HKUST Theoretical Computer Science Center Research Report HKUST-TCSC-2004-09

Optimal Expected-Case Planar Point Location*

Sunil Arya† Theocharis Malamatos‡ David M. Mount§ Ka Chun Wong†

Abstract

Point location is the problem of preprocessing a planar polygonal subdivision S of size n into a data structure in order to determine efficiently the cell of the subdivision that contains a given query point. We consider this problem from the perspective of expected query time. We are given the probabilities p_z that the query point lies within each cell $z \in S$. The entropy H of the resulting discrete probability distribution is the dominant term in the lower bound on the expected-case query time. We show that it is possible to achieve query time $H + O(\sqrt{H} + 1)$ with space $O(n)$, which is optimal up to lower order terms in the query time. We extend this result to subdivisions with convex cells, assuming a uniform query distribution within each cell. In order to achieve space efficiency, we introduce the concept of entropy-preserving cuttings.

*Preliminary results appeared in the papers "Nearly Optimal Expected-Case Planar Point Location" in *Proc. 41st Sympos. Foundations of Computer Science*, 2000, 208–218 and "Entropy Preserving Cuttings and Space Efficient Planar Point Location" in *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms, 2001, 256–261.* The work of Arya, Malamatos, and Wong was supported in part by the Research Grants Council, Hong Kong, China (HKUST6229/00E and HKUST6080/01E) and the work of Mount was supported by the National Science Foundation (CCR–0098151).

†Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. Email: {arya,odan}@cs.ust.hk.

‡Max Planck Institut für Informatik, Saarbrücken, Germany. Part of this research was conducted while the author was at the Hong Kong University of Science and Technology. Email: tmalamat@mpi-sb.mpg.de.

§Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland. Email: mount@cs.umd.edu.

1 Introduction

A planar straight-line graph defines a subdivision of the plane into (possibly unbounded) polygonal regions called *cells*. Planar point location is the problem of preprocessing such a polygonal subdivision S so that, given any query point q, the cell containing q can be computed efficiently. Throughout, we let n denote the total *size* of S, defined to be the total number of vertices, edges, and faces of S.

The point-location problem has a considerable history. The first asymptotically worst-case optimal result in the area was Kirkpatrick's elegant method based on hierarchical triangulations [17], which supported query processing in $O(\log n)$ time using $O(n)$ space. This was followed by a number of other optimal methods with better practical performance including the layered-DAG of Edelsbrunner, Guibas, and Stolfi [12], searching in similar lists by Cole [8], the method based on persistent search trees by Sarnak and Tarjan [24], and the randomized incremental algorithms of Mulmuley [21] and Seidel [25]. The important question of determining the exact constant factor in query time was raised in work by Goodrich, Orletsky, and Ramaiyer [14] and was solved subsequently by Seidel and Adamy [26], who showed that point-location queries can be answered in $\log_2 n + 2\sqrt{\log_2 n} + o(\sqrt{\log n})$ time and $O(n)$ space. They also provided a nearly matching lower bound.

All of this work was done in terms of worst-case query times. In many applications, point-location queries tend to be clustered in regions of greater interest. This raises the question of whether it is possible to use knowledge of the query distribution to achieve better query times in the expected case. We model this by assuming that, for each cell $z \in S$, we are given the probability p_z that a query point lies in z. We call the result a *weighted subdivision*. Unless otherwise stated we make no assumptions about the probability distribution within each cell. To avoid dealing with many special cases, we assume that the probability that the query point lies on an edge or vertex of the subdivision is zero, but this restriction can be overcome, for example, by treating edges and vertices of nonzero probability as cells that have infinitesimal width or extent.

An important concept in characterizing the complexity of the search is the *entropy* of S, denoted throughout as H:

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

(Unless otherwise stated all logarithms are taken in base 2.) It is well known [10] that entropy is maximized when all of the cells have equal probability, in which case $H = \log n_2$, where n_2 denotes the number of cells (faces of dimension 2) of S. Conversely, entropy decreases as the disparity among the probabilities increases.

For the 1-dimensional restriction of this problem, a classical result due to Shannon implies that the expected number of binary comparisons needed to answer such queries is at least as large as the entropy of the probability distribution [18, 27], and clearly this lower bound applies to the 2-dimensional case as well. Mehlhorn [20] showed that it is possible to build a binary search tree whose expected search time is at most $H + 2$. The related problem of computing the binary search tree that minimizes the expected search time is considered in [15, 18].

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 [2] first applied this approach to analyzing the complexity of approximate nearest neighbor queries. Arya, Cheng, Mount, and Ramesh [1] then applied this to point location in subdivisions having convex cells. They assumed that the x and y coordinates of the query point were chosen independently from some probability distribution. They showed that $O(H)$ expected query time was achievable, where the multiplicative constant factor was a function of the amount of space used. Arya, Malamatos, and Mount [5] presented a simple and practical randomized algorithm that answers queries in $O(H)$ expected time with $O(n)$ space, and Iacono [16] presented a similar deterministic method achieving the same bounds.

In the spirit of prior work on optimal search structures [14, 20, 26], a fundamental question is whether it is possible to achieve the expected query time of H including only additive lower order terms. In our earlier work on this problem [3, 4] we presented data structures achieving average query times of the form $H + o(H)$, but the space was linear only in special cases, for example, if the cells are axis-parallel rectangles. Otherwise, the space requirements were superlinear, ranging from $O(n \log^* n)$ up to $O(n^{1+\epsilon})$ depending on the nature of the subdivision and assumptions about the probability distribution.

All of the existing solutions fall short of the goal of producing a point location structure of linear space whose expected query time matches the information-theoretic lower bound of H (up to lower order terms) and which makes no restrictions on the probability distribution within each cell. In this paper we present such a solution. Here is our main result.

Theorem 1 *Consider a polygonal subdivision S of size n consisting of cells of constant combinatorial complexity and a query distribution presented as a weight assignment to the cells of S. In time $O(n \log n)$ it is possible to construct a search structure of space $O(n)$ that answers point location queries in expected time $H + O(\sqrt{H} + 1)$, where $H = entropy(S)$.*

Due to our reliance on geometric cuttings of line segments in the plane [7, 14], our construction is randomized, and so the $O(n \log n)$ construction time holds in expectation. Otherwise, our construction runs in $O(n \log n)$ time and is deterministic.

The requirement that cells have constant complexity only applies to cells of nonzero probability, since cells of zero probability can be triangulated without affecting the entropy of the subdivision. (This applies to the unbounded external cell of the subdivision as well.) If no assumptions are made about the query distribution, then the assumption that cells have constant cell complexity seems to be critical. In the next section, for example, we show that if the query distribution is arbitrary, then even determining whether a query point lies within a single n-sided convex polygon requires expected time $\Omega(\log n)$, irrespective of entropy. Nonetheless, we show that if we impose the restriction that the cells are convex polygons and the query distribution within each cell is uniform, then it is possible to answer point location queries as efficiently even if the cells have an arbitrary number of sides. A subdivision is said to be *convex* if all its faces are convex.

Theorem 2 *The complexity bounds of Theorem 1 apply as well if S is a weighted convex planar polygonal subdivision such that the query distribution within each cell is uniform.*

Before discussing our methods, we need to say a bit about our model of computation and the definition of expected query time. To establish Theorem 1 we assume that the answer to a query is based on the results of certain binary tests called *primitive comparisons*. They were introduced by Seidel and Adamy [26] as the *trapezoidal search graph model*, and they are special cases of *point-line comparisons*, which determine the relative position of a point and a line. There are two types of primitive comparisons. One determines whether the query point q lies to the left or right of a vertical line passing through a vertex of the subdivision. (See Fig. 1(a).) The other determines whether the query point lies above or below an edge of the subdivision. This latter comparison is only performed after we have already determined that the x-coordinates of the point lie between the x-coordinates of the endpoints of the edge. (See Fig. 1(b).) We make the usual general position assumption that no two vertices have the same x-coordinate and so no edge is vertical. This assumption can be overcome, for example, by standard perturbation methods [13].

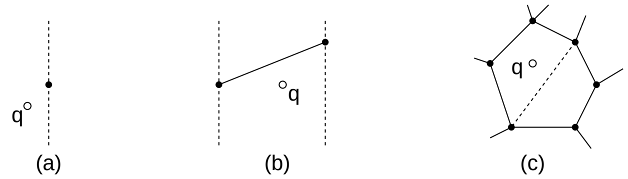

Fig. 1: Primitive comparisons ((a) and (b)) and extended primitive comparisons (c).

To establish Theorem 2, which involves convex subdivisions, it will be necessary to refine the subdivision through the insertion of new edges. For this purpose we define an *extended primitive comparison* to include comparisons involving line segments that are not necessarily part of the original subdivision, but that join two vertices of the original subdivision. (Consider the comparison between q and the broken line segment in Fig. 1(c).) Primitive comparisons and extended primitive comparisons form the basis of virtually all the worst-case optimal point location algorithms cited earlier. Our data structures for Theorem 1 use only primitive comparisons and for Theorem 2 use extended primitive comparisons.

Any point location algorithm that is based on binary comparisons can be modeled abstractly as a decision-tree structure called a *binary space partition* (BSP) [11]. (The search structure that we present is not a proper binary tree, since it allows sharing of substructures, but this only affects the space requirements, and not the query time.) For our purposes, a BSP is a rooted binary tree in which each internal node is associated with a line. This line subdivides the plane into two halfplanes, one open and one closed, which are then associated with the node's two children. (Since we assume that query points do not fall on edges, the question of which is open and which is closed is not significant.) Each node of a BSP is implicitly associated with a (possibly unbounded) convex polygon, called its *region*, which is the intersection of the halfplanes corresponding to the path from the root to this node.

