DISTRIBUTION-SENSITIVE POINT LOCATION IN CONVEX SUBDIVISIONS[*]

Sébastien Collette
Université Libre de Bruxelles

John Iacono
Polytechnic University

Stefan Langerman
Université Libre de Bruxelles

Pat Morin
Carleton University

ABSTRACT. A data structure is presented for point location in convex planar subdivisions when the distribution of queries is known in advance. The data structure has an expected query time that is within a constant factor of optimal.

Keywords: Planar point location, Entropy

1 Introduction

The planar point location problem is one of the classic problems in computational geometry. Given a planar subdivision G,[1] the planar point location problems asks us to construct a data structure so that, for any query point p, we can quickly determine which face of G contains p.[2] The history of the planar point location problem parallels, in many ways, the study of binary search trees.

After a few initial attempts [7, 14, 17], asymptotically optimal (and quite different) linear-space $O(\log n)$ query time solutions to the planar point location problem were obtained by Kirkpatrick [12], Sarnak and Tarjan [20], and Edelsbrunner *et al* [8] in the mid 1980s. These results were based on hierarchical simplification, data structural persistence, and fractional cascading, respectively. All three of these techniques have subsequently found many other applications. An elegant randomized solution, combining aspects of all three previous solutions, was later given by Mulmuley [16]. Preparata [18] gives a comprehensive survey of the results of this era.

In the 1990s, several authors became interested in determining the exact constants achievable in the query time. Goodrich *et al* [9] gave a linear size data structure that, for any query, requires at most $2\log n + o(\log n)$ point-line comparisons and conjectured that this query time was optimal for linear-space data structures. Here and throughout, logarithms are implicitly base 2 unless otherwise specified. The following year, Adamy and Seidel [1] gave a linear space data structure that answers queries using $\log n + 2\sqrt{\log n} + O(\log\log n)$ point-line comparisons and showed that this result is optimal up to the third term.

[*] The research presented in this article took place while the fourth author was a visiting researcher at the Université Libre de Bruxelles, supported by a grant from FNRS. The researchers are partially supported by NSERC and FNRS.

[1] A *planar subdivision* is a partitioning of the plane into points (called *vertices*), open line segments (call *edges*), and open polygons (called *faces*).

[2] In the degenerate case where p is vertex or contained in an edge of G any face incident on that vertex/edge may be returned as an answer.

Still not done with the problem, several authors considered the point location problem under various assumptions about the query distribution. All these solutions compare the expected query time to the *entropy bound*; in a planar subdivision with f faces, if the query point p is chosen from a probability measure over $\mathbb{R}^2$ such that p_i is the probability that p is contained in face i of G, then no algorithm that makes only binary decisions can answer queries using an expected number of decisions that is fewer than

$$H(p_1, \ldots, p_f) = \sum_{i=1}^{f} p_i \log(1/p_i) \ . \tag{1}$$

In the previous results on planar point location, none of the query times are affected significantly by the structure of G; they hold for arbitrary planar subdivisions. However, when studying point location under a distribution assumption the problem becomes more complicated and the results become more specific. A *convex subdivision* is a planar subdivision whose faces are all convex polygons, except the outer face, which is the complement of a convex polygon. A *triangulation* is a convex subdivision in which each face has at most 3 edges on its boundary. Note that, if every face of G has a constant number of sides, then G can be augmented, by the addition of extra edges, so that it is a triangulation without increasing (1) by more than a constant. Thus, in the following we will simply refer to results about triangulations where it is understood that these also imply the same result for planar subdivisions with faces of constant size.

Arya *et al* [2] gave two results for the case where the query point p is chosen from a known distribution where the x and y coordinates of p are chosen independently and G is a convex subdivision. They give a linear space data structure for which the expected number of point-line comparisons is at most $4H + O(1)$ and a quadratic space data structure for which the expected number of point-line comparisons is at most $2H + O(1)$. The assumption about the independence of the x and y coordinates of p is crucial to the these results.

For arbitrary distributions that are known in advance, several results exist. Iacono [10, 11] showed that, for triangulations, a simple variant of Kirkpatrick's original point location structure gives a linear space, $O(H + 1)$ expected query time data structure. A result by Arya *et al* [3] gives a data structure for triangulations that uses $H + O(H^{2/3} + 1)$ expected number of comparisons per query and $O(n \log n)$ space. The space requirement of this latter data structure was later reduced, by the same authors, to $O(n \log^* n)$ [4]. Finally, the same three authors [5] showed that a variant of Mulmuley's randomized algorithm gives, for triangulations, a simple $O(H + 1)$ expected query time, linear space data structure.

The above collection of results suggest that point location, and even distribution-sensitive point location, is a well-studied and well-understood problem, with solutions that are optimal up to constant factors. However, in the above results there is a glaring omission. Given a convex polygon P, a folklore $O(\log n)$ time algorithm exists to test if a query point p is contained in P and this algorithm is optimal, in the worst case [19]. Testing if $p \in P$ is a special case of point location in a convex subdivision in which the subdivision has only 2 faces. Thus, we might expect that, if p is drawn according to some distribution over $\mathbb{R}^2$, it may be possible to do better in many cases. How much better? It is certainly not possible to achieve the entropy bound in all cases since, when $f = 2$ the entropy bound is at most 1.

We begin our investigation of distribution-sensitive point location with the fundamental problem of testing if a query point p, drawn from an arbitrary distribution D over $\mathbb{R}^2$, is contained in a convex polygon P. We describe a hierarchical triangulation T of $\mathbb{R}^2$ that we use to simultaneously achieve two objectives:

1. T is used with a query algorithm to check if a point p is contained in P, and
2. T is used to give a lower bound on the expected cost of any linear decision tree that tests if a point p selected according to D is contained in P.

