Adaptive Point Location in Planar Convex Subdivisions*

Siu-Wing Cheng† Man-Kit Lau†

Abstract

We present a planar point location structure for a convex subdivision S. Given a query sequence of length m, the total running time is $O(\mathrm{OPT} + m \log\log n + n)$, where n is the number of vertices in S and OPT is the minimum time required by any linear decision tree for answering planar point location queries in S to process the same query sequence. The running time includes the preprocessing time. Therefore, for $m \geq n$, our running time is only worse than the best possible bound by $O(\log\log n)$ per query, which is much smaller than the $O(\log n)$ query time offered by a worst-case optimal planar point location structure.

1 Introduction

There has been extensive research on planar point location—a fundamental problem in computational geometry—to obtain worst-case optimal query time, preprocessing time, and space complexity [2, 16, 20, 21, 22, 23, 24, 25]. Some of them are now standard results in textbooks [8, 11]. Planar point location can be seen as a generalization of the one-dimensional dictionary problem to two dimensions. In any dimension, an information theoretic lower bound on processing a sequence of m queries, known as the *entropy-based* lower bound, follows from Shannon's work [26]. The lower bound is $\sum_z f(z) \cdot \log \frac{m}{f(z)}$, where $f(z)$ denotes the access frequency of an item z in the sequence of length m. The splay tree [27] has been designed such that, given an initially empty structure and a sequence of m insertions, deletions, and queries, the total running time for processing these operations is $O\left(\sum_z f(z) \cdot \log \frac{m}{f(z)}\right)$, where each insertion or deletion of z also contributes to the access frequency of z. Notice that the access frequencies of items are unknown beforehand. As a result, $o(\log n)$ amortized query time is possible in one dimension if the access frequencies of the items are substantially unequal.

For point location in a planar subdivision S, there are also previous works on making the performance adaptive to the access frequencies. When the regions in S have constant complexities and a fixed query distribution is given, there are several works by Arya together with several sets of collaborators [4, 5, 6, 7] and Iacono [17] to construct a data structure such that the expected query time is $O\left(\sum_z p_z \log \frac{1}{p_z}\right)$, where p_z is the probability of a query point falling into the region z. The algorithm of Iacono [17] uses $O(n)$ space and $O(n)$ preprocessing time. The algorithm of Arya et al. [7] uses $O(n)$ space and $O(n \log n)$ preprocessing time, and its expected running time per query is optimal up to the leading constant factor modulo some additive low-order terms. Arya et al. [7] show that one can design a convex polygon of n sides and a query distribution so that a query point lies in the polygon with probability $1/2$ and the expected number of point-line comparisons needed to decide whether a query point lies in the polygon is $\Omega(\log n)$. However, the entropy-based lower bound for a single query is only a

*Supported by FSGRF14EG26, HKUST. A preliminary version appears in Proceedings of the International Symposium on Algorithms and Computation, 2015, 14–21.

†Department of Computer Science and Engineering, HKUST, Hong Kong

constant in this case. This shows that the entropy-based lower bound is too weak for a convex subdivision.

As a result, when analogous results are developed for a given and fixed query distribution over a more general subdivision S in which the regions may have arbitrary complexities [1, 9, 10, 13], the results are compared against the best linear decision tree for answering planar point location queries in S. This is reasonable because the linear decision tree models the process for answering a query by point-line comparisons, and many existing point location structures are based on point-line comparisons.[1] A linear decision tree for locating a query point in a given planar subdivision S is a rooted binary tree. Each internal node v is associated with a linear inequality function φ_v that accepts the input query point as a parameter. When locating a query point q, one navigates down the tree starting at the root. At each internal node v, $\varphi_v(q)$ is evaluated. Depending on whether $\varphi_v(q) \leq 0$ or $\varphi_v(q) > 0$, the left or right child is visited next, respectively. Geometrically, it is equivalent to deciding whether q is below or above a line associated with v. When a leaf is reached, the name of the region of S stored at that leaf is reported. Geometrically, a leaf corresponds to a convex polygon contained in a region of S. The depth of a node in a linear decision tree is the number of edges on the tree path from the root to that node.

