Efficient Expected-Case Algorithms for Planar Point Location

Sunil Arya[1], Siu-Wing Cheng*[1], David M. Mount**[2], and H. Ramesh[3]

[1] Department of Computer Science, The Hong Kong University of Science and Technology,
Clear Water Bay, Kowloon, Hong Kong
{arya, scheng}@cs.ust.hk
[2] Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland
mount@cs.umd.edu
[3] Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India
ramesh@csa.iisc.ernet.in

Abstract. Planar point location is among the most fundamental search problems in computational geometry. Although this problem has been heavily studied from the perspective of worst-case query time, there has been surprisingly little theoretical work on expected-case query time.
We are given an n-vertex planar polygonal subdivision S satisfying some weak assumptions (satisfied, for example, by all convex subdivisions). We are to preprocess this into a data structure so that queries can be answered efficiently. We assume that the two coordinates of each query point are generated independently by a probability distribution also satisfying some weak assumptions (satisfied, for example, by the uniform distribution).
In the decision tree model of computation, it is well-known from information theory that a lower bound on the expected number of comparisons is $\mathit{entropy}(S)$. We provide two data structures, one of size $O(n^2)$ that can answer queries in $2\,\mathit{entropy}(S) + O(1)$ expected number of comparisons, and another of size $O(n)$ that can answer queries in $(4 + O(1/\sqrt{\log n}))\,\mathit{entropy}(S) + O(1)$ expected number of comparisons. These structures can be built in $O(n^2)$ and $O(n \log n)$ time respectively. Our results are based on a recent result due to Arya and Fu, which bounds the entropy of overlaid subdivisions.

1 Introduction

Planar point location is certainly among the most fundamental search problems in computational geometry. Given a polygonal subdivision S in the plane, the problem is to construct a data structure so that given any query point q in the

* Research supported in part by RGC Grant HKUST 6088/99E.
** Research supported in part by NSF Grant CCR–9712379.

plane, it is possible to determine efficiently which polygon of the subdivision contains q. This problem has been heavily studied in computational geometry. (For example, a search for "point location" found 77 papers in the computational geometry bibliography.) With only a few exceptions, previous work on this problem has dealt with the worst-case complexity of this problem. When expected-case complexity has been considered, it has been done under the assumption that both the subdivision and the query points are selected subject to various assumptions on distribution. Here, we consider search algorithms that are efficient in the expected-case for queries, and in the worst-case for subdivisions.

The planar point location problem is a generalization of the well-known one-dimensional search problem. In the one-dimensional case, we are given a set of n keys, and told the probabilities of accessing each key and the $n+1$ failure probabilities of falling in the gaps between the keys. If we assume that the probability of matching a key is zero, then this reduces to the expected-case complexity of solving a point location problem for $n+1$ disjoint subintervals of the unit interval. Consider any binary search tree whose leaves correspond to the intervals. It is easy to see that the expected number of comparisons is given by the *weighted external path length* [14] of the tree, where the weight of a leaf is the probability of the query point lying in the associated interval.

Let p_i denote the probability of falling in the ith interval. A fundamental information theoretic result due to Shannon implies that the weighted path length of any binary tree (and hence the expected number of comparisons) is at least the *entropy* of the probability distribution

$$\sum_i p_i \log\left(\frac{1}{p_i}\right).$$

(Unless otherwise stated, all logarithms are base 2.) Knuth [13] shows how to construct an optimum binary search tree in $O(n^2)$ time using dynamic programming. Hu and Tucker [11] presented a bottom-up construction of the tree, which takes $O(n \log n)$ time, but is quite complex. Mehlhorn [17] gives a simple construction of a binary search tree whose weighted path length is within a constant additive factor of the entropy-based lower bound. It is eminently natural to ask whether these results can be extended to planar subdivisions. To the best of our knowledge, this is the first paper to address this obvious and fundamental problem.

Consider a polygonal subdivision S. Given a region z in S, let p_z denote the probability that the query point lies inside region z. Define the *entropy* of S to be

$$\mathit{entropy}(S) = \sum_{z \in S} p_z \log\left(\frac{1}{p_z}\right).$$

The coordinates of the query points are assumed to be sampled independently from probability distributions over bounded intervals of the x-axes and y-axes. Both S and the probability distributions are assumed to satisfy some additional weak assumptions (see Section 2 for formal definitions).

Shannon's lower bound applies to the planar point location problem as well. We present two algorithms for the planar point location problem. The first uses quadratic space and can answer point location queries in $2\,\mathit{entropy}(S) + O(1)$ expected number of point-line comparisons (i.e., given a point and a directed line, one has to determine whether the point lies to the left of, on, or to the right of the line). The second uses $O(n)$ space, and can answer point location queries in nearly $4\,\mathit{entropy}(S) + O(1)$ expected number of point-line comparisons.

The paper is organized as follows. In Section 2 we present definitions and state our results formally. In Section 3 we present background on the planar point location problem. In Section 4 we present our algorithms for the case of uniformly distributed query points, and in Section 5 we generalize our results to a wider class of probability distributions.

