A Static Optimality Transformation with Applications to Planar Point Location

John Iacono Wolfgang Mulzer

November 9, 2012

Abstract

Over the last decade, there have been several data structures that, given a planar subdivision and a probability distribution over the plane, provide a way for answering point location queries that is fine-tuned for the distribution. All these methods suffer from the requirement that the query distribution must be known in advance.

We present a new data structure for point location queries in planar triangulations. Our structure is asymptotically as fast as the optimal structures, but it requires no prior information about the queries. This is a 2-D analogue of the jump from Knuth's optimum binary search trees (discovered in 1971) to the splay trees of Sleator and Tarjan in 1985. While the former need to know the query distribution, the latter are *statically optimal*. This means that we can adapt to the query sequence and achieve the same asymptotic performance as an optimum static structure, without needing any additional information.

1 Introduction

We consider the problem of finding a statically optimal data structure for planar point location in triangulations. This problem and related problems have a long history that goes back to the dawn of computer science. Thus, before giving a formal description of the problem and of our results, let us first provide some background on the history and motivation behind our work.

1.1 1-D History

Comparison-based predecessor search constitutes one of the oldest problems in computer science: given a set S from a totally ordered universe U, we would like to construct a data structure for answering *predecessor queries*. In such a query, we are given an element $x \in U$, and we need to return the largest $y \in S$ with $y \leq x$ (or $-\infty$, if no such y exists). In the most general *decision-tree* model, we are allowed to evaluate in each step an *arbitrary* function $f : U \to \{0, 1\}$ on x, where the choice of f may depend on the outcomes of the previous evaluations. The classic solution sorts S during preprocessing and answers queries in $O(\log n)$ steps through binary search, where n denotes the size of S. Information theoretic arguments imply that any such comparison-based algorithm requires $\Omega(\log n)$ steps in the worst case (see, e.g., Ailon et al. [2, Section 2] for more details).

However, the story does not end here. Early in the history of computer science, researchers realized that if the distribution of query outcomes is sufficiently biased, $o(\log n)$ expected-time query processing becomes possible. This insight led to the invention of *optimal search trees*. These are specialized data structures for the case that the query outcomes are drawn independently from a

known fixed distribution, and a wide literature studying their variants and extensions have been developed [10, 22–24, 28, 33, 35–37, 41–44, 51, 52]. In this context, optimality is characterized by the *entropy* of the distribution: if p_i denotes the probability of the ith outcome, the entropy $\mathcal{H}$ is defined as $\sum_i -p_i \log_2 p_i$. Information theory [48] shows that $\mathcal{H}$ is a lower bound for the expected number of steps that any comparison-based algorithm needs to answer a predecessor query, assuming that the searches are drawn independently from a fixed distribution (e.g., [2, Claim 2.2]).

All the above results require that the distribution, or a suitable approximation thereof, be known in advance. This situation changed in 1985, when Sleator and Tarjan [49] introduced *splay trees*. These trees have many amazing properties, not the least of which is called *static optimality*. This means that for any sufficiently long query sequence, splay trees are asymptotically as fast as optimal static search trees. For this, splay trees require no prior information on the query distribution.

1.2 2-D History

Planar point location is a fundamental problem in computational geometry. A *triangulation* S is a partition of the plane into (possibly infinite) triangles. Given S, we need to construct a data structure for *point location queries*: given a point $p \in \mathbb{R}^2$, return the triangle of S that contains it. Again, we use a decision-tree model. This means that in each step we may evaluate an arbitrary function $f : \mathbb{R}^2 \to \{0, 1\}$ on p, where f may depend on the previous comparisons.

There are several point location structures with $O(\log n)$ query time, which is optimal in our decision-tree model. These structures are notable not only for achieving optimality, but for doing so through very different methods, such as planar separators [39,40], Kirkpatrick's successive refinement approach [34], persistence [46], layered DAGs [20], or randomized incremental construction [45, 47].

Once again, it makes sense to consider biased query distributions. For a known fixed distribution of point location queries, there are several data structures that achieve optimal expected query time, assuming independence. These *biased* structures are analogous to optimal search trees. Thus, we can use the same information theoretic arguments to characterize the optimal expected query time by the entropy $\mathcal{H}$ of the probabilities of the queried regions [2, Claim 2.2].

