Adaptive Planar Point Location*

Siu-Wing Cheng[1] and Man-Kit Lau[2]

1 **Department of Computer Science and Engineering, HKUST, Hong Kong**
2 **Department of Computer Science and Engineering, HKUST, Hong Kong**

Abstract

We present a self-adjusting point location structure for convex subdivisions. Let n be the number of vertices in a convex subdivision S. Our structure for S uses $O(n)$ space and processes any online query sequence σ in $O(n + \mathrm{OPT})$ time, where OPT is the minimum time required by any linear decision tree for answering point location queries in S to process σ. The $O(n + \mathrm{OPT})$ time bound includes the preprocessing time. Our result is a two-dimensional analog of the static optimality property of splay trees. For connected subdivisions, we achieve a processing time of $O(|\sigma| \log\log n + n + \mathrm{OPT})$.

1998 ACM Subject Classification F.2.2 [Analysis of Algorithms and Problem Complexity] Nonnumerical Algorithms and Problems – Geometrical Problems and Computations

Keywords and phrases point location, planar subdivision, static optimality

Digital Object Identifier 10.4230/LIPIcs.SoCG.2017.30

1 Introduction

Planar point location is a fundamental problem in computational geometry that has been studied extensively. It calls for preprocessing a *planar subdivision* into a data structure so that for any query point, the region in the subdivision that contains the query point can be reported. There are several common types of planar subdivisions. A subdivision is *convex* if the boundary of every region (including the outer boundary) bounds a convex polygon. A subdivision is *connected* if the boundary of every region bounds a simple polygon. A subdivision is *general* if the boundary of every region bounds a polygon possibly with holes. In this paper, we are concerned with point location methods that use point-line comparisons.

Given a general subdivision with n vertices, point location structures with worst-case $O(\log n)$ query time, $O(n \log n)$ preprocessing time, and $O(n)$ space have been obtained [2, 11, 14, 15]. For connected subdivisions, the preprocessing time can be reduced to $O(n)$ [14] after triangulating every region in linear time [6].

When processing a sequence of query points that fall into different regions with vastly different frequencies, one may consider objectives other than minimizing the worst-case time to answer a single query. One scenario is that for every region r, the probability p_r of the query point falling into r is given. In this case, one may want to minimize the expected query time. The entropy $H = \sum_r p_r \log(1/p_r)$, where the sum is over all regions in S, is a lower bound to the expected query time according to Shannon's theory [16]. Arya, Malamatos, and Mount [3] and Iacono [12] studied subdivisions in which all regions have sizes bounded by some constant. They obtained structures that use $O(n)$ space and answer a query in $O(H)$ expected time. Later, Arya, Malamatos, Mount, and Wong [4] improved the expected query time to $H + O(\sqrt{H})$.

* Supported by Research Grants Council, Hong Kong, China (project no. 16201116).

If some regions have non-constant sizes, the entropy is a very weak lower bound. Indeed, Arya, Malamatos, Mount, and Wong [4] showed 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 $\Omega(\log n)$. Note that the entropy is only a constant. Several subsequent research works consider comparison against linear decision trees that answer point location queries in planar subdivisions. We call them *point location linear decision tree* for convenience. Given a connected subdivision S with n vertices and the query distribution, Collete et al. [9] designed a structure that uses $O(n)$ space and answers a query in $O(H^*)$ expected time, where H^* is the minimum expected time needed by any point location linear decision tree for S. Afshani, Barbay, and Chan [1] and Bose et al. [5] also obtained optimal solutions (with respect to linear decision trees) for several geometry query problems, including planar point location, when the query distribution is given.

In one dimension, optimal query performance can be obtained without knowing the query distribution. Sleator and Tarjan [17] designed splay trees for storing an ordered set of values such that any online query sequence σ can be processed in $O(|\sigma| + \sum_v f_v \log(|\sigma|/f_v))$ time, where the sum is over all values in the set and f_v denotes the frequency of v being queried in σ, provided that every value is accessed at least once. This result is known as the Static Optimality Theorem [17]. Note that $\sum_v f_v \log(|\sigma|/f_v)$ is the minimum time needed to process σ by any static binary search tree that stores the same set of values.

Does there exist an analog of the Static Optimality Theorem in the context of planar point location? Iacono and Mulzer [13] proposed a self-adjusting point location structure for triangulations. Given a triangulation S, their structure uses $O(n)$ space and processes any online query sequence σ in $O(n + \sum_t f_t \log(|\sigma|/f_t))$ time, where the sum is over all triangles in S and f_t denotes the frequency of a triangle t being hit by a query point in σ. The space usage is $O(n)$. The time bound includes the preprocessing time to construct the first structure before locating the first query point in σ. Note that $\sum_t f_t \log(|\sigma|/f_t)$ is a lower bound to the minimum time needed by a static point location structure for S to process σ. The handling of more general planar subdivisions is posed as an open problem in [13]. Recently, we made progress by designing a self-adjusting point location structure for convex subdivisions [8] based on the result in [13]. Given a convex subdivision S, our structure uses $O(n)$ space and processes any online query sequence σ in $O(|\sigma| \log\log n + n + \mathrm{OPT})$ time, where OPT is the minimum time needed by any point location linear decision tree for S to process σ.

In this paper, we prove an analog of the Static Optimality Theorem for convex subdivisions. We propose a self-adjusting point location structure that processes any online query sequence in $O(n + \mathrm{OPT})$ time, which includes the preprocessing time. The space usage is $O(n)$.