The lower bound in Point 2 matches, to within a constant factor, the expected query time of the algorithm in Point 1. Thus, among algorithms that can be expressed as linear decision trees, our algorithm is optimal. Our result is the first result to give *any* lower bound on the expected complexity of *any* point location problem that exceeds the entropy bound. Proving the lower bound is by far the hardest part of our result.

As an easy consequence of the above result we obtain a data structure for point location in convex subdivisions. The expected query time of the resulting algorithm is optimal in the linear decision tree model of computation. Note that all known algorithms for planar point location can be described in the linear decision tree model of computation. This data structure for point location in convex subdivisions where the query point is drawn according to an arbitrary distribution is the most general result known about planar point location and implies, to within constant factors, all of the results discussed above.

The remainder of this paper is organized as follows: Section 2 presents definitions and notations used throughout the paper. Section 3 discusses algorithms and lower bounds for point location in convex polygons. Section 4 presents algorithms and lower bounds for point location in convex subdivisions. Finally, Section 5 concludes with a discussion and points out directions for further research.

2 Preliminaries

In this section we give definitions, notations, and background required in subsequent sections.

Triangles. For the purposes of this paper, a *triangle* is the common intersection of at most 3 halfplanes. This includes triangles with infinite area and triangles having 0, 1, 2, or 3, vertices.

Classification Problems and Classification Trees. A *classification problem* over a domain $\mathcal{D}$ is a function $\mathcal{P} : \mathcal{D} \mapsto \{0, \ldots, k-1\}$. The special case in which $k = 2$ is called a *decision problem*. A d-ary *classification tree* is a full d-ary tree[3] in which each internal node v is labelled with a function $P_v : \mathcal{D} \mapsto \{0, \ldots., d-1\}$ and for which each leaf ℓ is labelled with a value $d(\ell) \in \{0, \ldots, k-1\}$. The *search path* of an input p in a classification tree T starts at the root of T and, at each internal node v, evaluates $i = P_v(p)$ and proceeds to the ith child of v. We denote by $T(p)$ the label of the final (leaf) node in the search path for p. We say that the classification tree T *solves* the classification problem $\mathcal{P}$ over the domain $\mathcal{D}$ if, for every $p \in \mathcal{D}$, $\mathcal{P}(p) = T(p)$.

Unless specifically mentioned otherwise, classification trees are binary classification trees. For a node v in a (binary) classification tree, its left child, right child, and parent are denoted by $\mathrm{left}(v)$, $\mathrm{right}(v)$ and $\mathrm{parent}(v)$, respectively.

Probability. For a distribution D and an event X, we denote by $D_{|X}$ the distribution D conditioned on X. That is, the distribution where the probability of an event Y is $\Pr(Y|X) = \Pr(Y \cap X)/\Pr(X)$.

[3] A full d-ary tree is a rooted ordered tree in which each non-leaf node has exactly d children.

The probability measures used in this paper are usually defined over $\mathbb{R}^2$. We make no assumptions about how these measures are represented, but we assume that an algorithm can perform the following two operations in constant time:

1. given an open triangle Δ, compute $\Pr(\Delta)$, and
2. given an open triangle Δ and a point t at the intersection of two of Δ's supporting lines, compute a line ℓ that contains t and that partitions Δ into two open triangles Δ_0 and Δ_1 such that $\Pr(\Delta_0) \leq \Pr(\Delta)/2$ and $\Pr(\Delta_1) \leq \Pr(\Delta)/2$.

Requirement 2 is used only for convenience in describing our data structure. In Section 5 we show that Requirement 2 is not really necessary and that Requirement 1 is sufficient to implement our data structure.

For a classification tree T that solves a problem $P : \mathcal{D} \mapsto \{0, \ldots, k-1\}$ and a probability measure D over $\mathcal{D}$, the *expected search time* of T is the expected length of the search path for p when p is drawn at random from $\mathcal{D}$ according to D. Note that, for each leaf ℓ of T there is a maximal subset $r(\ell) \subseteq \mathcal{D}$ such that the search path for any $p \in r(\ell)$ ends at ℓ. Thus, the expected search time of T (under distribution D) can be written as

$$\mu_D(T) = \sum_{\ell \in L(T)} \Pr(r(\ell)) \times \operatorname{depth}(\ell) \enspace ,$$

where $L(T)$ denotes the leaves of T and $\operatorname{depth}(\ell)$ denotes the length of the path from the root of T to ℓ.

The following theorem, which is a restatement of (half of) Shannon's Fundamental Theorem for a Noiseless Channel [21, Theorem 9], is what all existing results on distribution-sensitive planar point location use to establish their optimality:

Theorem 9. *Let $\mathcal{P} : \mathcal{D} \mapsto \{0, \ldots, k-1\}$ be a classification problem and let $p \in \mathcal{D}$ be selected from a distibution D such that $\Pr\{\mathcal{P}(x) = i\} = p_i$, for $0 \leq i < k$. Then, any d-ary classification tree T that solves $\mathcal{P}$ has*

$$\mu_D(T) \geq \sum_{i=0}^{k-1} p_i \log_d(1/p_i) \enspace . \tag{2}$$

Theorem 9 is applied to the point location problem by treating point location as the problem of classifying the query point p based on which face of G contains it. In this way, we obtain the lower bound in (1).

3 Point In Convex Polygon Testing

Let P be a convex n-gon whose boundary is denoted by ∂P and consider a probability measure D over $\mathbb{R}^2$. For technical reasons, we use the convention that P does not contain its boundary so that $p \in \partial P$ implies $p \notin P$. In this section we are interested in preprocessing P and D so that we can quickly solve the decision problem of testing whether a query point p, drawn according to D, is contained in P.

