HKUST Theoretical Computer Science Center Research Report HKUST-TCSC-2004-10

A Simple Entropy-Based Algorithm for Planar Point Location*

Sunil Arya† Theocharis Malamatos‡ David M. Mount§

Abstract

Given a planar polygonal subdivision S, point location involves preprocessing this subdivision into a data structure so that given any query point q, the cell of the subdivision containing q can be determined efficiently. Suppose that for each cell z in the subdivision, the probability p_z that a query point lies within this cell is also given. The goal is to design the data structure to minimize the average search time. This problem has been considered before, but existing data structures are all quite complicated. It has long been known that the entropy H of the probability distribution is the dominant term in the lower bound on the average-case search time. In this paper, we show that a very simple modification of a well-known randomized incremental algorithm can be applied to produce a data structure of expected linear size that can answer point-location queries in $O(H)$ average time. We also present empirical evidence for the practical efficiency of this approach.

1 Introduction

The planar point-location problem is one of the most fundamental query problems in computational geometry. The problem is to preprocess a polygonal subdivision S consisting of n edges into a data structure so that given any query point q, the polygonal cell of the subdivision containing q can be reported quickly. The first worst-case asymptotically optimal algorithm for this problem was due to Kirkpatrick, which achieved $O(n)$ space and $O(\log n)$ query time [10]. This was followed by a number of related methods with better practical performance by Edelsbrunner, Guibas, and Stolfi [7], Cole [5], and Sarnak and Tarjan [14]. In spite of their enhanced practicality, these methods lacked the simplicity of Kirkpatrick's method. A truly simple randomized incremental method was discovered by Mulmuley [13] and Seidel [15]. This method is based on randomly inserting the line segments of the subdivision and maintaining a trapezoidal map of the segments. The point-location data structure is simply a directed, acyclic graph that records the history of the various changes to the structure. For a fixed query point, the expected search time takes at most $5 \ln n + O(1)$ comparisons. Here the expectation is taken over all random permutations of the segments.

*A preliminary version of this paper appeared in the *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, 2001, 262–268. The work of Arya and Malamatos was supported in part by the Research Grants Council, Hong Kong, China (HKUST6229/00E) 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@cs.ust.hk.

‡Max Plank 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.

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 fundamental question of whether it is possible to answer queries efficiently in the average case. We are given a planar subdivision S and a probability distribution on the set of queries. 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. In practice these probabilities can be determined by observing a long sequence of queries. 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 each edge as a trapezoid of infinitesimal height and each vertex as a square of infinitessimal side length.

An important concept is that of the *entropy* of S, denoted H throughout, which is defined

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

(Throughout we use log to denote base-2 logarithm and ln to denote the natural logarithm.) For the 1-dimensional restriction of this problem, a classical result due to Shannon implies that the average number of comparisons needed to answer such queries is at least as large as the entropy of the probability distribution [11, 17]. Mehlhorn [12] showed that it is possible to construct a nearly optimal binary search tree whose average search time is at most $H + 2$.

Arya, Cheng, Mount, and Ramesh [1] showed that if the subdivision consists of convex polygons and the x and y coordinates of the query points are chosen independently from some probability distribution, the entropy bound can be achieved to within a constant multiplicative factor (2 using quadratic space and about 4 using linear space). These results were strengthened by Arya, Malamatos, and Mount [2, 3] for the case of polygonal subdivisions in which each cell has constant complexity. They presented an algorithm that answers queries in $H + O(H^{2/3}+1)$ average time and $O(n \log^* n)$ space. These methods rely on relatively complex constructions. In contrast, Mehlhorn's nearly optimal binary search tree for the 1-dimensional problem [12] is quite simple. This raises the question of whether there exists a data structure that achieves the simplicity of the randomized incremental point-location data structure, while achieving both good expected-case performance and low space requirements.

In this paper we give a positive answer by presenting a simple weighted variant of the randomized incremental algorithm. Our main results are given in Theorems 1 and 2 below. We assume that the cells of the subdivision have constant complexity, that is, they are each bounded by a constant number of sides. To minimize confusion we use the term *expected* when referring to variations that are caused by the random choices made by the algorithm and *average* when referring to variations due to the random distribution of query points. Our query times are presented in terms of the number of binary tests, called *primitive comparisons.* One determines whether the query point lies to the left or right of a vertical line passing through the endpoint of some edge of the subdivision, and the other determines whether the query point lies above or below an edge of the subdivision. This is the same as the model introduced by Seidel and Adamy [16] in their lower bound, and is used in almost all common point-location algorithms.

Theorem 1 *Consider a polygonal subdivision S of size n consisting of cells of constant 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 structure of space $O(n)$ that can answer point-location queries in average time $O(H+1)$. If S is presented as a trapezoidal map (defined below) then the average search time is $(5\ln 2)H + O(1)$. All bounds hold in expectation over random choices made by the construction algorithm.*

Note that the constructions are randomized, and the query time and space bounds hold in expectation over the algorithm's random choices. By repeating the construction and relaxing the constant factors, we can achieve the following result, which provides guarantees on the space and average query time bounds, while increasing the construction time only by a constant factor.

Theorem 2 *Given the same conditions of Theorem 1, in $O(n \log n)$ expected time it is possible to construct a structure of space $O(n)$ that can answer point-location queries in average time $O(H+1)$.*

We actually prove the stronger result that for any cell z of a trapezoidal map S, the expected number of comparisons to locate a query point lying within z is at most $5\ln(1/p_z)+O(1)$. Theorem 1 follows directly from this fact. It is also interesting to note that our search structure is efficient even in the worst case. For any query point q, the expected query time is $O(\log n)$. (This holds irrespective of the cell containing q, but holds in expectation over the random choices made by the algorithm.)