It is known that the optimal point location linear decision tree for an optimally triangulated subdivision has the same asymptotic performance as the optimal point location linear decision tree for the untriangulated subdivision. Therefore, our solution keeps a triangulation of the convex subdivision so that we can invoke Iacono and Mulzer's result [13]. The triangulation also allows us to extract some frequently accessed triangles and keep a separate, smaller point location structure for them. Then, query points in these triangles can be located faster. As observed in [13], the difficulty lies in efficiently computing the optimal triangulation, which depend on the access frequencies. As the access frequencies evolve, the subdivision will need to be retriangulated and the analysis has to address this issue. On the other hand, we showed in [8] that some canonical triangulation methods (independent of the access frequencies) can lower the average extra cost per query to $O(\log\log n)$. Our insight is to *recursively* extract

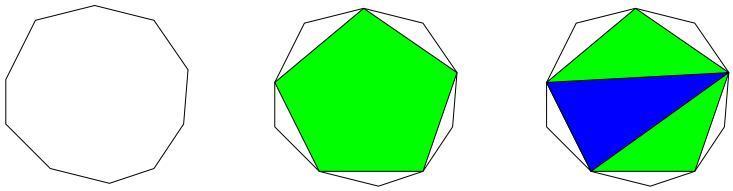


Figure 1 Triangulation of a bounded region in S.

frequently accessed triangles and generate a separate point location structure for them using these canonical triangulation methods. This results in a multi-level structure. The structure at the highest level is queried first and if that fails, we move down the levels. We devise an analysis that handles both successful and unsuccessful queries at each level. Intuitively, the performance improves as the number of levels increases, and thus we circumvent the difficulty of computing an optimal triangulation.

We also observe that the strategy in [8] works for connected subdivisions with the help of balanced geodesic triangulations. This gives a processing time of $O(|\sigma| \log\log n + n + \mathrm{OPT})$.

2 Basics

We state the result of Iacono and Muzler [13] below for future reference.

▶ **Theorem 1** ([13]). *For any planar triangulation T with n vertices, there is a point-line comparison based data structure that uses $O(n)$ space and processes any online sequence of point location queries in T in $O\left(\sum_{t\in T} f(t) \log \frac{N}{f(t)} + n\right)$ time, where N is the number of queries and $f(t)$ is the number of query points that fall into the triangle t in T. The time bounds includes the $O(n)$ preprocessing time.*

We review the canonical triangulation methods in [8] which will be used later.

Let S denote a convex subdivision with n vertices. For each bounded region r in S, the procedure TriReg is called to triangulate r. Figure 1 gives an example. TriReg runs in $O(|r|)$ time and produces a triangulation of $O(|r|)$ size. This triangulation method was first introduced by Dobkin and Kirkpatrick for convex polygon intersection detection [10]. Every line segment in r intersects $O(\log |r|)$ triangles.

TriReg(r)
1. If r is a triangle, then return.
2. Take any maximum subsequence α of vertices of r such that no two vertices in α are adjacent along the boundary of r, except possibly the first and the last ones.
3. Connect the vertices in α to form a convex polygon r'.
4. Call TriReg(r').

Second, we triangulate the exterior region of S. We pick three boundary edges of S such that removing them gives three boundary chains of S of roughly equal sizes. The support lines of these three edges bound a triangle, denoted by $\mathcal{B}_S$, that contains S.[1] We call the three interior-disjoint regions between the boundaries of $\mathcal{B}_S$ and S *boomerangs*. Each boomerang has two straight sides and a reflex chain. Figure 2(a) gives an example. For

[1] It is possible that $\mathcal{B}_S$ is unbounded, but we will assume that $\mathcal{B}_S$ is bounded for simplicity.

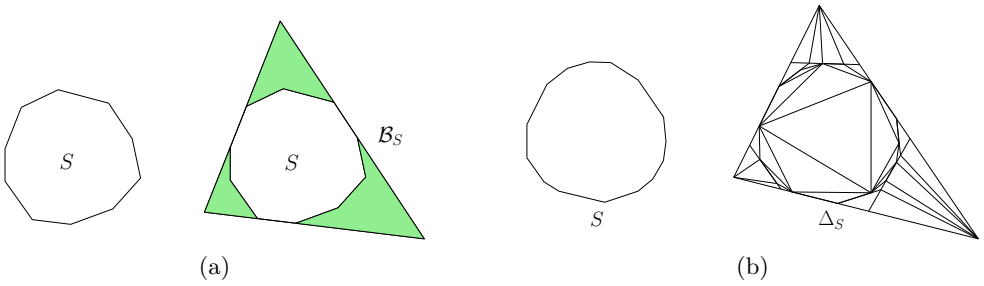


Figure 2 (a) Boomerangs (shown shaded). (b) An example in which S is just one convex polygon.

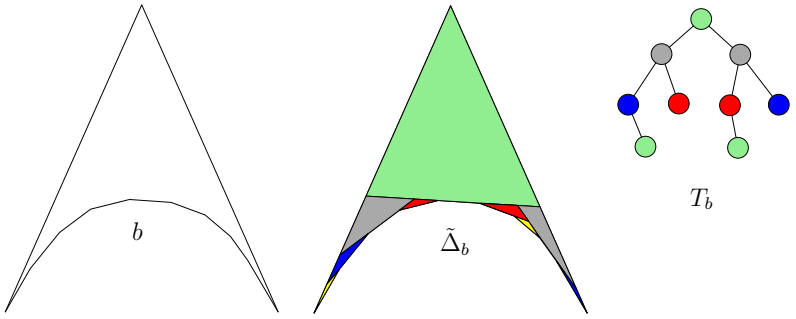


Figure 3 The nodes of T_b are given the same colors as the corresponding regions in $\tilde{\Delta}_b$.

each boomerang b, we call the procedure SplitBR to partition b hierarchically into triangular regions as well as to construct a binary tree that represents this hierarchy. Denote the output partition of b by $\tilde{\Delta}_b$ and the binary tree by T_b. Figure 3 gives an example. The binary tree T_b is not constructed in [8], but we will need it later. SplitBR runs in $O(|b|)$ time, $\tilde{\Delta}_b$ has $O(|b|)$ size, and T_b has $O(\log |b|)$ height.