In particular, we are interested in algorithms that can be described as *linear decision trees*. These are decision trees such that each internal node v contains a linear inequality $P_v(x,y) = [ax + by \geq c]$.[4] We require that, for every $p \in \mathbb{R}^2$ the leaf reached in the search path for p is labelled with a 1 if and only if $p \in P$. Geometrically, each internal node of T is labelled with a directed line and the decision to go to the left or right child depends on whether p is to the left or right (or on) this line.

Our exposition is broken up into three sections. We begin by describing a data structure (in fact, a decision tree) that tests if query point p is contained in a convex polygon P. Next, we give an (easy) analysis of the expected search time of this data structure. Finally, we give a (more difficult) proof that this expected search time is optimal.

3.1 Triangle Trees

At a high level, our data structure works by creating a sequence of successively finer approximations $A_0, \dots, A_k$ to ∂P. Each approximation A_i consists of two convex polygons; an *outer approximation* that contains P and an *inner approximation* that is contained in P.

Each approximation A_i is completely defined by a set S_i of points on ∂P. The inner approximation is simply the convex hull of S_i. The outer approximation has an edge tangent to P at each of the points of S_i. More precisely, for each point $x \in S_i$ there is an edge e of the outer approximation that contains x. If x is in the interior of an edge of P then e is contained in the same line that contains that edge. Otherwise (x is a vertex of P) e is supported by the line containing the edge incident to x that precedes x in counterclockwise order. We ensure that successive approximations have a containment relationship, i.e., $A_i \supseteq A_{i+1}$, by choosing our boundary points so that $S_i \subseteq S_{i+1}$.

Next we define the sets $S_0, \dots, S_k$ that define our approximations. The set S_0 is empty, and we use the convention that the outer approximation in this case is the entire plane and the inner approximation is the empty set. The set S_1 consists of any two points, x and y on ∂P such that $\Pr(h_1) \leq 1/2$ and $\Pr(h_2) \leq 1/2$, for each of the two open halfspaces h_1 and h_2 bounded by the line containing x and y. The existence of x and y is guaranteed, for example, by the planar Ham Sandwich Theorem [6].

We now show how, for $i \geq 1$, to obtain the set S_{i+1} from the set S_i. Let $p_0, \dots, p_{m_i - 1}$ be the points in S_i as they occur in conterclockwise order around the boundary of P. The approximation A_i thus consists of m_i open triangles $\Delta_0, \dots, \Delta_{m_i-1}$ where Δ_j is the intersections of the following halfspaces:

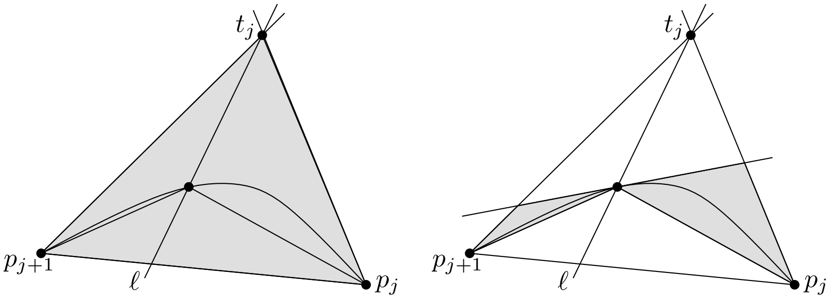
1. the open halfspace to the right of the directed line through p_j and p_{j+1},
2. the open halfspace bounded by the tangent to P at p_j and that contains p_{j+1}, and
3. the open halfspace bounded by the tangent to P at p_{j+1} and that contains p_j.

Let t_j be the intersection point of the two lines tangent to P at p_j and p_{j+1}.[5] Refer to Figure 1. For each triangle Δ_j that is not completely contained in P we add a new point to S_{i+1} as follows: we subdivide Δ_j into two open triangles $\Delta_{j,0}$ and $\Delta_{j,1}$ by a line ℓ through t_j and such that

$$\Pr(\Delta_{j,b}) \leq \Pr(\Delta_j)/2 \ ,$$

[4] Here, and throughout, we use Iverson's notation where $[X] = 0$ if X is false and $[X] = 1$ if X is true [13].

[5] Throughout this discussion, subscripts are implicitly taken modulo m_i.



(a) (b)

Figure 1: Starting with Δ_j we (a) subdivide Δ_j into two triangles $\Delta_{j,0}$ and $\Delta_{j,1}$ with a line through t_j and then (b) split Δ_j into its two children.

for $b \in \{0,1\}$. We then select a new point to add to S_{i+1} at the intersection of ℓ and ∂P that occurs in Δ_j. Note that the next level of approximation A_{i+1} now contains two triangles that are contained in Δ_j. We call these two triangles the *children* of Δ_j and we say that this operation *splits* Δ_j into these two triangles.

The entire process terminates at the first value of k for which A_k is completely contained in P. The approximations $A_0, \ldots, A_k$ are stored as a binary tree $T = T(P,D)$ that we call a *triangle tree* for P and D. Each node v of T at level i in T corresponds to an open triangle $\Delta(v)$ in approximation A_i and the two children of v correspond to the two open triangles into which $\Delta(v)$ is split. See Figure 2. A crucial property of this construction guaranteed by the splitting process is that, for any node v at level distance i from the root of T, $\Pr(\Delta(v)) \leq 1/2^i$.

To use the tree T to determine if a point p is contained in P we proceed top-down, starting at the root. For a point p contained in $\Delta(v)$ one of two things can happen: (1) p is contained in one of the two open triangles $\Delta(\mathrm{left}(v))$ or $\Delta(\mathrm{right}(v))$ in which case we recurse on the right or left child of v, respectively, or (2) we can determine in constant time if $p \in P$.