2 Definitions and Main Results

Let I and J be two arbitrary intervals of real numbers. In this paper, we only work with planar subdivisions that partition an underlying rectangle $I \times J$ into disjoint connected regions. We allow the underlying rectangle to be the infinite plane, in which case $I = J = (-\infty, \infty)$.

Given a query point q, let x_q and y_q denote its x and y coordinate. Throughout this paper, we assume that x_q and y_q are two *independent* random variables. We denote the probability distribution function for x_q by $P : I \to [0, 1]$ and the probability distribution function for y_q by $Q : J \to [0, 1]$. That is, $P(x)$ is the probability that the random variable x_q is less than or equal to x, and $Q(y)$ is the probability that the random variable y_q is less than or equal to y. We call (P, Q) a *well-behaved distribution* if P and Q are *continuous* and *strictly increasing*. For example, if $I \times J$ is the unit square, then picking x_q and y_q uniformly and independently from $[0, 1]$ yields a well-behaved distribution.

Let U be the unit square $[0, 1]^2$. We define a mapping f_{PQ} from $I \times J$ (call this *geometric space*) to U (call this *probability space*) as follows:

$$f_{PQ}(x, y) = (P(x), Q(y)).$$

If (P, Q) is well-behaved, then f_{PQ} is a bijection as P and Q are strictly increasing. We can also generalize f_{PQ} in the obvious way for a set of points. Let A be any set points in $I \times J$. Then

$$f_{PQ}(A) = \{(P(x), Q(y)) : (x, y) \in A\}.$$

In this paper, we assume that each evaluation of P, Q, P^{-1}, and Q^{-1} takes constant time.

We are now ready to state the main results of this paper.

Theorem 1. *Let I and J be two intervals of real numbers. Let $\mathcal{S}$ be a planar subdivision of $I \times J$ of n vertices. Suppose that a well-behaved distribution (P, Q) is given for the coordinates of the query point, and for each region $z \in \mathcal{S}$, $f_{PQ}(z)$ has at most a constant number of holes and the perimeter of $f_{PQ}(z)$ is bounded by a constant. Then*

(i) Using $O(n^2)$ space and $O(n^2)$ preprocessing time, it is possible to answer point location queries in $2\,entropy(\mathcal{S}) + O(1)$ expected number of point-line comparisons.

(ii) Using $O(n)$ space and $O(n \log n)$ preprocessing time, it is possible to answer point location queries in $(4+O(1/\sqrt{\log n}))\,entropy(\mathcal{S})+O(1)$ expected number of point-line comparisons.

Clearly this theorem applies if $I \times J$ is the unit square U and x_q and y_q are chosen uniformly and independently from $[0, 1]$. Thus we have the following theorem, which is an interesting special case of Theorem 1:

Theorem 2. *Let $\mathcal{S}$ be a planar subdivision of U of n vertices. Suppose that the coordinates of the query point are chosen uniformly and independently from $[0, 1]$, and for each region z in $\mathcal{S}$, z has at most a constant number of holes, and the perimeter of z is bounded by a constant. Then*

(i) Using $O(n^2)$ space and $O(n^2)$ preprocessing time, it is possible to answer point location queries in $2\,entropy(\mathcal{S}) + O(1)$ expected number of point-line comparisons.

(ii) Using $O(n)$ space and $O(n \log n)$ preprocessing time, it is possible to answer point location queries in $(4+O(1/\sqrt{\log n}))\,entropy(\mathcal{S})+O(1)$ expected number of point-line comparisons.

Remark: It is worth noting that any convex polygon in the geometric space is mapped by f_{PQ} to a region in the probability space that has bounded perimeter. This follows from the fact that any monotonic increasing (resp. decreasing) curve in the geometric space maps to a monotonic increasing (resp. decreasing) curve in the probability space. And the length of any monotonic curve in the unit square is bounded by 2. Thus Theorem 1 applies to any subdivision of the plane into (bounded and unbounded) convex polygons.

3 Background

For the planar point location problem, let n denote the number of vertices in the subdivision. The early work of Dobkin and Lipton [6] showed that a query time of $O(\log n)$ and space $O(n^2)$ could be achieved. Lipton and Tarjan [15] showed that the space requirement could be reduced to $O(n)$, but their approach was rather impractical. Since then a number of more practical methods have been proposed. These include Kirkpatrick's clever hierarchical method [12], the separator method by Edelsbrunner et al. [8], the persistent search tree method by Sarnak and Tarjan [21], and the randomized incremental method by Mulmuley [20]. All of these are based on worst-case analyses. Recently Adamy and Seidel [1] presented an $O(n)$ space data structure that achieves a worst-case query time of $\log n + 2\sqrt{\log n} + O(\log^{1/4} n)$ point-line comparisons, thus approaching the worst-case information theoretic lower bound.