The assumption that cells have a constant number of sides seems to be critical, if no other assumptions are made about the query distribution. This assumption is not required for worst-case optimal planar point location, since it is possible to refine any polygonal subdivision into one of constant combinatorial complexity while increasing the subdivision's size by just a constant factor. However the case of average-case query time is much different. Arya *et al.* [4] show that even for the restricted case in which the subdivision consists of a single n-sided convex polygon, there exists a query distribution such that no point location algorithm that is based on point-line comparisons can achieve an average query time that is solely a function of entropy.

As is common with randomized algorithms, our proofs of the space and query time both make use of *backwards analysis*. Such an analysis is based on the notion that, since objects are inserted randomly, at any given stage every object is equally likely to have been the last to be inserted. In our case this assumption does not hold. We overcome this problem by a trick of associating some number of "pebbles" with each of the edges of the subdivision. The number is a function of the query probabilities for the incident subdivision cells. The pebbles are drawn in random order, and a segment is inserted the first time that one of its pebbles is drawn. Subsequent pebble drawings for this segment are ignored.

Recently and independently, John Iacono has developed a algorithm of a similar flavor to ours [8, 9]. He has shown how to adapt Kirkpatrick's point location algorithm [10] in order to achieve query times that are proportional to entropy. As with Kirkpatrick's algorithm, his construction is deterministic, but is somewhat less practical due to the larger constant factors involved.

The remainder of the paper is organized as follows. In the next section we present the randomized incremental construction algorithm. In Sections 3.1 and 3.2 we show that the space and average time bounds hold in expectation (over random choices made in the construction). In Section 3.3 we show that these bounds can be guaranteed; although the construction time holds only in expectation. Finally, in Section 4 we present experimental evidence that, when the query distribution is highly nonuniform, our algorithm achieves significant speed-ups on average over the standard worst-case efficient algorithm.

2 Weighted Randomized Incremental Algorithm

Let S denote a polygonal subdivision whose cells are of constant combinatorial complexity. Our approach to answering point location queries in S is to reduce the problem to an appropriately weighted trapezoidal map (defined below) of the edges of S. We will show that (subject to variations due to the randomized construction) point location queries in a trapezoidal map of entropy H' can be answered in average time $(5 \ln 2)H' + O(1)$. Since the trapezoidal map is a refinement of S, the location of a query point in the trapezoidal map uniquely determines the location of the query point in S. Arya *et al.* [4] prove that if S is a weighted polygonal subdivision of entropy H whose cells have bounded combinatorial complexity, then such a reduction increases the above average query time only by an additive factor of $O(H + 1)$. The reweighting is computed as follows. Recall that p_z denotes the probability that a query point lies in cell z of S, and let f_z denote the number of disjoint cells into which z is decomposed by the trapezoidal map. Then the weight p_z is evenly redistributed among its fragments by assigning each fragment a weight of p_z/f_z. Henceforth, we may assume that S is presented as a weighted trapezoidal map, and we let H denote its entropy.

We now present a quick review of trapezoidal maps. We refer the reader to the text by de Berg *et al.* [6] for more detailed information. We assume that the input is any set of nonvertical line segments in the plane with disjoint interiors. We assume that the segments and all the queries are enclosed within a large bounding rectangle. The trapezoidal map is defined by the following subdivision process. From each segment endpoint, imagine shooting two vertical bullets, one up and one down. Each bullet continues until first hitting a segment, or the bounding rectangle if it hits no segment. The union of the segments together with the bullet paths defines a subdivision of the bounding rectangle. (See Fig. 1(a).) It is easy to see that the cells of the resulting subdivision have disjoint interiors and cover the bounding rectangle. Each cell is a (possibly degenerate) trapezoid, bounded by at most two vertical sides and a top and bottom segment. Note that subdivision is not a cell complex, since the interior of a single side of one trapezoid may be incident to many other trapezoids.

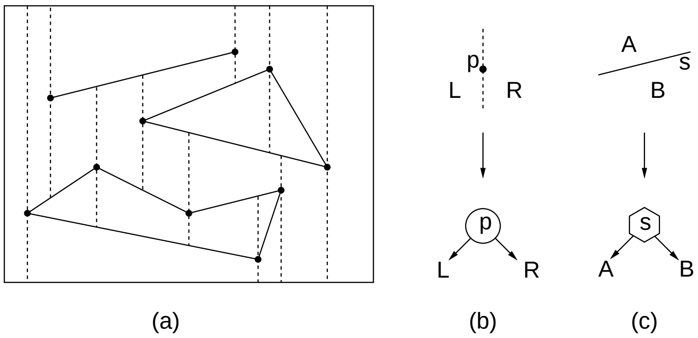


Fig. 1: Trapezoidal maps and node types.

The point-location structure for a trapezoidal map is a rooted directed acyclic graph in which each node either has two outgoing edges (an *internal node*) or has no outgoing edges (a *leaf node*). The structure can also be viewed as a binary tree in which each subtree may be shared by multiple parents. Each *leaf node* is uniquely associated with a trapezoid of the final map. There are two types of internal nodes. A *left-right internal node* is associated with an endpoint of some segment. (See Fig. 1(b).) It tests whether the query point's x-coordinate lies to the left or right of the x-coordinate of the associated endpoint and branches to the left or right child, respectively. An *above-below internal node* is associated with a segment. It tests whether the query point lies above or below the line containing the associated segment and branches to the left or right child, respectively. (See Fig. 1(c).)