3.2 Analysis of the Triangle Tree

Let $T = T(P,D)$ be a triangle tree for a convex n-gon P and a distribution D. Define, for each node v of T,

$$\Xi(v) = \Delta(v) \setminus (\Delta(\mathrm{left}(v)) \cup \Delta(\mathrm{right}(v)))$$

and define $\Pr(v) = \Pr(\Xi(v))$. Notice that the search for a point p terminates at v precisely when $p \in \Xi(v)$. Thus, $\Pr(v)$ is the probability that a search terminates at node v. For a set V of nodes in T we use the notation $\Pr(V) = \sum_{v \in V} \Pr(v)$ to denote the probability that the search path ends at some node in V.

Theorem 1. *A triangle tree T contains $O(n)$ nodes and can be constructed in $O(n)$ time. Using the triangle tree, the expected number of linear inequalities required to check if $p \in P$ for a point p drawn from D is at most*

$$\mu_D(T) \leq O(1) + O(1) \times \sum_{v \in T} \Pr(v) \log(1/\Pr(v)) \ .$$

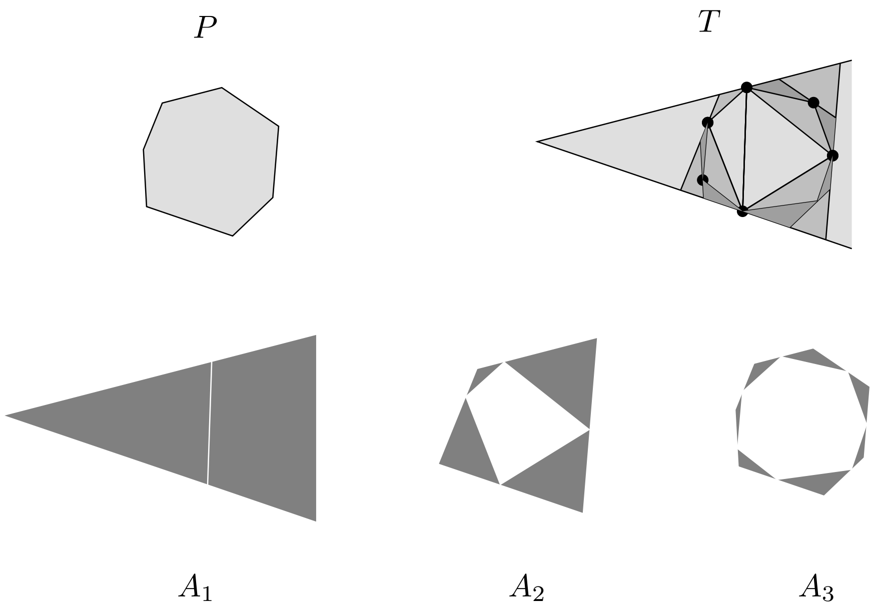


Figure 2: A convex polygon P, the triangles of the triangle tree T, and the sequence of approximations $A_1, \dots, A_3$ that approximate ∂P.

Proof. For each $1 \leq i \leq n$, let e_i be the ith edge of P, defined so that it does not include its first (clockwise) endpoint but does include its second (counterclockwise) endpoint. Observe that if, during some iteration of the algorithm for constructing T, we select a point $x \in e_i$ then e_i moves to the boundary of the outer approximation and no point of e_i will ever be selected again. This implies that T has $O(n)$ nodes.

To obtain an $O(n)$ time algorithm to construct T we apply a trick used by Mehlhorn [15] in the construction of biased binary search trees. Splitting a triangle Δ_j involves finding the line ℓ and computing the intersection of ℓ with $\partial P \cap \Delta_j$. The former operation takes, by assumption, $O(1)$ time. The latter operation, by using two exponential searches in parallel, can be done in $O(\log(\min\{m-k, k\}))$ time, where m is the number of edges of P that intersect Δ_j and k is the rank in this set of the edge that intersects ℓ. In this way, the overall running time of the construction algorithm is given by the recurrence

$$T(n) \leq T(n-k) + T(k) + O(\log(\min\{n-k, k\}))$$

which resolves to $O(n)$.

The expected running time of the query algorithm on T is follows immediately from the fact that, for any node v at a distance of i from the root of T, $\Pr(v) \leq \Pr(\Delta(v)) \leq 1/2^i$. □

3.3 Optimality of the Triangle Tree

Next we will show that the performance bound given by Theorem 1 is optimal. More precisely, we show that there is no linear decision tree whose expected search time (on distribution D) is asymptotically better than that of the triangle tree. The key ingredient in our argument is the following lemma:

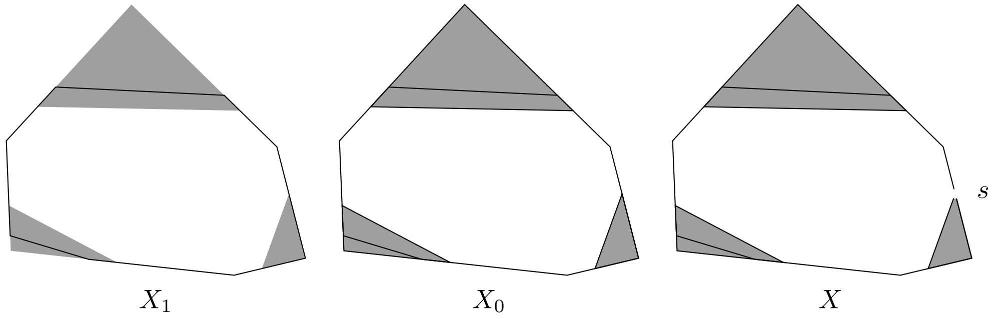


Figure 3: The point sets X_1, X_0 and X used in the proof of Lemma 1.

