Appeared in SODA 2001, pp. 262-268

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. 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. This problem has been considered before, but existing data structures are all quite complicated. 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 [8]. This was followed by a number of related methods with better practical performance by Edelsbrunner, Guibas, and Stolfi [7], Cole [6], and Sarnak and Tarjan [13]. 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 [11] and Seidel [14]. 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 involves at most $5 \ln n + O(1)$ comparisons.[1] Here the expectation is taken over all random permutations of the segments.

All of this work was done in terms of worst-case query times. An important variant is the average-case performance. We assume that for each polygon $z \in S$ we are given the probability p_z that a query point lies in z. As is common in computational geometry, we will assume that the probability that the query point lies on an edge or vertex of the subdivision or lies outside the subdivision is zero. We make no assumptions about the distribution of query points within each polygon, but we will introduce this later with some of our results.

The *entropy* of the S, denoted H throughout, is defined

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

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 [9, 15]. It has also been known for many years [10] that it is possible to construct a binary search tree whose average search time is at most $H + 2$.

Arya, Cheng, Mount, and Ramesh [2] showed that for subdivisions consisting of convex polygons, assuming that the x and y coordinates of the query point are chosen independently from some probability distribution, the entropy bound can be achieved to within a constant multiplicative factor (2 using quadratic space and about 4 using linear space). These results were strengthened by Arya, Malamatos, and Mount [3] for the case of polygonal subdivisions in which each cell has constant complexity. They presented an algorithm which answers queries in $H + O(H^{2/3} + 1)$ average time and $O(n \log n)$ space. Recently these same authors have announced a nearly optimal expected query time of $H + O(H^{2/3} + 1)$ using nearly optimal $O(n \log^* n)$ space [4].

*Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. Email: {arya,tmalamat}@cs.ust.hk. Research supported in part by a grant from the Hong Kong Research Grants Council.

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

[1] We use log to denote the base-2 logarithm and ln to denote the natural logarithm.

All of the above methods rely on relatively complex constructions. In contrast, Mehlhorn's nearly optimal binary search tree for the 1-dimensional problem [10] 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 present a positive answer, by presenting a simple weighted variant of the randomized incremental algorithm. Our main result is given below. To minimize confusion we use the term *expected* when referring to variations that are caused by the the random choices made by the algorithm and *average* when referring to variations due to the random distribution of query points.

THEOREM 1.1. *Consider an n-segment polygonal subdivision S in which each cell z has constant combinatorial complexity and is associated with a probability p_z that the query point falls within this cell. There is a randomized algorithm with expected running time $O(n \log n)$ that produces a data structure of expected space $O(n)$ such that for any fixed query point q, the expected time needed to locate q is at most $5 \ln(1/p_z) + O(1)$, where z is the cell containing q. (In all cases, the expectation is over the random choices made by the algorithm.) Thus the average query time (over the query distribution) is at most $(5 \ln 2)H + O(1)$.*

As in the standard randomized incremental algorithm, our analysis of the space and query times are based on a standard backwards analysis. Such analyses are 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. As a result, the analysis of both the space and expected query time is significantly more complicated than in a standard backwards analyses. 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 when its first pebble is drawn.

We assume for simplicity that no edge of S is vertical (but this assumption is easy to overcome). We make use of two standard types of *comparisons* in locating a query point, and measure the average query time in terms of the number of such comparisons needed. The first determines whether the query point lies to the left or right of 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 Adamy and Seidel [1] in their lower bound, and is used in all the common point location algorithms.

2 Weighted Randomized Incremental Algorithm

Before describing our algorithm, we present a quick review of trapezoidal maps. We refer the reader to the text by de Berg, et al. [5] for more detailed information. The algorithm can be applied to any set of line segments with disjoint interiors, and we will describe it assuming that the input is in this form. We assume that the segments are enclosed within a bounding rectangle. The trapezoidal map is formed by firing two bullet paths vertically up and down from the endpoints of each segment until hitting another segment or the bounding box. Each cell of the resulting subdivision is a (possibly degenerate) trapezoid, bounded by at most two vertical sides and a top and bottom segment.

The randomized incremental algorithm operates by inserting the segments one by one in random order and updating the subdivision after each insertion. (See Fig. 1.) The associated point location data structure contains three types of nodes: *left-right nodes*, each associated with the endpoint of some segment (indicated by a circle in the figure), *above-below nodes*, each associated with a segment (indicated by a hexagon in the figure), and *leaf nodes*, each associated with a trapezoidal cell (indicated by a square node in the figure). Left-right nodes test whether the query point's x-coordinate lies to the left or right of the x-coordinate of the associated endpoint and branch to the left or right child, respectively. Above-below nodes test whether the query point lies above or below of the line containing the associated segment, and branch to the left or right child, respectively.