Given a BSP, point location queries are answered by performing a simple descent in the tree. At each internal node we visit the child corresponding to the halfplane that contains the query point until arriving at a leaf. It is easy to see that the query point lies within the regions associated with each of the nodes along the search path. It follows that the BSP correctly solves the point location problem if and only if the region associated with every leaf of the tree lies entirely within a single cell of the subdivision. (If the region were to overlap two or more cells, we could not unambiguously determine which of these cells contains the query point.) When the search arrives at a leaf, the associated cell is returned as the answer. Given a query distribution, each leaf of the BSP is associated with the probability that the query point lies within this leaf's region. The *weighted external path length* of a BSP is the weighted average of the depths of all its leaves, where the weight is this probability [18]. We define the *expected query time* of a BSP to be this weighted external path length. An example is shown in Fig. 2, where (a) shows the original subdivision, (b) shows the binary space partition induced by three lines L_1, L_2, and L_3 and the associated probabilities with each region, and (c) shows the associated tree. In this case the weighted external path length is $1 \cdot p_1 + 2 \cdot p_2 + 3(p_3 + p_4)$.

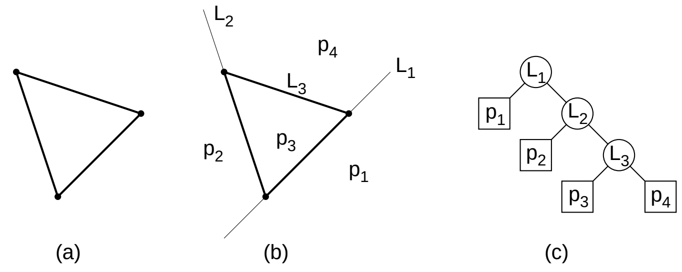

Fig. 2: A binary space partition (BSP) and the associated search tree.

The rest of the paper is organized as follows. The next section presents mathematical preliminaries and provides an overview of our approach. In Section 3 we present an algorithm for answering point-location queries that is optimal in expected time (up to lower order terms) but suboptimal in space. It answers queries in expected time $H + O(\sqrt{H} + 1)$ and space $O(n^{1+\epsilon})$, for any $\epsilon > 0$. In Section 4 we show how to reduce this to $O(n)$ space. We first introduce the notion of entropy-preserving cuttings, and in Sections 4.1 and 4.2 we show how to apply this concept to complete the space bound of Theorem 1. Finally, in Section 5, we show that these methods can be generalized to convex subdivisions with uniform query distributions within each cell in order to prove Theorem 2.

2 Preliminaries

Consider a weighted subdivision S. Viewing S as a planar graph, let n denote the total numbers of its vertices, edges, and faces (cells), respectively. The (unbounded) external face is also considered a cell. A well known consequence of Euler's formula is that if this graph is connected, then n is asymptotically bounded by the number of edges of S (see, e.g., [11]). Since the number of edges is clearly a lower bound on the space complexity of any point location structure, any $O(n)$ space structure is asymptotically optimal with respect to space.

2.1 On Subdivisions of Unbounded Cell Complexity

Throughout much of the paper we will concentrate on the case where the cells of S are bounded by a constant number of sides. We will show here why this assumption seems to be critical in the context of achieving query time bounds based on entropy. Note that this assumption is not required for worst-case optimal planar point location, since it is possible to refine any planar polygonal subdivision into one whose cells have a constant number of sides while increasing the size of the subdivision by just a constant factor. However, we show that if cells have unbounded complexity, even if they are convex, there exist query distributions such that any search structure based on point-line comparisons performs arbitrarily worse than the entropy bound in the expected case.

Lemma 1 *Given any convex polygon Z with n sides there exists a discrete query probability distribution such that the probability that a query point lies within Z is $1/2$, and the expected number of point-line comparisons needed to determine whether a point lies within Z is $\Omega(\log n)$.*

Proof: The probability distribution is defined as follows. Let the vertices of Z be $\{v_1, \ldots, v_n\}$. For each vertex v_i we consider two points a_i and b_i placed very close to v_i. The point a_i lies just inside of Z, and b_i lies just outside of Z. The points a_i and b_i all carry a query probability of $1/(2n)$. (See Fig. 3.) Observe that the probabilities sum to 1.

Now let T be any BSP that correctly determines membership in Z. Clearly for T to be correct, for each vertex v_i there must be some node of T whose associated line stabs the line segment $\overline{a_i b_i}$, since otherwise both a_i and b_i would lie in the same leaf region, implying that we cannot distinguish between inside and outside in this case. Because Z is convex, we can place the points a_i and b_i sufficiently close to each vertex so that any line can stab at most two such segments. Now consider any node of T. In order to minimize the expected search time, the best that we can hope to accomplish is that the remaining probability in the region is evenly split between the left and right children, implying that, other than the at most two vertices whose segments were stabbed, half of the remaining vertices lie on one side of the line and half on the other. It follows easily that $\Omega(\log n)$ such comparisons are needed along any search path of the optimum BSP. □

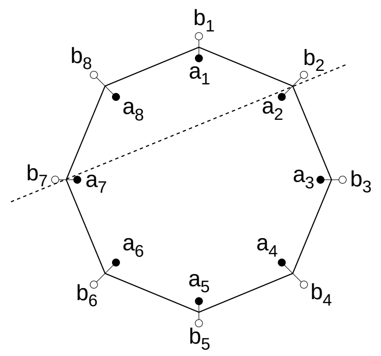

Fig. 3: The proof of Lemma 1 and a stabbing line.

A convex polygon defines a trivial subdivision consisting of two cells (inside and outside). Irrespective of n, the entropy of the subdivision described in the previous lemma is easily seen to be 1. Since the query time grows as $\Omega(\log n)$, it is not possible to bound the expected query time purely as a function of entropy. Thus we have:

Theorem 3 *If no restrictions are placed on cell complexity or query distribution, then no search structure based on point-line comparisons can guarantee an expected-case query time that is bounded purely by a function of entropy, even for convex subdivisions.*

2.2 Conditioning the Subdivision

If the cells of the subdivision all have constant combinatorial complexity then we claim that it is possible to condition the problem to bring it into a simpler form without adversely affecting the expected-case query time. There are two types of conditioning that will be of interest to us. First, we observe that it does not make sense to talk about query times that are smaller than 1, but the entropy H of a subdivision can be considerably smaller than 1. This can happen only if there is one cell whose probability is greater than 1/2. (For otherwise $\log(1/p_z) \geq 1$ for all cells z, from which it follows immediately that the entropy is at least 1.) There can be at most one such cell, and so to achieve this, we split the high probability cell into two fragments, each having half the weight of the original. The cost of performing a point location in the modified subdivision is larger by at most one comparison.

The second type of conditioning arises because many point-location data structures assume that the subdivision is presented in some canonical subdivision, e.g. a triangulation [17], a monotone subdivision [12], or a trapezoidal map [21, 25]. If the cells of S have constant combinatorial complexity, then any of these canonical subdivisions can be realized by refining each cell into at most a constant number of subcells.

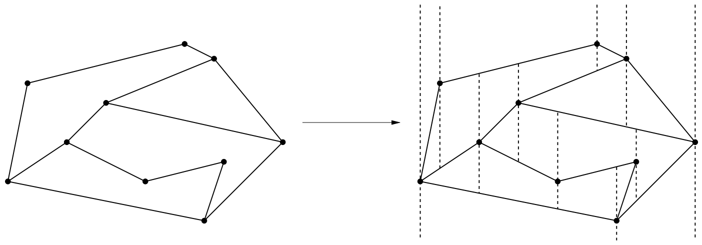

Fig. 4: A subdivision (left) and its trapezoidal map (right).

The most convenient canonical structure for our purposes is a *trapezoidal map*. This is a planar subdivision in which each cell is a trapezoid with vertical parallel sides. These are trapezoids in a general sense, and may be unbounded or degenerate to triangles. (To avoid the complexities of unbounded cells, it is common to enclose the entire subdivision in a large bounding rectangle, which contains all the query points.) Any polygonal subdivision can be converted into a trapezoidal map by adding two vertical segments between each vertex and the edges lying immediately above and below it. (See Fig. 4.) This can be done in $O(n \log n)$ time either by a straightforward modification of plane sweep [6, 11] or through a simple randomized incremental construction [21, 25].

We show that if the initial subdivision has cells of constant complexity, then it is possible to condition the input without significantly affecting the expected-case query time so that it is a trapezoidal map. (The method can be applied to produce any of the other canonical subdivision forms, but this is the one that will be most relevant to our construction.) Before giving a formal statement of the result, we first present a few definitions. Given a planar subdivision S, a *refinement* S^* is a planar subdivision such that each cell of S^* is contained within some cell of S. Given S and S^*, for each cell $z \in S$, we let $\mathcal{F}_z$ denote the subset of cells of S^* that are contained within z, called its *fragments*. Let $f_z = |\mathcal{F}_z|$ be the number of fragments. We say that S^* has *fragmentation* f, if $f_z \leq f$ for all cells z. Clearly answering point location queries for S can be reduced to answering queries in S^*.

Given a function $g : \mathbb{R}^+ \to \mathbb{R}^+$, we say that a point-location structure T is *g-efficient* for a weighted subdivision S with entropy H if T answers point location queries for S in expected time at most $H + g(H)$. For example, Theorem 1 asserts it is possible to construct an $O(\sqrt{H}+1)$-efficient point-location structure. Since g is designed to capture lower order terms of the complexity, we assume that g grows asymptotically no faster than linearly.

An important observation about the point-location construction described in Theorem 1 is that it is only given the probability p_z that a query point lies within each cell and knows nothing of the probability distribution within each cell. We say that such a construction is *distribution-oblivious*. It follows that any structure produced by such an algorithm must satisfy its expected-case query time bound for any choice of the probability distribution within each cell, provided of course that the probability that a point lies within cell z is indeed p_z. This issue arises because in the process of computing the point location structure we compute a refinement of the original subdivision. We do not know what the query distribution is within the cells of the refined subdivision, and so we cannot compute its expected query time. However, we know that entropy is maximized when all cells have the same probability. So, we invent a query distribution by splitting the probabilities evenly among the fragments of each refined cell. We then build an efficient structure for the resulting weighted subdivision using any distribution-oblivious construction. The following result shows formally that this strategy leads to an efficient solution to the original problem, irrespective of the actual query distribution.