Lemma 1. *Let V be a subset of the vertices of the triangle tree T such that no vertex in V is the descendent of any other vertex in V. Let $R = \bigcup_{v \in V} \Xi(v)$. Then, for any linear decision tree T',*

$$\mu_{D_{|R}}(T) \geq \frac{1}{4} \sum_{v \in V} \Pr(\Xi(v) \mid R) \log(1/\Pr(\Xi(v) \mid R)) \ .$$

Proof. We define a new model of computation, show that a lower bound in this new model implies a lower bound in the linear decision tree model and then prove the lower bound for the new model. Refer to Figure 3. Let $X_1 = \partial P \cup \bigcup_{v \in V} \Delta(v)$, let $X_0 = X_1 \cup \partial X_1$ be the closure of X_1, and let $X = X_0 \setminus \{s\}$ for some vertex s of some triangle $\Delta(v)$ with $v \in V$ and $s \in \partial P$.[6] A *chord* of X is a closed line segment with both endpoints on the boundary of X and whose interior is contained in X. Note that a chord may have length zero. If c is a chord of X then $X \setminus c$ has at most 3 connected components X_1^c, X_2^c, and X_3^c.

A *chord tree* is a 4-ary decision tree that solves the problem of testing whether a point $p \in X_0$ is contained in P. Each internal node v of a chord tree is labelled with a chord $c(v)$ of X and the four children of v correspond to the options, $p \in X_1^{c(v)}$, $p \in X_2^{c(v)}$, $p \in X_3^{c(v)}$, and $p \in c(v)$. Observe that, because X is bounded by two convex chains, any line intersects X in at most 2 chords. This immediately implies the following relationship between chord trees and linear decision trees.

Claim 1. *Let T be any linear decision tree that solves the problem of testing if $p \in P$. Then there exists a chord tree T' such that $\mu_{D_{|R}}(T') \leq 2\mu_{D_{|R}}(T)$.*

Now we need only prove a lower bound on $\mu_{D_{|R}}(T')$ for any chord tree T'. For two nodes $u, v \in V$, $u \neq v$, let p be a point in $\Delta(u)$ and let q be a point in $\Delta(v)$. We claim that the search path for p in T' and the search path for q in T' must end at different leaves of T'. Assume, for the sake of contradiction, that this is not the case and that both search paths terminate at the leaf ℓ. We will show that in this case T' does not solve the problem of testing if any point $p \in X$ is contained in P and thus contradicts the assumption that T' is a chord tree. First observe that, because the search paths for p and q both end at ℓ, there exists a closed curve ρ in X with one endpoint at p and the other endpoint at q and for which all points of ρ have a search path that ends at ℓ. Indeed, the shortest path, in X, from p to q has this property.

If either p or q is contained in P then we immediately obtain the desired contradiction since ρ

[6] The removal of s ensures that X is *simply connected*. That is, for any two points $p, q \in X$ any path, in X, from p to q can be continuously deformed into any other path, in X, from p to q while remaining in X throughout the deformation.

must contain one of the vertices r of (e.g.) $\Delta(u)$, a point that is not in P (recall that P does not contain ∂P). Otherwise, both p and q are not in P. Then the path ρ still contains one of the vertices r of $\Delta(u)$. Furthermore, none of the vertices on the search path in T' for p is associated with a chord that contains r (otherwise p and q would not have the same search path in T'). Therefore, there is a disk D, centered at r and having positive radius, for which the search path in T' for any point in $D \cap X$ terminates at ℓ. Again, we obtain a contradiction since $D \cap X$ contains points in the interior of P.

We have established that, for any two points p and q contained in different triangles $\Delta(u)$ and $\Delta(v)$ the search paths of p and q terminate at different leaves of T'. This implies that, by labelling the leaves of T' appropriately, we obtain a 4-ary classification tree that determines, for any $p \in \bigcup_{v \in V} \Delta(v)$ the node $v \in V$ such $\Delta(v)$ contains p. Therefore, by Theorem 9,

$$\begin{aligned}\mu_{D_{|R}}(T') &\geq \sum_{v \in V} \Pr(\Xi(v) \mid R) \log_4(1/\Pr(\Xi(v) \mid R)) \\ &= \frac{1}{2} \sum_{v \in V} \Pr(\Xi(v) \mid R) \log(1/\Pr(\Xi(v) \mid R)) \enspace ,\end{aligned}$$

and the lemma follows from Claim 1. □

The remainder of our argument involves carefully piecing together subsets of the nodes in T and applying Lemma 1 to these subsets. By doing this carefully, we will eventually obtain a lower bound that matches the upper bound in Theorem 1. Let

$$H = \sum_{v \in T} \Pr(v) \log(1/\Pr(v)) \tag{3}$$

be the entropy of the distribution of search paths in the triangle tree T. Note that, ignoring the $O(1)$ term, the upper bound of Theorem 1 is within a constant factor of H. Thus, our goal is to show that no linear decision tree has an expected search time in $o(H)$.

We start our analysis by partitioning the internal nodes of T in *groups* $G_1, G_2, \ldots$ where

$$G_i = \{v \in T : 1/2^i \leq \Pr(v) < 1/2^{i-1}\} \enspace .$$

In what follows, we fix some constant $0 < \alpha < 1$ that will be specified later. Our first result shows that, in our lower bound, we can discard a fairly large number of elements from each group without having much effect on the overall entropy:

Lemma 2. *For each i, let G'_i be obtained by deleting at most $2^{\alpha i}$ elements from G_i. Then*

$$\sum_{i=1}^{\infty} \sum_{v \in G'_i} \Pr(v) \log(1/\Pr(v)) \geq H - O(1) \enspace .$$

Proof.

