Nearly Optimal Expected-Case Planar Point Location

Sunil Arya* Theocharis Malamatos* David M. Mount†

Abstract

We consider the planar point location problem from the perspective of expected search time. We are given a planar polygonal subdivision S and for each polygon of the subdivision the probability that a query point lies within this polygon. The goal is to compute a search structure to determine which cell of the subdivision contains a given query point, so as to minimize the expected search time. This is a generalization of the classical problem of computing an optimal binary search tree for one-dimensional keys. In the one-dimensional case it has long been known that the entropy H of the distribution is the dominant term in the lower bound on the expected-case search time, and further there exist search trees achieving expected search times of at most $H + 2$. Prior to this work, there has been no known structure for planar point location with an expected search time better than $2H$, and this result required strong assumptions on the nature of the query point distribution. Here we present a data structure whose expected search time is nearly equal to the entropy lower bound, namely $H + o(H)$. The result holds for any polygonal subdivision in which the number of sides of each of the polygonal cells is bounded, and there are no assumptions on the query distribution within each cell. We extend these results to subdivisions with convex cells, assuming a uniform query distribution within each cell.

1. Introduction

The planar point location problem is one of the most fundamental query problems in computational geometry. The problem is to preprocess a polygonal subdivision S into a data structure so that given any query point q, the polygon of the subdivision containing q can be reported quickly. Dobkin and Lipton [7] showed that a query time of $O(\log n)$ is achievable with $O(n^2)$ space. Preparata showed how the space could be reduced to $O(n \log n)$ [16], and Kirkpatrick [11] presented a simple approach based on hierarchical triangulations, which reduced the space to $O(n)$. This was followed by a number of other methods with better performance in terms of constant factors and simplicity. These include the methods by Edelsbrunner, Guibas, and Stolfi [8], Cole [5], and Sarnak and Tarjan [17], and randomized methods by Mulmuley [14] and Seidel [18]. Recently, the question of the exact constant factor in query time was raised in work by Goodrich, Orletsky and Ramaiyer [9]. This question was answered definitively by Adamy and Seidel [1], who showed that point location queries can be answered in $\log_2 n + 2\sqrt{\log_2 n} + o(\sqrt{\log n})$ time and provided a similar lower bound.

All of this work was done in terms of worst-case query times. In this paper we consider the expected-case performance for planar point location. We assume that for each polygon $z \in S$ we are given the probability p_z that a query point lies in z. As is common in computational geometry, we will assume that the probability that the query point lies on an edge or vertex of the subdivision is zero. The problem is to produce a point location data structure whose expected search time is as low as possible. At this point we make no assumptions about the distribution of query points within each polygon, but we will introduce this later with some of our results.

This formulation 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 [10, 12]. If we think of the keys as being real numbers then they naturally define a subdivision S of the one-dimensional line into intervals. Following the geometric convention above, let us 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

*Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. Email: {arya,tmalamat}@cs.ust.hk. Research supported in part by RGC Grant HKUST 6229/00E.

†Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland. Email: mount@cs.umd.edu. Research supported in part by the National Science Foundation under grant CCR–9712379.

Subdivision Type	Query distribution within each cell	Space	Expected Query Time
Axis-parallel rectangles	Arbitrary	$O(n)$	
Triangles (bounded complexity)	Arbitrary	$O(n \log n)$	$H + O(H^{2/3}) + O(1)$
Convex polygons	Uniform	$O(n \log n)$	
Triangles (bounded complexity)	Arbitrary	$O(n^{1+\epsilon})$	$H + 2\sqrt{2H} + \log(\sqrt{H}+1) + O(1)$
Convex polygons	Uniform		

Table 1. Summary of results.

each such interval z let p_z denote the probability that a query point falls within this interval. The *entropy* of the S, denoted H throughout, is defined

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

(Throughout the paper all logarithms are taken base 2.) A classical result due to Shannon implies that the expected number of comparisons needed to answer such queries is at least as large as the entropy of the probability distribution [12, 19]. It has also been known for many years [13] that it is possible to construct a binary search tree whose expected search time is at most $H + 2$.

We consider whether we can generalize these one-dimensional results to the plane. In spite of the wealth of worst-case results on the planar point location problem it is quite remarkable that the expected-case complexity of the problem has received so little attention. Recently, Arya et al. [2] showed that for subdivisions consisting of convex polygons, assuming that the x and y coordinates of the query point are chosen *independently* from some probability distribution, the entropy bound can be achieved to within a constant multiplicative factor (2 using quadratic space and about 4 using linear space).