Lemma 2 *Consider a subdivision S and a query distribution on S. Let H denote its entropy. Let S^* be any refinement of S of fragmentation $O(1)$. In $O(n)$ time it is possible to assign nonnegative weights to the cells of S^*, thus producing a weighted subdivision $\widehat{S}$ with the following property. Let T be any g-efficient point-location structure for $\widehat{S}$ produced by a distribution-oblivious construction. Then T is an $O(g+1)$-efficient point-location structure for the original subdivision S.*

Before presenting the proof, we show how to apply this result to achieve the desired conditioning. Consider any weighted subdivision S of constant cell complexity and of total size n. In $O(n \log n)$ time we convert this subdivision into a trapezoidal decomposition $\widehat{S}$ through a refinement of fragmentation $O(1)$. Clearly the resulting refinement also has size $O(n)$. Now, we apply the above result to assign weights to this trapezoidal decomposition. Assuming that Theorem 1 holds for trapezoidal decompositions, it follows that in $O(n \log n)$ time we can construct an $O(\sqrt{H}+1)$-efficient point-location structure T for $\widehat{S}$. By the above result, T can be used as an $O(\sqrt{H}+1)$-efficient point-location structure for the original subdivision S, albeit with a higher constant factor hidden by the "O"-notation.

We devote the remainder of this section to proving Lemma 2. Let S denote the original subdivision of entropy H. Let S^* be a refinement of S of fragmentation f. For each cell z of S, recall that p_z is the probability that the query point lies within z. Also recall that $\mathcal{F}_z$ denotes the set of f_z fragments into which z has been subdivided. For each fragment $y \in \mathcal{F}_z$ there is some probability p_y that the query point lies within y, but the algorithm does not know this probability. Clearly $\sum_{y \in \mathcal{F}_z} p_y = p_z$. Let $H' = \sum_{z \in S} \sum_{y \in \mathcal{F}_z} p_y \log(1/p_y)$ denote the entropy of this (unknown) distribution.

For the purposes of the proof we need to assign probabilities to the fragments. Since we know that entropy is maximized when probabilities are distributed as evenly as possible, we split the weight evenly among the fragments by setting $w_y = p_z/f_z$, for each fragment $y \in \mathcal{F}_z$. Since $p_y \le p_z$, and since S^* has fragmentation f, we have $w_y \ge p_y/f$. Clearly $\sum_{y \in \mathcal{F}_z} w_y = p_z$. Let $\widehat{S}$ denote the resulting weighted subdivision, and let

$$\widehat{H} = \sum_{z \in S} \sum_{y \in \mathcal{F}_z} w_y \log \frac{1}{w_y}$$

denote its entropy based on this weight assignment. The assignment can easily be computed in $O(n)$ time. The following lemma asserts that the entropies of all these subdivisions are related to each other, up to an additive term of $\log f$.

Lemma 3 *Given the weighted subdivisions S, S^*, and $\widehat{S}$ defined above, their respective entropies satisfy*

$$H \;\leq\; H' \;\leq\; \widehat{H} \;\leq\; H + \log f$$

Proof: To prove the first inequality we observe that if y is a fragment of z then $p_y \leq p_z$ and so

$$H \;=\; \sum_{z \in S} p_z \log \frac{1}{p_z} \;=\; \sum_{z \in S} \sum_{y \in \mathcal{F}_z} p_y \log \frac{1}{p_z} \;\leq\; \sum_{z \in S} \sum_{y \in \mathcal{F}_z} p_y \log \frac{1}{p_y} \;=\; H'.$$

The second inequality is an immediate consequence of the fact that entropy is maximized when the probabilities (weights) are equal to each other [10], which is clearly the case for the weight assignment defining $\widehat{H}$. Finally, to prove the last inequality we use the facts that $f_z = |\mathcal{F}_z|$, $\sum_z p_z = 1$, and $f_z \leq f$ to obtain:

$$\begin{aligned}
\widehat{H} \;&=\; \sum_{z \in S} \sum_{y \in \mathcal{F}_z} w_y \log \frac{1}{w_y} \;=\; \sum_{z \in S} \sum_{y \in \mathcal{F}_z} \frac{p_z}{f_z} \log \frac{f_z}{p_z} \;=\; \sum_{z \in S} p_z \left(\log \frac{1}{p_z} + \log f_z \right) \\
&\leq\; \left(\sum_{z \in S} p_z \log \frac{1}{p_z} \right) + \log f \;=\; H + \log f.
\end{aligned}$$

$\square$

Returning now to the proof of Lemma 2, recall that $\widehat{S}$ is our uniformly weighted refined subdivision. Let T be a g-efficient point-location structure for $\widehat{S}$, which results from a distribution-oblivious construction. Let $\widehat{E}_T$ denote the expected query time for T assuming the weight assignment w_y given above for each cell y of $\widehat{S}$. Technically, $\widehat{E}_T$ can only be defined relative to a particular probability distribution such that the probability that a query point lies within y is w_y. However, given that the construction is distribution-oblivious, we know that for any choice of such a probability distribution we will have $\widehat{E}_T \leq \widehat{H} + g(\widehat{H})$. Since this is the only assumption we will make about $\widehat{E}_T$ we will tolerate this abuse.

Because T can be viewed abstractly as a BSP and by the remarks made in Section 2.1 on the correctness of BSPs for point location, it follows that the region associated with each leaf cell of T lies within some cell y of $\widehat{S}$. Thus, each fragment y is further decomposed by T into subfragments. Let $\mathcal{F}_y$ denote the subfragments of y, each associated with a leaf of T. (See Fig. 5.)

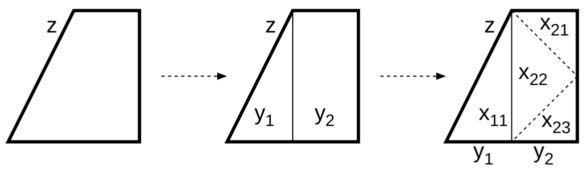

Fig. 5: Fragments ($\mathcal{F}_z = \{y_1, y_2\}$) and subfragments ($\mathcal{F}_{y_2} = \{x_{21}, x_{22}, x_{23}\}$).

For each $x \in \mathcal{F}_y$, let p_x denote the (unknown) probability that the query point lies within this subfragment, and let d_x denote its search depth in T, that is, the number of primitive comparisons needed to provide an answer for any query point in x. Since subfragments may have different search depths, we then define the (true) expected search depth for a fragment y of S^* to be

$$D_y = \frac{1}{p_y} \sum_{x \in \mathcal{F}_y} p_x d_x.$$

It follows easily that if we apply T as a point location structure for the original subdivision S, the expected search time, denoted E_T, is given by summing up the contributions from all the subfragments:

$$E_T \;=\; \sum_{z \in S} \sum_{y \in \mathcal{F}_z} \sum_{x \in \mathcal{F}_y} p_x d_x \;=\; \sum_{z \in S} \sum_{y \in \mathcal{F}_z} p_y D_y.$$

The values of the p_y's are not known to us, and so we cannot compute D_y. But because T is constructed by a distribution-oblivious algorithm, we know that the upper bound on the expected query time for $\widehat{S}$ holds irrespective of the choice of probability distribution within each of the fragments. We define such a probability distribution for $\widehat{S}$ by allocating weights among the subfragments in exactly the same proportions as their true values. Of course, this is done subject to the constraint that they sum to w_y. We also define the expected depth analogously:

$$w_x \;=\; \frac{p_x}{p_y} w_y \qquad \text{and} \qquad \widehat{D}_y \;=\; \frac{1}{w_y} \sum_{x \in \mathcal{F}_y} w_x d_x.$$

We observe that $\widehat{D}_y$ is equal to its counterpart in the true distribution:

$$\widehat{D}_y \;=\; \frac{1}{w_y} \sum_{x \in \mathcal{F}_y} w_x d_x \;=\; \frac{1}{w_y} \sum_{x \in \mathcal{F}_y} \frac{p_x}{p_y} w_y d_x \;=\; \frac{1}{p_y} \sum_{x \in \mathcal{F}_y} p_x d_x \;=\; D_y.$$

The expected search time $\widehat{E}_T$ under our constructed distribution is

$$\widehat{E}_T \;=\; \sum_{z \in S} \sum_{y \in \mathcal{F}_z} \sum_{x \in \mathcal{F}_y} w_x d_x \;=\; \sum_{z \in S} \sum_{y \in \mathcal{F}_z} w_y \widehat{D}_y.$$

By the obliviousness of T's construction and g-efficiency we have

$$\begin{aligned} g(\widehat{H}) \;&\geq\; \widehat{E}_T - \widehat{H} \;=\; \sum_{z \in S} \sum_{y \in \mathcal{F}_z} w_y \widehat{D}_y - \sum_{z \in S} \sum_{y \in \mathcal{F}_z} w_y \log \frac{1}{w_y} \\ &=\; \sum_{z \in S} \sum_{y \in \mathcal{F}_z} w_y \left(\widehat{D}_y - \log \frac{1}{w_y} \right). \end{aligned}$$

From the observation made prior to Lemma 3 that $w_y \geq p_y / f$ and the fact that the probabilities sum to 1, we have

$$\begin{aligned} g(\widehat{H}) \;&\geq\; \sum_{z \in S} \sum_{y \in \mathcal{F}_z} \frac{p_y}{f} \left(\widehat{D}_y - \log \frac{f}{p_y} \right) \;=\; \frac{1}{f} \sum_{z \in S} \sum_{y \in \mathcal{F}_z} p_y \left(\widehat{D}_y - \log \frac{1}{p_y} - \log f \right) \\ &=\; \frac{1}{f} \left(\left(\sum_{z \in S} \sum_{y \in \mathcal{F}_z} p_y \left(\widehat{D}_y - \log \frac{1}{p_y} \right) \right) - \log f \right). \end{aligned}$$

Next, we multiply both sides by f and add $\log f$, and then apply the substitution $\widehat{D}_y = D_y$ and the definitions of E_T and H' to obtain

$$f \cdot g(\widehat{H}) + \log f \;\geq\; \sum_{z \in S} \sum_{y \in \mathcal{F}_z} p_y \left(D_y - \log \frac{1}{p_y} \right) \;=\; E_T - H'.$$

Now, from Lemma 3 we know that $H' \leq \widehat{H} \leq H + \log f$. Since g grows no faster than linearly, it follows that $g(H + \log f) \leq g(H) + O(\log f)$. Now, combining these with the fact that the fragmentation f is a constant we obtain

$$\begin{aligned} E_T - H &\leq E_T - H' + \log f \;\leq\; f \cdot g(H + \log f) + 2 \log f \\ &\leq f(g(H) + O(\log f)) + 2 \log f \;=\; O(g(H) + 1). \end{aligned}$$

This implies that T is $O(g + 1)$-efficient as a point-location structure for S, and so completes the proof of Lemma 2.