A series of papers by Arya et al. [3–8] converge on two algorithms. The first one achieves query time $\mathcal{H} + O(\sqrt{\mathcal{H}} + 1)$ with $O(n)$ space, while the second, simpler, algorithm supports queries in time $(5 \ln 2)\mathcal{H} + O(1)$ and $O(n \log n)$ space.[1] The latter algorithm is a truly simple variant of randomized incremental construction [45, 47], where the random choices are biased according to the distribution. Both structures are randomized and have superlinear construction costs. Iacono [31] presented a data structure that supports $O(\mathcal{H})$ time queries in $O(n)$ space, but, unlike the aforementioned results, it is deterministic, can be constructed in linear time, and has terrible constants.

1.3 Creating a point location structure that is statically optimal

In view of the developments for binary search trees, one question presents itself: Is there a point location structure that is asymptotically as fast as the biased structures, *without* explicit knowledge of the query distribution? Or, put differently, can a point location structure achieve a running time similar to the static optimality bound of splay trees? This open problem, which we resolve here, explicitly appears in several previous works on point location, e.g., in Arya et al. [8, Section 6]:

[1] In this context, *query time* refers to the expected depth of the associated decision tree.

Taking this in a different direction, suppose that the query distribution is not known at all. That is, the probabilities that the query point lies within the various cells of the subdivision are unknown. In the 1-dimensional case it is known that there exist self-adjusting data structures, such as splay trees, that achieve good expected query time in the limit. Do such self-adjusting structures exist for planar point location?

There are several possible approaches towards statically optimal point location. One, suggested above, would be to create some sort of self-adjusting point location structure and to analyze it in a way similar to splay trees. This has not been done; we suspect that the main stumbling block is that all known efficient structures are *comparison DAGs* [20, 34, 45–47]: they can be represented as a directed acyclic graph with a unique source and out-degree 2, such that each node corresponds to a planar region. A point location query proceeds by starting at the source and by following in each step an edge that is determined by comparing the query point with a fixed line. The query continues until it reaches a sink, whose corresponding region constitutes the desired query outcome. In order to achieve reasonable space usage, it seems essential to use a DAG instead of a simple tree. Unfortunately, we do not know how to perform rotation-like local changes in such DAGs that would mimic the behavior of splay trees.

Another possible avenue is to use splay trees in an existing structure. Goodrich et al. [26] followed this approach, using essentially a hybrid of splay trees and the persistent line-sweep method. Unfortunately, their method does not give a result optimal with respect to the entropy of the original distribution of query outcomes, but rather to the entropy of the probabilities of querying regions of a *strip decomposition* of the triangulation. The latter is obtained by drawing vertical lines through every point of the triangulation. This strip decomposition could split a high-probability triangle into several parts and could potentially increase the entropy of the query result by $\Omega(\log n)$, the worst possible; see Figure 1 for an example.

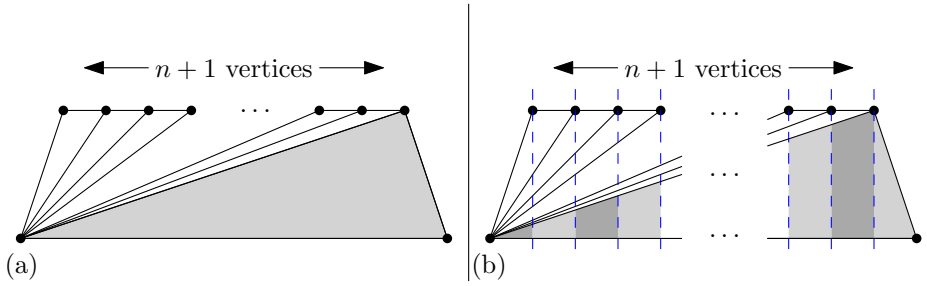

Figure 1: A bad example for the strip decomposition of [26]: (a) We have $n+3$ vertices and $n+1$ triangles. The small triangles each have query probability $1/n^2$, the large, shaded, triangle has query probability $1-1/n$. The entropy is $(1/n)\log n^2 + (1-1/n)\log(n/(n-1)) = O(1)$. (b) The strip decomposition partitions the large triangle into $n+1$ parts. Suppose each part has probability $(n-1)/n(n+1) \approx 1/n$. The resulting entropy is larger than $(1-1/n)\log(n(n+1)/(n-1)) = \Omega(\log n)$.

