query time of $H_u + O(H_u^{2/3} + 1)$.

By Lemma 4.1, the complexity of the subdivision associated with the internal nodes at level i is at most $\log^{(i-1)} n$, which implies that the depth of T is at most $\log^* n$. It is easy to check that the space used by the point-location search structure at the root is $O(n)$ and this increases by a constant factor with each successive level. Since the depth of T is $\log^* n$, the total space used is $O(n2^{O(\log^* n)})$.

By Lemma 4.4, the expected query time is $H_S + O(P_I(H_S^{2/3} + 1))$. In the remainder we will show that P_I is bounded by a constant, which implies the desired bound on the expected query time and completes the proof.

To show the bound on P_I, we will establish that the total probability of the internal nodes at any level i, $2 \leq i \leq \log^* n$, is no more than $1/2^i$. Summing over all the levels, it will follow that P_I is bounded by a constant. For the sake of contradiction suppose that for some $i \geq 2$, the total probability of the

internal nodes at level i is greater than $1/2^i$. By Lemma 4.1, the probability of any of these internal nodes is at most $\log^{(i-1)} n/n$. Consider the set of leaves generated by these internal nodes. Clearly any of these leaves has probability at most $\log^{(i-1)} n/n$. Thus the contribution to the entropy of these leaves is at least $(1/2^i)\log(n/\log^{(i-1)} n)$. Recall that $i \leq \log^* n$. Thus

$$H_L \geq \frac{1}{2^{\log^* n}} \left(\log n - \log^{(i)} n\right) \geq \frac{\log n - \log\log n}{2^{\log\log\log n}} = \frac{\log n}{\log\log n} - 1. \tag{4.5}$$

However, since the depth of T is at most $\log^* n$, P_I is at most $\log^* n$, and so by Lemma 4.3, H_L is at most $\log\log n + O(\log^* n)$. This contradicts Eq. (4.5) (for large enough n). Hence, the total probability of the internal nodes at level $i, i \geq 2$, is at most $1/2^i$. □

Theorem 4.1 *We can construct a search structure that answers point location queries in expected time $H_S + O(H_S^{2/3} + 1)$ using $O(n \log^* n)$ space.*

Proof. Recall that Lemma 4.6 gives us a search structure that provides expected query time of $H + O(H^{2/3} + 1)$ using $O(n2^{c\log^* n})$ space, where c is a suitable constant. We build a 3-level search tree T for the subdivision S as follows. For the root we construct an entropy-preserving $(1/r)$-cutting using $r = n/2^{c\log^* n}$, and use the search structure provided by Lemma 4.6 to do point location. The size of the subdivision corresponding to the nodes at level 2 is at most $2^{c\log^* n}$. For any internal node u at level 2, we do not compute a cutting, instead each cell in S_u becomes a child of internal node u. For these nodes we use the $O(d_u \log d_u)$ space auxiliary search structure that provides expected query time of $H_u + O(H_u^{2/3} + 1)$. It is easy to see that the space used by the search structure at the root is $O(n)$ and all the search structures at level 2 together use space $O(n \log^* n)$. Also, by Lemma 4.4, it is clear that the expected query time is bounded by $H_S + O(H_S^{2/3} + 1)$. □

4.4 Open Problems

A natural question that arises from our work in this chapter is whether $H + o(H)$ expected query time can be achieved in less than $O(n \log^* n)$ space. Indeed,

observe that starting with a method that achieves $H + o(H)$ expected query time and $O(n \log n)$ space, we have used entropy-preserving cuttings to obtain a method that achieves $H + o(H)$ expected query time and $O(n \log^* n)$ space (Theorem 4.1). Using the same approach again, but starting with the method of Theorem 4.1, we can reduce the space further to $O(n \log^{**} n)$, while still retaining $H + o(H)$ expected query time. (Here, $\log^{**}$ denotes the number of times we need to apply the $\log^*$ function in succession, starting with argument n to obtain a value less than one.) The technique can, in fact, be repeatedly applied leading to still better space bounds.

In light of this fact, it would be interesting to show that there is a method that uses $O(n\alpha(n))$ space, where $\alpha(n)$ is the inverse Ackermann function, as is the case with several other geometric problems. Of course, one might also be able to ultimately prove that $H + o(H)$ time is possible in linear space, thus matching the analogous result in one dimension.

CHAPTER 5

WEIGHTED RANDOMIZED INCREMENTAL METHOD

The solutions we considered so far for expected-case planar point location seem to be complicated. In contrast, for the 1-dimensional problem, Mehlhorn's nearly optimal binary search trees [24] or Seidel and Aragon's randomized search trees [37], for example, are quite simple. It is natural to ask if there are simple solutions in two-dimensions as well. In this chapter, we show that a very small modification of the well-known and simple randomized incremental algorithm can be applied to produce a data structure of expected linear size that can answer point location queries in $O(H)$ average time. We also present empirical evidence for the practical efficiency of this approach.

Our main result is given below. To minimize confusion we use the term *expected* when referring to variations that are caused by the the random choices made by the algorithm and *average* when referring to variations due to the random distribution of query points.