In this paper, we improve upon their results significantly. Let n denote the number of vertices in a given polygonal subdivision S, and let H denote its entropy. The polygonal faces of S are called *cells*. To have any chance of solving the problem we will need to make some limiting assumption 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. (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. 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. We show that we can extend our results to convex polygonal cells with an arbitrary number of sides, but to do so we need to add the assumption that the query distribution is uniform within each cell.

Our main result is that it is possible to design a point location search structure such that the expected query time is nearly optimal, growing[1] as $H + o(H)$. 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 $O(H^{2/3})$. Our results depend on the nature of the subdivision and assumptions on the query distribution. They are summarized in Table 1. The case of cells of bounded complexity is described in terms of triangles, but the generalization to cells of bounded complexity is straightforward. Throughout this paper we assume that the probability of the query point lying in the unbounded face of the subdivision is zero. It is well known that $H \leq \log n + O(1)$, and thus our best results on expected-case query time match the worst-case query time of Adamy and Seidel [1] in the dominant term, and are larger only by a factor of $\sqrt{2}$ in the second largest term.

The space used by our data structures is $O(n)$ for axis-parallel rectangles, $O(n \log n)$ for triangles (cells of bounded complexity) 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 1.)

The idea of using the entropy of the query distribution as the basis for an analysis for geometric data structures is a recent development in computational geometry. Arya and Fu [3] first applied this approach to analyzing the complexity of approximate nearest neighbor queries. A technique given in this paper was used in obtaining the expected-case planar point location results mentioned earlier [2].

Throughout the paper we assume that queries are

[1] Entropy generally increases with n, so it is reasonable to treat H as an asymptotic quantity.

answered through the use of the following standard test, and measure the expected query time in terms of the number of such tests needed. Our model considers a unit test to be a binary determination of the position of the query point in relation to a line (either above/below or left/right). Points lying on the line are classified uniformly to one side or the other. We call the test a *point-to-line comparison* or sometimes simply a comparison. For the case of axis-aligned rectangles and triangle cells, these lines are either vertical and pass through some vertex of the subdivision, or they lie along an edge of the subdivision. This is the same as the model introduced by Adamy and Seidel [1] in their lower bound, and is used in all the common point location algorithms. In the case of convex cells we generalize this slightly to allow lines that pass through any two vertices of the subdivision.

2. Main Ideas

Our point location data structure is 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.

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. The use of point-to-line comparisons naturally defines a binary space partition (BSP) tree [4], 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 must this tree possess so that point location queries can be answered efficiently?

To answer this question, observe that the expected query time is given by the weighted external path length [12], 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 [12], this suggests that the BSP tree should possess the following two properties:

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.

2.1. Property 1

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

$$\sum_{z \in S} p_z \log(f_z/p_z) \;=\; H + \sum_{z \in S} p_z \log(f_z). \qquad (1)$$

Observe that the bound in Eq. (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, such a tree construction is known only for certain special classes of subdivisions (e.g., axis-parallel rectangles). But when S may contain arbitrary oriented segments, it is not known whether such a tree can be constructed. (Note that the existence of such a tree would imply the existence of a BSP tree of linear size for arbitrary oriented segments, which is an outstanding open problem.)

A key insight of this paper 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. (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 [16].

2.2. Property 2

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 paper we present two different ways of achieving this property. The first approach is given in Section 3 and is based on a modification of the methods given by Preparata [16] and Adamy and Seidel [1]. The second approach is given in Section 4 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.

3. High Space Solution

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 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 $\epsilon > 0$. The preprocessing time is the same as the space. We defer discussion of the construction time to the full version.

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. (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 Adamy and Seidel [1], we build a weighted search tree [13] for each slab, where the weights of the trapezoids are assigned as follows. 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). 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 $region(x)$ 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 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 2 *The total number of nodes in the BSP tree T is at most $O(n2^t(\log(n)/t+1))$.*

Proof By Lemma 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 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 *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 bound on p'_x from the previous paragraph, this is at most $6p'_x/(2^t h)$.

□

Lemma 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*

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

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 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 [13], 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, $W \leq (p'_{x_i}/2^t)(1+6/h)$. 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

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

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

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

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

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

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. (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 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 4, this sum is at most

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

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 5. This yields an expected query time of at most $H + 2\sqrt{2H} + \log(\sqrt{H}+1) + O(1)$. Using Lemma 2 and noting that t is at most $O(\sqrt{\log n})$, we obtain a bound on the space of $O(n 2^{\sqrt{2H}} \log n/(\sqrt{H}+1))$. This completes the proof of Theorem 1.