The randomized incremental algorithm operates by inserting the segments one by one in random order and updating the subdivision after each insertion. The insertion of each segment results in the creation of four new bullet paths, two for each of its endpoints, and some of the existing bullet paths may now be blocked by the new segment. (See Fig. 2.) This in turn causes some of the existing trapezoidal cells to be replaced with new trapezoids. For each trapezoid that has been replaced, the incremental algorithm replaces the corresponding leaf node in the history graph with the appropriate set of internal nodes to ascertain which new trapezoid the query point now lies in. (The newly added nodes in the history graph are shaded in the right side of Fig. 2.) See de Berg *et al.* [6] for details.

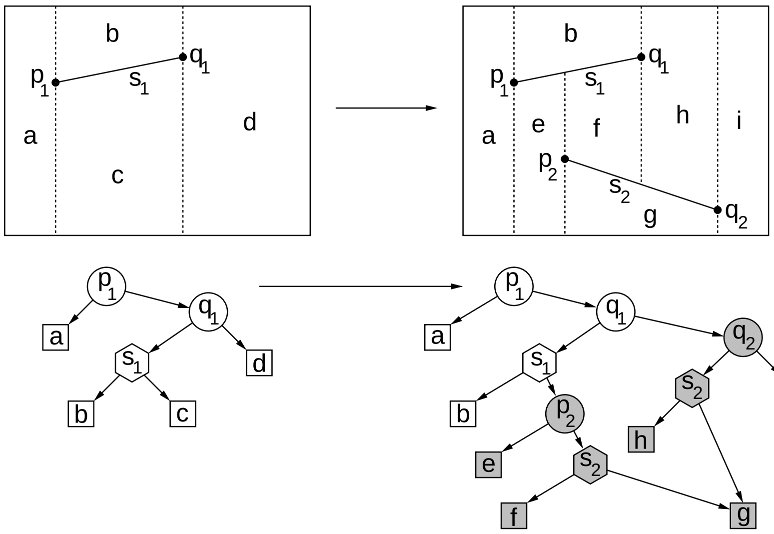


Fig. 2: Incremental algorithm for trapezoidal map and the associated history graph.

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

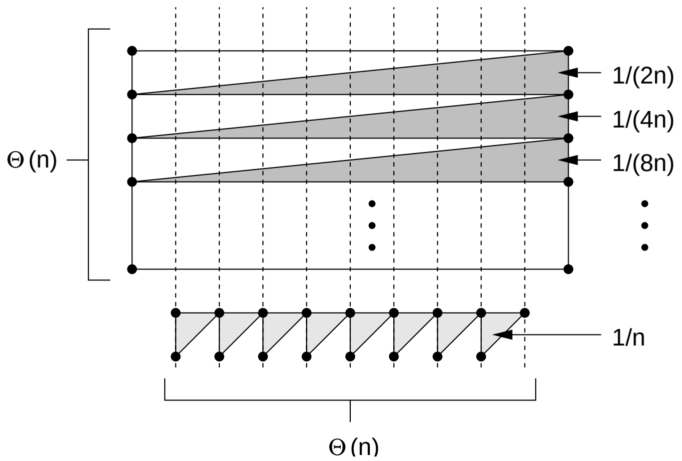
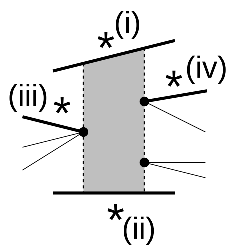


Fig. 3: A bad instance for simple weighting.

The problem is that an adversary can adjust the probabilities to cause the algorithm to insert the segments in an order so that the space of the resulting structure would be $\Theta(n^2)$. Consider the example shown in Fig. 3. Below there are $\Theta(n)$ triangles, each having a probability of $1/n$. Above there are $\Theta(n)$ triangles with probabilities of the form $1/(2^i n)$ with i increasing from the top to the bottom. The insertion order provided by the simple weighting scheme results in the insertion of segments bounding the $\Theta(n)$ lower triangles, thus creating $\Theta(n)$ vertical slabs. After this, the segments bounding the upper triangles will be inserted in order from top to bottom. Since each intersects all $\Theta(n)$ vertical slabs, each insertion results in $\Theta(n)$ updates to the trapezoidal map, which leads to $\Theta(n^2)$ total space. Note that the problem is not due merely to the absence of randomization, since even a randomized algorithm that samples segments according to the given weights will result in an expected total space of $\Theta(n^2)$.

At first it might seem that any weighted randomized incremental construction could suffer from this space problem. The source of the problem is not the weights themselves, but arises from the huge variation that is possible among the weights. The standard randomized incremental algorithm can be thought of as a method in which all segments have equal weight of $1/n$. Our solution is to modify the simple weighting scheme so that the ratio of any two segment weights is $O(n)$. Remarkably, this simple fix is all that is needed to eliminate the above space problem, and yet the representation of the probability distribution is close enough to achieve the desired entropy bounds on average search times.

Recall that S is a trapezoidal map, which is assumed to be defined by n segments, denoted by X. Each trapezoid of S can be associated by a subset of at most four *defining segments* of X, which have the properties that once all of these segments have been inserted by the incremental algorithm, this trapezoid will come into existence. These are:



(i) the segment defining the trapezoid's upper side,
(ii) the segment defining its lower side,
(iii) any segment whose endpoint lies on its left vertical side, and
(iv) any segment whose endpoint lies on its right vertical side.