Theorem 5.1 *Consider an n-segment polygonal subdivision S in which each cell z has constant combinatorial complexity and is associated with a probability p_z that the query point falls within this cell. There is a randomized algorithm with expected running time $O(n \log n)$ that produces a data structure of expected space $O(n)$ such that for any fixed query point q, the expected time needed to locate q is at most $5 \ln(1/p_z) + O(1)$, where z is the cell containing q. (In all cases, the expectation is over the random choices made by the algorithm.) Thus the average query time (over the query distribution) is at most $(5 \ln 2)H + O(1) = 3.47H + O(1)$.*

As in the standard randomized incremental algorithm, our analysis of the space and query times are based on a standard backwards analysis. Such analyses

are based on the notion that, since objects are inserted randomly, at any given stage every object is equally likely to have been the last to be inserted. In our case this assumption does not hold. As a result, the analysis of both the space and expected query time is significantly more complicated than in standard backwards analyses. We overcome this problem by a trick of associating some number of pebbles with each of the edges of the subdivision. The number is a function of the query probabilities for the incident subdivision cells. The pebbles are drawn in random order, and a segment is inserted when its first pebble is drawn.

As in Chapter 3, we assume that S consists of interior disjoint triangles and the probability of the query point lying in the unbounded face is zero. We also assume for simplicity that no edge of S is vertical.

5.1 Algorithm

The worst-case randomized incremental algorithm has been briefly described in Section 2.2.4. The reader is referred to the text by de Berg et al. [7] for more detailed information. Recall that the algorithm inserts the segments of S one by one in random order and updates the subdivision after each insertion (Figure 5.1).

Before presenting our algorithm, we begin by showing why the "obvious" approach to the problem does not work. Recall that the randomized incremental approach adds edges of the subdivision one by one in random order. Intuitively, we want the edges bounding triangles with high probability to be added early in the process, since then any query that falls within this triangle will have its location resolved near the root of the history graph. This suggests a *simple weighting scheme* in which each edge of the subdivision is assigned a weight that is derived from the probability that the query point lies in either of its incident triangles, say the sum of these two probabilities. Then we apply the incremental algorithm, but insert the edges in decreasing order of weight (rather than using a random permutation).

The problem is that an adversary can adjust the probabilities to cause the algorithm to insert the edges in an order so that the space of the resulting structure would be $\Theta(n^2)$. Consider the example shown in Figure 5.2. Below there are $\Theta(n)$ triangles, each having a probability of $1/n$. Above there are $\Theta(n)$ triangles

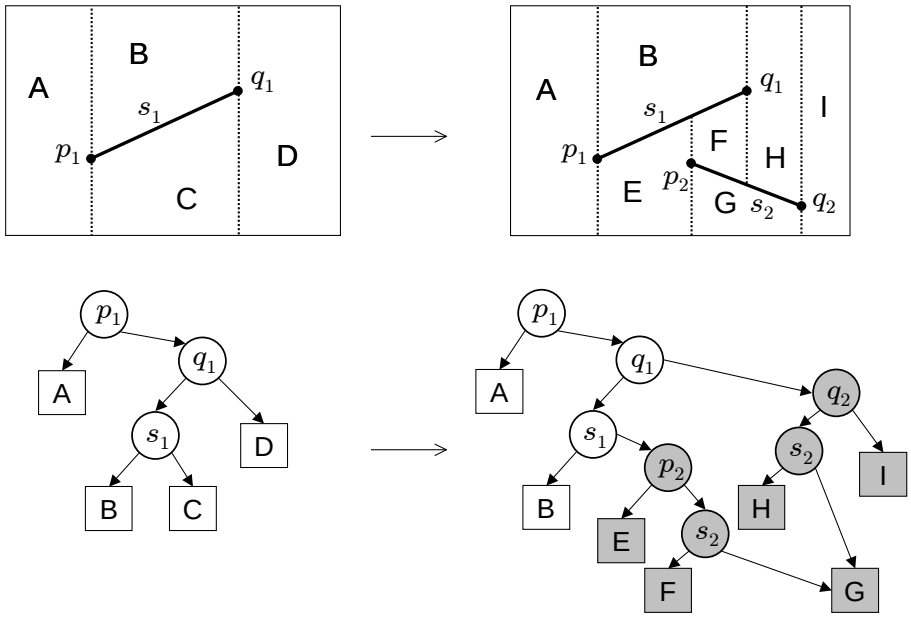


Figure 5.1 Incremental algorithm for trapezoidal map and the associated search structure.

with probabilities of the form $1/(2^i n)$ with i increasing from the top to the bottom. The insertion order provided by the simple weighting scheme results in the insertion of edges bounding the $\Theta(n)$ lower triangles, thus creating $\Theta(n)$ vertical slabs. After this, the edges bounding the upper triangles will be inserted in order from top to bottom. Since each intersects all $\Theta(n)$ vertical slabs, each insertion results in $\Theta(n)$ updates to the trapezoidal map, which leads to $\Theta(n^2)$ total space. Note that the problem is not due merely to the absence of randomization, since even a randomized algorithm which samples according to the given weights will result in a substantially similar situation. The source of the problem comes from the huge variation that is possible in the weights.