Existing work on expected case-performance has been based on the assumption that both the subdivision and the queries satisfy certain probabilistic assumptions. Edahiro et al. [7] proposed a practical algorithm for planar point location based on bucketing techniques. Their method may use $\Theta(n^2)$ space in the worst case. Methods using kd-trees, quad-trees, and R-trees are also popular in practice, but their analyses do not hold in the worst case. Mucke et al. [19] and Devroye et al. [5] have analyzed methods based on walking through subdivisions. For Delaunay triangulations of uniformly distributed data sets, these methods take expected time close to $O(n^{1/3})$ and $O(n^{1/4})$ in two and three dimensions, respectively.

Goodrich et al. [10] presented an interesting point location method, which adapts to the query distribution. Intuitively, if a cell is accessed more frequently, then the data structure is modified to ensure that the time for subsequent accesses to the cell is reduced. They show that the amortized time complexity for accessing cell i in a sequence of m queries is $O(\min\{\log n, \log(t(i) + 1), \log(m/f(i))\})$, where $t(i)$ is the number of different queries between two accesses to cell i, and $f(i)$ is the frequency of accesses to cell i. A limitation of their approach is that the cells are not the regions in the given subdivision; instead they are the trapezoids in the refined subdivision formed by passing a vertical line through each segment endpoint. This can adversely affect the query time.

4 The Uniform Distribution Case

We first present our techniques in attacking the case when $I \times J$ is U and the coordinates of the query point are chosen uniformly and independently from $[0, 1]$. The techniques are based on certain box decompositions of the planar subdivision $\mathcal{S}$ of U. In the case of general well-behaved distributions (P, Q), the key insight is that the map f_{PQ} transforms the problem in the geometric space to a problem in the probability space, where we are to locate query points in the subdivision $f_{PQ}(S)$ of U, and the coordinates of the query point are chosen uniformly and independently from $[0, 1]$. Thus, for general well-behaved distributions, it suffices to invoke the techniques for uniform distribution in the probability space to organize a point location data structure. Given a query point q, we use this data structure to locate the region z' in $f_{PQ}(\mathcal{S})$ containing $f_{PQ}(q)$, and the region in $\mathcal{S}$ containing z is then given by $f_{PQ}^{-1}(z')$. These claims will be proved formally in Section 5.

In the following, we focus on the uniform distribution case. We first present an algorithm, which uses $2\ \mathit{entropy}(\mathcal{S}) + O(1)$ expected number of point-line comparisons. The data structure needs $O(n^2)$ space and can be built in $O(n^2)$ time. Later we present another algorithm which reduces the space to $O(n)$ and the preprocessing time to $O(n \log n)$. The expected number of point-line comparisons goes up to nearly $4\ \mathit{entropy}(\mathcal{S}) + O(1)$.

A lemma proved in [2] will be very useful. We state it in a form which is applicable in two dimensions. The result concerns with overlaying two planar subdivisions of U. One subdivision is the given planar subdivision $\mathcal{S}$ of U. The

other subdivision is a decomposition of U into *cells* that enjoys the following properties, for some constants c_a and c_n:

(A.1) *Difference of Two Rectangles:* A cell is the set-theoretic difference of two axis-parallel rectangles, one enclosed within the other. We call these the *outer rectangle* and *inner rectangle* of the cell. Note that the inner rectangle need not be present. Given a cell u, we let u_O and u_I denote its outer and inner rectangle, respectively. Also, we define the size of u, denoted by s_u, to be the length of the longest side of u_O.

(A.2) *Bounded Aspect Ratio:* The outer rectangle and inner rectangle (if present) have aspect ratio (ratio of longest to shortest side) bounded by c_a. (In this case we say that the cell has aspect ratio at most c_a.)

(A.3) *Stickiness:* If the cell has an inner rectangle, then for each dimension, the separation between the corresponding faces of the inner and outer rectangle is either 0 or at least the length of the inner rectangle along that dimension.

(A.4) *Proximity to S*: For each cell u, there is some edge or vertex in $\mathcal{S}$ within a distance of $c_n \cdot s_u$ from any point in u_O.

(A.5) *Disjointness:* Given any two cells, either the outer rectangles of the two cells are disjoint or the outer rectangle of one cell is contained within the inner rectangle of the other.

We define a *fragment* to be a connected component in the intersection between a cell in the decomposition and a region in $\mathcal{S}$. Let $\mathcal{F}$ be the set of all fragments. Let $area(x)$ denote the area of region x.

Lemma 1. *Let $\mathcal{S}$ be a planar subdivision of U such that each region has at most a constant number of holes and the total boundary length of each region is bounded by a constant c_s. Let $\mathcal{D}$ be a decomposition of U that satisfies properties A.1, A.2, A.3, A.4, and A.5. Let $\mathcal{F}$ be the set of fragments in the overlay of $\mathcal{S}$ and $\mathcal{D}$. Then*

$$\sum_{x \in \mathcal{F}} area(x) \log \frac{1}{area(x)} \leq 2 \sum_{z \in \mathcal{S}} area(z) \log \frac{1}{area(z)} + O(1),$$

where the constant in the O-notation depends on c_a, c_s, and c_n.