(These are shown with bold lines in the figure to the right.) If a trapezoid is bounded on some side by a side of the bounding rectangle, we ignore this side since the sides of the bounding rectangle are present even before the algorithm begins.

Our algorithm works as follows. First, we redistribute the probabilities associated with the cells of S to the segments of S as follows. Recall that for each trapezoid z of S, p_z denotes the probability of the query point lying in z. Assign a probability of $p_z/4$ to each of the (up to) four defining segment of z. For a segment $x \in X$, let p_x denote the sum of the probabilities it derives from its incident trapezoids. Thus, $\sum_x p_x \leq 1$. Let $K > 0$ be a constant. (We shall see that the choice of K provides a trade-off between the multiplicative constant in the space bound and the additive constant in the query time. For example, $K = 1$ is a reasonable choice.) Assign a weight $w_x = \max(\lceil Kp_x n \rceil, 1)$ to each segment x.

After this weight assignment we run the standard randomized incremental algorithm in which we sample the segments based on their weights. Initially the map consists of just the bounding rectangle. Otherwise, if W denotes the total weight of all the uninserted segments, the probability that x is sampled for insertion is w_x/W.

Observe that the weight w_x of any segment x is an integer between 1 and $Kn+1$, and the total weight W of all the segments is at most $(K+1)n$. This follows because $\sum_x w_x \leq \sum_x (Kp_x n + 1) \leq (K+1)n$.

3 Analysis

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

3.1 Space Analysis

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

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

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

Reversing the order of summation gives

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

Recall that each trapezoid $\tau \in \mathcal{T}(P^i)$ is defined by at most four segments that bound its sides. Clearly the trapezoid τ is created in the last step if and only if one of the following four conditions hold:

(i) there is exactly one pebble in P^i such that the associated segment forms the top side of τ and this pebble is inserted last,
(ii) same as (i) for the bottom side,
(iii) there is exactly one pebble in P^i such that an endpoint of the associated segment lies on the left side of τ and this pebble is inserted last, and
(iv) same as (iii) for the right side.

Thus there are at most four pebbles in P^i with the property that if they were inserted last they would have created τ, i.e., $\sum_{p\in P^i}\delta(\tau,p) \le 4$, and so we obtain

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

Here we have used the fact that the number of segments in $\mathcal{T}(P^i)$ is at most i and so by standard results on the complexity of planar maps, it follows that $\mathcal{T}(P^i)$ contains at most $O(i)$ trapezoids [6]. Since the bound on the expected value of k_i does not depend on the choice of P^i, we can conclude that it holds unconditionally. Summing $E[k_i]$ over the W steps of the algorithm it follows that the expected total number of new trapezoids created is $O(W)$. Recalling that $W \le (K+1)n$, this gives a bound of $O(n)$ on the expected number of new trapezoids, and hence on the space used by the search structure.

3.2 Average Query Time

We now analyze the *expected* value of the *average* query time provided by the search structure, measured as the average number of primitive comparisons, or equivalently the weighted depth of each leaf of the search structure. (Recall our convention that the average query time refers to the average over the different locations of the query point. We are interested in the expectation of this quantity taken over the random choices made by the algorithm.)

We say that two query points are *equivalent* if they follow the same path through the search structure. Consider a partitioning of the plane into slabs by passing a vertical line through the endpoints of all the segments. Each slab is decomposed into trapezoids by the segments that cross the slab. It is easy to see that the query points inside any such trapezoid are equivalent. Let $\mathcal{T}$

denote the set of trapezoids in all the slabs. Then the average query time is given by $\sum_{\tau \in \mathcal{T}} p_\tau d_\tau$, where p_τ is the probability of the query point lying in τ, and d_τ is the length of the search path for the query points in τ. Note that τ is a subregion of a cell of the original subdivision. Since we make no assumptions about the query distribution within each cell, the value of p_τ is unknown to the algorithm. Our analysis holds irrespective of its value, subject to the obvious constraint that, for each cell $z \in S$, the sum of the probabilities p_τ for all $\tau \subseteq z$ equals the cell's probability, p_z.

Let q be a query point inside a trapezoid $\tau \in \mathcal{T}$. Let τ be contained within trapezoid $z \in S$. We will prove that the expected length of the search path for q, $E[d_\tau]$, is at most $5\ln(1/p_z) + O(1)$. By considering the contribution of all the trapezoids of S, it follows that the expected value of the average query time is

$$\sum_{z \in S} p_z \left(5 \ln \frac{1}{p_z} + O(1)\right) \;=\; (5\ln 2)\left(\sum_{z \in S} p_z \log \frac{1}{p_z}\right) + O(1) \;=\; (5\ln 2)H + O(1),$$

which is the desired bound as stated in Theorem 1.

Let ℓ_i be a random variable that takes the value 1 if the left side of the trapezoid containing q changes when the ith pebble is inserted and is 0 otherwise. Similarly define the random variables r_i, t_i, and b_i for the right, top, and bottom sides, respectively. We employ the following observation made by Seidel [15]: the length of the search path for q grows by at most $2\ell_i + r_i + t_i + b_i$ when the ith pebble is inserted.[1] Let $\ell = \sum_{1 \le i \le W} \ell_i$. Similarly define r, t, and b. By the linearity of expectation, we have

$$E[d_\tau] \;\le\; 2E[\ell] + E[r] + E[t] + E[b].$$

We will show that $E[\ell], E[r], E[t]$, and $E[b]$ are all bounded by $\ln(1/p_z) + O(1)$, which implies the desired bound on $E[d_\tau]$.