Weight assignment

The standard randomized incremental algorithm can be thought of as a method in which all segments have equal weight of $1/n$. Our solution is to modify the simple weighting scheme so that the ratio of any two segment weights is $O(n)$. Remarkably, this simple fix is all that is needed to eliminate the above space problem, and yet it provides a true enough representation of the probability

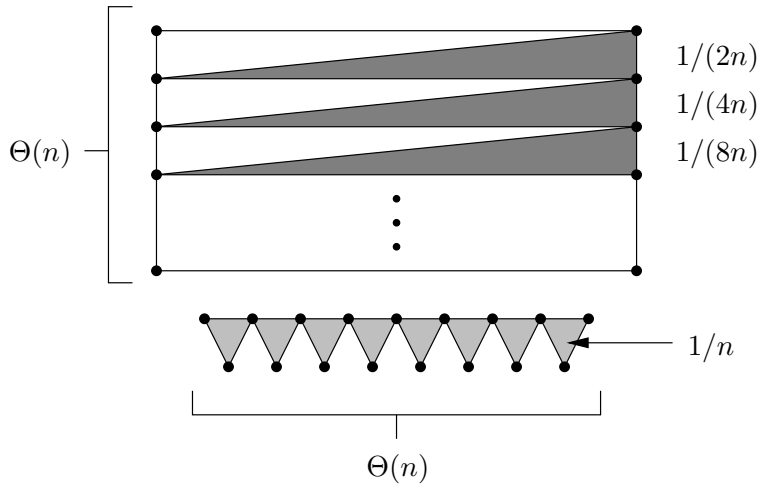


Figure 5.2 A bad instance for simple weighting.

distribution to achieve the desired entropy bounds on average search times.

In detail our algorithm works as follows. Let m denote the number of triangles in S. Let X denote the set of edges of these triangles. Let $n = O(m)$ denote the number of segments in X.

Let p_z denote the probability of the query point lying in a cell $z \in S$. Assign a probability of $p_z/3$ to each of the three segments bounding the triangle z. For a segment x, let p_x denote the sum of the two probabilities it derives from its incident triangles. Assign a weight $w_x = \lceil Kp_x n \rceil$ to each segment x, where K is any constant. We will see later that the choice of K affects only the multiplicative constant in the space bound and the additive constant in the query time. After this weight assignment we run the standard randomized incremental algorithm in which we sample the segments based on their weights.

5.2 Analysis

We now analyze the space and average query time of the search structure. To simplify the presentation we assume that no two distinct endpoints of segments have the same x-coordinate; however two or more segments are allowed to share an endpoint. Observe that the weight w_x of any segment x is an integer between 1 and $Kn + 1$, and the total weight W of all the segments is at most $(K + 1)n$ (because $\sum_x w_x \leq \sum_x (Kp_x n + 1) \leq (K + 1)n$).

For the purpose of analysis, it is useful to consider the following randomized

incremental algorithm, which can be easily verified is equivalent to the actual algorithm described in the last section. With each segment x we associate w_x *pebbles*. Note that by definition w_x is an integer. We use P to denote the set of pebbles associated with all the segments. At each step of the algorithm we pick any of the remaining pebbles with *equal* probability. If the pebble represents a segment that has not yet been inserted, then the segment is inserted as usual into the trapezoidal map and search structure. Otherwise the pebble is simply ignored (i.e., it has no affect on the trapezoidal map or search structure).

5.2.1 Space analysis

We first analyze the expected space used by the search structure. To compute this quantity we will bound the expected structural change in the trapezoidal map when the ith pebble is inserted, and sum this over the W steps of the algorithm. This will also give a bound on the space used by the search structure. Let k_i denote the number of new trapezoids created when the ith pebble is inserted. We bound the expected value of this quantity using backwards analysis.

Consider a fixed set of i pebbles $P^i \subseteq P$. Let $\mathcal{T}(P^i)$ denote the trapezoidal map for the set of segments associated with the pebbles in P^i. For a trapezoid $\triangle \in \mathcal{T}(P^i)$, let $\delta(\triangle, p) = 1$ if pebble p would have caused $\triangle$ to be created, had p been inserted last, and 0 otherwise. The expected value of k_i, subject to the condition that P^i is the set of first i pebbles, is given by

$$E[k_i|P^i] = \frac{1}{i} \sum_{p \in P^i} \sum_{\triangle \in \mathcal{T}(P^i)} \delta(\triangle, p).$$

Reversing the order of summation,

$$E[k_i|P^i] = \frac{1}{i} \sum_{\triangle \in \mathcal{T}(P^i)} \sum_{p \in P^i} \delta(\triangle, p).$$

Note that each trapezoid $\triangle \in \mathcal{T}(P^i)$ is defined by at most two segments that bound its top and bottom walls, and two endpoints whose vertical extensions form its left and right walls. Clearly the trapezoid $\triangle$ is created in the last step if and only if one of the following four conditions hold: (i) there is exactly one pebble in P^i such that the associated segment defines the top wall of $\triangle$ and this

pebble is inserted last; (ii) same as (i) for the bottom wall; (iii) there is exactly one pebble in P^i such that an endpoint of the associated segment defines the left wall of $\triangle$ and this pebble is inserted last; and (iv) same as (iii) for the right wall. Thus there are at most four pebbles in P^i with the property that if they were inserted last they would have created $\triangle$, i.e., $\sum_{p \in P^i} \delta(\triangle, p) \leq 4$ and we get

$$E[k_i | P^i] \leq \frac{1}{i} \sum_{\triangle \in \mathcal{T}(P^i)} 4 = \frac{1}{i} 4O(i) = O(1).$$

Here we have used the fact that the number of segments in $\mathcal{T}(P^i)$ is at most i and so by standard results $\mathcal{T}(P^i)$ contains at most $O(i)$ trapezoids. Since the expected value of k_i does not depend on the choice of P^i we can conclude that it holds unconditionally. Summing $E[k_i]$ over the W steps of the algorithm it follows that the expected total number of new trapezoids created is $O(W)$. Recalling that $W \leq (K+1)n$, this gives a bound of $O(n)$ on the expected number of new trapezoids, and hence on the space used by the search structure.