SplitBR(b)

1. If b is a triangle, then return.
2. Take the middle edge e of the reflex chain of b.
3. Cut b with the support line of e into a triangle t and two smaller boomerangs b_1 and b_2.
4. Call SplitBR(b_1) and SplitBR(b_2) to obtain $\tilde{\Delta}_{b_1}$, T_{b_1}, $\tilde{\Delta}_{b_2}$, and T_{b_2}.
5. $\tilde{\Delta}_b := \{t\} \cup \tilde{\Delta}_{b_1} \cup \tilde{\Delta}_{b_2}$.
6. Create the binary tree T_b with root v containing t. Make T_{b_1} and T_{b_2} left and right subtrees of v.
7. Return $\tilde{\Delta}_b$ and T_b.

Finally, for every triangular region r in $\tilde{\Delta}_b$, there is exactly one side e of r that bounds S. This side e contains $O(\log |b|)$ vertices. We call TriBR(b) to obtain a triangulation of b.

TriBR(b)

1. For each triangular region r in $\tilde{\Delta}_b$, do
   a. take the side e of r that bounds S,
   b. add edges to connect the vertices in e to the vertex of r opposite e.

Every line segment in b intersects $O(\log |b|)$ triangular regions in $\tilde{\Delta}_b$ and there are $O(\log |b|)$ triangles in each triangular region. So each line segment in b intersects $O(\log^2 |b|)$ triangles in the triangulation of b.

We use Δ_S to denote the resulting triangulation of the boomerangs and the bounded regions in S. Figure 2(b) gives an example. There are $O(n)$ vertices in Δ_S and Δ_S can be constructed in $O(n)$ time. Theorem 1 is applied to Δ_S to produce a data structure for answering point location queries in S.

▶ **Theorem 2** ([8]). *Let S be a planar convex subdivision with n vertices. There is a point-line comparison based data structure that uses $O(n)$ space and processes any online sequence σ of point location queries in S in $O(|\sigma| \log\log n + n + \mathrm{OPT})$ time, where* OPT *is the minimum time needed by any point location linear decision tree for S to process σ. The time bound includes the $O(n)$ preprocessing time.*

3 Planar convex subdivision

Let S denote a planar convex subdivision with n vertices. We first present a solution with processing time $O(|\sigma| \log\log\log n + n + \mathrm{OPT})$. Then, we bootstrap from this solution to obtain the optimal result.

3.1 First solution

We first compute Δ_S as described in Section 2. Let D_S be the point location structure in Theorem 2 for S. Note that D_S evolves as σ is processed. Let $f(n)$ denote the function $(\log_2 n)^6$. For any $j \geq 1$, whenever D_S has been used to answer the j-th subset of $f(n)$ queries, we extract a subset of triangles from Δ_S, construct a new triangulation Δ_j using this subset, and then compute a point location structure D_j for Δ_j. Once D_j has been constructed, the next query is answered using D_j first. If D_j locates the query point in a region (bounded or exterior) in S, we are done; otherwise, we use D_S to answer the query. We elaborate on the construction of Δ_j and D_j in the following sections.

3.1.1 Triangulation Δ_j

We extract the subset X of $f(n)(\log_2 n)^{-2}$ triangles in Δ_S that have the highest $f(n)(\log_2 n)^{-2}$ access frequencies currently. Some triangles in X lie in bounded regions in S and some may lie outside S. To extract X quickly, we maintain a doubly linked list A such that the i-th entry of A stores a doubly linked list of triangles with the i-th highest frequency. Whenever we need to output X, we scan the lists in the entries of A in order until we have collected $f(n)(\log_2 n)^{-2}$ triangles. Whenever the frequency of a triangle $t \in \Delta_S$ is incremented, we need to relocate t within A. Suppose that t is currently stored in the list at the i-th entry of A. If the triangles in the list at the $(i-1)$-th entry of A have the same frequency as t, then we move t to the end of the list at the $(i-1)$-th entry. Otherwise, the triangles in the list at the $(i-1)$-th entry have a higher frequency than t, and so we insert a new entry of A between the $(i-1)$-th and the i-th entries and make t a singleton list at this new entry of A. If the list at the i-th entry of A becomes empty after moving t, we delete this entry of A. So each update of A takes $O(1)$ time.

We can assume that for each triangle t in Δ_S, if t lies in a region r in S, then t stores the region id r. Moreover, if r is the exterior region of S, then $t \subseteq \tilde{\Delta}_b$ for some boomerang b and t also stores the id of the triangular region in $\tilde{\Delta}_b$ that contains t. By sorting the triangles in X with respect to their region ids, we can find the triangles in $r \cap X$ for every region r in S.

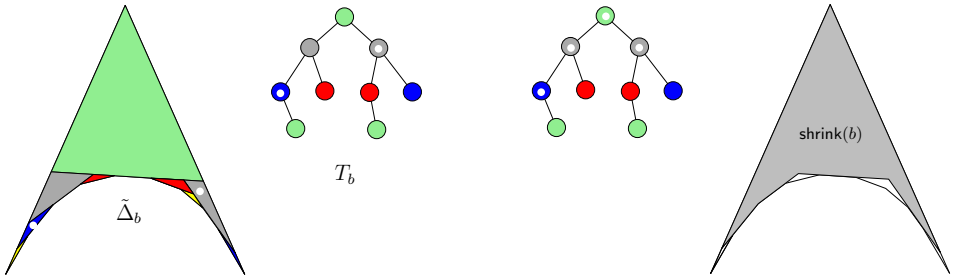


Figure 4 The two triangular regions in $\tilde{\Delta}_b$ with white dots contain some triangles in X. Corresponding nodes in T_b are also marked with white dots. Then, all ancestors of these nodes in T_b are marked, and the union of the corresponding triangular regions in $\tilde{\Delta}_b$ is a boomerang $\mathsf{shrink}(b)$ (shown shaded).

For each bounded region r in S, let $\mathrm{conv}(r \cap X)$ denote the convex hull of the triangles in $r \cap X$ and we can compute $\mathrm{conv}(r \cap X)$ in $O(|r \cap X| \log |r \cap X|)$ time. Then, we call $\mathsf{TriReg}(\mathrm{conv}(r \cap X))$ to triangulate $\mathrm{conv}(r \cap X)$. Each resulting triangle stores the region id r. Summing over all bounded regions in S, the total running time is $O(|X| \log |X|) = O(f(n)/\log n)$ and the total number of triangles produced is $O(|X|) = O(f(n)(\log n)^{-2})$.