4.1 Quadratic Space Solution

We prove Theorem 2(i) in this subsection. Let $\mathcal{S}$ be the given planar subdivision of U such that each region has at most a constant number of holes, and the total boundary length of each region is bounded by a constant. We construct a hierarchical decomposition of U by building a box-decomposition tree (BD-tree) on the vertices of $\mathcal{S}$ [4, 22]. Initially, the BD-tree contains only one node which is the root. Each node represents a cell and the root represents U. We keep expanding the tree until some terminating condition is satisfied. The leaf cells form the desired decomposition of U. We describe how to construct children for a node u below. For convenience, we also use u to denote the cell it represents.

If u contains at most one vertex, then u is a leaf cell. Otherwise, it can be guaranteed inductively that u is a rectangle and we recursively construct two children of u as follows. Split u orthogonally at the midpoint of its longest side to obtain two rectangles v and w. If both v and w contain some vertex, then we make v and w children of u. This operation is called a *midpoint split*. Otherwise, if v or w is empty, then we recursively apply the midpoint splitting rule to the non-empty rectangle, until we obtain a rectangle v' such that v' will be split into two non-empty rectangles. We make v' and $u \setminus v'$ children of u. This operation is called a *shrink*. Note that $u \setminus v'$ is a leaf cell and it contains no vertex.

To construct the tree efficiently, we use a standard trick due to Vaidya [22] for partitioning the points. We store the data points contained in a cell in d separate lists, each sorted by one of the coordinates, that are cross-referenced with each other. Instead of updating the lists after each split, we update them after a sequence of splits is performed, until each of the resulting subsets contains fewer than half the initial number of points. Also, assuming a model of of computation in which exclusive-or, integer floor, powers of 2, and integer logarithm can be computed on point coordinates, the shrink operation can be performed in $O(d)$ time. (For example, see Bern [3]). Straightforward modification of the argument given by Vaidya leads to a construction time of $O(n \log n)$. (We mention that we can achieve the same construction time without using non-algebraic operations by building the *sliding-midpoint tree* [16, 18] instead. It can be shown that Lemma 1 holds for the fragments induced by the leaves of the sliding-midpoint tree. The query algorithm and the rest of the analysis given in this section can also be easily adapted.)

The cells associated with the leaves of the BD-tree satisfy properties A.1, A.2 ($c_a = 2$), A.3, A.4 ($c_n = 2$), and A.5. In addition, the BD-tree has the following property, which is important for our analysis.

Lemma 2. *Let $\mathcal{T}$ be a BD-tree constructed on some point set in U. For any query point q, the number of point-line comparisons needed in traversing $\mathcal{T}$ to locate the leaf cell y containing q is at most* $\log \frac{1}{area(y)} + O(1)$.

Proof. Suppose that we arrive at a node representing a cell u in traversing $\mathcal{T}$ and we need to decide which of its child cells should be visited. Let v and w be its child cells. If v and w are formed by a midpoint split, then one point-line comparison is needed to determine whether v or w contains q. Note that the area of v and w are both half the area of u. If v and w are formed by a shrink, then one is a leaf cell, say v, and it encloses the other child cell, say w. Let i, $1 \leq i \leq 4$, denote the number of sides that the inner box of v does not share with the boundary of u. Thus, it takes i point-line comparisons to decide whether v or w contains the query point q. Note that the area of w is at most $1/2^i$ times the area of u, for $2 \leq i \leq 4$. Therefore, in both cases of midpoint split or shrink, if we spend i point-line comparisons to decide the next child cell to visit and this child cell is not a leaf cell, then the area of this child cell is at most $1/2^i$ times the area of its parent. The area of U is 1. Therefore, the number of point-line comparisons needed to reach the leaf cell y containing q is

at most $\log(1/area(y)) + O(1)$, where the $O(1)$ additive term comes from the last i point-line comparisons, $1 \leq i \leq 4$, spent at the parent of y.

Let y denote any leaf cell of the BD-tree. Observe that y is either a rectangle containing at most one vertex of $\mathcal{S}$, or it is the set-theoretic difference of an outer and inner rectangle, in which case it contains no vertex of $\mathcal{S}$. In each case we partition y into at most four rectangles whose interior contains no vertex of $\mathcal{S}$ (we call them *subcells*). If y is a rectangle and contains no vertex of $\mathcal{S}$ in its interior, then the subcell is y itself. Otherwise if it contains a vertex of $\mathcal{S}$ in its interior, then we split it into two subcells by a vertical line passing through this vertex. Otherwise it must be the set-theoretic difference of an outer and inner rectangle. In this case we partition it into at most four subcells by passing lines coinciding with the vertical sides of the inner rectangle.