$$\begin{aligned}H &= \sum_{v \in T} \Pr(v) \log(1/\Pr(v)) \\ &= \sum_{i=1}^{\infty} \sum_{v \in G'_i} \Pr(v) \log(1/\Pr(v)) + \sum_{i=1}^{\infty} \sum_{v \in G_i \setminus G'_i} \Pr(v) \log(1/\Pr(v))\end{aligned}$$

$$
\begin{aligned}
&\le \sum_{i=1}^{\infty}\sum_{v\in G'_i}\Pr(v)\log(1/\Pr(v))+\sum_{i=1}^{\infty}\sum_{v\in G_i\setminus G'_i}(1/2^{i-1})\log(2^i)\\
&\le \sum_{i=1}^{\infty}\sum_{v\in G'_i}\Pr(v)\log(1/\Pr(v))+\sum_{i=1}^{\infty}\sum_{v\in G_i\setminus G'_i}2i(1/2^i)\\
&\le \sum_{i=1}^{\infty}\sum_{v\in G'_i}\Pr(v)\log(1/\Pr(v))+\sum_{i=1}^{\infty}2i(1/2^{i-\alpha i})\\
&= \sum_{i=1}^{\infty}\sum_{v\in G'_i}\Pr(v)\log(1/\Pr(v))+2\sum_{i=1}^{\infty}i/(2^{1-\alpha})^i\\
&= \sum_{i=1}^{\infty}\sum_{v\in G'_i}\Pr(v)\log(1/\Pr(v))+O(1)\ \ ,
\end{aligned}
$$

since $2^{1-\alpha} > 1$. □

In order to use Lemma 1 we must partition the vertices of T into subsets that are compatible with the conditions of the lemma.

Lemma 3. *Each group G_i can be partitioned into t_i subgroups $G_{i,1},\dots,G_{i,t_i}$ such that*

1. $|G_{i,t_i}| \le 2^{\alpha i}$.
2. $|G_{i,j}| \ge 2^{\alpha i}/i$, *for all* $1 \le j < t_i$, *and*
3. *for every* $1 \le j < t_i$ *and every* $u, v \in G_{i,j}$ *node* u *is not an ancestor of node* v *in* T.

Proof. Assume that $|G_i| > 2^{\alpha i}$, otherwise there is nothing to prove. Observe that all vertices in G_i appear within the first i levels of T. Thus, any node in G_i has at most $i-1$ ancestors in T.

We can obtain the first subgroup $G_{i,1}$ by first defining all nodes of G_i to be *unmarked* and *unselected*. To obtain $G_{i,1}$ we repeatedly *select* any unselected and unmarked node $v \in G_i$ that does not have any descendants in G_i and *mark* the (at most $i-1$) ancestors of v in T. This process selects at least

$$|G_i|/i \ge 2^{\alpha i}/i$$

elements to take part in $G_{i,1}$. We can then apply this process recursively on $G_i \setminus G_{i,1}$ to obtain the sets $G_{i,2},\dots,G_{i,t_i-1}$. Once this is done, we place the at most $2^{\alpha i}$ remaining elements in group G_{i,t_i}. □

We now have all the tools we need to prove our lower bound.

Theorem 2. *Let $\epsilon > 0$ be an arbitrarily small constant. Any linear decision tree T' that solves the problem of testing if any query point $p \in \mathbb{R}^2$ is contained in P has*

$$\mu_D(T') \ge \left(\frac{1}{4}-\epsilon\right)H - O(1)\ \ .$$

Proof. Our proof is an application of the little-birdie principle. We work in a slightly stronger model of computation in which we are given a triangle tree $T = T(P, D)$ and the partitioning of the vertices of T into the sets $G_{i,j}$ described in Lemma 3. In this model, an algorithm consists of a whole collection of linear decision trees $T_{i,j}$; one for each group $G_{i,j}$. Now, when the point p is selected according to D, a *little birdie* tells the algorithm which group $G_{i,j}$ contains the vertex v such that $p \in \Xi(v)$. The algorithm then uses the information provided by the little birdie to select the decision tree $T_{i,j}$ and uses $T_{i,j}$ to determine if $p \in P$. The cost of this is the cost of searching in $T_{i,j}$. Thus, the expected cost of a search in this model of computation is

$$\mu = \sum_{i=1}^{\infty}\sum_{j=1}^{t_i} \Pr(G_{i,j})\mu_{D_{i,j}}(T_{i,j}) \enspace ,$$

where $D_{i,j}$ denotes the probability distribution D conditioned on the search for p ending at a node in $G_{i,j}$. Clearly this model of computation is at least as strong as the linear decision tree model since, in this model, there is always the option of ignoring the birdie's advice by creating a single linear decision tree T' and setting $T_{i,j} = T'$ for all i and j.

Now, applying Lemma 1 to each group $G_{i,j}$ we obtain

$$\begin{aligned}
\mu &= \sum_{i=1}^{\infty}\sum_{j=1}^{t_i} \Pr(G_{i,j})\mu_{D_{i,j}}(T_{i,j}) \\
&\geq \sum_{i=1}^{\infty}\sum_{j=1}^{t_i-1} \Pr(G_{i,j})\mu_{D_{i,j}}(T_{i,j}) \\
&\geq \sum_{i=1}^{\infty}\sum_{j=1}^{t_i-1} \Pr(G_{i,j}) \times \left(\frac{1}{4}\sum_{v\in G_{i,j}} \Pr(v \mid G_{i,j})\log(1/\Pr(v \mid G_{i,j}))\right) \\
&= \frac{1}{4}\sum_{i=1}^{\infty}\sum_{j=1}^{t_i-1} \Pr(G_{i,j}) \times \left(\sum_{v\in G_{i,j}} \Pr(v \mid G_{i,j})\log(1/\Pr(v \mid G_{i,j}))\right) \\
&= \frac{1}{4}\sum_{i=1}^{\infty}\sum_{j=1}^{t_i-1}\sum_{v\in G_{i,j}} \Pr(v)\log(\Pr(G_{i,j})/\Pr(v)) \\
&= \frac{1}{4}\sum_{i=1}^{\infty}\sum_{j=1}^{t_i-1}\sum_{v\in G_{i,j}} \Pr(v)(\log(1/\Pr(v)) + \log(\Pr(G_{i,j})) \\
&\geq \frac{1}{4}\sum_{i=1}^{\infty}\sum_{j=1}^{t_i-1}\sum_{v\in G_{i,j}} \Pr(v)(\log(2^{i-1}) + \log(2^{\alpha i}/i2^{i}) \\
&= \frac{1}{4}\sum_{i=1}^{\infty}\sum_{j=1}^{t_i-1}\sum_{v\in G_{i,j}} \Pr(v)(i - 1 + \alpha i - i - \log i) \\
&= \frac{1}{4}\sum_{i=1}^{\infty}\sum_{j=1}^{t_i-1}\sum_{v\in G_{i,j}} \Pr(v)(\alpha i - 1 - \log i) \\
&\geq \left(\frac{1}{4} - \epsilon\right)H - O(1) \enspace ,
\end{aligned}$$