A natural question is whether we can obtain a self-adjusting planar point location structure that can adapt to a query sequence without knowing the access frequencies of the regions beforehand. There has been only one such result. It is due to Iacono and Mulzer [19] and it works for triangulations only. The total running time is $O\left(n + \sum_z f(z) \cdot \log \frac{m}{f(z)}\right)$, including the preprocessing time to construct the initial structure before processing the query sequence. In this paper, we study the adaptive point location problem for a convex subdivision S. That is, the regions in S and the complement of the unbounded region are convex polygons. We do not require the regions in S to have constant complexities. Note that Voronoi diagrams are convex subdivisions and so our result is applicable in the important problem of nearest neighbor searching. Given a sequence of m queries, our method runs in $O(\mathrm{OPT} + m \log\log n + n)$ total time, where OPT is the minimum time required by any linear decision tree for answering point location queries in S to process the same query sequence. Our time bound includes the preprocessing time. Therefore, for $m \geq n$, our running time is only worse than the best possible bound by $O(\log\log n)$ time per query, which is much smaller than the $O(\log n)$ query time offered by a worst-case optimal planar point location structure.

One can build another auxiliary planar point location structure so that a query can be executed on our adaptive structure and this auxiliary point location structures simultaneously until one of the two structures returns an answer. The advantage is that this auxiliary point location structure can offer additional properties. For example, if the outer boundary of the subdivision is a unit square, one can use the distance-sensitive planar point location structure [3] so that queries far away from any region boundary can be answered fast too. This data structure uses $O(n)$ space and locates a query point q in $O\left(\min\left(\log n, 1 + \log \frac{1}{\Delta_q^2}\right)\right)$ time, where Δ_q is the distance from q to the boundary of the region containing it. Alternatively, if one uses the proximate planar point location structure [18] as the auxiliary structure, then the current query point can be located faster if it is close to the previous query point. This data structure uses $O(n \log\log n)$ space and locates a query q_i in $O(\log d(q_{i-1}, q_i))$ time where q_{i-1} is the previous query point and $d(q_{i-1}, q_i)$ is the number of regions intersected by some diamond-shaped region enclosing q_{i-1} and q_i.

[1] Methods that employ indexing (e.g. [15]) and bit tricks (e.g. [12]) do not fall under the linear decision tree model.