Define a *pseudo-fragment* to be a connected component in the intersection of any subcell with a region in $\mathcal{S}$. Clearly each fragment is partitioned into at most four pseudo-fragments. Let z be any subcell. Observe that z contains no vertex of $\mathcal{S}$ and intersects $O(n)$ edges of the subdivision $\mathcal{S}$. Thus z is partitioned into at most $O(n)$ pseudo-fragments. Since the subdivision inside z is so simple, we can locate the pseudo-fragment in z containing the query point by searching an auxiliary structure associated with z.

If there is an edge that intersects two opposite sides of z, then let s be one of the sides intersected. The edges intersecting s divide z into super-fragments which can be linearly ordered along s. (See Figure 1.) Each super-fragment is either a pseudo-fragment by itself, or it is further subdivided by other edges into pseudo-fragments which can be linearly ordered within the super-fragment. (There are at most two super-fragments which are further subdivided; these are shown shaded in the figure.) Thus, we first organize a weighted search tree [17] for the super-fragments with their area as weights. Each super-fragment points to another weighted search tree storing the linearly ordered pseudo-fragments within the super-fragment (the area of the pseudo-fragments are the weights in this second level tree). If there is no edge that intersects two opposite sides of z, we can do the above using any side s of z.

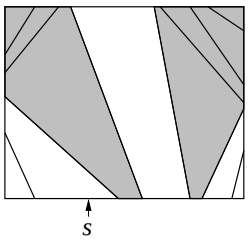

Fig. 1. Super-fragments inside a subcell.

A single query is now answered by first locating the leaf cell in the BD-tree that contains the query point q. Then we determine which of the at most

four subcells associated with this leaf cell contains q. Then we query the auxiliary structure for the subcell to locate the pseudo-fragment containing q. Each pseudo-fragment lies inside a region in $\mathcal{S}$ and hence we have the solution to the query.

We analyze the time to answer a single query as follows. By Lemma 2, the number of point-line comparisons needed to reach a leaf cell y of the BD-tree is $\log(1/\mathit{area}(y)) + O(1)$. It takes $O(1)$ point-line comparisons to find the subcell z containing q. Then we query the auxiliary structure associated with z. It is known [17] that querying a weighted search tree takes at most $\log(K/k) + 2$ comparisons, where K is the total weight of all the items, and k is the weight of the item being searched for. Therefore, querying the auxiliary structure takes $\log(\mathit{area}(z)/\mathit{area}(z')) + \log(\mathit{area}(z')/\mathit{area}(x)) + 4$ point-line comparisons, where z' and x are the super-fragment and pseudo-fragment containing the query point, respectively. Hence, the total number of point-line comparisons is at most $\log(1/\mathit{area}(y)) + \log(\mathit{area}(z)/\mathit{area}(z')) + \log(\mathit{area}(z')/\mathit{area}(x)) + O(1) \leq \log(1/\mathit{area}(x)) + O(1)$.

The probability of the query point lying in a pseudo-fragment x is clearly $\mathit{area}(x)$. Thus, the expected number of point-line comparisons to answer a query is at most

$$\sum_{x \in \mathcal{F}'} \mathit{area}(x) \left(\log \frac{1}{\mathit{area}(x)} + O(1) \right) = \mathit{entropy}(\mathcal{F}') + O(1),$$

where $\mathcal{F}'$ is the set of pseudo-fragments. Since each fragment is partitioned into at most four pseudo-fragments, it is easy to see that $\mathit{entropy}(\mathcal{F}') = \mathit{entropy}(\mathcal{F}) + O(1)$. Therefore, the expected number of point-line comparisons is at most $\mathit{entropy}(\mathcal{F}) + O(1)$, which is at most $2\,\mathit{entropy}(\mathcal{S}) + O(1)$ by Lemma 1.

We analyze the space of the entire data structure. The space needed by the BD-tree is $O(n)$. Since there are $O(1)$ subcells for each leaf cell, and $O(n)$ pseudo-fragments for each subcell, the auxiliary structure at each leaf cell also takes $O(n)$ space. Thus, the total space is $O(n^2)$. As mentioned earlier the BD-tree can be contructed in $O(n \log n)$ time. A weighted search tree of m sorted items can be constructed in $O(m)$ time [17]. Thus, the auxiliary structure at each leaf cell can be constructed in $O(n)$ time which leads to a total preprocessing time of $O(n^2)$. This completes the proof of Theorem 2(i).

4.2 Linear Space Solution

We prove Theorem 2(ii) in this subsection. First, we also build a decomposition tree on the vertices of $\mathcal{S}$, but it is different from the BD-tree in the quadratic space solution. The cell at each node of the tree will be rectangles of bounded aspect ratio. We will classify the leaf cells of the tree into two types, *S-type* and *L-type*. The root of the tree represents the unit square U. Inductively suppose that we are to construct the children of a cell u.