We will only prove that $E[\ell] \le \ln(1/p_z) + O(1)$, since the other bounds can all be proved in a similar way. For $1 \le s \le 4$, let x_s denote the four segments that define z, and let M_s be a random variable denoting the step of the algorithm in which a pebble associated with segment x_s is first inserted, and let M be the random variable $\max_s(M_s)$. M is the step at which all the defining segments are present, and hence no further changes can occur to the trapezoid containing q. We can write $E[\ell]$ as follows:

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

Since no changes occur to the trapezoid containing q after step M, we have $E[\ell_i \mid M = j]$ for $i > j$ is 0. Obviously $E[\ell_j \mid M = j] \le 1$ since the largest value that the random variable ℓ_j can take is one.

We next prove that $E[\ell_i \mid M = j] \le 1/i$ for $i < j$. This is based on backwards analysis. Suppose that the jth pebble inserted is associated with segment x_k, where $1 \le k \le 4$. Let $P^{j-1} \subseteq P$ be any fixed set of $j-1$ pebbles that contains at least one pebble associated with the three segments x_s, where $s \in \{1,2,3,4\} - \{k\}$, and no pebble associated with x_k. We claim that, for $i < j$, the expected value of ℓ_i subject to the condition that P^{j-1} is the set of the first $j-1$ pebbles inserted

[1] The source of the rather unexpected asymmetry that results in the extra factor of 2 times ℓ_i is evident in example shown on the left side of Fig. 2. Trapezoid a whose right side touches the segment is at depth 1, whereas trapezoid d whose left side touches the segment is at depth 2.

is at most $1/i$, irrespective of the choice of x_k and P^{j-1}. By the definition of M, it is clear that this claim would imply that $E[\ell_i \mid M = j] \le 1/i$ for $i < j$.

To prove this claim, let $P^i \subseteq P^{j-1}$ be any fixed set of i pebbles. We compute the expected value of ℓ_i subject to the condition that P^i is the set of the first i pebbles inserted. Let σ be the trapezoid in $\mathcal{T}(P^i)$ that contains q. The left side of σ would have changed in step i if and only if there is exactly one pebble in P^i such that an endpoint of the associated segment lies on the left side of σ, and this pebble is inserted last (i.e., in step i). The probability of this event is at most $1/i$, and thus the expected value of ℓ_i is at most $1/i$. Note that this bound holds irrespective of the choice of P^i. This establishes the claim at the end of the last paragraph.

In summary we have three cases, $E[\ell_i \mid M = j] \le \langle 0, 1, 1/i \rangle$ for $i > j$, $i = j$, and $i < j$, respectively, from which we obtain

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

Substituting this in Eq. (1) we obtain

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

Using the fact that the natural logarithm is a concave function and Jensen's inequality, it follows that $E[\ln M] \le \ln E[M]$. In the remainder we will prove that $E[M] \le O(1/p_z)$. Using these facts in Eq. (2) gives the desired bound on $E[\ell]$.

Let w_s and p_s denote respectively the number of pebbles and probability associated with the segment x_s, defined above. Since the algorithm samples W pebbles without replacement, it can be easily shown that the expected number of trials needed to select one of the w_s pebbles associated with x_s is $(W + 1)/(w_s + 1)$. Thus

$$E[M_s] = \frac{W+1}{w_s+1} \le \frac{W}{w_s}.$$

By definition,

$$w_s \;=\; \max(\lceil K p_s n \rceil, 1) \;\ge\; K p_s n \;\ge\; K n p_z / 4,$$

and $W \le (K + 1)n$. It follows that $E[M_s] \le 4(1 + 1/K)/p_z$. Since $M = \max_{1 \le s \le 4} M_s$, it follows that $M \le \sum_{s=1}^{4} M_s$. Thus $E[M] \le \sum_{s=1}^{4} E[M_s] \le 16(1 + 1/K)/p_z$. This completes the analysis of the expected value of the average query time.

Remark: Working out the constants in our analysis, we obtain an upper bound on the expected query time of $(5 \ln 2)H + 5[\ln(1 + 1/K) + \ln 16 + 2]$. Note that as K increases the additive constant in the expected query time decreases. However, since the space used is $O((K + 1)n)$, we obtain a poorer bound on the space.

Remark: Our goal was to minimize average-case query time, and so it is interesting to consider just how bad query times might be for query points lying in cells of very low probability. Interestingly, even for these points the expected query time (averaged over the random order of pebble insertion) is at most $O(\log n)$. Intuitively, this follows from the fact that our pebble-based algorithm can be viewed as being identical to the standard randomized incremental, but with Kn segments, many of which are duplicates. More formally, observe that $w_s \geq 1$ and $W \leq (K+1)n$, and so we have $E[M] \leq 4(K+1)n$. Thus $E[\ell] \leq \ln(n) + \ln(K+1) + \ln 4 + 2$, which implies that the expected length of the search path for q is at most $5\ln(n) + O(1)$, for fixed K. Using this it also follows that the construction time is $O(n \log n)$ in the expected case.

This concludes the proof of Theorem 1.