One might also try to create a structure with the *working set property*. This property, originally used in the analysis of splay trees, states that the processing time for a query q is logarithmic in the number of distinct queries since the last query that returned the same result as q. The working set property implies static optimality [30]; it has also proved useful in several other contexts [15, 16, 21, 29, 32]. Most importantly, there is a general transformation from a *dynamic*

$O(\log n)$ time structure into one with the working set property [30]. Unfortunately, even though several dynamic data structures for predecessor searching are known (e.g., AVL trees [1] or red-black trees [27]), it remains a prominent open problem to develop a point location structure that supports insertions, deletions, and queries in $O(\log n)$ time. (Note that [50] claims to modify Kirkpatrick's method to allow for $O(\log n)$ time insertions, deletions, and queries. The claimed result is wrong. [2])

Our solution to the problem of statically optimal point location is very simple: we take a biased structure that needs to be initialized with distributional information, and we rebuild it periodically using the observed frequencies for each region. We do not store all the regions in the biased structure—this would make the rebuilding step too expensive. Instead, upon rebuilding we create a structure storing only the n^β most frequent items observed so far, where $\beta \in (0, 1)$ is some suitable constant. We resort to a static $O(\log n)$ time structure to complete queries for the remaining regions. The rebuilding takes place after every n^α queries, for some constant $\alpha \in (\beta, 1 - \beta)$. This is a simple and general method of converting biased structures into statically optimal ones, and it enables us to waive the requirement of distributional knowledge present in all previous biased point location structures, at least for triangulations. Our approach can be seen as a generalization and simplification of a method by Goodrich for dictionaries [25].

2 Notation

Let U be some universal set, and let S be a partition of U into n pieces. The elements of U are called *points*, the subsets in S are called *regions*. A *location query* takes some point $p \in U$ and returns the region $s \in S$ with $p \in s$. The result of a location query with input p is denoted by $q(p)$. A data structure for location queries is called a *location query structure*.

Let $P = \langle p_1, p_2, \ldots, p_m \rangle$ be a sequence of m queries, and let $Q := \langle q(p_1), q(p_2), \ldots, q(p_m) \rangle$ denote the results of these queries. Let $f_t(s)$ be the number of occurrences of s in the first t elements of Q, and define $f(s) := f_m(s)$, the number of times s occurs in the entire sequence. Furthermore, let $t_j(s)$ be the time of the j^{th} occurrence of s in Q; thus $f_{t_j(s)}(s) = j$.

We use $\log x$ to refer to $\max(1, \log_2 x)$; this avoids clutter generated by additive terms that would otherwise be needed to handle degenerate cases of our analysis. We next define the notion of a biased structure.

Definition 2 *Let S be set of n regions, and let D be a location query structure for S. We say that D is* biased *if the following holds: There exists a function $c_D : \mathbb{N} \to \mathbb{N}$ such that given any weight function $w : S \to \mathbb{R}^+$, D executes any query sequence P in total time*

$$O\left(c_D(n) + \sum_{s \in S} f(s) \log \frac{\sum_{r \in S} w(r)}{w(s)}\right).$$

The function c_D is called the construction cost *of the structure.*

Suppose we choose $w(s)$ proportional to the number of queries that return the given region, e.g., $w(s) := f(s) + 1$. In this case, a biased location query structure achieves an amortized query time that is (of the order of) the entropy $\mathcal{H}$ of the query distribution. As we argued in the introduction, this is optimal for our decision-tree model. We now define the notion of static optimality.

[2]The method presented makes the assumption that given a triangle T in a triangulation of size n on which a Kirpatrick hierarchy has been built, the complexity of the intersection of T with any level of the hierarchy is constant; this is false as examples where the intersection has size $\sqrt{n}$ are easy to produce.

Definition 3 *Let S be set of n regions, and let D be a location query structure for S. We say that D is* statically optimal *if there exists a function $c_D : \mathbb{N} \to \mathbb{N}$ such that D executes any query sequence P of length m in total time*[3]

$$O\left(c_D(n) + \sum_{s \in S} f(s) \log \frac{m}{f(s)}\right).$$

We call c_D the construction cost *of D.*

Note that a statically optimal structure is given neither the frequency function f nor any weights in advance, in particular, the structure does not need to be static.