1. If $\mathit{area}(u) < 1/n$, we label u an S-type leaf cell.

2. If $area(u) \geq 1/n$ and u intersects no edge of $\mathcal{S}$, then u must be completely contained in some region of $\mathcal{S}$; we store the name of this region with u. In addition, we label u an L-type leaf.
3. If $area(u) \geq 1/n$, u intersects some edge of $\mathcal{S}$, and each region in $u \cap \mathcal{S}$ has area less than $1/n$, then we label u an S-type leaf cell
4. Otherwise, $area(u) \geq 1/n$, u intersects some edge of $\mathcal{S}$, and some region in $u \cap \mathcal{S}$ has area at least $1/n$. We split u using a midpoint split into two cells v and w, and make them children of u. Then we recursively construct the descendants of v and w.

We denote this decomposition tree by $\mathcal{T}(\mathcal{S})$. The cells associated with the leaves of the tree satisfy properties A.1, A.2 ($c_a = 2$), A.3, A.4 ($c_n = 2$), and A.5. We also have the following result which is analogous to Lemma 2.

Lemma 3. *For any query point q, the number of point-line comparisons needed in traversing $\mathcal{T}(\mathcal{S})$ to locate the leaf cell y containing q is at most* $\log \frac{1}{area(y)} + O(1)$.

The final step of preprocessing is to construct the worst-case planar point location data structure for $\mathcal{S}$ invented by Adamy and Seidel [1]. This data structure uses $O(n)$ space and can be constructed in $O(n \log n)$ time. A point location query can be answered using $\log n + 2\sqrt{\log n} + O(\log^{1/4} n)$ point-line comparisons.

Given a query point q, we first descend $\mathcal{T}(\mathcal{S})$ to find the leaf cell x containing q. If x is an L-type leaf cell, then we report the region of $\mathcal{S}$ containing x and terminate. Otherwise, x is an S-type leaf cell, and we simply resort to Adamy and Seidel's data structure to answer the point location query.

Space analysis Each leaf cell of $\mathcal{T}(\mathcal{S})$ has area at least $1/2n$ and they partition the unit square U. This implies that $\mathcal{T}(\mathcal{S})$ has $O(n)$ leaves and hence $O(n)$ nodes. Adamy and Seidel's data structure use $O(n)$ space. Thus, the total space needed is $O(n)$.

Query time analysis Recall that a *fragment* is a connected component of the intersection of the leaf cells of $\mathcal{T}(\mathcal{S})$ and $\mathcal{S}$. Let $\mathcal{F}$ denote the set of all fragments. By Lemma 1, we have $entropy(\mathcal{F}) \leq 2\, entropy(\mathcal{S}) + O(1)$. In the following, we show that the expected number of point-line comparisons to answer a query is $(2 + O(1/\sqrt{\log n}))\, entropy(\mathcal{F})$ and so the desired bound of $(4 + O(1/\sqrt{\log n}))\, entropy(\mathcal{S}) + O(1)$ follows.

We call a fragment *large* if its area is at least $1/n$, and *small* otherwise. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ denote the set of large and small fragments, respectively. A large fragments is exactly an L-type leaf cell and vice versa. Small fragments lie inside S-type leaves.

We analyze the time to locate a query point q. Suppose that q lies inside a fragment $x \in \mathcal{F}$. If x is large, then x is an L-type leaf, and the number of comparisons needed to reach x is $\log(1/area(x))$ by Lemma 3. If x is small, then the query procedure will first locate the leaf cell y containing x and

then query the worst-case data structure. By Lemma 3, the number of point-line comparisons needed to reach y is $\log(1/\mathit{area}(y))$. Since $\mathit{area}(y) \geq \mathit{area}(x)$, $\log(1/\mathit{area}(y)) \leq \log(1/\mathit{area}(x))$. Adding the number of point-line comparisons needed for querying the worst-case data structure, the total number of comparisons is at most $\log(1/\mathit{area}(x)) + \log n + 2\sqrt{\log n} + O(\log^{1/4} n)$. Since x is small, $\mathit{area}(x) < 1/n$ which implies that $\log(1/\mathit{area}(x)) > \log n$. So the total number of comparisons is at most $(2 + O(1/\sqrt{\log n}))\ \log(1/\mathit{area}(x))$.

The probability that q lies in a fragment x is clearly $\mathit{area}(x)$. Thus, the expected number of point-line comparisons to answer a query is bounded by

$$\sum_{x \in \mathcal{F}_1} \mathit{area}(x) \log \frac{1}{\mathit{area}(x)} + \sum_{x \in \mathcal{F}_2} \left(2 + O\left(\frac{1}{\sqrt{\log n}}\right)\right)\ \mathit{area}(x) \log \frac{1}{\mathit{area}(x)}$$
$$\leq \left(2 + O\left(\frac{1}{\sqrt{\log n}}\right)\right)\ \mathit{entropy}(\mathcal{F}).$$

Preprocessing time When we construct the child cells of a cell u during preprocessing, the most time consuming part of the construction is to determine whether each region in $u \cap \mathcal{S}$ has area less than $1/n$. We describe a method to carry out this computation efficiently.