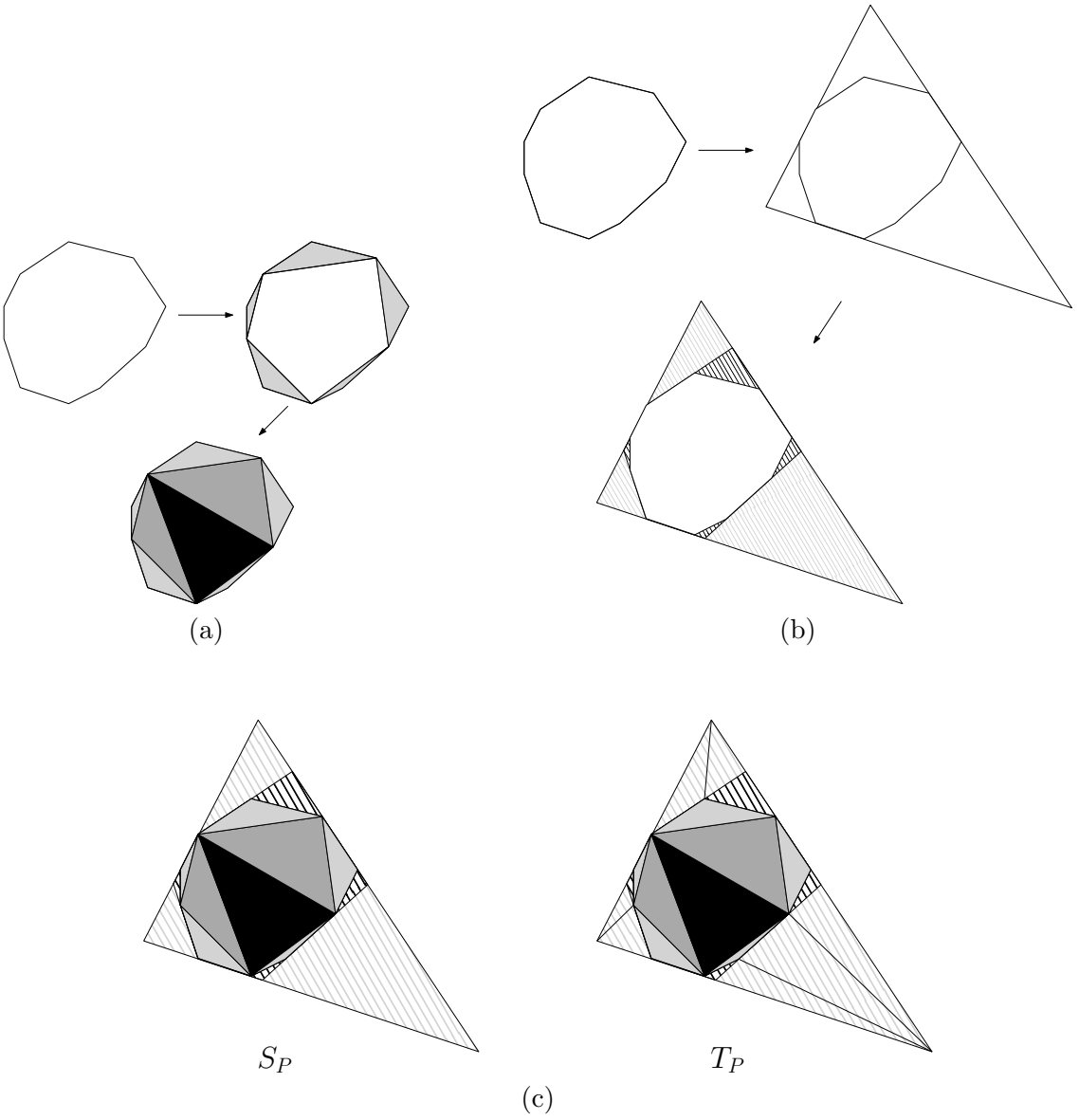

Figure 1: (a) The lightly shaded, shaded, and black triangles are obtained from the first, second and third convex hulls, respectively. (b) The triangle enclosing P is Q_0. The lightly shaded triangles are clipped from Q_0 to form Q_1. Then the shaded triangles are clipped from Q_1 to form $Q_2 = P$. (c) The planar subdivision S_P and the final triangulation T_P.

2 Triangulation of a convex polygon

Let P be a convex region in S with n_P vertices in counterclockwise order $(v_0, v_1, ..., v_{n_P-1})$. We first triangulate the interior of P as follows. Select every other vertex of P. (When n_P is odd, the last vertex selected is adjacent to the first vertex selected.) Let P_1 be the convex hull of these selected vertices. Clearly, $P_1 \subset P$, $P \setminus P_1$ is a collection of triangles, and the number of vertices of P_1 is at most $\lceil n_P/2 \rceil$. Then, we recurse on P_1 to construct P_2 and so on until we produce a convex hull P_j that is a single triangle or a single line segment. The triangulation of P is the collection of triangles in $P \setminus P_1$, $P_1 \setminus P_2$, etc. Figure 1(a) shows an example. This hierarchical triangulation was first introduced by Dobkin and Kirkpatrick [14] in the context of detecting intersection between two convex polygons and polyhedra. Note that $O(\log n_P)$ P_i's are constructed because the size of the P_i's decreases repeatedly by a constant factor. The time to produce each P_i is $\lceil n_P/2^i \rceil$. Therefore, the total time to compute T_P is $O\left(\sum_{i=0}^{\infty} n_P/2^i\right) = O(n_P)$. A line segment ℓ in P intersects the boundary of each P_i's in at most two points. It follows that ℓ intersects at most two triangles in $P_{i-1} \setminus P_i$, and therefore, ℓ intersects $O(\log n_P)$ triangles.