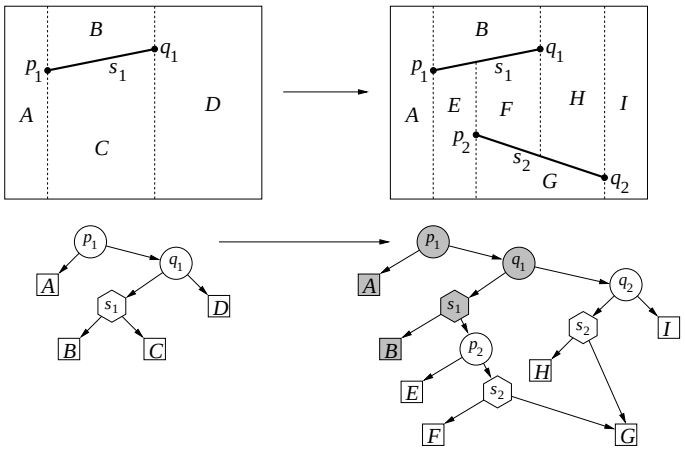


Figure 1: Incremental algorithm for trapezoidal map and the associated history graph.

The insertion of a segment results in the creation of four new bullet paths and some of the existing bullet paths being blocked by the new segment. (See the right side of Fig. 1.) This in turn causes some of the existing trapezoidal cells to be replaced with new trapezoids. For each trapezoid which has been replaced, the incremental algorithm replaces the corresponding leaf node in the history graph with the appropriate set of tests to ascertain which new trapezoid the query point now lies in. (The newly added nodes in the history graph are shown unshaded in the right side of Fig. 1.)

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 edges of the subdivision one by one in random order. Intuitively, we want the edges bounding triangles with high probability to be added early in the process, since then any query that falls within this triangle will have its location resolved near the root of the history graph. This suggests a *simple weighting scheme* in which each edge of the subdivision is assigned a weight that is derived from the probability that the query point lies in either of its incident triangles, say the sum of these two probabilities. Then we apply the incremental algorithm, but insert the edges in decreasing order of weight (rather than using a random permutation).

The problem is that an adversary can adjust the probabilities to cause the algorithm to insert the edges in an order so that the space of the resulting structure would be $\Theta(n^2)$. Consider the example shown in Fig. 2. 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 figure only shows the relevant triangles, but the subdivision can be completed to a triangulation through the addition of triangles of zero probability.) The insertion order provided by the simple weighting scheme results in the insertion of edges bounding the $\Theta(n)$ lower triangles, thus creating $\Theta(n)$ vertical slabs. After this, the edges 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 which samples according to the given weights will result in a substantially similar situation.

The source of the problem comes from the huge variation that is possible in 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

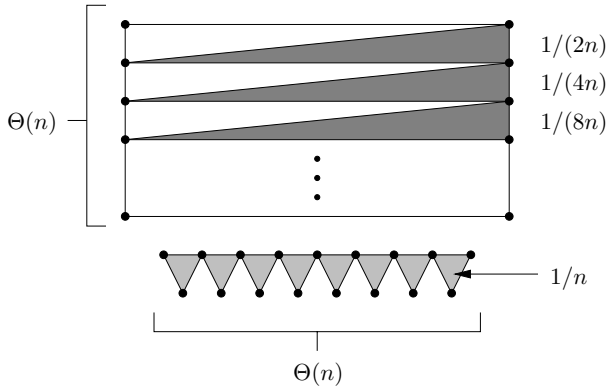


Figure 2: A bad instance for simple weighting.

scheme so that the ratio of any two segment weights is $O(n)$. Remarkably, this simple fix is all that that is needed to eliminate the above space problem, and yet it provides a true enough representation of the probability distribution to achieve the desired entropy bounds on average search times.

In detail our algorithm works as follows. For the sake of simplicity we assume that the given subdivision S is a triangulation (we can easily generalize our results to any subdivision in which the cells have bounded complexity). Let m denote the number of triangles in S. Let X denote the set of edges of these triangles. Let $n = O(m)$ denote the number of segments in X.

Let p_z denote the probability of the query point lying in a cell $z \in S$. Assign a probability of $p_z/3$ to each of the three segments bounding the triangle z. For a segment x, let p_x denote the sum of the two probabilities it derives from its incident triangles. Assign a weight $w_x = \lceil K p_x n \rceil$ to each segment x, where K is any constant. (We will see that the choice of K affects only the multiplicative constant in the space bound and the additive constant in the query time). After this weight assignment we run the standard randomized incremental algorithm in which we sample the segments based on their weights.