2.3 Overview of Our Methods

Before presenting our algorithms we provide an overview of our methods. Our point-location data structure is based on many of the same methods used in the construction of worst-case efficient point-location structures, particularly the methods given by Preparata [22] and Seidel and Adamy [26]. Establishing efficient expected query time involves considerably different techniques from worst-case query time. As mentioned above, a point location data structure that is based on linear comparisons can be viewed abstractly as a BSP. What properties must the associated partition tree possess in order to answer point-location queries efficiently on average? Observe that since the expected query time is the tree's weighted external path length, the contribution of each leaf to the expected query time is its tree depth times its probability. 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 each cell. Our strategy will be to construct a tree in which the depth of any leaf generated from a cell $z \in S$ is close to $\log(1/p_z)$.

As we shall see this will lead to a method, which while optimal with respect to expected-case query time, is not optimal with respect to space. In particular it requires $O(n^{1+\epsilon})$ space for any $\epsilon > 0$. To reduce the storage further we employ a common strategy in computational geometry, called cuttings. Given a subdivision S with m edges, and a parameter $r \geq 1$, a $(1/r)$-*cutting* [7] is a partition of the plane into $O(r)$ trapezoids such that the interior of each trapezoid is intersected by at most m/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. Goodrich, Orletsky, and Ramaiyer [14] and later Seidel and Adamy [26] applied cuttings in a divide-and-conquer manner to produce the most space-efficient data structure.

Cuttings cannot be applied directly in our case, however, since the partitioning process may refine the subdivision in a way that significantly increases its entropy, and this increases the expected query time. An important contribution of this paper 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. This will be presented in Section 4. Our approach will be to apply entropy-preserving cuttings to build a two-level search structure. For the first level we construct an entropy-preserving cutting of an appropriately chosen size and build the initial point location structure described in Section 3 for the cutting. This achieves good expected-case query time, but leaves a number of regions to search. We show that the probability that the query point lies within any of these remaining regions is so small, that a relatively sloppy worst-case optimal point location algorithm suffices to achieve our desired results.

Extending these results to convex subdivisions with uniform query distributions involves a simple refinement step. Each convex cell is triangulated by a process that extracts a triangle whose area is a constant fraction of the total area, and then recurses on each of the three resulting fragments. We show that if the query distribution is uniform, then the increase in entropy in the resulting refined subdivision is at most an additive constant.

3 Initial Solution

In this section we present an algorithm for answering point-location queries that is optimal in expected time but suboptimal in space. Recall that from the results of Section 2 we may assume that the subdivision S is presented as a weighted trapezoidal map and its entropy H is at least 1.

Theorem 4 *Given a trapezoidal map S of size n, 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*

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

where H is the entropy of S. The space for the data structure is

$$O\left(n 2^{\sqrt{2H}} \frac{1}{\sqrt{H}} \log n\right).$$

Other than the probabilities p_z we make no assumptions about the query probability distribution within each cell. Recall that $1 \le H \le \log n$. It is easy to verify that $2^{\sqrt{2H}}/\sqrt{H}$ increases monotonically for $H \ge 1/(\sqrt{2}\ln 2)^2 \approx 1.04$, and so the space is maximized when $H = \log n$. Thus, the space is at most $O(n 2^{\sqrt{2\log n}}\sqrt{\log n})$, which is $O(n^{1+\epsilon})$ for any $\epsilon > 0$. The preprocessing time is $O(N + n\log n)$, where N is the total space of the data structure. Thus the asymptotic preprocessing time is dominated by the above space bound.

3.1 Construction of the Search Tree

As mentioned earlier, our data structure is based on constructing a BSP T for S. Before building the tree we map the cell probabilities to an assignment of *weights* to the vertices of the subdivision as follows. Recall that n is the combinatorial complexity of S. For each trapezoid $z \in S$, we assign a weight of $w_z/4$ to each of its (at most four) corner vertices, where

$$w_z \;=\; \frac{1}{2}\max\left(p_z, \frac{1}{n}\right).$$

(See Fig. 6.) If a vertex is a corner of multiple trapezoids, then its weight is the sum of the contributions from all such trapezoids. It is easy to see that the total weight of all the trapezoids is at most 1, and this holds as well for the total weight of all the vertices. The $1/n$ term in the definition of the weight will be important in limiting the fragmentation of cells of very small probability. This in turn will be needed to establish our space bounds.

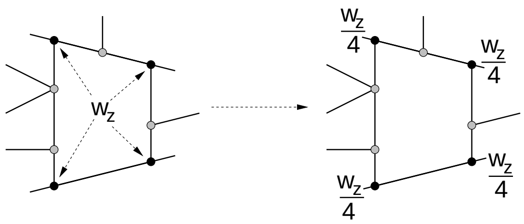

Fig. 6: Assignment of weights.

The tree T is built recursively in a top-down fashion. Each node u of T will be associated with a trapezoidal region, denoted $R(u)$. We define the *weight* of a region to be the sum of the weights of the subdivision vertices in the interior of the region, and we define the *weight* of a node u to be the weight of $R(u)$. The construction is based on a recursive process, which is broken into *stages*. Each stage replaces an existing leaf node whose associated region overlaps two or more cells of S with a new subtree. Let us consider one such stage. Suppose that we are working on the subdivision contained within a trapezoid $R(u)$ associated with some node u. (Initially u is the root of the tree and $R(u)$ is the entire space.) Let w_u denote u's weight. (For example, in upper left of Fig. 7(a) w_u is the sum of the weights of the black and gray vertices lying in the interior of the trapezoid $R(u)$.) We split $R(u)$ into two vertical *slabs*, by passing a vertical line through a subdivision segment endpoint, such that the weight of each slab is at most $w_u/2$. We repeat this for t levels, where $t \geq 1$ is a suitable parameter (to be fixed later), each time ensuring that the sum of the weights is halved. This partitioning can be represented in a natural way by a balanced tree having up to 2^t leaves, representing the 2^t vertical slabs. (There may in fact be fewer, since some slabs may contain no interior vertices before the splitting process ends.)

After this, each slab is further partitioned into trapezoids by the segments of the subdivision that completely cross it. (See Fig. 7(b).) Following Seidel and Adamy [26], we build a weighted search tree [20] for each slab. There is a technical difficulty, however. A trapezoid that contains no vertices in its interior (called an *empty trapezoid*) has a weight of 0, and so we cannot reasonably bound its depth in the tree. To handle this the weighted search tree is based on the following set of *adjusted weights*. The adjusted weight of a non-empty trapezoid is just its weight, that is, the sum of weights of its interior vertices. The adjusted weight of an empty trapezoid cell is defined to be $w_z/(h2^t)$, where z is the trapezoid in S that contains this cell, and where $h \geq 1$ is a suitable parameter (to be fixed later). (See Fig. 7(b).) After building the weighted search tree for the trapezoids of each slab, we recurse on each of the non-empty trapezoids. The process ends when there are no non-empty trapezoids.

For the purpose of analysis, it is convenient to view this partitioning scheme as a *multi-way tree* as follows. Each stage involves splitting a trapezoid u into 2^t vertical slabs, each of which is then partitioned into smaller trapezoids. 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

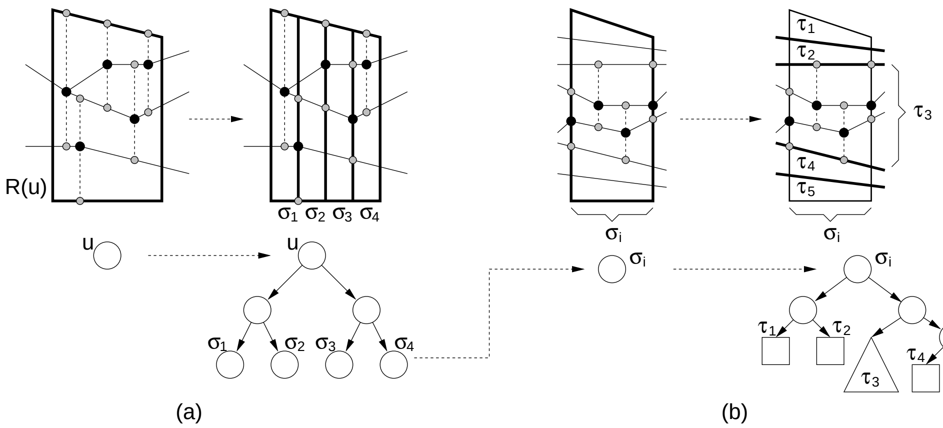

Fig. 7: Construction of the initial search structure for $t = 2$.

multi-way tree. Note that each node of T' corresponds to a unique node of T and represents the same region.

This construction can be implemented efficiently as follows. We assume that we maintain the following for each node u: (1) the intersection of the subdivision S with $R(u)$, denoted by S_u, and (2) a list of the vertices of S that lie within the interior of $R(u)$ and their associated weights, sorted according to their x-coordinates. We assume that S_u is represented in a manner that supports efficient processing and traversal, say as a *doubly connected edge list* (DCEL) [11]. We also assume that there are cross links between each vertex of the sorted list and its corresponding vertex of the DCEL. This information can be computed for the root of the tree in $O(n \log n)$ time.