Second, we triangulate the complement of P. Identify three edges of P the removal of which split the boundary of P into three chains of roughly equal length, i.e., $\lceil n_P/3 \rceil - 1$, $\lfloor n_P/3 \rfloor - 1$, and $n_P - 1 - \lceil n_P/3 \rceil - \lfloor n_P/3 \rfloor$ edges. The support lines of these three edges bound a triangle Q_0 that encloses P. Each vertex of Q_0 faces a boundary chain of P. For each vertex v of Q_0, we take the middle edge on the corresponding boundary chain of P and use the support line of that edge to clip off the vertex v of Q_0. This gives a convex polygon Q_1 with six vertices and $Q_1 \subset Q_0$. We repeat to obtain a nested hierarchy of convex polygons $Q_0 \supset Q_1 \supset \ldots P$. Figure 1(b) gives an example. The number of vertices is doubled as we go from Q_i to Q_{i+1} until some vertex of Q_i is also a vertex of P. Then, if some vertex in Q_i is still not a vertex of P, it faces a boundary chain of P with only a constant number of edges, and so clipping can be repeated another constant number of rounds to obtain P. Therefore, there are $O(\log n_P)$ convex polygons in the hierarchy. The unbounded region, the collection of triangles in $Q_0 \setminus Q_1$, $Q_1 \setminus Q_2$, etc, and the triangulation of P form a planar subdivision which we denote by S_P. The left image in Figure 1(c) gives an example of S_P. All regions in S_P have a triangular boundary including the unbounded region. For every bounded region r in $S_P \setminus P$, r has exactly one side e that contains an edge of P. The other two sides of r contains no vertex in their interior, but e may contain up to $O(\log n_P)$ vertices in its interior as e may bound several convex polygons Q_i's. A line segment ℓ intersects the boundary of each Q_i in at most two points. Therefore, ℓ intersects at most two triangles in $Q_i \setminus Q_{i+1}$, and hence a total of $O(\log n_P)$ regions in $S_P \setminus P$, including the unbounded region.

We refine S_P into a triangulation T_P as follows: for every bounded region r in $S_P \setminus P$ and every vertex v in the interior of the side e of r that bounds P, add an edge to connect v to the vertex of r opposite e. We regard the unbounded region as a region in T_P. So T_P covers the entire plane. The right image in Figure 1(c) gives an example of T_P. Since each region in $S_P \setminus P$ is split into $O(\log n_P)$ regions in T_P, a line segment intersects $O(\log^2 n_P)$ regions in $T_P \setminus P$. The triangulation of the interior of P introduce no new vertex. In $S_P \setminus P$, each edge of P is extended bidirectionlly to produce two new vertices. Therefore, T_P has $O(n_P)$ vertices, and it can clearly be constructed in $O(n_P)$ time.

In the following, we prove an upper bound on the entropy of T_P that is closely related to the performance of any linear decision tree.

Lemma 2.1 *Let P be a convex polygon in $\mathbb{R}^2$. Let $\mathcal{D}$ be an arbitrary linear decision tree for determining whether a query point in $\mathbb{R}^2$ lies in P. Let $L_{\mathcal{D}}$ be the set of leaves of $\mathcal{D}$. For every leaf $\nu \in L_{\mathcal{D}}$, let r_ν denote the convex region represented by ν. Consider an arbitrary query*

sequence of length m. For every region $r \subseteq \mathbb{R}^2$, let $f(r)$ denote the number of query points that fall inside r. Then, the entropy $H(T_P)$ of T_P satisfies the following inequality.

$$\begin{aligned} H(T_P) &= \sum_{t \in T_P} f(t) \cdot \log \frac{m}{f(t)} \\ &\leq \sum_{\nu \in L_{\mathcal{D}}} f(r_\nu) \cdot \Big(\text{depth}(\nu) + O(\log(\text{depth}(\nu))) + O(\log \log n) \Big) \end{aligned}$$

Proof. Let q be a query point that falls in the convex polygon r_ν for some leaf $\nu \in L_{\mathcal{D}}$. Let k_ν denote the number of sides of r_ν. We have $\text{depth}(\nu) \geq k_\nu$ because each internal node on the path from the root of $\mathcal{D}$ to ν corresponds to a cut along a line. Since the linear decision tree $\mathcal{D}$ must output at ν whether q is in P, the polygon r_ν is either completely inside or outside P.

We can expand the linear decision tree $\mathcal{D}$ to another linear decision tree $\mathcal{D}''$ that allows us to identify the region $t \in T_P$ containing q. The construction of $\mathcal{D}''$ works in two steps as follows. For each leaf $\nu \in L_{\mathcal{D}}$, if $k_\nu > 3$, we recursively add a chord to split r_ν into two convex polygons, each having at most $(\lceil \frac{k}{2} \rceil + 1)$ sides. At the same time, we attach two child nodes of ν to represent these smaller convex polygons. The recursion stops when r_ν is triangulated.[2] The recursive triangulation of r_ν produces a subtree rooted at ν of height $O(\log k_\nu) = O(\log(\text{depth}(\nu)))$. Let $\mathcal{D}'$ denote this intermediate linear decision tree obtained after expanding each leaf of $\mathcal{D}$ to a subtree as described above. Each leaf of $\mathcal{D}'$ represents a region t' with at most three sides that lies in r_ν for some $\nu \in L_{\mathcal{D}}$. As r_ν is either completely inside or outside P, so is t'.