3.3 Guarantees on Space and Query Time

In the last section, we showed that, in $O(n \log n)$ time, we can construct a data structure of $O(n)$ space that provides average query time $O(H+1)$. The preprocessing time, space, and average query time, all hold in the expected case. In this section we strengthen this result by showing that we can achieve these bounds on space and average query time, in the worst case. In particular, we prove Theorem 2.

Recall that the expectations in our results are computed over the random choices made by the algorithm. Applying Markov's inequality, it is easy to see that the asymptotic bounds on both space and average query time hold with a constant probability. Thus, after constructing the search structure, by computing its space and average query time, and rebuilding if the desired bounds on either space or average query time are violated, we would succeed in finding the desired structure after a constant number of trials. However, it turns out that we cannot determine the average query time precisely, because a cell $z \in S$ may generate a large number of leaves, possibly at different depths, and the input does not specify how the cell probability is distributed among these leaves. We now discuss how to overcome this problem.

First, observe that the average query time is at most $\sum_z p_z d_z^{\max}$, where $d_z^{\max}$ is the maximum depth of any leaf generated from z. Note that the quantity $\sum_z p_z d_z^{\max}$ can be easily computed in $O(n)$ time by traversing the search structure. We will show that, with constant probability, $\sum_z p_z d_z^{\max}$ is $O(H+1)$ and the space used is $O(n)$. Thus, after repeating the construction an expected constant number of times, we will produce a search structure satisfying the average query time and space bounds.

We now give the details. We borrow all the notation and terminology from Section 3.2. In particular, recall that $z \in S$ is the trapezoid that contains the query point q. The four segments defining z are denoted x_1, x_2, x_3, and x_4. The random variable M is defined as $\max_s M_s$, where M_s is the algorithm step in which a pebble associated with x_s is first added.

Lemma 1 *Let $\gamma > 0$ be a parameter and let $M_0 = 4\gamma(1+1/K)\frac{1}{p_z}\ln\frac{2}{p_z}$. Then $\Pr[M > M_0] \leq 4(p_z/2)^\gamma$.*

Proof: By definition, M can exceed M_0 only if at least one of M_1, M_2, M_3, or M_4 exceeds M_0. It follows that $\Pr[M > M_0] \leq \sum_{s=1}^{4} \Pr[M_s > M_0]$. Let w_s denote the number of pebbles associated with segment x_s. The probability that the algorithm fails to select a pebble associated with x_s in M_0 successive trials is at most $(1 - w_s/W)^{M_0}$. Since $w_s \geq Knp_z/4$ and $W \leq (K+1)n$, we obtain

$$\Pr[M_s > M_0] \;\leq\; \left(1 - \frac{Knp_z/4}{(K+1)n}\right)^{M_0}.$$

Using the inequality $1-x \le e^{-x}$, it is easy to show that the quantity on the right hand side is at most $(p_z/2)^{\gamma}$. $\square$

Lemma 2 *Let $\lambda > 0$ be a real parameter and j be any positive integer. Let z be any cell in S. Let $d_z^{\max}$ be the maximum depth of a leaf generated from cell z. Then*

$$\Pr[d_z^{\max} \ge 5(\lambda \ln(2/p_z)+1) \mid M=j] \;\le\; 2j^2(p_z/2)^{\lambda \ln 2}.$$

Proof: Throughout this proof, we compute probabilities conditioned on the assumption that $M=j$. Let the jth pebble be associated with segment x_k, and let $P^{j-1} \subseteq P$ be any fixed set of $j-1$ pebbles that contains at least one pebble associated with the three segments x_s, where $s \in \{1,2,3,4\} - \{k\}$, and no pebble associated with x_k. In the remainder of this proof, we assume that x_k is fixed and P^{j-1} is the set of the first $j-1$ pebbles added. Under these conditions, we will compute an upper bound on $\Pr[d_z^{\max} \ge \lambda \ln(2/p_z)+1]$. Our bound will not depend on the choice of x_k and P^{j-1} and hence will apply irrespective of these two choices (i.e., assuming only that $M=j$).

For the purpose of analysis, imagine that we partition cell z into a set $\mathcal{T}_z$ of at most $2j-1$ trapezoids by passing a vertical line through the endpoints of each segment corresponding to a pebble in P^{j-1}. Let τ denote any trapezoid in $\mathcal{T}_z$. Note that given any search structure built assuming the above conditions on the pebble set (irrespective of insertion order), the search path is the same for all the query points in τ. Let d_τ be a random variable that denotes the length of this search path. Let q be any query point in τ. Define the random variables ℓ, r, t, and b as in the last section. We will show that

$$\Pr[\ell \ge \lambda \ln(2/p_z)+1] \;\le\; j(p_z/2)^{\lambda \ln 2}.$$

Let us assume this for now. By symmetry, a similar probability bound also holds for r, t, and b. Recalling that $d_\tau \le 2\ell + r + t + b$, we obtain

$$\Pr[d_\tau \ge 5(\lambda \ln(2/p_z)+1)] \;\le\; j(p_z/2)^{\lambda \ln 2}.$$

Since $d_z^{\max} = \max_{\tau \in \mathcal{T}_z} d_\tau$ and $|\mathcal{T}_z| \le 2j-1$, it follows that

$$\Pr[d_z^{\max} \ge 5(\lambda \ln(2/p_z)+1)] \;\le\; 2j^2(p_z/2)^{\lambda \ln 2},$$

which is the desired result.