Now consider the single-stage construction for a fixed internal node u of the multi-way tree. Let n_u denote the combinatorial complexity of S_u. Let m_u denote the total number of empty and non-empty trapezoids that result from one stage of the construction, or equivalently, the total number of nodes in all the weighted trees for all of u's slabs. In $O(n_u)$ time it is possible to scan the sorted vertex list from left to right, to generate the (up to) 2^t slabs with the desired weights. We then cut S_u into slabs by tracing along each of the $2^t - 1$ vertical lines that define the slab boundaries. Because each cell of S_u is of constant combinatorial complexity, by standard results on DCELs [11] we can do this in $O(1)$ time for each intersection between a segment of S_u and a cutting line. The overall time is $O(m_u)$. As we are cutting out each slab, we note which of the segments of S cut clear through the slab. From this information we can determine which of the resulting trapezoids are empty and which are non-empty. Then, for each non-empty trapezoid, we can construct the DCEL representation of the subdivisions lying within the trapezoid in time proportional to its size. This takes time $O(n_u)$ when summed over all the non-empty trapezoids. After this the subdivisions are ready for the next recursive step.

Next we consider how to update the sorted vertex list. Consider each non-empty trapezoid τ_i. We traverse the associated subdivision and, for each vertex lying in the interior of the trapezoid, we access the cross link to the sorted list and label the associated list entry with the integer i. Through the use of any stable integer sorting algorithm (e.g., counting sort [9]) we sort these labels in $O(n_u)$ time. By collecting adjacent entries with the same label, we can partition the sorted vertex lists among the various non-empty trapezoids, while maintaining the sorted order, all in $O(n_u)$ time.

Thus, the total processing time for node u is $O(n_u + m_u)$. The sum of the m_u terms over all the nodes is essentially equal to the number of nodes of T', which is bounded above by the total number of nodes of T. Because each subdivision vertex appears in the region associated with at most one node at each level of T', the sum of the n_u terms for each level is $O(n)$. In Lemma 4 below we show that the number of levels in T' is $(1/t)\log n + O(1)$. Therefore the overall construction time is $O(N + n\log n)$, where N is the total number of nodes in the tree. We will bound N later in Lemma 5.

3.2 Analysis of the Space and Query Time

We now analyze the space and expected query time as a function of the parameters t and h. For a node u of T, let p_u denote the probability of the query point lying in its associated region $R(u)$ (or equivalently, the probability of visiting u during point location). Recall that w_u denotes the weight of all the vertices in the interior of $R(u)$. The following lemma bounds the number of levels in multi-way tree T'.

Lemma 4 *The number of levels in T' is at most $(1/t)\log n + O(1)$.*

Proof: Consider any path in the multi-way tree T' descending from the root to a leaf. Along any edge that leads from one internal node to another the weight decreases by a factor of at least 2^t. Since the weight of the root is at most 1, the weight of any internal node u at level i of T' is at most $1/2^{t(i-1)}$. Since u is internal, it must contain the corner vertex of some trapezoid z in its interior. Recall that the weight of each of the corner vertices of trapezoid z is at least $w_z/4$ (more if it is a corner of multiple trapezoids). It follows that

$$\frac{1}{2^{t(i-1)}} \geq \frac{w_z}{4}.$$

Simplifying this gives

$$i \;\leq\; \frac{1}{t}\left(\log\frac{1}{w_z} + \log 4\right) + 1 \;\leq\; \frac{1}{t}\log\frac{1}{w_z} + 3.$$

Since $w_z \geq 1/(2n)$ it follows that the level of any internal node is at most $(1/t)\log n + 4$, and the number of levels is higher by 1. $\square$

Using this, we can bound the total size of the tree.

Lemma 5 *The total number of nodes of T is at most $O(n2^t((\log n)/t + 1))$.*

Proof: Consider any trapezoid z of S. We first show that the number of leaves in T generated by z is at most

$$4 \cdot 2^t\left(\frac{1}{t}\log n + O(1)\right).$$

It follows from our construction that any internal node of the multi-way tree T' that overlaps the interior of z must contain at least one of the four corner vertices of z. Thus, there are at most four such internal nodes at any level of T'. (In fact, a more careful analysis shows that there are at most two such nodes, one for the left side and one for the right side.) From Lemma 4 it follows that the total number of internal nodes of T' that overlap the interior of z is at most $4((1/t)\log n + O(1))$. Since any node of T' can have at most 2^t children that overlap the interior of z, the above bound follows.

By summing this bound over all $O(n)$ trapezoids, the total number of leaves in T is at most $O(n2^t((\log n)/t + 1))$. Since T is binary, the number of internal nodes cannot be larger. $\square$

In Lemma 7, we bound the depth of a leaf generated from a trapezoid $z \in S$. To this end, we need the following technical result.

Lemma 6 *Let u be an internal node in the multi-way tree T'. Let σ be any of the 2^t vertical slabs into which $R(u)$ is partitioned. Then the total adjusted weight of the weighted search tree corresponding to σ is at most $(w_u/2^t)(1 + 4/h)$.*

Proof: Recall that the segments in S partition σ into empty and non-empty trapezoids. Since the weight associated with σ is at most $w_u/2^t$, the total adjusted weight of all the non-empty trapezoids is at most $w_u/2^t$. We will show that the total adjusted weight of the empty trapezoids is at most $4w_u/(2^t h)$, which will complete the proof.

Let $\mathcal{T}_u$ denote the set of trapezoids in S that overlap $R(u)$. The construction implies that the trapezoids in $\mathcal{T}_u$ have at least one of their four corner vertices in the interior of $R(u)$. Since w_u is the sum of the weights of all the vertices in the interior of $R(u)$, it follows that $w_u \geq \sum_{z\in\mathcal{T}_u} w_z/4$.

Next observe that a trapezoid in $\mathcal{T}_u$ can generate at most one empty trapezoid in slab σ. Recall that the adjusted weight of an empty trapezoid generated by trapezoid $z \in S$ is $w_z/(2^t h)$. Thus the total adjusted weight of the empty trapezoids in σ is at most $\sum_{z\in\mathcal{T}_u} w_z/(2^t h)$. By the bound on w_u from the previous paragraph, this is at most $4w_u/(2^t h)$. $\square$

The following lemma establishes the essential property given in Section 2.3, by showing that the depth of any leaf associated with a cell is proportional to the logarithm of its reciprocal probability.

Lemma 7 *Let u be a leaf in T generated by any trapezoid $z \in S$. Then the depth of u in T is at most*

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

Proof: Let $P = u_1, u_2, \ldots, u_\ell$ be the path from the root to the leaf $u = u_\ell$ in the multi-way tree T'. Recall that each node of T' corresponds to a unique node of T. Consider a fixed i, $1 \leq i < \ell$. Let σ denote the vertical slab in $R(u_i)$ that contains the trapezoid associated with u_{i+1}, and let v denote the node in T corresponding to σ. To prove the lemma, we will separately bound the length of the paths in T from u_i to v and from v to u_{i+1}. By construction, the length of the path in T from u_i to v is t.

To bound the length of the path in T from v to u_{i+1}, recall that u_{i+1} is a leaf in the weighted search tree for slab σ. By standard results on weighted search trees [20], the length of the path in T from v to u_{i+1} is at most $\log(W/w) + 2$, where W is the total adjusted weight of all the trapezoids in slab σ, and w is the adjusted weight of the trapezoid associated with u_{i+1}. By Lemma 6, $W \leq (w_{u_i}/2^t)(1 + 4/h)$. We now consider two cases: (i) $1 \leq i \leq \ell - 2$ and (ii) $i = \ell - 1$. In the first case, u_{i+1} is a non-empty trapezoid, so its adjusted weight w is the same as its weight $w_{u_{i+1}}$.

Thus, the length of the path in T from v to u_{i+1} is at most

$$\log\left(\frac{(w_{u_i}/2^t)(1+4/h)}{w_{u_{i+1}}}\right)+2 \;=\; (\log w_{u_i}-\log w_{u_{i+1}})-t+\log\left(1+\tfrac{4}{h}\right)+2.$$

In the second case, $u_{i+1}=u_\ell$ is an empty trapezoid, so its adjusted weight w is $w_z/(2^t h)$. Thus, the length of the path in T from v to u_ℓ is at most

$$\log\left(\frac{(w_{u_{\ell-1}}/2^t)(1+4/h)}{w_z/(2^t h)}\right)+2 \;=\; (\log w_{u_{\ell-1}}-\log w_z)+\log(h+4)+2.$$

By using the above claim to bound the lengths of the paths in T between adjacent pairs of vertices in P, the fact that $w_{u_1}\le 1$, and summing and cancelling the telescoping probability terms, it is easy to see that the depth of u in T is at most

$$\log\frac{1}{w_z}+t+\left(2+\log\left(1+\frac{4}{h}\right)\right)(\ell-2)+\log h+O(1). \tag{1}$$

Recall the trapezoid z in the statement of the lemma. Since $u_{\ell-1}$ must contain at least one of the corner vertices of trapezoid z, it follows that $w_{u_{\ell-1}}\ge w_z/4$. Also, since the weight of a node at level i is at most $1/2^{t(i-1)}$, we have $w_{u_{\ell-1}}\le 1/2^{t(\ell-2)}$. Thus,

$$\ell-2 \;\le\; \frac{1}{t}\log\frac{1}{w_{u_{\ell-1}}} \;\le\; \frac{1}{t}\left(\log\frac{1}{w_z}+\log 4\right) \;\le\; \frac{1}{t}\left(\log\frac{1}{w_z}+2\right).$$

Substituting this value of ℓ in Eq. (1), and using the fact that $\log(1+4/h)=O(1/h)$, after some simplification, we get the bound on the depth of u given in the statement of the lemma. □

We can now bound the expected query time.