Next, we handle the triangles in X outside S. Let b be one of the boomerangs between the boundaries of $\mathcal{B}_S$ and S. For each triangle $t \in b \cap X$, we mark the triangular region r in the binary tree T_b that contains t and we also mark all ancestors of r in T_b. We form the union of the marked triangular regions. Denote the union by $\mathsf{shrink}(b)$. Note that $\mathsf{shrink}(b)$ is a boomerang and every edge in the reflex chain of $\mathsf{shrink}(b)$ supports an outer boundary edge of S. Figure 4 gives an example. Also, $b \cap X \subseteq \mathsf{shrink}(b)$, $|\mathsf{shrink}(b)| = O(|b \cap X| \log |b|)$, and $\mathsf{shrink}(b)$ can be computed in $O(|b \cap X| \log |b|)$ time. We call $\mathsf{SplitBR}(\mathsf{shrink}(b))$ and then $\mathsf{TriBR}(\mathsf{shrink}(b))$ to obtain a triangulation of $\mathsf{shrink}(b)$. Each resulting triangle stores the id of the exterior region of S. Summing over all three boomerangs between the boundaries of $\mathcal{B}_S$ and S, the total running time is $O(|X| \log n) = O(f(n)/\log n)$ and the total number of triangles produced is $O(|X| \log n) = O(f(n)/\log n)$.

Collect all $O(f(n)/\log n)$ triangles computed in the above. By a plane sweep, we can add edges in $O\left(\frac{f(n)}{\log n} \log \frac{f(n)}{\log n}\right)$ time to form a triangulation that contains all triangles collected and has size $O(f(n)/\log n)$. This is the triangulation Δ_j desired. The total construction time of Δ_j is $O\left(\frac{f(n)}{\log n} \log \frac{f(n)}{\log n}\right)$. The extra triangles added by the plane sweep do not store the id of any region in S, and therefore, query points that fall into such triangles are not located successfully in S.

3.1.2 Structure D_j, querying, and frequencies

The access frequencies in Δ_S are initialized to be zero before processing σ. The subscript of Δ_j and D_j increases monotonically as we process σ. When the construction of Δ_j and D_j completes, we forget about Δ_{j-1} and D_{j-1} and reuse their storage. The access frequencies in Δ_j are initialized to be zero.

D_j consists of two point location structures D'_j and D''_j. D'_j is obtained by invoking Theorem 1, the result in [13], on Δ_j. D''_j is a worst-case optimal planar point location structure (e.g. [14]). The querying procedure works as follows. Let q be the next query point. We check in $O(1)$ time whether q lies inside $\mathcal{B}_S$. If not, we just output that q is outside S. Suppose that q lies inside $\mathcal{B}_S$. We query D_j by alternating the search steps in D'_j and D''_j.

We stop as soon as a triangle t in Δ_j containing q is found. If t stores a region (bounded or exterior) of S, then we output that region. Otherwise, we use D_S to locate the triangle t' in Δ_S that contains q, and we output the region of S (bounded or exterior) stored at t'.

After locating q, we update the access frequencies in Δ_S or Δ_j. This update is important because the frequencies govern how the method in [13] will adjust D_S and D_j in order to adapt to incoming queries. If q lies outside $\mathcal{B}_S$, we do not change any frequency in Δ_S and Δ_j. Suppose that q lies inside $\mathcal{B}_S$. If q is located in a triangle $t \in \Delta_j$ and t stores a region of S, then we increment the frequency of t in Δ_j and we are done. The frequencies in Δ_S do not change in this case. On the other hand, if the search in D_j does not report a region of S, then q is subsequently located in S by D_S and we increment the frequency of the triangle in Δ_S that contains q. The frequencies in Δ_j do not change in this case.

There are some consequences due to our frequency update. Consider two online query sequences $\alpha_j \subseteq \alpha$ such that D_j can successfully locate in S the query points in α_j, but not the query points in $\alpha \backslash \alpha_j$. Therefore, query points in $\alpha \backslash \alpha_j$ do not cause any change to D_j. Let $D_j(\alpha)$ denote the running time of D_j on α (excluding the preprocessing time of D_j). We conclude that

$$D_j(\alpha) = O(D_j(\alpha_j) + |\alpha \backslash \alpha_j| \log |\Delta_j|) \tag{1}$$

because each query in $\alpha \backslash \alpha_j$ can be answered by D_j'' in $O(\log |\Delta_j|)$ worst-case time.

3.1.3 Analysis

We first analyze the performance of D_j. Among all point location linear decision trees for S, let D be the one that takes the minimum time to process σ. We convert D to a point location linear decision tree for Δ_j as follows.

Each leaf node v of D corresponds to a convex polygon ρ in a region of S. If ρ has k sides, then v has depth at least k as each node of D applies a cut along a line. Therefore, we can expand v into a linear decision subtree so that the leaf nodes of this subtree correspond to a triangulation of ρ and the height of this subtree is at most $k - 2$.

Let $D^{\sigma}_{\min}$ denote the linear decision tree obtained by expanding D as described above. The triangular regions at the leaf nodes of $D^{\sigma}_{\min}$ form a refinement of S. Locating a query point q in this refinement of S using $D^{\sigma}_{\min}$ has the same asymptotical complexity as locating q in S using D.

Let t be the triangle at a leaf node of $D^{\sigma}_{\min}$. We discuss how to expand this leaf node to a linear decision subtree depending on whether t lies in a bounded or unbounded region of S.