Define L_u to be the set of regions of area at least $1/n$ in $u \cap \mathcal{S}$. The observation is that any region in L_v at a child v of u must be contained in some region in L_u. Therefore, our strategy is to compute L_u for each node u inductively.

Let r be the root of $\mathcal{T}(\mathcal{S})$ and so $r \cap \mathcal{S} = \mathcal{S}$. We simply traverse $\mathcal{S}$ in $O(n)$ time to collect all regions of area at least $1/n$ in L_r. Inductively, let v be the child of u and we are to compute L_v. For each region z in L_u, we claim that we can compute the intersection $z \cap v$ in time proportional to the size of z. (Note that $z \cap v$ may consist of several connected components.)

This can be done by clipping z with four halfplanes successively. We describe the first clipping as follows. Let ℓ be the bounding line of a halfplane. For convenience, denote the size of z by $|z|$. First, compute the $O(|z|)$ intersections between ℓ and the boundary of z in $O(|z|)$ time by brute-force. Second, apply Jordan sorting to sort these intersections in order of their appearance on ℓ. This can also be done in $O(|z|)$ time [9]. Third, start a clockwise traversal from some vertex of z within the halfplane. If we come to an intersection on ℓ, then we use the sorted list of intersections to jump to the next intersection along ℓ. The traversal stops when we come back to a visited vertex, and we have traversed the boundary of one connected component of the clipped z. Then we repeat the traversal from an unvisited vertex of z within the halfplane and so on until no such vertex is left. This traverses all connected components in the clipped z. Since each vertex of z and each intersection on ℓ is visited at most once, this also takes $O(|z|)$ time. This completes the first clipping. Each subsequent clipping is done the same way. Since we have added at most $O(|z|)$ new vertices after a clipping and there are four clippings, we conclude that each clipping takes $O(|z|)$ time.

After obtaining $z \cap v$, we can then retain only the components in $z \cap v$ that has area at least $1/n$ and include them in L_v. Repeating this for each region in L_u yields L_v. The total time needed is then proportional to the sum of sizes of regions in L_u. Let E_i denote the set of edges on the boundaries of regions in L_u for all nodes u at level i of $\mathcal{T}(\mathcal{S})$. We claim that for any level i, the number of edges in E_i is $O(n)$. Since $\mathcal{T}(\mathcal{S})$ is constructed using midpoint split and each leaf cell has area at least $1/2n$, the number of levels in the tree is $O(\log n)$, and it follows that the preprocessing time is $O(n \log n)$.

To see that there are $O(n)$ edges in E_i, note that there are two categories of edges in E_i. The first category consists of edges that lie on the sides of a cell, and the second category consists of edges that lie on edges of $\mathcal{S}$. Observe that edges in the first category can be charged against edges in the second category, so we only need to show that the number of edges in the second category is $O(n)$. The second category can be divided into two groups. The first group consists of edges that are incident to a vertex of $\mathcal{S}$ inside a cell at level i, and the second group consists of the remaining edges. It is clear that the number of edges of the first group can be no more than the total degree of the vertices of $\mathcal{S}$, which is $O(n)$. To count the number of edges of the second group, first observe that the number of regions with area at least $1/n$ in cells at level i is at most n. Second, each such region can have at most four boundary edges that are not incident to a vertex of $\mathcal{S}$ inside a cell at level i. Thus the number of edges of the second group is at most $4n$. Hence the total number of edges in E_i is $O(n)$, which is the desired claim.

5 General Well-behaved Distributions

Given a planar subdivision $\mathcal{S}$ and a well-behaved distribution (P, Q), our main idea is that we can organize our point location data structure (the quadratic space version or linear space version) in Theorem 2 in the probability space. Then given a query point q, we locate the region z' in $f_{PQ}(\mathcal{S})$ containing $f_{PQ}(q)$ and then return $f_{PQ}^{-1}(z')$.

For the above strategy to work, there are several requirements. First, the x and y coordinates of $f_{PQ}(q)$ should be uniformly and independently chosen from $[0, 1]$. Second, $f_{PQ}(\mathcal{S})$ is a planar subdivision, and each region has at most a constant of holes and the perimeter of each region is bounded by a constant. Third, if we were to apply Theorem 2 directly, we would require that $f_{PQ}(\mathcal{S})$ be a polygonal planar subdivision, but this is usually untrue. Instead, we will map back and forth between the geometric and probability spaces using f_{PQ} and f_{PQ}^{-1} to construct and query our data structure in the probability space. We describe below how these requirements are satisfied.

Let x'_q be the x-coordinate of $f_{PQ}(q)$. The probability $prob(x'_q \leq x')$ that x'_q is less than or equal to x' for some $0 \leq x' \leq 1$ is equal to $prob(P(x_q) \leq x')$, where x_q is the x-coordinate of q. But $prob(P(x_q) \leq x') = prob(x_q \leq P^{-1}(x'))$ which is equal to $P(P^{-1}(x')) = x'$ by definition of P. So $prob(x'_q \leq x') = x'$ and

x'_q is uniformly picked from $[0,1]$. Similarly, we can show that the y-coordinate of $f_{PQ}(q)$ is uniformly picked in $[0,1]$.