Lemma 8 *The expected query time using the BSP T is at most*

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

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

$$\sum_{z\in S}\sum_{u\in\mathcal{L}_z} p_u d_u,$$

where p_u denotes the probability that the query point lies in the region associated with node u, and d_u denotes the depth of u. By applying Lemma 7 it follows that this sum is at most

$$\sum_{z\in S}\sum_{u\in\mathcal{L}_z} p_u\left[\log\frac{1}{w_z}+t+\left(2+O\left(\frac{1}{h}\right)\right)\frac{1}{t}\log\frac{1}{w_z}+\log h+O(1)\right].$$

Using the facts that $\sum_{u\in\mathcal{L}_z}p_u=p_z$ and $\sum_{z\in S}p_z=1$ this is at most

$$\left(\sum_{z\in S}p_z\log\frac{1}{w_z}\right)+t+\left(2+O\left(\frac{1}{h}\right)\right)\frac{1}{t}\left(\sum_{z\in S}p_z\log\frac{1}{w_z}\right)+\log h+O(1).$$

Noting that $w_z \geq p_z/2$ and simplifying we obtain the desired bound. $\square$

In order to obtain the best bound on the expected query time, we choose $t = \left\lceil \sqrt{2H} \right\rceil$ and $h = \sqrt{H}$ in Lemma 8. This yields an expected query time of at most

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

Using Lemma 5 and noting that t is at most $O(\sqrt{\log n})$, we obtain a bound on the space of

$$O\left(n 2^{\sqrt{2H}} \frac{\log n}{\sqrt{H}}\right).$$

This completes the proof of Theorem 4.

Remark: By setting $t = \left\lceil \sqrt{2H} \right\rceil + c$, where c is a fixed positive integer, and $h = \sqrt{H}$, it is easy to see from Lemma 7 that the maximum depth in the search tree, that is, the worst-case query time is $(1+O(1/c))\log n$. Simultaneously, the bound on expected performance given by Theorem 4 also holds.

Remark: For the next section on entropy-preserving cuttings, it will be necessary to derive a bound on query times that is sensitive to the cell containing the query point. The following lemma establishes this for us.

Lemma 9 *We are given a trapezoidal map S of n segments, together with a non-negative weight w_z for each cell $z \in S$, such that $\sum_{z \in S} w_z \leq 1$ and a real parameter $1 \leq B \leq \sqrt{\log n}$. Then we can build a data structure that answers point location queries for each cell $z \in S$ in time*

$$\left[1 + O\left(\frac{1}{B}\right)\right] \log \frac{1}{w_z} + O(B).$$

The space and preprocessing time for the data structure are $O(n 2^{\sqrt{2\log n}} \sqrt{\log n})$.

Proof:

Let $h = B$ and $t = \lceil B\sqrt{2} \rceil$. From Lemma 7 it follows that the time to locate a query point in any cell z of S is at most

$$\begin{aligned}
&\log \frac{1}{w_z} + t + \left(2 + O\left(\frac{1}{h}\right)\right) \frac{1}{t} \log \frac{1}{w_z} + \log h + O(1) \\
&= \left[1 + \frac{1}{t}\left(2 + O\left(\frac{1}{h}\right)\right)\right] \log \frac{1}{w_z} + (t + \log h + O(1)) \\
&= \left[1 + O\left(\frac{1}{B}\right)\right] \log \frac{1}{w_z} + O(B),
\end{aligned}$$

as desired.

From Lemma 5, the total number of nodes of T is at most

$$O\left(n 2^t \left(\frac{\log n}{t} + 1\right)\right) = O\left(n 2^{B\sqrt{2}} \left(\frac{\log n}{B} + 1\right)\right).$$

For $B \geq 1/(\sqrt{2}\ln 2) \approx 1.02$, it is easy to verify that $2^{B\sqrt{2}}/B$ is an increasing function of B, and since $B \leq \sqrt{\log n}$, it follows that the space is at most

$$O\left(n2^{\sqrt{2\log n}}\left(\frac{\log n}{\sqrt{\log n}}+1\right)\right) \;=\; O\left(n2^{\sqrt{2\log n}}\sqrt{\log n}\right).$$

Recall that the preprocessing time is $O(N+n\log n)$, where N is the total space of the data structure. Thus the preprocessing time is dominated by the above space bound. This completes the proof. □

4 Entropy-Preserving Cuttings

The results of the previous section provide optimal expected query time (up to lower order terms), but the space required is $O(n^{1+\epsilon})$, for any $\epsilon > 0$. In this section we show how to apply the well known notion of cuttings in the expected-case setting to produce a structure of linear space. Let X be a set of line segments in the plane such that each $x \in X$ is associated with a positive weight w_x. Let $W = \sum_{x\in X} w_x$ denote the total weight. Consider a positive parameter r. For our purposes we define a $(1/r)$-*cutting* of X to be a partition of plane into trapezoids (in general these are canonical shapes of constant combinatorial complexity) such that the total weight of the segments of X that intersect the interior of any trapezoid is at most W/r. It is known that it is possible to compute such a cutting of size $O(r)$ in $O(n\log n)$ time by a randomized algorithm, and it can be computed deterministically in polynomial time [7, 14]. The construction also provides for each trapezoid of the cutting, the set of segments that intersect this trapezoid. Furthermore, this construction has the property that each trapezoid in this cutting is bounded from above and below by a subsegment of some segment in X, and the linear extension of each vertical side passes through the endpoint of some segment of X. (See Fig. 8 below.)

The point location construction of the previous section was based primarily on an assignment of weights to the vertices of the subdivision. Our approach here will involve a similar assignment of weights to the edges of the subdivision. Recall that (because of our conditioning) we may assume that the input S is a trapezoidal decomposition of a subdivision of combinatorial complexity n, whose edges (through perturbation) are nonvertical. Let X denote the set of line segments forming the edges of S. For each cell $z \in S$, recall that p_z denotes the probability that the query point lies within z, and that we make no other assumptions about the query distribution within a cell.

We begin with some notation. For any region Δ, let $S_\Delta = S \cap \Delta$ (i.e., the original subdivision S clipped to within Δ). (See Fig. 8.) Let p_Δ be the probability that the query point lies within Δ. We will use X_Δ to denote the set of segments in X that intersect the interior of Δ and T_Δ to denote the set of trapezoids in S that intersect the interior of Δ.

Each trapezoid of S can be associated by a subset of at most four *defining segments* of X, which together define its four sides. These are the segments defining the trapezoid's upper and lower sides, and any segment whose endpoint lies on its left vertical side, and any segment whose endpoint lies on its right vertical side. (These segments are indicated with an asterisk in Fig. 9.) We begin by assigning weights to the segments of X as follows. For each trapezoid $z \in S$, we assign a weight of

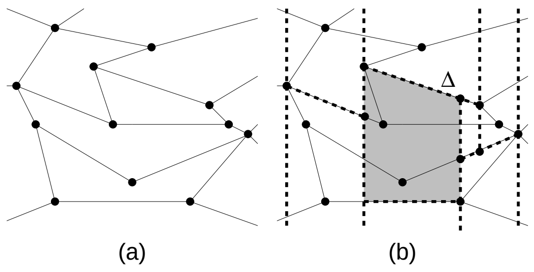

Fig. 8: A subdivision (a), a cutting (shown with broken lines) and a cell of the cutting (shaded) (b), and the restriction S_Δ (c).

$w_z/4$ to each of its defining segments where

$$w_z \;=\; \frac{1}{2}\max\left(p_z, \frac{1}{n}\right).$$

If a segment is a defining segment of multiple trapezoids, then its weight is the sum of the contributions from all such trapezoids. Note that the total weight of all the segments is at most 1.

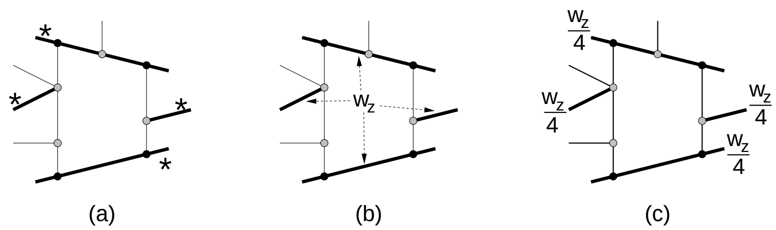

Fig. 9: Defining segments (a) and the weight distribution ((b) and (c)).

For any region Δ, we define its weight $\widehat{w}_\Delta$ to be the total weight of all the segments of X that intersect the interior of Δ, that is, $\widehat{w}_\Delta = \sum_{x \in X_\Delta} w_x$. The main result of this section is to show the existence of entropy-preserving cuttings in Lemma 11. The following lemma will be useful in proving this result.

Lemma 10 *Let Δ be any trapezoid such that at least one segment in X intersects its interior. Then*

(i) $\sum_{z \in T_\Delta} p_z = O(\widehat{w}_\Delta)$.
(ii) $p_\Delta = O(\widehat{w}_\Delta)$.
(iii) $|X_\Delta| = O(n\widehat{w}_\Delta)$.

Proof: By definition, the weight w_x of any segment $x \in X$ exceeds the sum of $p_z/8$ over all trapezoids $z \in S$ that have x as a defining segment. It follows that, for any $x \in X_\Delta$, w_x exceeds the sum of $p_z/8$ over all trapezoids $z \in T_\Delta$ that have x as a defining segment. Since every trapezoid

$z \in T_\Delta$ is defined by some segment in X_Δ, we obtain

$$\sum_{z \in T_\Delta} \frac{p_z}{8} \;\leq\; \sum_{x \in X_\Delta} w_x = \widehat{w}_\Delta,$$

which implies (i). From this we have

$$p_\Delta \;=\; \sum_{z \in S_\Delta} p_z \leq \sum_{z \in T_\Delta} p_z \;=\; O(\widehat{w}_\Delta),$$

which implies (ii).

Finally, since the weight of each segment in X is $\Omega(1/n)$, it follows that there can be at most $O(n\widehat{w}_\Delta)$ segments in X_Δ. □

Lemma 11 (Entropy Preserving Cuttings) *For any $r \geq 1$, in time $O(n \log n)$ we can partition the plane into $O(r)$ trapezoids satisfying the following properties. Let $\mathcal{C}$ denote the subdivision consisting of these $O(r)$ trapezoids, S^* denote the subdivision formed by superimposing $\mathcal{C}$ on S, and Δ denote any trapezoid in $\mathcal{C}$.*

(i) *Either Δ is a subset of some trapezoid in S or the probability p_Δ that the query point lies within Δ is $O(1/r)$.*
(ii) *The number of segments, $|X_\Delta|$, intersecting the interior of Δ is $O(n/r)$.*
(iii) $\sum_{\Delta \in \mathcal{C}} \sum_{z \in T_\Delta} p_z = O(1)$.
(iv) *Let $H_S = \sum_{z \in S} p_z \log \frac{1}{p_z}$ and $H_{S^*} = \sum_{u \in S^*} p_u \log \frac{1}{p_u}$ be the respective entropies of these subdivisions. Then the increase in entropy, $H_{S^*} - H_S$, is at most a constant.*

Proof: Using the weight assignment w_x defined above, we compute a standard weighted $(1/r)$-cutting for X. As mentioned at the start of this section, such a cutting has size $O(r)$ and can be computed in $O(n \log n)$ time. Furthermore, each trapezoid of this cutting is bounded from above and below by a subsegment of some segment of X, and the linear extension of each vertical side passes through the endpoint of some segment of X. Let $\mathcal{C}^*$ denote this cutting.