5.2.2 Average query time

We now analyze the *expected* value of the *average* query time provided by the search structure. (Recall that the average query time refers to the average over the different locations of the query point. We are interested in the expectation of this quantity over the random choices made by the algorithm.)

We call two query points equivalent if they follow the same path through the search structure. Consider a partitioning of the plane into slabs by passing a vertical line through the endpoints of all the segments. Each slab is decomposed into trapezoids by the segments that cross the slab. It is easy to see that the query points inside any such trapezoid are equivalent. Let $\mathcal{T}$ denote the set of trapezoids in all the slabs. Then the average query time is given by $\sum_{\triangle \in \mathcal{T}} p_\triangle d_\triangle$, where $p_\triangle$ is the probability of the query point lying in $\triangle$, and $d_\triangle$ is the length of the search path for the query points in $\triangle$.

Let q be a query point inside a trapezoid $y \in \mathcal{T}$. Let y be contained within triangle $z \in S$. We will prove that the expected length of the search path for q, $E[d_y]$, is at most $5 \ln(1/p_z) + O(1)$. It is easy to see that this implies that the expected value of the average query time is $(5 \ln 2)H + O(1)$.

Let ℓ_i be a random variable that takes the value 1 if the left wall of the trapezoid containing q changes when the ith pebble is inserted, and is 0 otherwise. Similarly define the random variables r_i, t_i, and b_i for the right, top, and bottom walls, respectively. We employ the following observation made by Seidel [35]: the length of the search path for q grows by at most $2\ell_i + r_i + t_i + b_i$ when the ith pebble is inserted. Let $\ell = \sum_{1 \leq i \leq W} \ell_i$. Similarly define r, t, and b. By linearity of expectation, $E[d_y]$ is at most $2E[\ell] + E[r] + E[t] + E[b]$. We will show that $E[\ell], E[r], E[t]$, and $E[b]$ are all bounded by $\ln(1/p_z) + O(1)$, which implies the desired bound on $E[d_y]$.

We will only prove that $E[\ell] = \ln(1/p_z) + O(1)$; the other bounds can all be proved in a similar way. Let u be the trapezoid in the final trapezoidal decomposition $\mathcal{T}(X)$ that contains q. Let u be contained within triangle $z \in S$. Let a_1, a_2, and a_3 denote the three sides of triangle z. Let $M_r, 1 \leq r \leq 3$, be a random variable denoting the step of the algorithm in which a pebble associated with segment a_r is first inserted, and let M be the random variable $\max_{1 \leq r \leq 3} M_r$. We can write $E[\ell]$ as follows:

$$E[\ell] = \sum_{j=1}^{W} \Pr[M = j] \cdot E[\ell | M = j]. \tag{5.1}$$

In order to compute $E[\ell | M = j]$ observe first that the trapezoid containing q does not change after a pebble associated with each side of z has already been inserted. Thus $E[\ell_i | M = j]$ for $i > j$ is 0. Obviously $E[\ell_j | M = j] \leq 1$ since the largest value that the random variable ℓ_j can take is one.

We next prove that $E[\ell_i | M = j] \leq 1/i$ for $i < j$. This analysis is based on backwards analysis. Suppose that the jth pebble inserted is associated with segment a_k, where $1 \leq k \leq 3$, and $P^{j-1} \subseteq P$ is any fixed set of $j-1$ pebbles that contains at least one pebble associated with the two segments $a_r, r \in \{1,2,3\} - \{k\}$, and no pebble associated with a_k. We claim that the expected value of ℓ_i subject to the condition that P^{j-1} is the set of the first $j-1$ pebbles inserted is at most $1/i$, irrespective of the choice of a_k and P^{j-1}. By the definition of M, it is clear that this claim would imply that $E[\ell_i | M = j] \leq 1/i$ for $i < j$.

To prove this claim, let $P^i \subseteq P^{j-1}$ be any fixed set of i pebbles. We compute the expected value of ℓ_i subject to the condition that P^i is the set of the first i pebbles inserted. Let $\triangle$ be the trapezoid in $\mathcal{T}(P^i)$ that contains q. The left

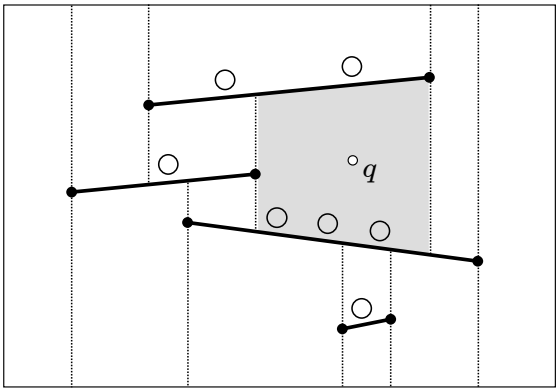


Figure 5.3 Trapezoidal map $\mathcal{T}(P^i)$ after step $i = 7$ (pebbles appear as circles).

wall of $\triangle$ would have changed in step i if and only if there is exactly one pebble in P^i such that an endpoint of the associated segment defines the left wall of $\triangle$ and this pebble is inserted last (i.e., in step i). The probability of this event is at most $1/i$, and thus the expected value of ℓ_i is at most $1/i$. Note that this bound holds irrespective of the choice of P^i. This establishes the claim at the end of the last paragraph.