Given two real numbers $\alpha, \beta \in I$, since P is strictly increasing, $\alpha \leq \beta$ iff $P(\alpha) \leq P(\beta)$ and equality holds exactly when $\alpha = \beta$. Therefore, the left-right ordering of points by x-coordinate in $I \times J$ is preserved in U after the mapping. A similar reasoning about Q shows that the above-below ordering of points by y-coordinate is also preserved. Also, a point p is on a line segment ξ in $I \times J$ iff $f_{PQ}(p)$ is on $f_{PQ}(\xi)$ in U. Thus, incidence relations in $\mathcal{S}$ are preserved in $f_{PQ}(\mathcal{S})$ and hence $f_{PQ}(\mathcal{S})$ is a planar subdivision of U. In Theorem 1, it is already assumed that each region in $f_{PQ}(\mathcal{S})$ has at most a constant number of holes, and the perimeter of each region is bounded by a constant.

We now deal with issue that $f_{PQ}(\mathcal{S})$ may not be a polygonal planar subdivision. In constructing our data structure in U, we need to perform two primitives. The first primitive is to determine whether a vertex lies above, below, to the left, or to the right of an orthogonal line. (This is needed in shrinking.) The second primitive is to compute the intersection between an (possibly curvy) edge ξ' and an orthogonal line segment. (This is needed in midpoint split and shrinking.) Since ordering and incidence relations are preserved, these two primitives can be provided by first going back to the geometric space, perform the computation, and map the result back to the probability space. Note that a vertical/horizontal line segment is always mapped to a vertical/horizontal line segment and vice versa. Also, for the second primitive, we would be intersecting $f_{PQ}^{-1}(\xi')$, which must be a line segment, with an orthogonal line segment in the geometric space. This can be done in constant time in the geometric space, and then we map the result using f_{PQ} to the intersection desired in the probability space.

To see the correctness of our approach to answer a query, first observe that, by continuity of P and Q, a closed curve in $I \times J$ is mapped by f_{PQ} to a closed curve in U. Since ordering and incidence relations are preserved, a point p lies inside/on/outside a closed curve ξ in $I \times J$ iff $f_{PQ}(p)$ lies inside/on/outside $f_{PQ}(\xi)$. Thus, given a query point q in the geometric space and a region z' in $f_{PQ}(\mathcal{S})$ containing $f_{PQ}(q)$, $f_{PQ}^{-1}(z')$ is the region in $\mathcal{S}$ containing q. In searching our data structure, we need to tell whether the query point $f_{PQ}(q)$ in U lies above, below, to the left, or to the right of an orthogonal line or a curvy edge ξ'. We have seen that this can be done for an orthogonal line. For a curvy edge ξ', we simply return the relation between q and $f_{PQ}^{-1}(\xi')$, which must be a line segment, in the geometric space. This establishes the correctness of our approach to answer a query.

In all, Theorem 1 holds assuming that each evaluation of the functions P, Q, P^{-1}, and Q^{-1} takes constant time.

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 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`.

[3] M. Bern, D. Eppstein, and S.-H. Teng. Parallel construction of quadtrees and quality triangulations. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes in Computer Science*, pages 188–199. Springer-Verlag, 1993.

[4] P. B. Callahan and S. R. Kosaraju. A decomposition of multi-dimensional point-sets with applications to k-nearest-neighbors and n-body potential fields. In *Proc. 24th Ann. ACM Sympos. Theory Comput.*, pages 546–556, 1992.

[5] L. Devroye, E. P. Mücke, and B. Zhu. A note on point location in Delaunay triangulations of random points. *Algorithmica*, 22:477–482, 1998.

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

[7] M. Edahiro, I. Kokubo, and T. Asano. A new point-location algorithm and its practical efficiency — Comparison with existing algorithms. *ACM Trans. Graph.*, 3(2):86–109, 1984.

[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] K.Y. Fung, T.M. Nicholl, R.E. Tarjan, and C.J. Van Wyk. Simplified linear-time jordan sorting and polygon clipping. *Inform. Process. Lett.*, 35:85–92, 1990.

[10] 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.

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

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

[13] D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1971.

[14] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 1973.

[15] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9:615–627, 1980.

[16] S. Maneewongvatana and D. M. Mount. Analysis of approximate nearest neighbor searching with clustered point sets. In *ALENEX*, 1999.

[17] K. Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5:287–295, 1975.

[18] D. M. Mount and S. Arya. ANN: A library for approximate nearest neighbor searching. 2nd Annual CGC Workshop on Computational Geometry, URL: `http://www.cs.umd.edu/~mount/ANN`, 1997.

[19] E. P. Mücke, I. Saias, and B. Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 274–283, 1996.

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

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

[22] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.*, 4:101–115, 1989.