We modify the cutting $\mathcal{C}^*$ by applying the following procedure on each trapezoid $z \in S$. If a trapezoid of the cutting lies within z, then by the properties of the cutting, this trapezoid is bounded from above and below by the same segments that bound z from above and below. Thus, any adjacent trapezoids of the cutting that are contained within z must be separated from each other by a vertical line segment, and hence a collection of adjacent cutting trapezoids within z occurs as a contiguous sequence. If z contains such a contiguous sequence of trapezoids of $\mathcal{C}^*$, we merge each such maximal subsequence into a single trapezoid. (See Fig. 10.) It is easy to see that after this modification, we still have a weighted $(1/r)$-cutting. Let $\mathcal{C}$ denote this new cutting. We claim that $\mathcal{C}$ satisfies the four properties given in the statement of the lemma.

Let Δ denote any trapezoid in $\mathcal{C}$. We distinguish between two cases: (a) Δ is completely contained within a trapezoid in S and (b) its interior intersects two or more trapezoids of S. In the former case, (i) and (ii) obviously hold. Consider the latter case. Because each trapezoid of C is bounded above and below by a segment of X, it follows in this case that some segment of

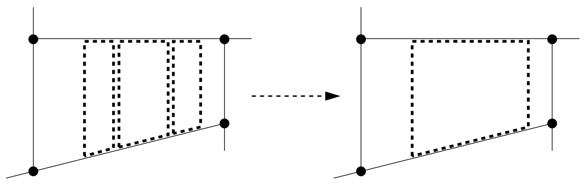

Fig. 10: Merging contiguous trapezoids of the cutting.

X intersects the interior of Δ. By Lemma 10(ii), $p_\Delta = O(\widehat{w}_\Delta) = O(1/r)$ and, by Lemma 10(iii), $|X_\Delta| = O(n\widehat{w}_\Delta) = O(n/r)$. This proves (i) and (ii).

To prove (iii), recall that S^* is 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. For any trapezoid $z \in S$, let $\mathcal{F}_z$ denote the set of fragments contained within z and let $f_z = |\mathcal{F}_z|$. Observe that $\sum_{\Delta \in \mathcal{C}} \sum_{z \in T_\Delta} p_z$ is the same as $\sum_{z \in S} p_z f_z$. We will show that the latter quantity is $O(1)$, which will prove (iii).

We distinguish between two kinds of fragments in $\mathcal{F}_z$. Let $u \in \mathcal{F}_z$ be the intersection of z with a trapezoid $\Delta \in \mathcal{C}$; if $\Delta \subseteq z$, we call it a *type-1 fragment*, otherwise it is a *type-2 fragment*. Let f'_z and f''_z denote the number of fragments in z of type 1 and type 2, respectively. In view of the merging process (recall Fig. 10), it is clear that there must be a fragment of type 2 between any two fragments of type 1, and so $f'_z \leq f''_z + 1$. Thus

$$\sum_{z \in S} p_z f_z \;=\; \sum_{z \in S} p_z (f'_z + f''_z) \;\leq\; \sum_{z \in S} p_z (2f''_z + 1) \;=\; 1 + 2\sum_{z \in S} p_z f''_z. \tag{2}$$

To analyze this sum, let $\mathcal{C}''$ denote the set of trapezoids in $\mathcal{C}$ that are not completely contained within a cell of S. Note that $\sum_{z \in S} p_z f''_z$ is the same as $\sum_{\Delta \in \mathcal{C}''} \sum_{z \in T_\Delta} p_z$. By Lemma 10(i), for each $\Delta \in \mathcal{C}''$, $\sum_{z \in T_\Delta} p_z = O(\widehat{w}_\Delta) = O(1/r)$. Since $|\mathcal{C}''| \leq |\mathcal{C}| = O(r)$, it follows that $\sum_{z \in S} p_z f''_z = O(1)$. Substituting in Eq. (2), we have $\sum_{z \in S} p_z f_z = O(1)$, which completes the proof of (iii).

For convenience we express part (iv) of the lemma as

$$\sum_{u \in S^*} p_u \log \frac{1}{p_u} - \sum_{z \in S} p_z \log \frac{1}{p_z} \;=\; O(1). \tag{3}$$

The left hand side of Eq. (3) can be written as

$$\sum_{z \in S} \left[\left(\sum_{u \in \mathcal{F}_z} p_u \log \frac{1}{p_u} \right) - p_z \log \frac{1}{p_z} \right].$$

Since $\sum_{u \in \mathcal{F}_z} p_u = p_z$, it follows from basic properties of entropy [10] that $\sum_{u \in \mathcal{F}_z} p_u \log(1/p_u)$ is maximized when the probability of all the fragments $u \in \mathcal{F}_z$ is equal, i.e. p_z/f_z. Thus

$$\sum_{u \in S^*} p_u \log \frac{1}{p_u} - \sum_{z \in S} p_z \log \frac{1}{p_z} \;\leq\; \sum_{z \in S} \left[f_z \cdot \frac{p_z}{f_z} \log \frac{f_z}{p_z} - p_z \log \frac{1}{p_z} \right] \;=\; \sum_{z \in S} p_z \log f_z \;\leq\; \sum_{z \in S} p_z f_z.$$

In proving (iii) above, we showed that $\sum_{z \in S} p_z f_z = O(1)$, which establishes Eq. (3). $\square$

4.1 Space Reduction through Entropy Preserving Cuttings

We are now ready to describe our data structure. Setting the parameter $r = n/(2^{\sqrt{2\log n}}\sqrt{\log n})$, we construct the subdivision $\mathcal{C}$ described in Lemma 11. Let $c = \sum_{\Delta\in\mathcal{C}}\sum_{z\in T_\Delta} p_z$, which by Lemma 11(iii) is $O(1)$. For each trapezoid $\Delta \in \mathcal{C}$, we assign it a weight $w_\Delta = \sum_{x\in T_\Delta} p_x/c$. (Note that this weight function is used for the purposes of building the point location structure, and is not to be confused with $\widehat{w}_\Delta$ used in the analysis of the previous section.) By our choice of c we have $\sum_{\Delta\in\mathcal{C}} w_\Delta = 1$. From our initial conditioning, we know that $H_S \geq 1$. So using these weights, and setting $B = \sqrt{H_S}$ we can now apply Lemma 9 to build the point location data structure for $\mathcal{C}$. Since $|\mathcal{C}| = O(n/(2^{\sqrt{2\log n}}\sqrt{\log n}))$, it follows that the space used by this structure is $O(n)$.

Let $\mathcal{C}'$ denote the set of trapezoids in $\mathcal{C}$ that are contained within some cell of S and let $\mathcal{C}''$ denote the rest of the trapezoids in $\mathcal{C}$. For each trapezoid $\Delta \in \mathcal{C}''$, we build any standard worst-case point location data structure for the subdivision S_Δ. This data structure uses $O(|X_\Delta|)$ space and answers queries in time $O(\log|X_\Delta|)$. Each such structure can be computed in $O(|X_\Delta|\log|X_\Delta|)$ time by standard algorithms [11]. By Lemma 11 (ii) we know that $|X_\Delta|$ is $O(n/r)$, and since there are $O(r)$ trapezoids in $\mathcal{C}$, it follows that the total time to build all these structures is $O(n\log n)$.

The method for answering queries is very simple. First we use the point location structure for $\mathcal{C}$ to identify the trapezoid $\Delta \in \mathcal{C}$ that contains the query point q. If $\Delta \in \mathcal{C}'$, then we can directly output the cell $z \in S$ that contains q. Otherwise we use the point location structure for S_Δ to locate the cell in S_Δ (and hence in S) that contains q.

We now analyze the space requirements. By Lemma 11(ii), for each trapezoid $\Delta \in \mathcal{C}''$, $|X_\Delta| = O(2^{\sqrt{2\log n}}\sqrt{\log n})$. Since the number of trapezoids in $\mathcal{C}''$ is $O(n/(2^{\sqrt{2\log n}}\sqrt{\log n}))$, the space used by all the point location structures corresponding to the trapezoids $\Delta \in \mathcal{C}''$ is $O(n)$. Recall that the space used by the point location structure for $\mathcal{C}$ is also $O(n)$. Thus the total space used is $O(n)$. In the next section we analyze the expected query time, which is the last step needed to establish Theorem 1.

4.2 Analysis of Expected-Case Query Time

We begin by borrowing some terminology from the proof of Lemma 11. Recall that S^* be the subdivision formed by superimposing $\mathcal{C}$ on S. The cells of this refined subdivision are called *fragments*, and for any trapezoid $z \in S$, let $\mathcal{F}_z$ denote the set of fragments contained within z. We distinguish two types of fragments. Consider a fragment u that arises from the intersection of a cutting cell Δ and a subdivision cell z. If Δ is contained within z, this is a *type-1 fragment*, and otherwise it is a *type-2 fragment.* The set of type-1 and type-2 fragments of z are denoted $\mathcal{F}'_z$ and $\mathcal{F}''_z$, respectively. Finally, let $\mathcal{F}' = \cup_{z\in S}\mathcal{F}'_z$ and $\mathcal{F}'' = \cup_{z\in S}\mathcal{F}''_z$.