where the last inequality follows from Lemma 2 and by setting $\alpha = 1 - \delta$ for a sufficiently small constant $\delta = \delta(\epsilon) > 0$. □

4 Convex Subdivisions

In this section we consider the problem of point location in convex subdivisions. Our data structure is simple. For each internal face F_i of the convex subdivision we triangulate the interior of F_i using the internal edges of the triangles of a triangle tree $T_i = (F_i, D_{|F_i})$ for the polygon F_i and the distribution $D_{|F_i}$. For the outer face, we do the same but keep only the external edges of the triangles. Next, we preprocess the resulting triangulation using Iacono's distribution-sensitive point location data structure for triangulations [10].

We have three lower bounds on the expected query time of any linear classification tree for point location. The first lower bound follows from the fact that any classification tree with more than 1 class must have at least one internal node:

$$B_0 = \Omega(1)$$

The second lower bound is the entropy bound:

$$B_1 = \sum_{i=1}^{f} \Pr(F_i) \log(1/\Pr(F_i)) \enspace .$$

The third lower bound is a bit more subtle. It follows from the fact that, inside any classification tree for point location is a decision tree for testing, for each $1 \leq i \leq f$, if a query point p is contained in F_i. From Theorem 2 we know that, for any decision tree T' that determines if a query point p is in F_i

$$\mu_{D_{|F_i}(T')} \geq \Omega(1) \times \sum_{v \in T_i} \Pr(v|F_i) \log(1/\Pr(v|F_i)) \enspace .$$

Summing over all faces, we obtain the third lower bound:

$$B_2 = \Omega(1) \times \sum_{i=1}^{f} \sum_{v \in T_i} \Pr(F_i) \Pr(v|F_i) \log(1/\Pr(v|F_i)) \enspace .$$

Now, because we store all the triangles of each T_i in a point location structure that achieves the entropy bound, the resulting structure has an expected query time of

$$\begin{aligned}
\mu &= O(1) + O(1) \times \sum_{i=1}^{f} \sum_{v \in T_i} \Pr(v) \log(1/\Pr(v)) \\
&= O(1) + O(1) \times \sum_{i=1}^{f} \Pr(F_i) \times \sum_{v \in T_i} \Pr(v|F_i) \log(1/\Pr(v)) \\
&= O(1) + O(1) \times \sum_{i=1}^{f} \Pr(F_i) \times \sum_{v \in T_i} \Pr(v|F_i) \log(\Pr(F_i)/\Pr(F_i)\Pr(v)) \\
&= O(1) + O(1) \times \left(\sum_{i=1}^{f} \Pr(F_i) \log(1/\Pr(F_i)) + \sum_{i=1}^{f} \sum_{v \in T_i} \Pr(F_i) \Pr(v|F_i) \log(1/\Pr(v|F_i)) \right) \\
&\leq O(1) + O(1) \times (B_1 + B_2) \\
&\leq O(1) \times \max\{B_1, B_2, B_3\} \enspace .
\end{aligned}$$

This completes the proof of our main result:

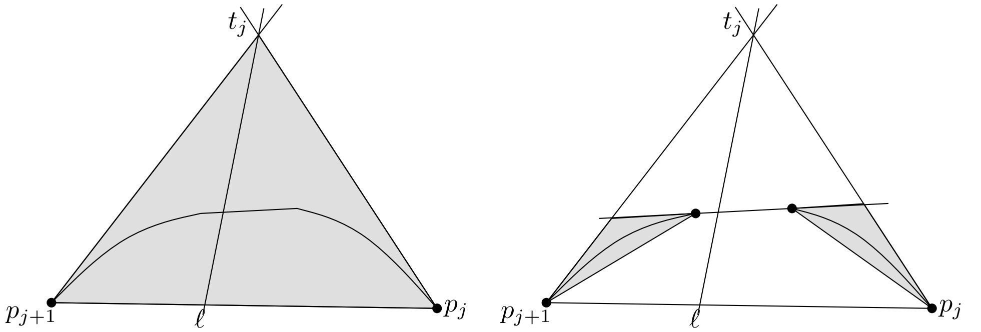


Figure 4: The data structure can be realized using only lines through the vertices of P.

Theorem 3. *Given a convex subdivision G with n vertices and a probability measure D over $\mathbb{R}^2$, a data structure of size $O(n)$ can be constructed in $O(n)$ time that answers point location queries in G. The expected query time of this data structure, for a point p drawn according to D is $O(\mu_D(T))$, where T is any linear classification tree that answers point location queries in G.*

5 Discussion

We have presented a data structure for distribution-sensitive point location in convex subdivisions. Our data structure achieves, up to constant factors, the best possible query time for any data structure in the linear decision tree model of computation. All known data structures for point location in planar subdivisions fit into the linear decision tree model.