Every region in $T_P \cap P$ has its vertices on the boundary of P, and every region in $S_P \setminus P$ has one side that bounds P. Therefore, if t' intersects a region in $T_P \cap P$ or $S_P \setminus P$, the boundary of t' must intersect the boundary of that region in $T_P \cap P$ or $S_P \setminus P$. As discussed in the construction of T_P, a line segment intersects $O(\log n_P)$ regions in $T_P \cap P$ and $S_P \setminus P$. Each region in $S_P \setminus P$ is split into $O(\log n_P)$ regions in $T_P \setminus P$. Therefore, t' intersects $O(\log^2 n_P)$ regions in T_P. This motivates us to expand the leaves of $\mathcal{D}'$ as follows. For every leaf ν' of $\mathcal{D}'$, replace ν' by a linear decision tree that corresponds to an optimal worst-case planar point location structure on the $O(\log^2 n_P)$ regions in T_P that intersect the region corresponding to ν'. The resulting linear decision tree is $\mathcal{D}''$. The height of $\mathcal{D}''$ is $O(\log \log n)$ more than the height of $\mathcal{D}'$.

If q is a query point inside r_ν for some $\nu \in L_{\mathcal{D}}$, we can follow the search path from the root of $\mathcal{D}$ to ν and then from ν to a leaf ν'' of $\mathcal{D}''$. The length of the path traversed is $\text{depth}(\nu'') \leq \text{depth}(\nu) + O(\log(\text{depth}(\nu))) + O(\log \log n)$. The entropy of T_P is an information-theoretic lower bound to answering point location queries in T_P [4, 26]. In particular, this lower bound applies to the linear decision tree $\mathcal{D}''$. Therefore,

$$\begin{aligned} H(T_P) &= \sum_{t \in T_P} f(t) \cdot \log \frac{m}{f(t)} \\ &\leq \sum_{\text{leaf } \nu'' \text{ of } \mathcal{D}''} f(r_{\nu''}) \cdot \text{depth}(\nu'') \\ &\leq \sum_{\nu \in L_{\mathcal{D}}} f(r_\nu) \cdot \Big(\text{depth}(\nu) + O(\log(\text{depth}(\nu))) + O(\log \log n) \Big) \end{aligned}$$

□

Lemma 2.1 is related to Lemma 5 in [13] in the context of planar point location when the query point distribution is known. Given a linear decision tree T for answering point location

[2] If r_ν is unbounded, there is an unbounded region with three sides in the triangulation of r_ν.

queries in a planar subdivision S with known query distribution, let $\mu(T)$ denote the expected query time, let L_T denote the set of leaves of T, and for every leaf $\nu \in L_T$, let $d_T(\nu)$ denote its depth in T and let r_ν denote the convex polygon corresponding to ν. It is proved in [13] that for every linear decision tree T^* that answers point location queries in S, there exists a linear decision tree T for answering point location queries in S such that every leaf of T corresponds to a triangle in $\mathbb{R}^2$ and $\mu(T) \leq \mu(T^*) + \sum_{\nu \in L_{T^*}} \Pr(r_\nu) \cdot O(\log d_T(\nu)) \leq \mu(T^*) + O(\log \mu(T^*))$.

3 Point location in a convex subdivision

Let S be a convex subdivision with n vertices. For every convex region P in S, we triangulate the interior of P as described in Section 2. We also triangulate the unbounded region of S as described in Section 2. The collection of all resulting regions form a triangulation T_S. As in Section 2, T_S includes the unbounded region and so T_S covers $\mathbb{R}^2$. Clearly, T_S has $O(n)$ vertices and it can be constructed in $O(n)$ time.

Theorem 3.1 ([19]) *For every planar triangulation T with n vertices, there is a point location structure that can execute any query sequence of length m in time*

$$O\left(\sum_{t \in T} f(t) \log \frac{m}{f(t)} + n\right),$$

where $f(t)$ is the number of query points that fall into the region $t \in T$.

Remark 1. Since the outer boundary of T_S has three sides, any query point that falls into the unbounded region of T_S can be located optimally in $O(1)$ time. Therefore, when we apply Theorem 3.1 to T_S, we can include the unbounded region and count the number of query points falling into it without affecting the asymptotic running time bound.