Suppose that t lies in a bounded region r of S. Recall that X is the subset of triangles extracted from Δ_S for constructing Δ_j. All vertices of $\text{conv}(r \cap X)$ lie on the boundary of r, so t intersects $O(\log |r \cap X|) = O(\log \log n)$ triangles in $\Delta_j \cap \text{conv}(r \cap X)$, which refine t into a planar subdivision P_t of size $O(\log \log n)$. We expand the leaf node of $D^{\sigma}_{\min}$ storing t to a linear decision subtree L_t that performs point location in P_t in $O(\log \log \log n)$ worst-case query time. Some leaf nodes of L_t correspond to regions in the refinement of t that are outside $\text{conv}(r \cap X)$. We need to expand such leaf nodes further in order to locate query points that fall into $t \setminus \text{conv}(r \cap X)$ in a triangle in Δ_j. We will not be interested in the query time for such query points, so we can expand these leaf nodes of L_t arbitrarily.

Suppose that t lies in the unbounded region of S. We expand the leaf node storing t into a linear decision subtree L'_t of $O(1)$ height such that each leaf node of L'_t corresponds to a triangle that lies inside or outside $t \cap \mathcal{B}_S$. At each leaf node of L'_t outside $t \cap \mathcal{B}_S$, we can output the exterior of $\mathcal{B}_S$. All leaf nodes of L'_t inside $\mathcal{B}_S$ lie in a boomerang b between the boundaries of $\mathcal{B}_S$ and S. Take such a leaf node of L'_t and let t' be the triangle stored

there. We are concerned with the overlay of t' and $\mathsf{shrink}(b)$. Since every edge of the reflex chain of $\mathsf{shrink}(b)$ supports an outer boundary edge of S, every triangular region in $\mathsf{shrink}(b)$ produced by SplitBR is incident on an outer boundary edge of S. Since the interior of t' cannot intersect the boundary of S or any bounded region in S, the interior of t' contains at most one vertex in the reflex chain of $\mathsf{shrink}(b)$. Thus, the boundary of t' inside $\mathsf{shrink}(b)$ consists of $O(1)$ line segments. Each segment intersects $O(\log|\mathsf{shrink}(b)|)$ triangular regions in $\mathsf{shrink}(b)$ produced by SplitBR, and each triangular region contains $O(\log|\mathsf{shrink}(b)|)$ triangles produced by TriBR. As a result, t' intersects $O(\log^2|\mathsf{shrink}(b)|) = O((\log\log n)^2)$ triangles in the triangulation of $\mathsf{shrink}(b)$. Therefore, we can expand the leaf node storing t' into a linear decision subtree as in the previous paragraph.

Let D' be the linear decision tree obtained by expanding $D^{\sigma}_{\min}$ as described above. Let q be a query point. If q can be located successfully in S by D_j, the search in D' traverses a root-to-leaf path in $D^{\sigma}_{\min}$ and then another path of length $O(\log\log\log n)$ to a leaf of D'. Let $D^{\sigma}_{\min}(\alpha)$ and $D'(\alpha)$ denote the running times of $D^{\sigma}_{\min}$ and D' on an online query sequence α, respectively. Then, for any online query sequence α_j such that query points in α_j can be located successfully in S by D_j,

$$D'(\alpha_j) = O(D^{\sigma}_{\min}(\alpha_j) + |\alpha_j|\log\log\log n). \tag{2}$$

For every pair of online query sequences $\alpha_j \subseteq \alpha$ such that α_j is the maximum subsequence of α that can be located successfully in S by D_j, by (1), $D_j(\alpha) = O(D_j(\alpha_j) + |\alpha\backslash\alpha_j|\log|\Delta_j|)$. By Theorem 1, D_j performs no worse than D' on α_j. Let D^{pre}_j denote the $O(|\Delta_j|\log|\Delta_j|)$ preprocessing time to construct both Δ_j and D_j. Therefore,

$$\begin{aligned}
&D_j(\alpha) + D^{\mathrm{pre}}_j \\
&= O\big(D_j(\alpha_j) + |\alpha\backslash\alpha_j|\log|\Delta_j|\big) + O\big(|\Delta_j|\log|\Delta_j|\big) \\
&= O\big(D'(\alpha_j) + |\Delta_j| + |\alpha\backslash\alpha_j|\log|\Delta_j|\big) + O\big(|\Delta_j|\log|\Delta_j|\big) && (\because \text{Theorem 1}) \\
&= O\big(D^{\sigma}_{\min}(\alpha_j) + |\Delta_j|\log|\Delta_j| + |\alpha_j|\log\log\log n + |\alpha\backslash\alpha_j|\log|\Delta_j|\big). && (\because (2))
\end{aligned}$$

The next result summarizes the discussion above.

▶ **Lemma 3.**

(i) *For every pair of online query sequences $\alpha_j \subseteq \alpha$ such that α_j is the maximum subsequence of α that can be located successfully in S by D_j, $D_j(\alpha) + D^{\mathrm{pre}}_j = O\big(D^{\sigma}_{\min}(\alpha_j) + |\Delta_j|\log|\Delta_j| + |\alpha\backslash\alpha_j|\log\log n + |\alpha_j|\log\log\log n\big)$.*

(ii) *$D^{\sigma}_{\min}(\sigma) = O(\mathrm{OPT})$, where OPT is the minimum time needed by any point location linear decision tree for S to process σ.*

We are ready to analyze the performance of the first solution.

▶ **Lemma 4.** *Let S be a planar convex subdivision with n vertices. There is a point-line comparison based data structure that processes any online sequence σ of point location queries in S in $O(|\sigma|\log\log\log n + n + \mathrm{OPT})$ time. The time bound includes the preprocessing time.*

Proof. Let σ_S denote the subsequence of queries in σ that are answered by D_S. For each $j \geq 1$, we use σ_j to denote the subsequence of σ that are located successfully in S by D_j. Therefore, $\bigcup_{j\geq 1}\sigma_j = \sigma\backslash\sigma_S$. Note that $\sigma_j \cap \sigma_k = \emptyset$ if $j \neq k$. Let Γ denote the total processing time required by all D_j's, including the preprocessing time D^{pre}_j. Note that Γ also includes the time spent on unsuccessfully locating some query points in σ_S by the D_j's. Lemma 3(i) implies that

$$\Gamma = O\Big(\sum_j D^{\sigma}_{\min}(\sigma_j) + \sum_j |\Delta_j|\log|\Delta_j| + |\sigma_S|\log\log n + |\sigma\backslash\sigma_S|\log\log\log n\Big).$$