It remains to show that $\Pr[\ell \ge \lambda \ln(2/p_z)+1] \le j(p_z/2)^{\lambda \ln 2}$. We write $\ell = \sum_{i=1}^{W} \ell_i$, where ℓ_i is the random variable that takes the value 1 if the left side of the trapezoid containing q changes when the ith pebble is inserted and is 0 otherwise. Clearly $\ell_j \le 1$ and $\ell_i = 0$ for $i > j$. Thus $\ell \le 1 + \sum_{i=1}^{j-1} \ell_i$. Letting $L = \sum_{i=1}^{j-1} \ell_i$, it is clear that $\Pr[\ell \ge \lambda \ln(2/p_z)+1] \le \Pr[L \ge \lambda \ln(2/p_z)]$.

In order to bound the latter probability, we will use essentially the same method as used in computing a tail estimate for the worst-case query time of the randomized incremental algorithm ([6]). As in the analysis of worst-case query time, a technical difficulty we face is that the ℓ_i's are, strictly speaking, not independent. This can be handled using a standard trick, which allows us to define 0-1 independent and identically distributed random variables $\ell_i', 1 \le i \le j-1$, such that $\ell_i \le \ell_i'$, $\Pr[\ell_i' = 1] = 1/i$, and $\Pr[\ell_i' = 0] = 1 - 1/i$ (see [6] for details). Letting $L' = \sum_{i=1}^{j-1} \ell_i'$, it is clear that $\Pr[L \geq \lambda \ln(2/p_z)] \leq \Pr[L' \geq \lambda \ln(2/p_z)]$. Next we show that $\Pr[L' \geq \lambda \ln(2/p_z)]$ is at most $j(p_z/2)^{\lambda \ln 2}$, which will complete the proof.

For any $t > 0$ we have

$$\Pr\left[L' \geq \lambda \ln \frac{2}{p_z}\right] \;=\; \Pr\left[e^{tL'} \geq e^{t\lambda \ln \frac{2}{p_z}}\right] \;\leq\; e^{-t\lambda \ln \frac{2}{p_z}} E[e^{tL'}] \;=\; (p_z/2)^{\lambda t} E[e^{tL'}], \tag{3}$$

where we have used Markov's inequality in the second to last step. Further, since the ℓ'_i are independent,

$$E[e^{tL'}] \;=\; E\left[e^{\sum_{i=1}^{j-1} t\ell'_i}\right] \;=\; E\left[\prod_{i=1}^{j-1} e^{t\ell'_i}\right] \;=\; \prod_{i=1}^{j-1} E[e^{t\ell'_i}].$$

Setting $t = \ln 2$, we obtain

$$E[e^{t\ell'_i}] \;=\; e^t \cdot \frac{1}{i} + e^0 \cdot \left(1 - \frac{1}{i}\right) \;=\; \frac{1+i}{i}.$$

Thus $E[e^{tL'}] = \prod_{i=1}^{j-1} E[e^{t\ell'_i}] = j$. Substituting this in Eq. (3), we have the desired result. $\square$

Lemma 3 *Let $\lambda > 0$ be a real parameter. Let z be any cell in S. Let $d_z^{\max}$ be the maximum depth of a leaf generated from cell z. Then there exist positive constants c_1 and c_2 (depending only on K) such that*

$$\Pr[d_z^{\max} \geq 5(\lambda \ln(2/p_z) + 1)] \leq (p_z/2)^{c_1\lambda - c_2}.$$

Proof: Let M_0 be as defined in Lemma 1. We have

$$\Pr[d_z^{\max} \geq 5(\lambda \ln(2/p_z) + 1)] \;\leq\; \Pr[M > M_0] + \Pr[d_z^{\max} \geq 5(\lambda \ln(2/p_z) + 1) \mid M \leq M_0].$$

By Lemma 1, the first term is at most $4(p_z/2)^\gamma$, and by Lemma 2 the second term is at most $2M_0^2(p_z/2)^{\lambda \ln 2}$. Setting $\gamma = \lambda$ and simplifying, the lemma follows. $\square$

Lemma 4 *There exists a constant c such that, with probability at least $3/4$, for all cells $z \in S$, $d_z^{\max} \leq c(\ln(1/p_z) + 1)$.*

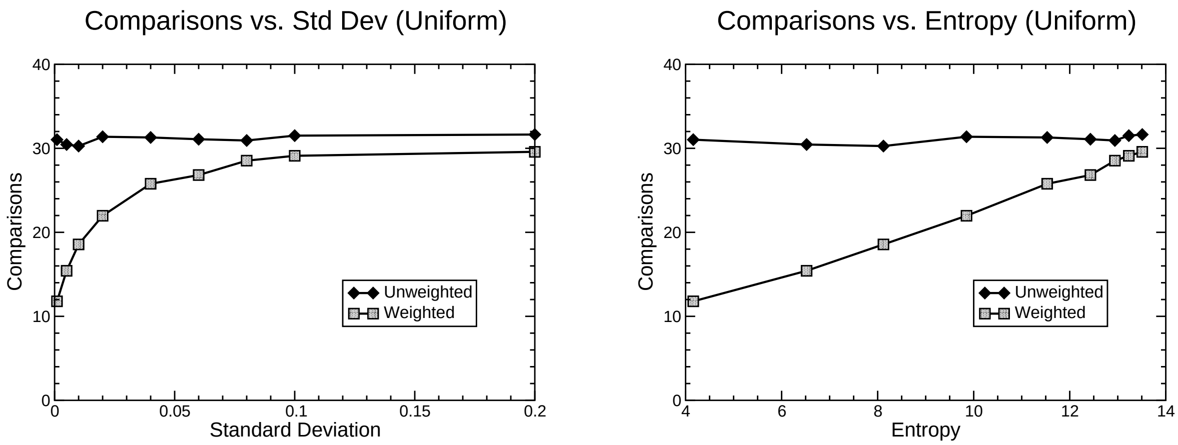
Proof: Setting λ to be a sufficiently large constant in Lemma 3, it follows that $\Pr[d_z^{\max} \geq 5(\lambda \ln(2/p_z) + 1)] \leq (p_z/2)^2$. Thus the probability that $d_z^{\max} \geq 5(\lambda \ln(2/p_z) + 1)$ for some cell z is at most $\sum_{z \in S}(p_z/2)^2 \leq 1/4$. The lemma now follows. $\square$