3 Analysis

We now analyze the space and average query time of the search structure. To simplify the presentation we assume that no two distinct endpoints of segments have the same x-coordinate; however two or more segments are allowed to share an endpoint. 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 $(K+1)n$ (because $\sum_x w_x \leq \sum_x (K p_x n + 1) \leq (K+1)n$).

For the purpose of analysis, it is useful to consider

the following randomized incremental algorithm, which can be easily verified is 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 structural change 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. 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 trapezoidal map for the set of segments associated with the pebbles in P^i. For a trapezoid $\triangle \in \mathcal{T}(P^i)$, let $\delta(\triangle, p) = 1$ if pebble p would have caused $\triangle$ 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 first i pebbles, is given by

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

Reversing the order of summation,

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

Note that each trapezoid $\triangle \in \mathcal{T}(P^i)$ is defined by at most two segments that bound its top and bottom walls, and two endpoints whose vertical extensions form its left and right walls. Clearly the trapezoid $\triangle$ 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 defines the top wall of $\triangle$ and this pebble is inserted last; (ii) same as (i) for the bottom wall; (iii) there is exactly one pebble in P^i such that an endpoint of the associated segment defines the left wall of $\triangle$ and this pebble is inserted last; and (iv) same as (iii) for the right wall. Thus there are at most four pebbles in P^i with the property that if they were inserted last they would have created $\triangle$, i.e., $\sum_{p \in P^i} \delta(\triangle, p) \leq 4$ and we get

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

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 $\mathcal{T}(P^i)$ contains at most $O(i)$ trapezoids. Since 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 $W \leq (K+1)n$, 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. (Recall 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 over the random choices made by the algorithm.)

We call two query points 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 $\sum_{\triangle \in \mathcal{T}} p_\triangle d_\triangle$, where $p_\triangle$ is the probability of the query point lying in $\triangle$, and $d_\triangle$ is the length of the search path for the query points in $\triangle$.

Let q be a query point inside a trapezoid $y \in \mathcal{T}$. Let y be contained within triangle $z \in S$. We will prove that the expected length of the search path for q, $E[d_y]$, is at most $5\ln(1/p_z) + O(1)$. It is easy to see that this implies that the expected value of the average query time is $(5\ln 2)H + O(1)$.

Let ℓ_i be a random variable that takes the value 1 if the left wall 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 walls, respectively. We employ the following observation made by Seidel [14]: 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. Let $\ell = \sum_{1 \leq i \leq W} \ell_i$. Similarly define r, t, and b. By linearity of expectation, $E[d_y]$ is at most $2E[\ell] + E[r] + E[t] + E[b]$. 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 $E[d_y]$.

We will only prove that $E[\ell] = \ln(1/p_z) + O(1)$; the other bounds can all be proved in a similar way. Let u be the trapezoid in the final trapezoidal decomposition $\mathcal{T}(X)$ that contains q. Let u be contained within triangle $z \in S$. Let a_1, a_2, and a_3 denote the three sides of triangle z. Let $M_r, 1 \leq r \leq 3$, be a random variable denoting the step of the algorithm in which a pebble associated with segment a_r is first inserted, and let M be the random variable $\max_{1 \leq r \leq 3} M_r$. We can write $E[\ell]$ as follows:

$$E[\ell] = \sum_{j=1}^{W} \Pr[M = j] \cdot E[\ell | M = j]. \tag{3.1}$$

In order to compute $E[\ell | M = j]$ observe first that the trapezoid containing q does not change after a pebble associated with each side of z has already been inserted. Thus $E[\ell_i | M = j]$ for $i > j$ is 0. Obviously $E[\ell_j | M = j] \leq 1$ since the largest value that the random variable ℓ_j can take is one.

We next prove that $E[\ell_i | M = j] \leq 1/i$ for $i < j$. This analysis is based on backwards analysis. Suppose that the jth pebble inserted is associated with segment a_k, where $k \in \{1, 2, 3\}$. Let $P^{j-1} \subseteq P$ be any fixed set of $j-1$ pebbles that contains at least one pebble associated with the two segments a_r, $r \in \{1, 2, 3\} - \{k\}$, and no pebble associated with a_k. We claim that the expected value of ℓ_i subject to the condition that P^{j-1} is the set of the first $j-1$ pebbles inserted is at most $1/i$, irrespective of the choice of a_k and P^{j-1}. By the definition of M, it is clear that this claim would imply that $E[\ell_i | M = j] \leq 1/i$ 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 $\triangle$ be the trapezoid in $\mathcal{T}(P^i)$ that contains q. The left wall of $\triangle$ 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 defines the left wall of $\triangle$ 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.