Remark: 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 4 that the worst-case query time is $(1 + O(1/c)) \log n$. Simultaneously, the bound on expected performance given by Theorem 1 also holds.

4. Low Space Solutions

In this section we prove the following theorem. We defer discussion of the preprocessing time to the full version.

Theorem 2 *Let S be an n-vertex planar subdivision. Assume that the query distribution within each cell is unknown.*

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

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

Note that the space used in these solutions is better than the space used in Theorem 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 axis-parallel rectangles, 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.

4.1. Transforming the BSP Tree

The following lemma is crucial to the proof of Theorem 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 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*

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

point-to-line comparisons. The space for the search structure is $O(N)$.

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* 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 $c(d_x - 1)$, where d_x denotes the depth of x. By property (b), the depth of leaf x is at most $\log_{3/2}(1/w_x) + 2$. Thus the number of comparisons needed is at most $c(\log_{3/2}(1/w_x) + 1)$.

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', representing 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 Adamy and Seidel [1] 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 $\alpha + O(\sqrt{\alpha})$,

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 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 (denote it 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 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 6.

4.2. Axis-parallel Rectangles

Theorem 2(i) is based on the fact that we can construct an $O(n)$ size BSP tree, when S consists of axis-parallel rectangles. This important result was proved by Paterson and Yao; Amore and Franciosa showed how to improve the constant factor in the size of the tree.

Theorem 3 (Paterson and Yao [15], Amore and Franciosa [6]) *Let S be an n-vertex planar subdivision consisting of axis-parallel rectangles. Then in $O(n \log n)$ time we can construct a simple BSP tree that partitions each rectangle in S into at most six fragments.*

We construct the desired search structure in two steps. In the first step we construct the BSP tree T described in Theorem 3. We assign a weight to each leaf of T as follows. For a rectangle $z \in S$, define $\hat{p}_z = \max(p_z, 1/m)$, where m denotes the number of rectangles in S. If a rectangle 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 6. We answer point location queries by descending T'' to find the leaf containing the query point. We now analyze the query time.

Lemma 8 *Let q be a query point contained in a rectangle $z \in S$. Then using T'' the number of point-to-line 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, rectangle z generates at most six leaves, and so the weight w_x assigned to each of the leaves is at least $\hat{p}_z/12$. The lemma now follows by applying Lemma 6. □

Lemma 9 *The expected query time using the tree T'' is at most $[1 + O(1/\sqrt{\alpha})]H + O(\alpha)$.*

Proof For a rectangle $z \in S$, let $\mathcal{L}_z$ denote the set of leaves in T generated from z. Using Lemma 8, 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 $\alpha = \lceil H^{2/3} \rceil$ in Lemma 9. 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 2(i).

4.3. Triangles

First we build a BSP tree T for S as in Section 3. The construction is carried out with the parameter t set to one. There is only one significant 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 6.

Note that T is a simple BSP tree since the region associated with each node is a trapezoid. By Lemma 2, the number of nodes in T is $O(n \log n)$. Setting t to one in Lemma 1 implies the following lemma.

Lemma 10 *The number of leaves in the BSP tree T generated by triangle $z \in S$ is at most $O(\log(1/\hat{p}_z)+1)$.*

We now analyze the time for answering queries using T''. The proof is similar to that given for axis-parallel rectangles in Section 4.2.

Lemma 11 *Let q be a query point contained in a triangle $z \in S$. Then using T'' the number of point-to-line 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 10, 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 6. □

Lemma 12 *The expected query time using the tree T'' is at most $[1 + O(1/\sqrt{\alpha})] \cdot [H + \log(H + 1)] + O(\alpha)$.*

Proof For a triangle $z \in S$, let $\mathcal{L}_z$ denote the set of leaves in T generated from z. Using Lemma 11, it follows that the expected query time is at most

$$\begin{aligned} \sum_{z \in S} \sum_{x \in \mathcal{L}_z} p_x \Bigg(& \left[1 + O\left(\frac{1}{\sqrt{\alpha}} \right) \right] \\ & \left[\log \frac{1}{\hat{p}_z} + \log \left(\log \frac{1}{\hat{p}_z} + 1 \right) \right] + O(\alpha) \Bigg) \\ \leq \quad & \left[1 + O\left(\frac{1}{\sqrt{\alpha}} \right) \right] \sum_{z \in S} \Bigg[p_z \log \frac{1}{p_z} \\ & + p_z \log \left(\log \frac{1}{p_z} + 1 \right) \Bigg] + O(\alpha). \end{aligned} \tag{3}$$

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

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

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

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

The lemma now follows by substituting this in Eq. (3) and simplifying. □

For the best bound, we set $\alpha = \lceil H^{2/3} \rceil$ in Lemma 12. 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 2(ii).