Theorem 3.1 gives us a point location data structure for T_S. We will prove that this point location data structure guarantees that every query sequence of length m can be processed in $O(\mathrm{OPT} + m \log\log n + n)$ time, where OPT is the minimum time required by any linear decision tree to process that query sequence.

The method of Iacono and Mulzer [19] is based on rebuilding from time to time. Initially, an optimal worst-case data structure W_0 is built on all triangles in T_S, and we start answering queries using W_0 until $\Theta(n^\alpha)$ queries have been answered for some $\alpha \in (0, 1)$. Then we identify the n^β most frequently queried triangles for some $\beta \in (0, 1)$ such that $\alpha \in (\beta, 1-\beta)$, triangulate their exterior, and then build a point location structure W_1 that is distribution-sensitive with respect to frequency counts in these n^β triangles [17]. These frequency counts are fixed when the rebuilding starts. The counts and this distribution-sensitive structure will not be updated as more queries are processed. Until the next rebuilding after another $\Theta(n^\alpha)$ queries, we first submit every query to W_1, and if W_1 does not report a triangle in the input triangulation, we resort to W_0 to answer the query. The challenge in [19] lies in proving that the total time to answer any query sequence of length m matches the entropy bound.

Theorem 3.2 *Let S be a convex subdivision of n vertices in the plane. There is a point-line comparision based algorithm that answers any sequence of m point location queries in S in $O(\mathrm{OPT} + m \log\log n + n)$ time, where* OPT *is the minimum time required by any linear decision tree for answering point location queries in S to process the same query sequence. The preprocessing time is included in our running time bound.*

Proof. We apply Theorem 3.1 to construct a point location structure $\mathcal{D}_S$ for T_S. By Remark 1, the unbounded region of T_S is included in the running time bound in Theorem 3.1. The total time spent by $\mathcal{D}_S$ on any query sequence of length m is

$$O\left(n + \sum_{t \in T_S} f(t) \cdot \log \frac{m}{f(t)}\right). \tag{1}$$

It takes $O(n)$ time to construct T_S which is subsumed by the above time bound.

Let $\mathcal{D}$ be an arbitrary linear decision tree for answering point location queries in S. We use $L_{\mathcal{D}}$ to denote the set of leaves of $\mathcal{D}$. For every leaf $\nu \in L_{\mathcal{D}}$, let r_ν denote the convex polygon corresponding to ν. We expand the leaves of $\mathcal{D}$ as in the proof of Lemma 2.1 to form a linear decision tree $\mathcal{D}''$ for answering point location queries in T_S. Since each leaf ν of $\mathcal{D}$ is expanded into a subtree of height $O\big(\log(\text{depth}(\nu)) + \log\log n\big)$, the total running time taken by queries that go through ν in $\mathcal{D}''$ is $f(r_\nu) \cdot (\text{depth}(\nu) + O(\log(\text{depth}(\nu)) + \log\log n))$. Therefore, $\mathcal{D}''$ gives a total query time of

$$\sum_{\nu \in L_{\mathcal{D}}} f(r_\nu) \cdot \Big(\text{depth}(\nu) + O(\log(\text{depth}(\nu)) + \log\log n)\Big).$$

The entropy $\sum_{t \in T_S} f(t) \cdot \log \frac{m}{f(t)}$ is an information theoretic lower bound for answering a sequence of m point location queries in T_S [4, 26]. Therefore,

$$\sum_{t \in T_S} f(t) \cdot \log \frac{m}{f(t)} \le \sum_{\nu \in L_{\mathcal{D}}} f(r_\nu) \cdot \Big(\text{depth}(\nu) + O(\log(\text{depth}(\nu))) + O(\log\log n)\Big). \tag{2}$$

Substituting (2) into (1), we obtain

$$\begin{aligned} & O\left(n + \sum_{t \in T_S} f(t) \cdot \log \frac{m}{f(t)}\right) \\ = \; & O\left(n + \sum_{\nu \in L_{\mathcal{D}}} f(r_\nu) \cdot \Big(\text{depth}(\nu) + O(\log(\text{depth}(\nu))) + O(\log\log n)\Big)\right) \\ = \; & O\left(\sum_{\nu \in L_{\mathcal{D}}} f(r_\nu) \cdot \text{depth}(\nu)\right) + O(m \log\log n + n). \end{aligned}$$

The first term is $O(\text{OPT})$ because we can choose $\mathcal{D}$ to be the optimal linear decision tree. □

4 Conclusion

We propose a self-adjusting point location structure for planar convex subdivisions. Its performance is worse than the optimal static linear decision tree by an $O(\log\log n)$ factor per query only. It is an open problem whether there exists a self-adjusting planar point location structure that is asymptotically as good as any static linear decision tree.