We provide a simple method for making a biased location query structure statically optimal, assuming a few technical conditions. The main such condition is that we should be able to construct the biased query structure not just on the set S, but on any subset S' of S. We require that a location query structure for S' performs as quickly as a biased structure for S when a region in S' is queried, and that it reports failure in $O(\log n)$ time if the query lies outside of S'. Formally:

Definition 4 *Let S be set of n regions, and let D be a location query structure. We call D* subset-biased *on S if the following holds: there exists a function $c'_D : \mathbb{N} \to \mathbb{N}$ such that given a subset $S' \subseteq S$ of size n' and a weight function $w' : S' \to \mathbb{R}^+$, the structure D executes any query sequence P of length m in time*

$$O\left(c'_D(n') + \sum_{s' \in S'} f(s') \log \frac{\sum_{r' \in S'} w'(r')}{w'(s')} + \left(m - \sum_{s' \in S'} f(s')\right) \log n\right).$$

For each query $p \in P$, we require that D reports the region $s' \in S'$ with $p \in s'$, if it exists, and that D reports a failure otherwise. The function c'_D is called the construction cost *of the structure.*

Note that $m - \sum_{s' \in S'} f(s')$ is just the number of queries that result in failure. Given Definition 4, we may now state our main theorem:

Theorem 5 *Let S be a set of n regions. Suppose we have an $O(\log n)$ time location query structure on S with construction cost $O(n)$ and a subset-biased structure on S with construction cost $O(n' \log n')$. Then we can construct a statically optimal structure on S with construction cost $O(n)$.*

3 The transformation

We now describe the construction for Theorem 5. By assumption, we are given a set S of n regions, and we have available an $O(\log n)$ time location query structure D on S with construction cost $O(n)$ as well as a subset-biased structure with construction cost $O(n' \log n')$.

[3] By convention, $f(s) \log(m/f(s)) := 0$ if $f(s) = 0$.

3.1 Description of the structure

Let α and β be two constants such that $0 < \beta < \alpha < 1 - \beta < 1$ (e.g., $\alpha = 1/2$ and $\beta = 1/3$). The simple idea behind our transformation is as follows: after every n^α queries, we build a subset-biased structure for the n^β most commonly accessed regions, in $O(n^\beta \log n) = o(n^\alpha)$ time. We also keep a static $O(\log n)$ time structure as a backup for failed queries in the subset-biased structure. Formally, the structure has several parts:

1. A static $O(\log n)$ query time structure.
2. A structure that keeps track of how often each region was queried and that is capable of reporting the k most popular regions in $O(k)$ time. Since in each step we increment the count for a single region by 1, we can easily maintain such a structure in linear space and constant time per update. (The additional space overhead can be made sublinear at the expense of determinism thorough the use of a streaming algorithm for the so-called *heavy hitters* problem (e.g., [12]). This shows that our transformation is also useful in a context where additional space is at a premium, for example for implicit data structures or when the data resides in read-only memory [9]).
3. A subset-biased structure D' that is built after $2n^\alpha$ queries and rebuilt every $n^{\alpha\text{th}}$ query thereafter. The structure contains the at most n^β most popular regions at the time of the rebuilding that have been queried at least $2n^\alpha$ times. In the choice of these regions, we break ties arbitrarily. The weight of a region s, denoted $w'(s)$, is the number of queries to s at the time of the rebuilding. More precisely, if the rebuilding is at time t, we set $w'(s) := f_t(s)$. Computing the n^β most popular regions and the weight function w' takes time $O(n^\beta)$ with the structure from Part 2. By assumption, the construction cost of D' is $O(n^\beta \log n)$.

A search is executed on the subset-biased structure first. If it fails (at amortized cost $O(\log n)$), it is executed in the static $O(\log n)$ time structure.

3.2 Initial analysis of the structure

We will now analyze the properties of our structure. Our first lemma describes a key property of the rebuilding process: for any sufficiently popular region s, the amortized query time for s is proportional to the amortized query time a biased structure would achieve if it were weighted with the frequencies observed so far.