Remark: Setting $m = \lceil H^{2/3} \rceil + c$, where c is a positive integer, it is easy to see from Lemma 11 that the worst-case query time is $(1 + O(1/\sqrt{c})) \log n$. Simultaneously, the bound on expected performance given by Theorem 2(ii) also holds. A similar remark can also be made for S consisting of axis-parallel rectangles.

Remark: We observe that the results given in Theorems 1 and 2(ii) 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 and 4.3. 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.

5. Convex Polygons with Uniform Interior Distribution

The main result of this section is the following.

Lemma 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 1 and 2(ii) also apply to any subdivision S consisting of convex polygons, assuming that the query distribution is uniform within each polygon.

The proof of Lemma 13 relies on the following observation.

Lemma 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 $\overline{v_1 v_2}$. 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 $\overline{v_1 v_2}$. Let v_4 denote the vertex farthest from $\overline{v_1 v_2}$ among vertices that are below it. Let R denote the rectangle defined by the two lines parallel to $\overline{v_1 v_2}$, passing through v_3 and v_4, respectively, and by the two lines perpendicular to $\overline{v_1 v_2}$, and passing through v_1 and v_2, respectively. It is an easy geometric exercise to show that P is completely contained with 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 13)

We triangulate each convex polygon in S as follows. Let z denote any convex polygon. By Lemma 14, we can find a triangle in z whose area if 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

$$\mathit{entropy}(\mathcal{T}_z) \leq p_z \log \frac{1}{p_z} + 8p_z, \tag{4}$$

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 basis case, z has 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. 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

$$\begin{aligned} & p_y \log \frac{1}{p_y} + \sum_{i=1}^{3} \mathit{entropy}(\mathcal{T}_{z_i}) \\ \leq \;& p_y \log \frac{1}{p_y} + \sum_{i=1}^{3} \left(p_{z_i} \log \frac{1}{p_{z_i}} + 8p_{z_i} \right) \\ = \;& \left(p_y \log \frac{1}{p_y} + \sum_{i=1}^{3} p_{z_i} \log \frac{1}{p_{z_i}} \right) + 8 \sum_{i=1}^{3} p_{z_i}. \end{aligned} \tag{5}$$

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. (5) we obtain

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

which completes the proof by induction.

Summing both sides of Eq. (4) over all the polygons in S, it follows that the entropy of triangulation exceeds the entropy of S by at most 8.

□

6. Acknowledgements

We would like to thank Siu-Wing Cheng and Ramesh Hariharan for helpful discussions.

References

[1] U. Adamy and R. Seidel. Planar point location close to the information-theoretic lower bound. In *Proc. 9th ACM-SIAM Sympos. Discrete Algorithms*, 1998.

[2] 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*, volume 1851 of *Lecture Notes Comput. Sci.*, pages 353–366. Springer-Verlag, 2000.

[3] S. Arya and H. Y. Fu. Expected-case complexity of approximate nearest neighbor searching. In *Proc. 11th ACM-SIAM Sympos. Discrete Algorithms*, pages 379–388, 2000. Extended version appears as HKUST Technical Report HKUST-TCSC-2000-03, URL: `http://www.cs.ust.hk/tcsc/RR`.

[4] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[5] R. Cole. Searching and storing similar lists. *J. Algorithms*, 7:202–220, 1986.

[6] F. d'Amore and P. G. Franciosa. On the optimal binary plane partition for sets of isothetic rectangles. *Inform. Process. Lett.*, 44:255–259, 1992.

[7] D. P. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM J. Comput.*, 5:181–186, 1976.

[8] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986.

[9] 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*, pages 757–766, 1997.

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

[11] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[12] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, second edition, 1998.

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

[14] K. Mulmuley. A fast planar partition algorithm, I. *J. Symbolic Comput.*, 10(3–4):253–280, 1990.

[15] M. S. Paterson and F. F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Comput. Geom.*, 5:485–503, 1990.

[16] F. P. Preparata. A new approach to planar point location. *SIAM J. Comput.*, 10(3):473–482, 1981.

[17] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, July 1986.

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

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