Combining the three cases: (i) $E[\ell_i|M = j] = 0$ for $i > j$, (ii) $E[\ell_j|M = j] \leq 1$, and (iii) $E[\ell_i|M = j] \leq 1/i$ for $i < j$, we get

$$E[\ell|M = j] \leq 1 + \sum_{i=1}^{j-1} \frac{1}{i} \leq \ln(j) + 2.$$

Substituting this in Eq. (5.1) we obtain

$$E[\ell] \leq \sum_{j=1}^{W} \Pr[M = j] \cdot (\ln(j) + 2) = E[\ln(M)] + 2. \tag{5.2}$$

Since ln() is a concave function, by Jensen's inequality we have $E[\ln(M)] \leq \ln(E[M])$. In the remainder we will prove that $E[M] \leq O(1/p_z)$. Using these facts in Eq. (5.2) gives the desired bound on $E[\ell]$.

Since the algorithm samples W pebbles without replacement, it can be easily shown that the expected number of trials needed to select one of the w_{a_r} pebbles associated with a_r is $(W+1)/(w_{a_r}+1)$. Thus $E[M_r] = (W+1)/(w_{a_r}+1) \leq W/w_{a_r}$. Since $w_{a_r} = \lceil Kp_{a_r}n \rceil \geq Kp_{a_r}n = Knp_z/3$ and $W \leq (K+1)n$, it follows that $E[M_r] \leq 3(1+1/K)/p_z$. Since $M = \max_{1 \leq r \leq 3} M_r$, it follows that $M \leq \sum_{r=1}^{3} M_r$.

Thus $E[M] \leq \sum_{r=1}^{3} E[M_r] \leq 9(1 + 1/K)/p_z$. This completes the analysis of the expected value of the average query time.

Parameter K

Working out the constants in our analysis, we obtain an upper bound on the expected query time of $(5 \ln 2)H + 5[\ln(1 + 1/K) + \ln 9 + 2]$. Note that as K increases the additive constants in the expected query time decrease. However, since the space used is $O((K + 1)n)$, we obtain a poorer guarantee on the space.

Worst-case performance and preprocessing time

Our analysis also shows that for any fixed query point the expected query time is at most $O(\log n)$. Since $w_{a_r} \geq 1$ and $W \leq (K + 1)n$, so $E[M] \leq 3(K + 1)n$. Thus $E[\ell] \leq \ln(n) + \ln(K + 1) + \ln 3 + 2$, which implies that the expected length of the search path for q is $5 \ln(n) + O(1)$. Using this it can be shown that the construction time is $O(n \log n)$ in the expected case.

Alternative method based on weighted random sampling

Mulmuley introduced a worst-case point location data structure based on generalizing the idea of skip lists to two dimensions [27]. By making a small modification to his method, we get another way of achieving $O(H)$ expected time and $O(n)$ space. We briefly describe the main idea, assuming the reader is familiar with the skip list approach. Our idea is to build a weighted skip list by associating pebbles with the segments, exactly as in the weighted randomized incremental algorithm. Starting with all the pebbles at the bottom level, we randomly promote half of the pebbles at each level to the next higher level. A segment is stored at all the levels at which an associated pebble is present. The rest of the construction and the search algorithm are equivalent to Mulmuley's method, except that we terminate the search as soon as we reach a trapezoid whose interior is not intersected by any segment in the subdivision.

5.3 Experimental Results

We have run experiments on the weighted randomized incremental algorithm comparing its performance to the standard unweighted version. We measured the query time of the algorithm by counting the average number of comparisons needed to answer a query and the space used by the search structure. In the weighted algorithm, we used $K = 5$ for the weight assignment (i.e., $w_x = \lceil 5p_x n \rceil$).

We used a uniform and a non-uniform subdivision for our experiments. These subdivisions are Delaunay triangulations of 10,000 points generated as follows. For the uniform subdivision, the points were generated uniformly in a unit square. For the non-uniform subdivision, ten points were chosen from the uniform distribution and a Gaussian distribution with a standard deviation of 0.04 was centered at each point (we will refer to this as the Clustered Gaussian distribution).

The query points were generated from the Clustered Gaussian distribution with ten centers. We obtained different distributions by varying the standard deviation of the Gaussian distribution from 0.001 to 0.2, which gave us a way of controlling the entropy of the subdivision.

For each subdivision and query distribution, we used a training set of 100,000 points to estimate the probability of the query point lying in each cell. We conducted ten runs for a given subdivision and fixed cell probabilities. In each run, we built the search structure for both the weighted and unweighted randomized incremental algorithms, and computed the average number of comparisons over 30,000 query points. Finally we computed the average of this over the ten runs, and plotted it as a function of the standard deviation and the entropy of the subdivision.