Each Δ_j has $O(f(n)/\log n)$ size and Δ_j is constructed after answering $f(n)$ new queries using D_S. So $|\Delta_j|\log|\Delta_j|$ can be charged to these queries, i.e., $\sum_j |\Delta_j|\log|\Delta_j| = O(|\sigma|)$. Therefore,

$$\Gamma = O\Big(D^{\sigma}_{\min}(\sigma\backslash\sigma_S) + |\sigma| + |\sigma_S|\log\log n + |\sigma\backslash\sigma_S|\log\log\log n\Big).$$

Let Γ_0 denote the total processing time required by D_S on σ_S, including the $O(n)$ preprocessing time to construct Δ_S and D_S. By Theorem 2, $\Gamma_0 = O(D^{\sigma}_{\min}(\sigma_S)+|\Delta_S|+|\sigma_S|\log\log n)$. Therefore,

$$\Gamma_0 + \Gamma = O\Big(D^{\sigma}_{\min}(\sigma) + n + |\sigma| + |\sigma_S|\log\log n + |\sigma\backslash\sigma_S|\log\log\log n\Big). \tag{3}$$

We will show that the term $|\sigma_S|\log\log n$ can be absorbed by other terms in (3). For query points in σ_S that end in leaf nodes of $D^{\sigma}_{\min}$ at depth greater than $\log_2\log_2 n$, their contribution to $|\sigma_S|\log\log n$ can be absorbed by $D^{\sigma}_{\min}(\sigma_S)$. We bound the number of remaining query points in σ_S in Claim 5 below.

▶ **Claim 5.** *Let $\hat{\sigma}_S$ be the subsequence of σ_S such that each query point in $\hat{\sigma}_S$ lies in some triangle (leaf node) in $D^{\sigma}_{\min}$ at depth $\log_2\log_2 n$ or less. Then, $|\hat{\sigma}_S| = O(\log^9 n + |\sigma_S|/\log n)$.*

Proof. We will make use of the following facts:

Fact 1: At most $2^{1+\log_2\log_2 n} - 1 = 2\log_2 n - 1$ nodes in $D^{\sigma}_{\min}$ have depth at most $\log_2\log_2 n$ because $D^{\sigma}_{\min}$ is a binary tree.

Fact 2: For each triangle $t \in \Delta_S$, if the current access frequency of t is at least $|\sigma_S|(\log_2 n)^2/f(n)$, then t must be included in the set X for the next construction of Δ_j and D_j. The reason is that the sum of frequencies in Δ_S is at most $|\sigma_S|$, so there are at most $f(n)(\log_2 n)^{-2}$ triangles in Δ_S with frequencies at least $|\sigma_S|(\log_2 n)^2/f(n)$, implying that t is one of the top $f(n)(\log_2 n)^{-2}$ frequently accessed triangles.

Let Z be the subset of triangles in Δ_S that overlap with some triangle (leaf node) in $D^{\sigma}_{\min}$ at depth $\log_2\log_2 n$ or less. By Fact 1, there are at most $2\log_2 n - 1$ triangles (leaf nodes) in $D^{\sigma}_{\min}$ at depth $\log_2\log_2 n$ or less. Each such triangle must lie inside a region (bounded or exterior) of S in order that $D^{\sigma}_{\min}$ answers a point location query correctly. So each such triangle intersects $O(\log^2 n)$ triangles in Δ_S. It follows that

$$|Z| = O(\log^3 n). \tag{4}$$

Consider a triangle $t \in \Delta_S$ that contains a query point in $\hat{\sigma}_S$. Thus, $t \in Z$ because t must overlap with some triangle (leaf node) in $D^{\sigma}_{\min}$ at depth $\log_2\log_2 n$ or less. If the frequency of t in Δ_S never reaches $|\sigma_S|(\log_2 n)^2/f(n)$, then at most $|\sigma_S|(\log_2 n)^2/f(n)$ query points in t are from $\hat{\sigma}_S$. Suppose that the frequency of t in Δ_S reaches $|\sigma_S|(\log_2 n)^2/f(n)$, say after the construction of Δ_j and before the construction of Δ_{j+1}. At most $f(n)$ queries can be answered by D_S during this period. It means that the frequency of t in Δ_S is at most $f(n) + |\sigma_S|(\log_2 n)^2/f(n)$ before the construction of Δ_{j+1}. By Fact 2, t will be included in Δ_k for all $k > j$. Every query point that falls in t after the construction of Δ_{j+1} will be located successfully in S by D_k for some $k \geq j+1$. Thus, the frequency of t in Δ_S will not be increased further and at most $f(n) + |\sigma_S|(\log_2 n)^2/f(n)$ query points in t are from $\hat{\sigma}_S$. Hence, $|\hat{\sigma}_S| \leq (f(n) + |\sigma_S|(\log_2 n)^2/f(n)) \cdot |Z|$. Recall that $f(n) = (\log_2 n)^6$. Since $|Z| = O(\log^3 n)$ by (4), we obtain $|\hat{\sigma}_S| = O(\log^9 n + |\sigma_S|/\log n)$. ◀

If $|\sigma_S \backslash \hat{\sigma}_S| < |\hat{\sigma}_S|$, we obtain the following from (3):

$$\begin{aligned}
\Gamma_0 + \Gamma &= O\big(D^{\sigma}_{\min}(\sigma) + n + |\sigma| + |\sigma_S \backslash \hat{\sigma}_S| \log\log n + |\hat{\sigma}_S| \log\log n + \\
&\qquad |\sigma \backslash \sigma_S| \log\log\log n\big) \\
&= O\big(D^{\sigma}_{\min}(\sigma) + n + |\sigma| + |\hat{\sigma}_S| \log\log n + |\sigma \backslash \sigma_S| \log\log\log n\big) \\
&= O\big(D^{\sigma}_{\min}(\sigma) + n + |\sigma| + |\sigma_S| + \log^9 n \log\log n + \\
&\qquad |\sigma \backslash \sigma_S| \log\log\log n\big) \qquad (\because \text{Claim 5}) \\
&= O\big(D^{\sigma}_{\min}(\sigma) + n + |\sigma| + |\sigma \backslash \sigma_S| \log\log\log n\big).
\end{aligned}$$