Acknowledgment

We thank the anonymous referees for their helpful comments.

References

[1] P. Afshani, J. Barbay, and T. Chan. Instance optimal geometric algorithms. *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, 2009, pp. 129–138.

[2] U. Adamy and R. Seidel. On the exact worst case query complexity of planar point location. *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1998, pp. 609–618.

[3] B. Aronov, M. de Berg, M. Roeloffzen, and B. Speckmann. Distance-sensitive planar point location. *Computational Geometry: Theory and Applications*, 54 (2016), 17–31.

[4] S. Arya, S. W. Cheng, D. M. Mount, and H. Ramesh. Efficient expected-case algorithms for planar point location. *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory*, 2000, pp. 353–366.

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

[6] S. Arya, T. Malamatos, and D. M. Mount. A simple entropy-based algorithm for planar point location. *ACM Transactions on Algorithms*, vol. 3, no. 2, 2007, article 17.

[7] S. Arya, T. Malamatos, D. Mount, and K. Wong. Optimal expected-case planar point location. *SIAM Journal on Computing*, vol. 37, no. 2, 2007, pp. 584–610.

[8] J.D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, 1998.

[9] P. Bose, L. Devroye, K. Douïeb, V. Dujmovic, J. King, and P. Morin. Point location in disconnected planar subdivisions. arXiv:1001.2763v1 [cs.CG], 15 January 2010.

[10] P. Bose, L. Devroye, K. Douïeb, V. Dujmovic, J. King, and P. Morin. Odds-On Trees, arXiv:1002.1092v1 [cs.CG], 5 February 2010.

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

[12] T.M. Chan and M. Pătraşcu. Transdichotomous results in computational geometry, I: point location in sublogarithmic time. *SIAM Journal on Computing*, vol. 39, no. 2, 2009, pp. 703–729.

[13] S. Collette, V. Dujmović, J. Iacono, S. Langerman, and P. Morin. Entropy, triangulation, and point location in planar subdivisions. *ACM Transactions on Algorithms*, vol. 8, no. 3, 2012, article 29.

[14] D.P. Dobkin and D.G. Kirkpatrick. Determining the separation of preprocessed polyhedra—a unified approach, in *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, 1990, pp. 400–413.

[15] M. Edahiro, I. Kokubo, and T. Asano. A new point-location algorithm and its practical efficiency—comparison with existing algorithms. *ACM Transactions on Graphics*, vol. 3, no. 2, 1984, pp. 86–109.

[16] H. Edelsbrunner, L. J Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, vol. 15, no. 2, 1986, pp. 317–340.

[17] J. Iacono. Expected asymptotically optimal planar point location. *Computational Geometry: Theory and Applications*, vol. 29, no. 1, 2004, pp. 19–22.

[18] J. Iacono and S. Langerman. Proximate planar point location. *Proceedings of the 19th Annual Symposium on Computational Geometry*, 2003, pp. 220–226.

[19] J. Iacono and W. Mulzer. A static optimality transformation with applications to planar point location. *International Journal of Computational Geometry and Applications*, vol. 22, no. 4, 2012, pp. 327-340.

[20] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, vol. 12, no. 1, 1983, pp. 28–35.

[21] D.T. Lee and F.P. Preparata. Location of a point in a planar subdivision and its applications. *SIAM Journal on Computing*, vol. 6, no. 3, 1977, pp. 594–606.

[22] K. Mulmuley. A fast planar partition algorithm, I. *Journal of Symbolic Computation*, vol. 10, issues 3–4, 1990, pp. 253–280.

[23] F.P. Preparata. A new approach to planar point location. *SIAM Journal on Computing*, vol. 10, no. 3, 1981, pp. 473–483.

[24] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Communications of ACM*, vol. 29, no. 7, 1986, pp. 669–679.

[25] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, vol. 1, no. 1, 1991, pp. 51–64.

[26] C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, 2001, pp. 3–55.

[27] D.D. Sleator and R.E. Tarjan. Self-adjusting binary search trees, *Journal of ACM*, vol. 32, no. 3, 1985, pp. 652–686.