Combining the three cases: (i) $E[\ell_i | M = j] = 0$ for $i > j$, (ii) $E[\ell_j | M = j] \leq 1$, and (iii) $E[\ell_i | M = j] \leq 1/i$ for $i < j$, we get

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

Substituting this in Eq. (3.1) we obtain

$$\begin{aligned} E[\ell] &\leq \sum_{j=1}^{W} \Pr[M = j] \cdot (\ln(j) + 2) \\ &= E[\ln(M)] + 2. \end{aligned} \tag{3.2}$$

Since $\ln()$ is a concave function, $E[\ln(M)] \leq \ln(E[M])$. In the remainder we will prove that $E[M] \leq O(1/p_z)$. Using these facts in Eq. (3.2) gives the desired bound on $E[\ell]$.

Since the algorithm samples W pebbles without replacement, it is easy to show that the expected number of trials needed to select one of the w_{a_r} pebbles associated with a_r is $(W+1)/(w_{a_r}+1)$. Thus $E[M_r] = (W+1)/(w_{a_r}+1) \leq W/w_{a_r}$. Since $w_{a_r} = \lceil K p_{a_r} n \rceil \geq K p_{a_r} n = K n p_z/3$ and $W \leq (K+1)n$, it follows that $E[M_r] \leq 3(1 + 1/K)/p_z$. Since $M = \max_{1 \leq r \leq 3} M_r$, it follows that $M \leq \sum_{r=1}^{3} M_r$. Thus $E[M] \leq \sum_{r=1}^{3} E[M_r] \leq 9(1 + 1/K)/p_z$. 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 $(5 \ln 2)H + 5[\ln(1 + 1/K) + \ln 9 + 2]$. Note that as K increases the additive constants in the expected query time decrease. However, since the space used is $O((K+1)n)$, we obtain a poorer guarantee on the space.

Remark: Our analysis also shows that for any fixed query point the expected query time is at most $O(\log n)$. Since $w_{a_r} \geq 1$ and $W \leq (K+1)n$, so $E[M] \leq 3(K+1)n$. Thus $E[\ell] \leq \ln(n) + \ln(K+1) + \ln 3 + 2$, which implies that the expected length of the search path for q is $5 \ln(n) + O(1)$. Using this it can be shown that the construction time is $O(n \log n)$ in the expected case.

Remark: Mulmuley introduced a worst-case point location data structure based on generalizing the idea of skip lists to two dimensions [12]. By making a small modification to his method, we get another way of achieving $O(H)$ expected time and $O(n)$ space. We briefly describe the main idea, assuming the reader is familiar with the skip list approach. Our idea to build a weighted skip list by associated pebbles with the segments. (Exactly as in the weighted randomized incremental algorithm.) Starting with all the pebbles at the bottom level, we randomly promote half of the pebbles at each level to the next higher level. A segment is stored at all the levels at which an associated pebble is present. The rest of the construction and the search algorithm are equivalent to Mulmuley's method, except that we terminate the search as soon as we reach a trapezoid whose interior is not intersected by any segment in the subdivision. Details will be left for the full version of this paper.

4 Experimental Results

We have run preliminary 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. For the weighted algorithm we used $K = 5$ for the weight assignment (i.e., $w_x = \lceil 5p_x n \rceil$).

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).

The query points were generated from the Clustered Gaussian distribution with ten centers. 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.

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 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 Figure 3 and 4, 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 (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 relation to the standard deviation (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 $3.47H + 21.90$ 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 pointing out an error in the analysis of Section 3.2 in an earlier version of this paper.

References

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

[2] S. Arya, S.-W. Cheng, D. M. Mount, and H. Ramesh. Efficient expected-case algorithms for planar point location. In *Proc. 7th Scand. Workshop Algorithm Theory*, volume 1851 of *Lecture Notes Comput. Sci.*, pages 353–366. Springer-Verlag, 2000.

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

[4] S. Arya, T. Malamatos, and D. M. Mount. Entropy-preserving cuttings and space efficient planar point location. In *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, 2001. (to appear).