If $|\sigma_S \backslash \hat{\sigma}_S| \geq |\hat{\sigma}_S|$, we obtain the following from (3):

$$\begin{aligned}
\Gamma_0 + \Gamma &= O\big(D^{\sigma}_{\min}(\sigma) + n + |\sigma| + |\sigma_S \backslash \hat{\sigma}_S| \log\log n + |\hat{\sigma}_S| \log\log n + \\
&\qquad |\sigma \backslash \sigma_S| \log\log\log n\big) \\
&= O\big(D^{\sigma}_{\min}(\sigma) + n + |\sigma| + |\sigma_S \backslash \hat{\sigma}_S| \log\log n + |\sigma \backslash \sigma_S| \log\log\log n\big) \\
&= O\big(D^{\sigma}_{\min}(\sigma) + n + |\sigma| + |\sigma \backslash \sigma_S| \log\log\log n\big).
\end{aligned}$$

In the last step above, we use the fact that $D^{\sigma}_{\min}(\sigma_S) = \Omega(|\sigma_S \backslash \hat{\sigma}_S| \log\log n)$, which is true because each query point in $\sigma_S \backslash \hat{\sigma}_S$ lies in a triangle (leaf node) in $D^{\sigma}_{\min}$ at depth greater than $\log_2 \log_2 n$.

As a result, no matter whether $|\sigma_S \backslash \hat{\sigma}_S|$ or $|\hat{\sigma}_S|$ is greater than the other, we have $\Gamma_0 + \Gamma = O\left(D^{\sigma}_{\min}(\sigma) + n + |\sigma| \log\log\log n\right) = O\left(\mathrm{OPT} + n + |\sigma| \log\log\log n\right)$ by Lemma 3(ii). ◀

3.2 Optimal solution

We apply the method in Section 3.1 recursively to obtain a multi-level data structure. To facilitate the description of this new strategy, we revise our notation as follows. We relabel each triangulation Δ_j and each point location structure D_j in Section 3.1 as $\Delta_{1,j}$ and $D_{1,j}$. The extra subscript 1 signifies that these are triangulations and structures at the first level. We use $\Delta_{0,1}$ and $D_{0,1}$ to denote Δ_S and D_S, respectively. At any level $i \geq 1$, a new triangulation $\Delta_{i,j}$ and a new point location structure $D_{i,j}$ will be constructed from time to time to replace $\Delta_{i,j-1}$ and $D_{i,j-1}$. At level 0, $\Delta_{0,1}$ and $D_{0,1}$ will never be replaced, and there are no other triangulation and point location structure at level 0. When it is not important to distinguish the current index j at a level, we use $\Delta_{i,*}$ and $D_{i,*}$ to denote the current triangulation and point location structure at level i.

We have multiple levels of triangulations and point location structures at any time: $(\Delta_{0,1}, D_{0,1})$, $(\Delta_{1,*}, D_{1,*}), \ldots, (\Delta_{m,*}, D_{m,*})$, where m is the highest level currently. Let q be the next query point. We first check if q lies inside $\mathcal{B}_S$ in $O(1)$ time. If not, we output the exterior region of S. Suppose that q lies inside $\mathcal{B}_S$. We first query $D_{m,*}$ with q. If we fail to locate a region in S containing q, then we try $D_{m-1,*}$. If that also fails, we try $D_{m-2,*}$ and so on. The location of q will succeed by $D_{0,1}$ the latest.

After locating q, we need to update the access frequencies in the triangulations. This update is important because the frequencies govern how the method in [13] adapts the point location structures to incoming queries. If q lies outside $\mathcal{B}_S$, we do not change any frequency in any triangulation. If q is located in a triangle that stores a region of S at level i, we increment the frequency of the triangle in $\Delta_{i,*}$ that contains q. The frequencies in triangulations at other levels do not change.

The number of levels increases monotonically as we process σ. The triangulation and point location structure at each level are rebuilt from time to time. We use *i-rebuild* to refer to a rebuild at level i. Use n_0 to denote n and define $n_i = f(n_{i-1})/\log_2 n_{i-1}$ for $i \geq 1$.

For any $i \geq 0$, if $f(n_i)$ query points are located successfully in S by $D_{i,*}$ since the last $(i+1)$-rebuild and no i-rebuild has happened during these $f(n_i)$ queries, then we perform a new $(i+1)$-rebuild. That is, if level $i+1$ does not exist, then we construct $\Delta_{i+1,1}$ from $\Delta_{i,*}$ and then $D_{i+1,1}$ for $\Delta_{i+1,1}$; otherwise, if $\Delta_{i+1,k}$ and $D_{i+1,k}$ are currently stored at level $i+1$, then we replace them by constructing $\Delta_{i+1,k+1}$ from $\Delta_{i,*}$ and then $D_{i+1,k+1}$ for $\Delta_{i+1,k+1}$. The construction works as follows.