An immediate corollary of Lemma 4 is that $\sum_{z \in S} p_z d_z^{\max} = O(H + 1)$ with probability at least $3/4$. By Markov's inequality, the space used is also $O(n)$ with probability, say, at least $3/4$. Thus the bounds on both space and $\sum_{z \in S} p_z d_z^{\max}$ hold with probability at least $1/2$. By repeating the construction expected constant number of times, we obtain a linear size data structure that guarantees $O(H + 1)$ expected query time. This concludes the proof of Theorem 2.

4 Experimental Results

We have run experiments on the weighted randomized incremental algorithm, comparing its performance to the standard unweighted version. We measured the query time of the algorithm by counting the average number of comparisons needed to answer a query and the space used by the search structure. We tested the method on a number of randomly generated Delaunay triangulations. Rather than using the weighting scheme described in Section 2, we instead chose to use a slightly different weighting method, which we felt is more natural for triangulations. For each cell z we assigned a weight of $p_z/3$ to each of its three sides, where p_z is the probability that the query point lies in z. For each edge x, let p_x denote the sum of its assigned weights. The number of pebbles assigned to x was then chosen to be $w_x = \lceil 5p_x n \rceil$. (Thus $K = 5$ in our implementation.)

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

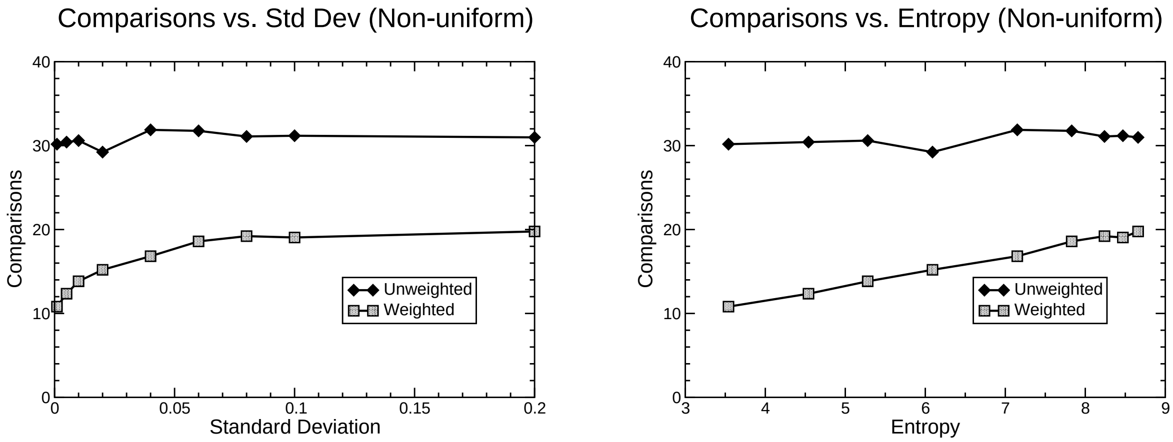


(a) (b)

Fig. 4: Uniform subdivision: Number of comparisons versus (a) standard deviation and (b) entropy.

For both subdivision types, the query points were generated from a clustered Gaussian distribution with ten uniformly distributed centers. (The centers are different from the ones used in generating data points for the clustered Gaussian case.) We obtained different distributions by varying the standard deviation of the Gaussian distribution from 0.001 to 0.2, which gave us a way of controlling the entropy of the subdivision. Intuitively, lower standard deviations correspond to lower entropies.

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



(a) (b)

Fig. 5: Non-uniform subdivision: Number of comparisons versus (a) standard deviation and (b) entropy.

it as a function of the standard deviation and the entropy of the subdivision.

The results for the uniform and non-uniform subdivision are shown in Figs. 4 and 5, respectively. We summarize the key observations:

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

(b) The average number of comparisons for the unweighted algorithm shows no significant relation to the standard deviation (and hence the entropy) of the distribution. In contrast the average number of comparisons for the weighted algorithm appears to grows linearly with the entropy. (By fitting a line to the data, it appears that the average number of comparisons grows as $1.94H + 3.11$ for the uniform subdivision, and as $1.75H + 4.49$ for the non-uniform subdivision. This suggests that the actual performance of the algorithm is better than the bound of about $3.47H + 24.77$ predicted by our theoretical analysis.)

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

5 Acknowledgements

We would like to thank Raimund Seidel for helpful comments and pointing out an error in the analysis of Section 3.2 in an earlier version of this paper.

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, 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.

[3] 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.

[4] S. Arya, T. Malamatos, D. M. Mount, and K. C. Wong. Optimal expected-case planar point location. Technical Report HKUST-TCSC-2004-09, Department of Computer Science, HKUST, 2004.

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

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

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

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

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

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

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

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

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

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

[15] 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.

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

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