Our data structure, as described in Section 3, uses comparisons between the query point p and precomputed lines determined by points on the edges of P that are, in turn, determined by the distribution D. We note that we can obtain an equally efficient structure that only does comparisons involving lines through two vertices of G. To achieve this, we simply modify the splitting process at the nodes of the triangle tree so that, instead of placing a single point in the interior of an edge e of P (Figure 1), we place one point on each of the endpoints of e (Figure 4). This modification has another nice property; it can be implemented so that the only mechanism needed to access the probability measure D is a primitive for computing the probability of a triangle. An interesting corollary of this result is that, given a convex polygon P and a probability distribution over the interior of P, the above construction gives a linear-time constant factor approximation for the minimum entropy triangulation of P.

In practice, performing computations with probability distributions is at best awkward. However, one potential real application of our data structures is when the distribution D is uniform over a set of m points. Such distributions are easily obtained by sampling some (unknown) continuous distribution and are often a good enough approximation of the continuous distribution. In this case, it is fairly straightforward to construct our point-location structure in $O(n + m\log(mn))$ time.

When discussing distribution-sensitive point location, the exact definition of the problem and the choice of computational model is important. For example, our definition of the problem requires that our data structure outputs the correct answer for every input point in $\mathbb{R}^2$. This means that, for example, we can obtain non-trivial lower bounds for testing if $p \in P$ even in the case when $\Pr\{p \in P\} = 1$. One could imagine another definition of the problem in which the data structure is only required to answer correctly for points that are in the support of D.

We have given upper and lower bounds on the expected complexity of testing if a point is in a convex polygon and these bounds match to within a constant factor. If we wish to be more precise, the expected number of linear inequalities tested by any linear decision tree is at least $(1/4 - \epsilon)H - o(H)$ and our data structure tests $3H + o(H)$ linear inqequalities. (The constant 3 appears in the upper bound because the expected depth of a search in the triangle tree is at most H and each node in the triangle tree tests the query point against 3 lines.) If we allow $O(n \log^* n)$ space, the upper bound can be reduced to $H + o(H)$ by storing the triangles of the triangle tree in the data structure of Arya *et al* [4]. Alternatively, we can maintain the $O(n)$ space requirement while decreasing the upper bound to $(1 + \epsilon)H + o(H)$ by storing sufficiently large subtrees of the triangle tree in the point location data structure of Arya *et al* [3, 4]. Even with this trick there still remains a factor of $4 + \epsilon$ between the upper and lower bounds.

Open Problem 1. *Close the gap between the lower bounds and upper bounds on the expected cost of testing if a point is contained in a convex polygon.*

We have studied point location in convex polygons and convex subdivisions. From here we could try to extend our results to simple polygons and general planar subdivisions whose faces are arbitrary simple polygons. Perhaps a more realistic next step is to study the related problem of *vertical ray shooting*: Preprocess a set S of points and open line segments so that, for any query point p, we can determine the first object in S intersected by an upward vertical ray originating at p. Vertical ray shooting is often used interchangeably with planar point location but is a strictly harder problem since it sometimes requires the data structure to distinguish between edges of the same face. At the heart of the vertical ray shooting problem is the following decision problem:

Open Problem 2. *Let P be a simple polygon whose boundary consists of one line segment and one x-monotone chain joining the two endpoints of the line segment and let D be a probability measure over $\mathbb{R}^2$. Preprocess P and D into a data structure that can test if a query point p is contained in P and whose expected query time is optimal in the linear decision tree model of computation.*

References

[1] U. Adamy and R. Seidel. On the exact worst case query complexity of planar point location. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 609–618, 1998.

[2] S. Arya, S. W. Cheng, D. M. Mount, and H. Ramesh. Efficient expected-case algorithms for planar point location. In *Proceedings of the seventh Scandinavian Workshop on Algorithm Theory*, pages 353–366, 2000.

[3] S. Arya, T. Malamatos, and D. M. Mount. Nearly optimal expected-case planar point location. In *Proceedings of the 41st annual Symposium on Foundations of Computer Science*, pages 208–218, 2000.

[4] S. Arya, T. Malamatos, and D. M. Mount. Entropy-preserving cuttings and space-efficient planar point location. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on 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 *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 262–268, 2001.

[6] W. A. Beyer and A. Zardecki. The early history of the ham sandwich theorem. *American Mathematical Monthly*, 111(1):58–61, 2004.

[7] D. Dobkin and R. Lipton. Multidimensional searching problems. *SIAM Journal on Computing*, 5:181–186, 1976.

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

[9] M. Goodrich, M. Orletsky, and K. Ramaiyer. Methods for achieving fast query times in point location data structures. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 757–766, 1997.

[10] J. Iacono. Optimal planar point location. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 240–241, 2001.

[11] J. Iacono. Expected asymptotically optimal planar point location. *Computational Geometry Theory and Applications*, 29(1):19–22, 2004.

[12] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[13] D. E. Knuth. Two notes on notation. *American Mathematical Monthly*, 99(5), 1992.

[14] D. T. Lee and F. P. Preparata. Location of a point in a planar subdivision and its applications. *SIAM Journal on Computing*, 6:594–606, 1977.

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

[16] K. Mulmuley. A fast planar partition algorithm. *Journal of Symbolic Computation,* 10:253–280, 1990.

[17] F. P. Preparata. A new approach to planar point location. *SIAM Journal on Computing*, 10:473–483, 1981.

[18] F. P. Preparata. Planar point location revisited: A guided tour of a decade of research. *International Journal of Foundations of Computer Science*, 1(1):71–86, 1990.

[19] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New-York, 1985.

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

[21] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, pages 379–423 and 623–656, 1948.