Lemma 6 *Consider the query p_t at time t, and let $s := q(p_t)$ denote the resulting region. Suppose that $f_t(s) \geq 2n^\alpha$. Then the amortized cost for query p_t is*

$$O\left(\log \frac{t}{f_t(s) - n^\alpha}\right).$$

Proof: Since $f_t(s) \geq 2n^\alpha$, we have $t \geq 2n^\alpha$. Thus, we first query the subset-biased structure D'. Suppose that D' has been rebuilt last at time $t' \geq t - n^\alpha$. There are two cases.

Suppose first that s is contained in D'. Definition 4 ensures that the amortized time for the query in D' is $O(\log(W'/f_{t'}(s)))$, where W' denotes the total number of queries for the regions in

D' at time t'. We have $W' \leq t$ (there have been t queries so far) and $f_{t'}(s) \geq f_t(s) - n^\alpha$ (there have been at most n^α queries since rebuilding). The lemma follows.

Now suppose that s is not in D'. In this case, the query takes $O(\log n)$ amortized time in D' and $O(\log n)$ time in the static structure. We know that at time t', there were n^β regions at least as popular as s. Thus, $n^\beta f_{t'}(s) \leq t' \leq t$. It follows that

$$\beta \log n = \log n^\beta \leq \log \frac{t}{f_{t'}(s)} \leq \log \frac{t}{f_t(s) - n^\alpha},$$

and the claimed bound suffices to account for the $O(\log n)$ query time. □

Using Lemma 6, we can now bound the running time in terms of the query frequencies.

Lemma 7 *Let S be a set of n regions. Our structure executes any query sequence P on S of length m in time*

$$O\left(\sum_{s \in S}\left(\underbrace{\min(f(s), 2n^\alpha)\log n}_{\text{first } 2n^\alpha \text{ queries to } s} + \underbrace{\sum_{j=2n^\alpha}^{f(s)} \log \frac{t_j(s)}{f_{t_j(s)}(s) - n^\alpha}}_{\text{queries to } s \text{ after the } 2n^\alpha \text{th}}\right) + \underbrace{\left\lfloor \frac{m}{n^\alpha} \right\rfloor n^\beta \log n}_{\substack{\text{rebuild biased} \\ \text{structure}}} + \underbrace{n}_{\substack{\text{static} \\ \text{structure} \\ \text{construction}}}\right).$$

Proof: The main summation is over the regions in S. For each region s, the initial $2n^\alpha$ (or less) queries take time $O(\log n)$, since during these queries s is never in the subset-biased structure. The running times for the remaining queries (if any) are bounded using Lemma 6. The first additional term comes from the $O(n^\beta \log n)$ construction cost of the subset-biased structure, incurred every n^α operations. The final term is the linear one-time cost to build the static structure. □

3.3 Technical Lemmas

In order to simplify the bound in Lemma 7, we need two technical lemmas to deal with the various terms. The first lemma shows how to simplify the summation for the later queries.

Lemma 8 *Let S be a set of n regions, and let P be a query sequence on S of length m. For each region $s \in S$, we have*

$$\sum_{j=2n^\alpha}^{f(s)} \log \frac{t_j(s)}{f_{t_j(s)}(s) - n^\alpha} \leq f(s)\left(3 + \log \frac{m}{f(s)}\right).$$

Proof: Since $f_{t_j(s)}(s) = j \geq 2n^\alpha$, we have $f_{t_j(s)}(s) - n^\alpha \geq j/2$. Also, $t_j(s) \leq m$. Thus,

$$\sum_{j=2n^\alpha}^{f(s)} \log \frac{t_j(s)}{f_{t_j(s)}(s) - n^\alpha} \leq \sum_{j=1}^{f(s)} \log \frac{m}{j/2} = \log \frac{(2m)^{f(s)}}{f(s)!} \leq \log \frac{(2em)^{f(s)}}{f(s)^{f(s)}} \leq f(s)\left(3 + \log \frac{m}{f(s)}\right).$$

Here, we used Stirling's formula to bound $f(s)! \geq (f(s)/e)^{f(s)}$. □

The second lemma deals with the time for the initial queries.