The results for the uniform and non-uniform subdivision are shown in Figure 5.4 and 5.5, respectively. We summarize the key observations:

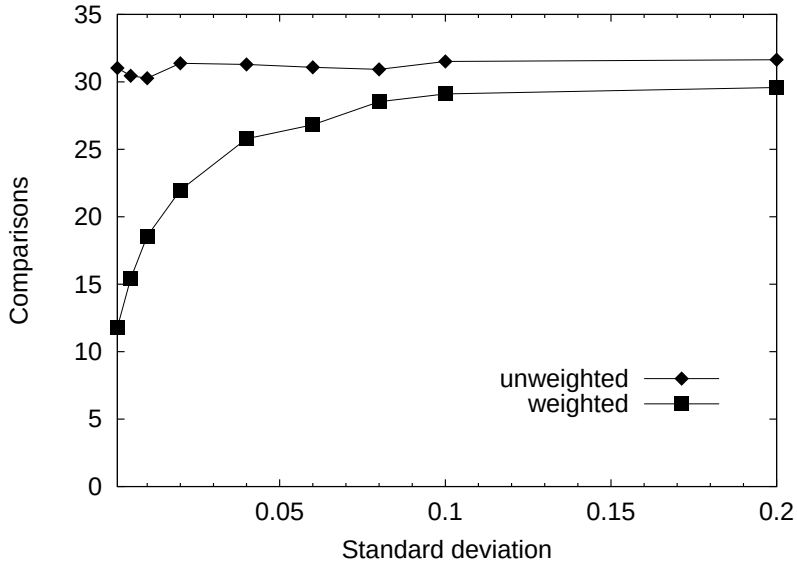
(a) The average number of comparisons for the weighted algorithm is always less than that for the unweighted algorithm. As expected when the standard deviation (entropy) is small, the advantage of the weighted over the unweighted algorithm is much larger. For example, when the standard deviation is 0.01, the weighted algorithm uses about 40–50% fewer comparisons than the unweighted algorithm.

(b) The average number of comparisons for the unweighted algorithm shows no relation to the standard deviation (entropy) of the distribution. In contrast the average number of comparisons for the weighted algorithm appears to grows linearly with the entropy. (By fitting a line to the data, it appears that the average number of comparisons grows as $1.94H + 3.11$ for the uniform subdivision, and as $1.75H + 4.49$ for the non-uniform subdivision. This suggests that the actual performance of the algorithm is better than the bound of about $3.47H + 21.90$ predicted by our theoretical analysis.)

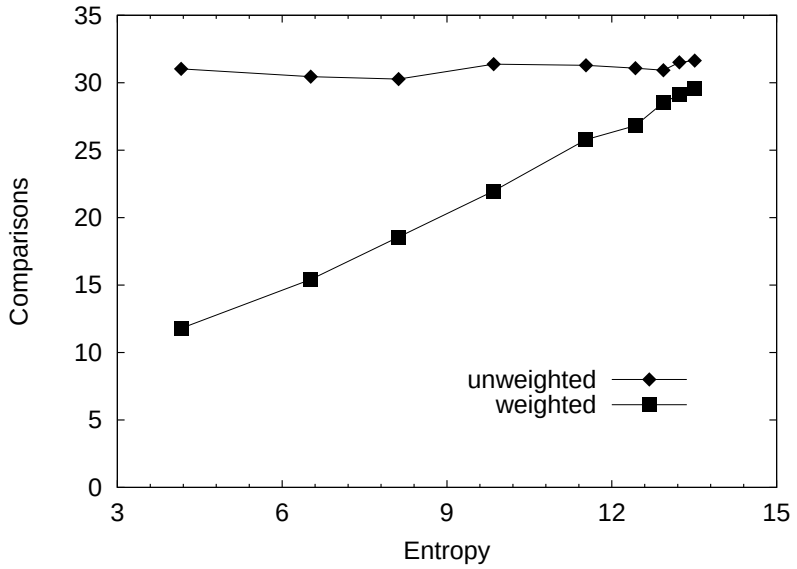
(c) The total number of nodes in the search structure for the weighted and unweighted algorithm is similar and is about $9n$, where n is the number of segments. The space used does not seem to depend on the entropy of the subdivision.

5.4 Open Problems

We observe from the experiments that the constant factor in front of the entropy term in the average query time tends to be much smaller than the constant factor we obtained by our analysis (i.e., the constant is estimated below 2 in practice, and around 3.5 in theory). An open problem is to find a more accurate analysis that reduces the difference between these two constant factors.

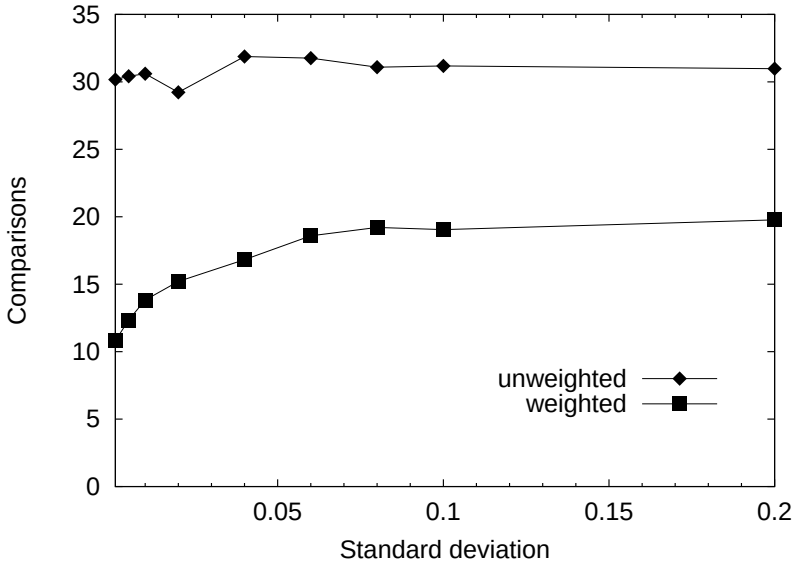


(a)