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

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

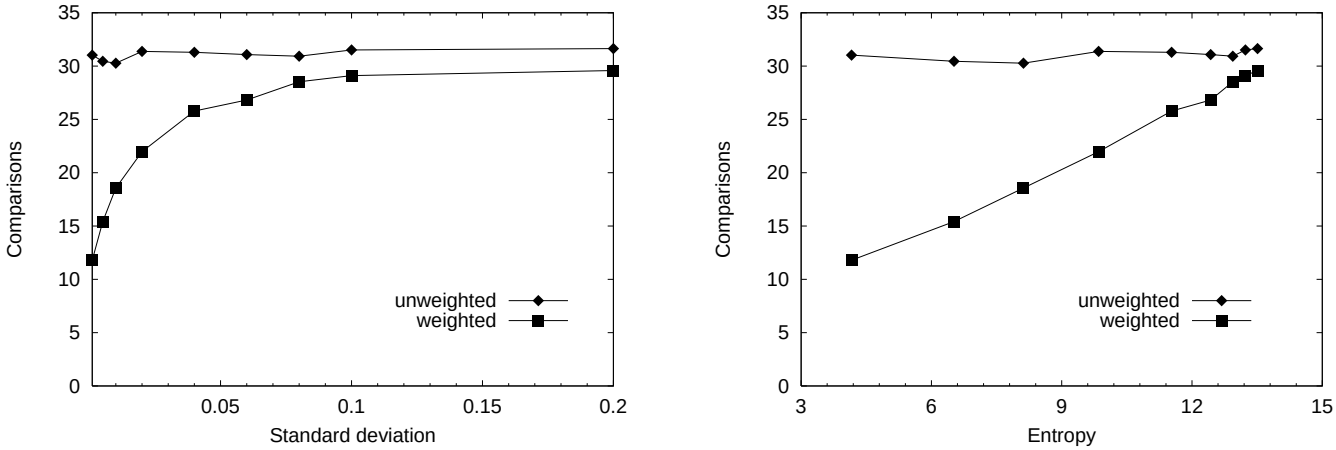
[7] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986.

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

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

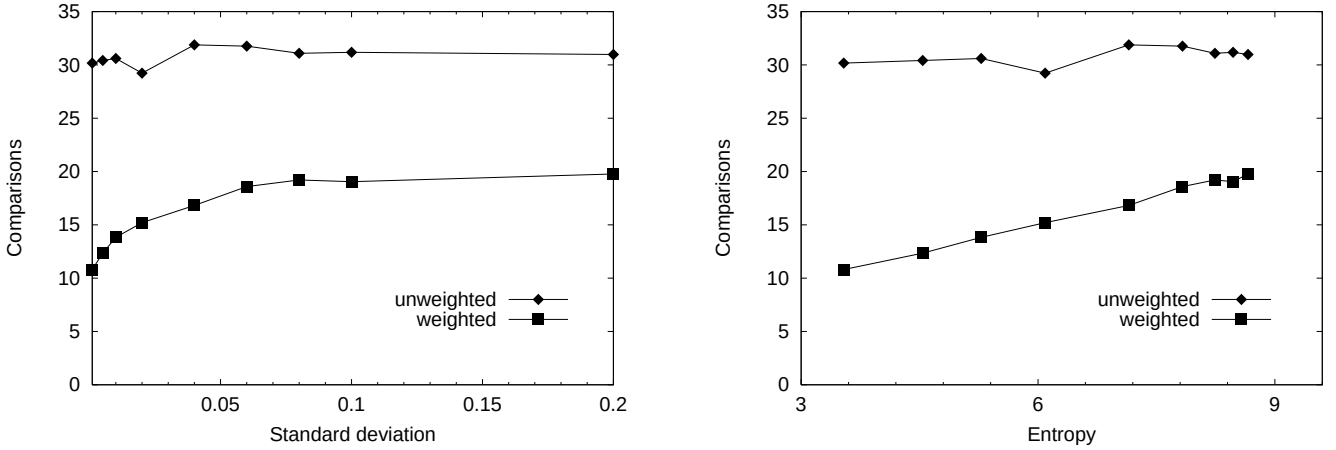
[10] K. Mehlhorn. Best possible bounds on the weighted path length of optimum binary search trees. *SIAM J. Comput.*, 6:235–239, 1977.

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



(a) (b)

Figure 3: Uniform subdivision: Comparisons versus (a) standard deviation and (b) entropy.



(a) (b)

Figure 4: Non-uniform subdivision: Comparisons versus (a) standard deviation and (b) entropy.

[12] K. Mulmuley. Randomized multidimensional search trees: Dynamic sampling. In *Proc. 7th ACM Sympos. on Computational Geometry*, pages 121–131, 1991.

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

[14] 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.

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