Lemma 9 *Let γ be a constant with $\alpha < \gamma < 1$. If $m \geq n^\gamma$, then*

$$\min(f(s), 2n^\alpha) \log n = O\left(f(s) \log \frac{m}{f(s)}\right).$$

Proof: Set $\delta := (\alpha + \gamma)/2$. If $f(s) \leq n^\delta$, the lemma holds since

$$f(s) \log(m/f(s)) \geq f(s) \log(m/n^\delta) = \Omega(f(s) \log n) = \Omega(\min(f(s), 2n^\alpha) \log n).$$

If $f(s) > n^\delta$, then

$$f(s) \log(m/f(s)) > n^\delta \geq 2n^\alpha \log n,$$

for n large enough, as desired (recall that we defined $\log x$ to be at least 1). □

3.4 Main theorem

We can now prove our main theorem.

Proof: [of Theorem 5] By Definition 3, we need to prove that the execution time is

$$O\left(n + \sum_{s \in S} f(s) \log \frac{m}{f(s)}\right).$$

By Lemma 7, the running time is bounded by

$$O\left(\sum_{s \in S} \left(\min(f(s), 2n^\alpha) \log n + \sum_{j=2n^\alpha}^{f(s)} \log \frac{t_j(s)}{f_{t_j(s)}(s) - n^\alpha}\right) + \left\lfloor \frac{m}{n^\alpha} \right\rfloor n^\beta \log n + n\right).$$

We now apply Lemma 8 and note that $\left\lfloor \frac{m}{n^\alpha} \right\rfloor n^\beta \log n = o(m)$ to obtain a running time bound of

$$O\left(\sum_{s \in S} \left(\min(f(s), 2n^\alpha) \log n + f(s) \left(3 + \log \frac{m}{f(s)}\right)\right) + n + m\right).$$

Since we defined $\log x$ to be at least 1, this simplifies to

$$O\left(\sum_{s \in S} \left(\min(f(s), 2n^\alpha) \log n + f(s) \log \frac{m}{f(s)}\right) + n\right).$$

If $m \leq n^{1-\beta}$, the sum over $s \in S$ is at most $n^{1-\beta} \log n = o(n)$. In this case, the bound simplifies to $O(n)$, and the theorem is proved. Otherwise, if $m > n^{1-\beta}$, Lemma 9 applies with $\gamma := 1 - \beta$ (a legal choice by our assumption on α and β), and the term $\min(f(s), 2n^\alpha) \log n$ collapses into $f(s) \log(m/f(s))$ to give the theorem. □

4 Point location

Theorem 10 *There is a data structure for point location in a planar triangulation of size n that can execute any query sequence of length m in time*

$$O\left(\sum_{s\in S} f(s)\log\frac{m}{f(s)} + n\right).$$

Proof: It is easy to apply our general transformation to the problem of planar point location in a triangulation, as all of the required ingredients are well known. We assume that the triangulation is given in a standard representation, such as a doubly-connected edge list (e.g., [11, §2.2]).

For the static structure with $O(\log n)$ query time and $O(n)$ construction time, Kirkpatrick's algorithm [34] can be used. For the subset-biased structure, the provided subset of n^β triangles may not be a connected triangulation and thus needs to be triangulated; this takes time $O(n^\beta \log n)$ using the classic line sweep approach [38]. This creates $O(n^\beta)$ new triangles, which are marked specially and given small weights. The resultant triangulation and weighting is given to a biased structure such as Iacono's [31]. The marking can be used to detect whether a query to the subset-biased structure was successful. With all ingredients in hand, the claim now follows from Theorem 5. □

Our choice of structures reflects a desire for the strongest asymptotic bounds possible. Thus, we have avoided structures that are randomized or that have non-linear construction cost; such structures, however, have far superior constants than the ones we use. If we took a data structure for the static $O(\log n)$ time queries with an $O(n \log n)$ construction cost instead of $O(n)$, this would simply change the linear additive term in Theorem 10 to $n \log n$.

5 Point location in polygonal subdivisions with non-constant sized cells

Our work applies to point location in triangulations. It can also be extended to polygonal subdivisions where each region has constant complexity. Indeed, suppose every region has $k+2$ edges. We can just triangulate each region and then apply our result. As mentioned in the introduction, this operation could increase the entropy of the query outcomes. However, the *log sum inequality* [19, Theorem 2.7.1] implies that $\sum_{i=1}^{k} p_i \log(1/p_i) \le p\log(k/p)$ for any nonnegative $p_1, p_2, \ldots, p_k$ and $p = \sum_{i=1}^{k} p_i$. Thus, if we subdivide a region with probability p into k triangles, the entropy increases by at most $p \log k$. It follows that the overall entropy grows by at most $\log k$, which is acceptable if k is constant.