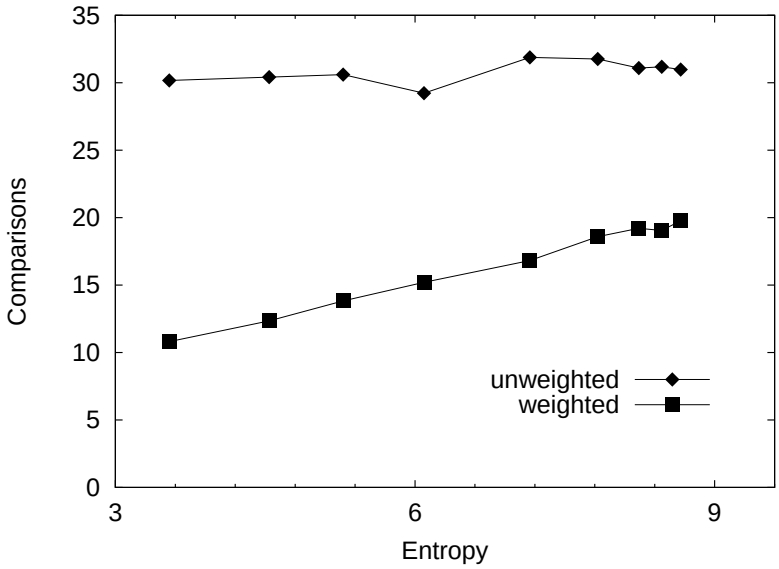


(b)

Figure 5.4 Uniform subdivision: Comparisons versus (a) standard deviation and (b) entropy.



(a)



(b)

Figure 5.5 Non-uniform subdivision: Comparisons versus (a) standard deviation and (b) entropy.

CHAPTER 6

CONCLUSIONS

Prior to this thesis, there has been very limited work on planar point location from the perspective of expected query time. Further, this work either led to a large constant in front of the entropy term or required strong assumptions on the query distribution. Here we have developed methods for answering planar point location queries in nearly optimal expected query time and using close to linear space. The results hold for subdivisions consisting of cells with bounded complexity and known probabilities. Our first method has expected query time $H + O(H^{1/2})$ and uses $O(n^{1+\epsilon})$ space, while our second method has expected query time $H + O(H^{2/3})$ and uses $O(n \log n)$ space. Both methods partition each cell into a small number of fragments according to the cell probability, and place the fragments at an appropriate height in the tree, reflecting their visiting frequency. It would be interesting to obtain expected-case results for more general subdivisions as well.

Assuming that the only information on the query distribution available is the probability of the query point lying in each cell, we have proved an almost tight lower bound on the expected query time of $H + \Omega(H^{1/2})$. It remains an open problem to determine the second largest term in the expected query time, assuming that we have full knowledge of the query distribution.

Following a different strategy, we have shown in a third method that $H + O(H^{2/3})$ expected query time can be achieved with $O(n \log^* n)$ space. This space reduction is accomplished by the use of *entropy-preserving* cuttings. This new type of cutting performs divide and conquer simultaneously in both the geometric and the probability space, and increases the entropy of the subdivision, a critical lower bound, only by an additive constant. In the future, we would like to solve the important problem of whether $H + o(H)$ is possible with linear space, as is the case in one dimension.

On the practical side, we have presented a linear-space data structure for the same problem with expected query time bounded by $O(H)$, based on a simple modification of the well known randomized incremental algorithm for trapezoidal maps. First, each segment of the subdivision is assigned a suitable weight. Then, the segments are picked randomly according to their weights, instead of uniformly as in the original algorithm. In our experiments, we have observed that, for non-uniform query distributions, this scheme gives significantly faster query times than the standard approach.

Finally, another challenging question is to construct an adaptive data structure for planar point location, that is, a data structure that achieves amortized query time $O(H)$ without any prior knowledge of the query distribution. Such behavior is known to hold in one dimension for splay trees and it would have many applications in the plane.

REFERENCES

[1] U. Adamy, "Very fast planar point location?," Diplomarbeit, Univ. Des Saarlandes, FB Informatik, 1998.

[2] U. Adamy and R. Seidel, "Planar point location close to the information-theoretic lower bound," in *Proc. 9th ACM-SIAM Sympos. Discrete Algorithms*, pp. 609–618, 1998.

[3] S. Arya, S.-W. Cheng, D. M. Mount, and H. Ramesh, "Efficient expected-case algorithms for planar point location," in *Proc. 7th Scand. Workshop Algorithm Theory*, vol. 1851 of *Lecture Notes Comput. Sci.*, pp. 353–366, Springer-Verlag, 2000.

[4] S. Arya, T. Malamatos, and D. M. Mount, "Nearly optimal expected-case planar point location," in *Proc. 41 Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 208–218, 2000.

[5] S. Arya, T. Malamatos, and D. M. Mount, "Entropy-preserving cuttings and space efficient planar point location," in *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, pp. 256–261, 2001.

[6] S. Arya, T. Malamatos, and D. M. Mount, "A simple entropy-based algorithm for planar point location," in *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, pp. 262–268, 2001.

[7] M. D. Berg, M. V. Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*. Berlin: Springer-Verlag, 1997.

[8] P. B. Callahan and S. R. Kosaraju, "A decomposition of multi-dimensional point sets with applications to k-nearest-neighbors and n-body potential fields," *Journal of the ACM*, vol. 42, no. 1, pp. 67–90, 1995.