Let q denote a query point in a fragment $u \in \mathcal{F}_z$. Let T_1 denote the time it takes to determine the trapezoid $\Delta \in \mathcal{C}$ that contains q. By Lemma 9 and using the fact that $w_\Delta \geq p_z/c$ (where c is the constant defined in the first paragraph of Section 4.1) we have

$$\begin{aligned} T_1 &\le \left[1 + O\left(\frac{1}{\sqrt{H_S}}\right)\right] \log \frac{1}{w_\Delta} + O\left(\sqrt{H_S}\right) \;\le\; \left[1 + O\left(\frac{1}{\sqrt{H_S}}\right)\right] \log \frac{c}{p_z} + O\left(\sqrt{H_S}\right) \\ &\le \left[1 + O\left(\frac{1}{\sqrt{H_S}}\right)\right] \log \frac{1}{p_z} + O\left(\sqrt{H_S}\right). \end{aligned}$$

If $u \in \mathcal{F}'_z$ (i.e., $\Delta = u$), then we are done after finding Δ and so the query time is T_1. Otherwise $u \in \mathcal{F}''_z$ and we need an additional time $T_2 = O(\log |X_\Delta|)$ to search S_Δ and determine u. By Lemma 11(ii), $|X_\Delta| = O(2^{2\sqrt{\log n}}\sqrt{\log n})$, so $T_2 = O(\sqrt{\log n})$.

Putting it all together, we have shown that the expected query time is

$$\sum_{z \in S} \left[\sum_{u \in \mathcal{F}_z} p_u \left(\left[1 + O\left(\frac{1}{\sqrt{H_S}}\right)\right] \log \frac{1}{p_z} + O\left(\sqrt{H_S}\right) \right) + \sum_{u \in \mathcal{F}''_z} p_u O\left(\sqrt{\log n}\right) \right].$$

Using the facts that $\sum_{u \in \mathcal{F}_z} p_u = p_z$, $\sum_{z \in S} p_z = 1$ and writing H_S in place of $\sum_{z \in S} p_z \log(1/p_z)$, this simplifies to

$$H_S + O\left(\sqrt{H_S}\right) + O\left(p'' \sqrt{\log n}\right), \tag{4}$$

where $p'' = \sum_{z \in S} \sum_{u \in \mathcal{F}''_z} p_u = \sum_{u \in \mathcal{F}''} p_u$.

Before continuing further, we establish a relationship between p'' and the entropy H_S in the following lemma. Intuitively, the lemma shows that p'' is small when the entropy H_S is small.

Lemma 12 *Let p'' be the probability that the query point falls in a fragment $u \in \mathcal{F}''$. Then $p'' = O(H_S / \log n)$.*

Proof: We first compute a lower bound on H_{S^*}. Clearly

$$H_{S^*} \;=\; \sum_{u \in S^*} p_u \log(1/p_u) \;\ge\; \sum_{u \in \mathcal{F}''} p_u \log(1/p_u).$$

By Lemma 11(i), for each $\Delta \in \mathcal{C}''$, $p_\Delta = O(2^{\sqrt{2 \log n}}\sqrt{\log n}/n)$. Let $p_m = \max_{u \in \mathcal{F}''} p_u$. Since any fragment in $\mathcal{F}''$ is a subset of some $\Delta \in \mathcal{C}''$, it follows that $p_m = O(2^{\sqrt{2 \log n}}\sqrt{\log n}/n)$. Recall that $p'' = \sum_{u \in \mathcal{F}''} p_u$. By basic properties of entropy [10], the entropy is minimized by maximizing the disparity among the probabilities. This is done by setting as many of the p_u's to p_m as possible, subject to the condition that they sum to p''. Thus, we obtain a lower bound on $\sum_{u \in \mathcal{F}''} p_u \log(1/p_u)$ by assuming that $\lfloor p''/p_m \rfloor$ of the fragments $u \in \mathcal{F}''$ have $p_u = p_m$, one fragment $u \in \mathcal{F}''$ has the leftover probability $p_u = p'' - \lfloor p''/p_m \rfloor p_m$, and all remaining fragments $u \in \mathcal{F}''$ have $p_u = 0$. Thus

$$H_{S^*} \;\ge\; \left(\frac{p''}{p_m} - 1\right) \cdot p_m \log \frac{1}{p_m} \;\ge\; p'' \log \frac{1}{p_m} - O(1) \;\ge\; \Omega(p'' \log n - 1).$$

Also, by Lemma 11(iv), $H_{S^*} \le H_S + O(1)$. Combining the lower and upper bound on H_{S^*}, we obtain $H_S + O(1) = \Omega(p'' \log n - 1)$. Recalling that $H_S \ge 1$, the lemma follows. □

Applying Lemma 12 to Eq. (4), we see that the expected query time is

$$H_S + O\left(\sqrt{H_S}\right) + O\left(H_S/\sqrt{\log n}\right) \;\leq\; H_S + O\left(\sqrt{H_S}\right),$$

where we have used the fact that $H_S = O(\log n)$.

This completes the proof of our main result, Theorem 1.

5 Convex Polygons with Uniform Distribution

The main result of this section is the following.

Lemma 13 *Let S be a planar subdivision of size n 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 subdivision exceeds the entropy of S by at most an additive constant. The new subdivision can be constructed in $O(n \log n)$ time.*

By applying Theorem 1 to the resulting triangulation, it readily follows from the above lemma that the bounds on space and expected query time given in the theorem also apply to any subdivision S whose cells are convex polygons, assuming that the query distribution is uniform within each polygon.

The proof of Lemma 13 relies on the following straightforward observation. We present its proof for completeness.

Lemma 14 *Given a convex polygon P with m 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. Moreover, these vertices can be found in $O(m)$ time.*

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 $\overline{v_1 v_2}$. By standard results, the vertices v_1 and v_2 can found in $O(m)$ time [23], and by computing the distance to all vertices, v_3 too can be found in $O(m)$ time. 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. $\square$

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 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

$$entropy(\mathcal{T}_z) \leq p_z \log \frac{1}{p_z} + 8p_z, \tag{5}$$

where $entropy(\mathcal{T}_z)$ 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, $entropy(\mathcal{T}_{z_i}) \leq p_{z_i} \log(1/p_{z_i}) + 8p_{z_i}$, for $1 \leq i \leq 3$. Thus $entropy(\mathcal{T}_z)$ can be written as

$$\begin{aligned} & p_y \log \frac{1}{p_y} + \sum_{i=1}^{3} entropy(\mathcal{T}_{z_i}) \\ \leq \;& p_y \log \frac{1}{p_y} + \sum_{i=1}^{3} \left(p_{z_i} \log \frac{1}{p_{z_i}} + 8p_{z_i} \right) \\ = \;& \left(p_y \log \frac{1}{p_y} + \sum_{i=1}^{3} p_{z_i} \log \frac{1}{p_{z_i}} \right) + 8 \sum_{i=1}^{3} p_{z_i}. \end{aligned} \tag{6}$$

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 basic properties of entropy [10], 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. (6) 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. (5) over all the polygons in S, it follows that the entropy of the triangulation exceeds the entropy of S by at most 8.

Finally, we discuss the time it takes to construct the triangulation. Let m denote the size of a convex polygon $z \in S$. By Lemma 14, it takes $O(m)$ time to find the first triangle y in z, and decompose $z - y$ into (at most) three convex polygons z_1, z_2, and z_3. If $z - y$ consists of just one polygon with $m - 1$ vertices, then continuing in this way, it may take $O(m^2)$ time to complete the triangulation of z. To reduce this to $O(m \log m)$, we make a small change to the construction. At each step in the recursion, we partition the current polygon into two polygons with roughly equal number of vertices, and then use Lemma 14 to find a triangle with large area in each of these two polygons. This reduces the depth of the recursion to $O(\log m)$, and since the time taken for each level of the recursion is $O(m)$, the total time becomes $O(m \log m)$. It is now a simple exercise to extend the above proof to show that this modified triangulation algorithm also preserves the entropy (only the constant 8 increases). □

6 Concluding Remarks

We have shown that, given a polygonal subdivision S of size n consisting of cells of constant combinatorial complexity and a query distribution presented as a weight assignment to the cells of S, it is possible to answer point location queries $H + O(\sqrt{H} + 1)$, where H is the entropy of the subdivision. Our data structure requires $O(n)$ space, and it can be constructed in $O(n \log n)$ time by a randomized algorithm. We make no assumptions about the distribution of query points within each cell of the subdivision. We have also shown that this result can be extended to convex subdivisions of arbitrary combinatorial complexity assuming that the query distribution is uniform within each cell.

There are a number of open problems suggested here. If point location is based on the results of primitive comparisons, it is known that the entropy is a lower bound on the expected running time, and so our results are optimal up to lower order terms. Assuming that the query distribution within the cells is unknown, a stronger lower bound of $H + \sqrt{H} - O(1)$ is known in the trapezoidal search graph model [19]. Can the lower-bound analysis be refined to justify the presence of the $O(\sqrt{H})$ term, when information on the query distribution within the cells is available?

A related observation is that, in the case of worst-case query time, Seidel and Adamy [26] showed an analogous lower order term of $2\sqrt{\log n}$. An interesting question along these lines is whether it is possible to reduce the lower order term in Theorem 4, say to $2\sqrt{H}$? Another interesting question is the computational complexity of computing the BSP that minimizes the expected search time, assuming, say, the use of extended primitive comparisons and an oracle that can answer questions about the query distribution.

7 Acknowledgements

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

References

[1] 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.

[2] S. Arya and H. Y. Fu. Expected-case complexity of approximate nearest neighbor searching. *SIAM J. Comput.*, 32(3):793–815, 2003.

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

[4] 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*, pages 256–261, 2001.

[5] 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*, pages 262–268, 2001.

[6] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, September 1979.

[7] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

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

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

[10] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, NY, 1991.

[11] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

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

[13] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990.

[14] 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.

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

[16] J. Iacono. Optimal planar point location. In *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, pages 340–341, 2001.

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

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

[19] T. Malamatos. *Expected-case planar point location*. PhD thesis, Dept. of Computer Science, Hong Kong University of Science and Technology, 2002.

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

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

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

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

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

[25] 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.

[26] R. Seidel and U. Adamy. On the exact worst case complexity of planar point location. *J. Algorithms*, 37:189–217, 2000.

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