Recently, several data structures have been developed for optimal point location where the distribution is known in advance for convex connected [17], connected [18], and arbitrary polygonal [14] subdivisions of the plane, as well as the more general odds-on trees [13]. Unfortunately, these structures are not biased according to our definition, since entropy-based lower bounds are not meaningful for them: a convex k-gon splits the plane into two regions, so the entropy of the query outcomes is constant. Nonetheless, some distributions require $\Omega(\log n)$ time for a point location query (in a reasonable model of computation that is more restrictive than the one described here).

The entropy-sensitive structures for non-triangulations all basically work by triangulating the given subdivision as a function of the provided probability distribution, and then using one of the biased structures on the resultant triangulation. The main conceptual problem in using our framework with such a structure is that it is unclear how to triangulate during the rebuilding process,

since the optimal triangulation is not known in advance. One could imagine that triangulating during each rebuild based on the observed queries so far would work well, but proving this would require a more complex and specialized analysis than what has been presented in this paper.

Acknowledgments

The second author would like to thank Pat Morin for suggesting the problem to him, for stimulating discussions on the subject, and for hosting him during a wonderful stay at the Computational Geometry Lab at Carleton University. We would also like to thank the anonymous referees for insightful comments that helped improve the presentation of the paper.

References

[1] G. M. Adel′son-Vel′skiĭ and E. M. Landis. An algorithm for organization of information. *Dokl. Akad. Nauk SSSR*, 146:263–266, 1962.

[2] N. Ailon, B. Chazelle, K. L. Clarkson, D. Liu, W. Mulzer, and C. Seshadhri. Self-improving algorithms. *SIAM J. Comput.*, 40(2):350–375, 2011.

[3] S. Arya, S.-W. Cheng, D. M. Mount, and R. Hariharan. Efficient expected-case algorithms for planar point location. In *Proc. 7th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 1851 of *Lecture Notes in Computer Science*, pages 353–366. Springer-Verlag, 2000.

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

[5] S. Arya, T. Malamatos, and D. M. Mount. Entropy-preserving cuttings and space-efficient planar point location. In *Proc. 12th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 256–261, 2001.

[6] S. Arya, T. Malamatos, and D. M. Mount. A simple entropy-based algorithm for planar point location. In *Proc. 12th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 262–268, 2001.

[7] S. Arya, T. Malamatos, and D. M. Mount. A simple entropy-based algorithm for planar point location. *ACM Trans. Algorithms*, 3(2):Art. 17, 17 pp, 2007.

[8] S. Arya, T. Malamatos, D. M. Mount, and K. C. Wong. Optimal expected-case planar point location. *SIAM J. Comput.*, 37(2):584–610, 2007.

[9] T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *JoCG*, 2(1):46–68, 2011.

[10] S. W. Bent, D. D. Sleator, and R. E. Tarjan. Biased search trees. *SIAM J. Comput.*, 14(3):545–568, 1985.

[11] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, third edition, 2008.

[12] R. Berinde, P. Indyk, G. Cormode, and M. J. Strauss. Space-optimal heavy hitters with strong error bounds. *ACM Trans. Database Syst.*, 35(4):Art. 26, 28 pp, 2010.

[13] P. Bose, L. Devroye, K. Douïeb, V. Dujmovic, J. King, and P. Morin. Odds-on trees. `arXiv:1002.1092`, 2010.

[14] P. Bose, L. Devroye, K. Douïeb, V. Dujmovic, J. King, and P. Morin. Point location in disconnected planar subdivisions. `arXiv:1001.2763`, 2010.

[15] P. Bose, K. Douïeb, V. Dujmovic, and J. Howat. Layered working-set trees. In *Proc. 9th Latin American Theoretical Informatics Symposium (LATIN)*, volume 6034 of *Lecture Notes in Computer Science*, pages 686–696. Springer-Verlag, 2010.

[16] P. Bose, K. Douïeb, and S. Langerman. Dynamic optimality for skip lists and B-trees. In *Proc. 19th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 1106–1114, 2008.