[9] B. Chazelle, "A theorem on polygon cutting with applications," in *Proc. 23th Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 339–349, 1982.

[10] B. Chazelle and J. Friedman, "A deterministic view of random sampling and its use in geometry," *Combinatorica*, vol. 10, no. 3, pp. 229–249, 1990.

[11] R. Cole, "Searching and storing similar lists," *J. Algorithms*, vol. 7, pp. 202–220, 1986.

[12] D. P. Dobkin and R. J. Lipton, "Multidimensional searching problems," *SIAM J. Comput.*, vol. 5, pp. 181–186, 1976.

[13] M. Edahiro, I. Kokubo, and T. Asano, "A new point-location algorithm and its practical efficiency — comparison with existing algorithms," *ACM Trans. Graph.*, vol. 3, no. 2, pp. 89–109, 1984.

[14] H. Edelsbrunner, L. J. Guibas, and J. Stolfi, "Optimal point location in a monotone subdivision," *SIAM J. Comput.*, vol. 15, no. 2, pp. 317–340, 1986.

[15] M. T. Goodrich, M. Orletsky, and K. Ramaiyer, "Methods for achieving fast query times in point location data structures," in *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pp. 757–766, 1997.

[16] M. T. Goodrich and K. Ramaiyer, "Geometric data structures," in *Handbook of Computational Geometry* (J. Sack and J. Urrutia, eds.), Elsevier Science, 2000.

[17] T. C. Hu and A. Tucker, "Optimum computer search trees," *SIAM J. of Applied Math.*, vol. 21, pp. 514–532, 1971.

[18] J. Iacono, "Optimal planar point location," in *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, pp. 340–341, 2001.

[19] D. G. Kirkpatrick, "Optimal search in planar subdivisions," *SIAM J. Comput.*, vol. 12, no. 1, pp. 28–35, 1983.

[20] D. E. Knuth, "Optimum binary search trees," *Acta Informatica*, vol. 1, pp. 14–25, 1971.

[21] D. E. Knuth, *Sorting and Searching*, vol. 3 of *The Art of Computer Programming*. Reading, MA: Addison-Wesley, second ed., 1998.

[22] D. T. Lee and F. P. Preparata, "Location of a point in a planar sudvision and its applications," *SIAM J. Comput.*, vol. 6, pp. 594–606, 1977.

[23] R. J. Lipton and R. E. Tarjan, "Application of a planar separator theorem," in *Proc. 18th Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 162–170, 1977.

[24] K. Mehlhorn, "Best possible bounds on the weighted path length of optimum binary search trees," *SIAM J. Comput.*, vol. 6, pp. 235–239, 1977.

[25] E. P. Mücke, I. Saias, and B. Zhu, "Fast randomized point location without preprocessing in two- and three-dimensional delaunay triangulations," *Comp. Geom. Theory & Applications*, vol. 12, no. 1–2, pp. 63–83, 1999.

[26] K. Mulmuley, "A fast planar partition algorithm, I," *J. Symbolic Comput.*, vol. 10, no. 3–4, pp. 253–280, 1990.

[27] K. Mulmuley, "Randomized multidimensional search trees: Dynamic sampling," in *Proc. 7th ACM Sympos. on Comp. Geom.*, pp. 121–131, 1991.

[28] K. Mulmuley, *Computational Geometry: An Introduction through Randomized Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, 1994.

[29] S. V. Nagaraj, "Optimal binary search trees," *Theoretical Computer Science*, vol. 188, no. 1–2, pp. 1–44, 1997.

[30] F. P. Preparata, "A new approach to planar point location," *SIAM J. Comput.*, vol. 10, no. 3, pp. 473–482, 1981.

[31] F. P. Preparata, "Planar point location revisited," *International Journal of Foundations of Computer Science*, vol. 1, no. 1, pp. 71–86, 1990.

[32] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. Springer-Verlag, 3rd ed., 1990.

[33] H. Samet, *The Design and Analysis of Spatial Data Structures*. Reading: Addison-Wesley, 1990.

[34] N. Sarnak and R. E. Tarjan, "Planar point location using persistent search trees," *Commun. ACM*, vol. 29, no. 7, pp. 669–679, 1986.

[35] R. Seidel, "A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons," *Comput. Geom. Theory Appl.*, vol. 1, no. 1, pp. 51–64, 1991.

[36] R. Seidel and U. Adamy, "On the exact worst case query complexity of planar point location," *J. Algorithms*, vol. 37, pp. 189–217, 2000.

[37] R. Seidel and C. R. Aragon, "Randomized search trees," *Algorithmica*, vol. 16, no. 4/5, pp. 464–497, 1996.

[38] C. E. Shannon, "A mathematical theory of communication," *Bell Sys. Tech. Journal*, vol. 27, pp. 379–423, 623–656, 1948.

[39] D. D. Sleator and R. E. Tarjan, "Self-adjusting binary search trees," *Journal of the ACM*, vol. 32, no. 3, pp. 652–686, 1985.

[40] J. Snoeyink, "Point location," in *Handbook of Discrete and Computational Geometry* (J. E. Goodman and J. O'Rourke, eds.), CRC Press, 1997.

[41] C. D. Tóth, "A note on binary space partitions," in *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pp. 151–156, 2001.

[42] C. D. Tóth, "Binary space partitions for line segments with a limited number of directions," in *Proc. 13th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pp. 465–471, 2002.