[17] S. Collette, V. Dujmovic, J. Iacono, S. Langerman, and P. Morin. Distribution-sensitive point location in convex subdivisions. In *Proc. 19th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 912–921, 2008.

[18] S. Collette, V. Dujmovic, J. Iacono, S. Langerman, and P. Morin. Entropy, triangulation, and point location in planar subdivisions. `arXiv:0901.1908`, 2009.

[19] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley-Interscience, second edition, 2006.

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

[21] A. Elmasry. A priority queue with the working-set property. *Internat. J. Found. Comput. Sci.*, 17(6):1455–1465, 2006.

[22] G. N. Frederickson. Implicit data structures for weighted elements. *Inform. and Control*, 66(1-2):61–82, 1985.

[23] M. L. Fredman. Two applications of a probabilistic search technique: Sorting $x+y$ and building balanced search trees. In *Proc. 7th Annu. ACM Sympos. Theory Comput. (STOC)*, pages 240–244, 1975.

[24] A. M. Garsia and M. L. Wachs. A new algorithm for minimum cost binary trees. *SIAM J. Comput.*, 6(4):622–642, 1977.

[25] M. T. Goodrich. Competitive tree-structured dictionaries. In *Proc. 11th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 494–495, 2000.

[26] M. T. Goodrich, M. Orletsky, and K. Ramaiyer. Methods for achieving fast query times in point location data structures. In *Proc. 8th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 757–766, 1997.

[27] L. J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 8–21, 1978.

[28] T. C. Hu and A. C. Tucker. Optimal computer search trees and variable-length alphabetical codes. *SIAM J. Appl. Math.*, 21:514–532, 1971.

[29] J. Iacono. Improved upper bounds for pairing heaps. In *Proc. 7th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 1851 of *Lecture Notes in Computer Science*, pages 32–45. Springer-Verlag, 2000.

[30] J. Iacono. Alternatives to splay trees with $O(\log n)$ worst-case access times. In *Proc. 12th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 516–522, 2001.

[31] J. Iacono. Expected asymptotically optimal planar point location. *Comput. Geom. Theory Appl.*, 29(1):19–22, 2004.

[32] J. Iacono. Key-independent optimality. *Algorithmica*, 42(1):3–10, 2005.

[33] J. H. Kingston. A new proof of the Garsia-Wachs algorithm. *J. Algorithms*, 9(1):129–136, 1988.

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

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

[36] J. F. Korsh. Greedy binary search trees are nearly optimal. *Inform. Process. Lett.*, 13(1):16–19, 1981.

[37] H.-P. Kriegel and V. K. Vaishnavi. Weighted multidimensional B-trees used as nearly optimal dynamic dictionaries. In *Proc. 10th Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 118 of *Lecture Notes in Computer Science*, pages 410–417. Springer-Verlag, 1981.

[38] D. T. Lee and F. P. Preparata. Location of a point in a planar subdivision and its applications. *SIAM J. Comput.*, 6(3):594–606, 1977.

[39] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979.

[40] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[41] K. Mehlhorn. Nearly optimal binary search trees. *Acta Inform.*, 5(4):287–295, 1975.

[42] K. Mehlhorn. A best possible bound for the weighted path length of binary search trees. *SIAM J. Comput.*, 6(2):235–239, 1977.

[43] K. Mehlhorn. Dynamic binary search. *SIAM J. Comput.*, 8(2):175–198, 1979.

[44] K. Mehlhorn. Arbitrary weight changes in dynamic trees. *RAIRO Inform. Théor.*, 15(3):183–211, 1981.

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

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

[47] 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.

[48] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. The University of Illinois Press, Urbana, Ill., 1949.

[49] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.

[50] A. Z. B. H. Talib, M. Chen, and P. Townsend. Three major extensions to Kirkpatrick's point location algorithm. In *Proc. Conference on Computer Graphics International (CGI)*, pages 112–121, 1996.

[51] K. Unterauer. Dynamic weighted binary search trees. *Acta Inform.*, 11(4):341–362, 1978/79.

[52] F. F. Yao. Efficient dynamic programming using quadrangle inequalities. In *Proc. 12th Annu. ACM Sympos. Theory Comput. (STOC)*, pages 429–435, 1980.