We extract the subset X of $f(n_i)(\log_2 n_i)^{-2}$ triangles in $\Delta_{i,*}$ that have the highest $f(n_i)(\log_2 n_i)^{-2}$ access frequencies in $\Delta_{i,*}$. Then, we proceed as in Section 3.1.1 to produce $\Delta_{i+1,k+1}$ from X. Details are given below. For each bounded region r of S, we compute $\mathrm{conv}(r \cap X)$ and then triangulate it by calling $\mathsf{TriReg}(\mathrm{conv}(r \cap X))$. This produces $O(|r \cap X|)$ triangles and takes $O(|r \cap X| \log |r \cap X|)$ time. Each resulting triangle stores the region id r. Summing over all bounded regions, we obtain $O(|X|) = O(f(n_i)(\log n_i)^{-2}) = o(n_{i+1})$ triangles in $O(|X| \log |X|) = O(n_{i+1} \log n_{i+1})$ time. Consider the processing of triangles in X outside S. In a 1-rebuild as described in Section 3.1.1, for each boomerang b between the boundaries of $\mathcal{B}_S$ and S, we compute another boomerang $\mathsf{shrink}(b) \subseteq b$. Suppose that there is a 2-rebuild before the next 1-rebuild. Note that any triangle outside S that is selected in this 2-rebuild must be contained in $\mathsf{shrink}(b')$ for some boomerang b' between the boundaries of $\mathcal{B}_S$ and S. Assume that some triangles lying in $\mathsf{shrink}(b)$ are selected. Note that these triangles belong to the triangulation of $\mathsf{shrink}(b)$ produced by $\mathsf{SplitBR}$ and TriBR. Since $\mathsf{shrink}(b)$ is a boomerang, there is also a hierarchy on the triangular regions produced by $\mathsf{SplitBR}$ as in Figure 4. As in Section 3.1.1, we first mark the triangular regions in the hierarchy that store the selected triangles and then mark their ancestors in the hierarchy. The union of the marked triangular regions is another boomerang $\mathsf{shrink}(\mathsf{shrink}(b)) \subseteq \mathsf{shrink}(b)$ and it is triangulated by calling $\mathsf{SplitBR}$ and TriBR. Each resulting triangle stores the id of the exterior region of S. Each triangle also stores the id of the triangular region containing it, which is produced by the call $\mathsf{SplitBR}(\mathsf{shrink}(\mathsf{shrink}(b)))$. Note that $\mathsf{shrink}(\mathsf{shrink}(b))$ has $O(f(n_1)/\log n_1)$ size and its processing takes $O(f(n_1)/\log n_1)$ time. In general, in an $(i+1)$-rebuild, the processing of triangles in X outside S takes $O(f(n_i)/\log n_i) = O(n_{i+1})$ time and produces at most three boomerangs of $O(n_{i+1})$ size and $O(n_{i+1})$ triangles in these boomerangs.

The triangles computed above may form disconnected components. We apply a plane sweep in $O(n_{i+1} \log n_{i+1})$ time to connect them with triangles. These extra triangles do not store any region in S, so query points that fall into them are not located successfully in S. The resulting triangulation is $\Delta_{i+1,k+1}$. The frequencies of all triangles in $\Delta_{i+1,k+1}$ are initialized to be zero. We apply Theorem 1 to $\Delta_{i+1,k+1}$ to obtain the point location structure $D_{i+1,k+1}$. In summary, the $(i+1)$-rebuild takes $O(n_{i+1} \log n_{i+1})$ time and $|\Delta_{i+1,k+1}| = O(n_{i+1})$.

We do not increase the number of levels anymore when the highest level m reaches the value such that $n_m < 30^5$ for the first time.[2] The triangulation size is $O(n_m) = O(1)$. However, we will still perform i-rebuild for any $i \in [1, m]$. Also, if a query point is located successfully in S at this highest possible level m, we do not change any frequency in $\Delta_{m,*}$. The query time is only $O(1)$ anyway.

▶ Remark. Let m be the highest level currently. When an i-rebuild is performed for some $i < m$, the triangulations at levels $i+1, \ldots, m$ are unaffected. Query answering will still start from level m. The selected triangles on which the construction of $\Delta_{i+1,*}$ was based may not be related to the selected triangles on which the construction of the new $\Delta_{i,*}$ is based.

[2] This particular choice goes well with the proof of Claim 8.

▶ **Theorem 6.** *Let S be a planar convex subdivision with n vertices. There is a point-line comparison based structure that uses $O(n)$ space and processes any online sequence σ of point location queries in S in $O(n + \mathrm{OPT})$ time, where* OPT *is the minimum time required by any point location linear decision tree for S to process σ. The time bound includes the $O(n)$ preprocessing time.*

Proof. For any online query sequence α, we use $D_{i,j}(\alpha)$ to denote the time needed by $D_{i,j}$ to process α and $D^{\mathrm{pre}}_{i,j}$ to denote the preprocessing time to construct both $\Delta_{i,j}$ and $D_{i,j}$. Define the following subsets of query points:

$$\begin{aligned}\sigma_{i,j} &= \{q \in \sigma : q \text{ is located successfully in } S \text{ using } D_{i,j}\ \},\\ \sigma_i &= \{q \in \sigma : q \text{ is located successfully in } S \text{ at level } i\},\\ \sigma_{<i} &= \{q \in \sigma : q \text{ is located successfully in } S \text{ at some level less than } i\}.\end{aligned}$$

By definition, $\sigma = \bigcup_{i,j} \sigma_{i,j}$, the $\sigma_{i,j}$'s are mutually disjoint, $\sigma_i = \bigcup_j \sigma_{i,j}$, and $\sigma_{<i} = \bigcup_{a=0}^{i-1} \sigma_a$. Claim 7 below is analogous to Lemma 3(i) and it can be proved by the same argument.

▶ **Claim 7.** *For all $i \in [1, m]$ and all online query sequences $\alpha_{i,j} \subseteq \alpha$ such that $\alpha_{i,j}$ is the maximum subsequence of α that can be successfully located in S by $D_{i,j}$, $D_{i,j}(\alpha) + D^{\mathrm{pre}}_{i,j} = O(D^{\sigma}_{\min}(\alpha_{i,j}) + |\Delta_{i,j}| \log |\Delta_{i,j}| + |\alpha \backslash \alpha_{i,j}| \log n_i + |\alpha_{i,j}| \log\log n_i)$.*

Let Γ_i denote the total processing time required by $D_{i,j}$ over all j, including the preprocessing time $D^{\mathrm{pre}}_{i,j}$. Recall that $D^{\mathrm{pre}}_{i,j} = O(|\Delta_{i,j}| \log |\Delta_{i,j}|) = O(n_i \log n_i)$ for $i > 0$ and $D^{\mathrm{pre}}_{0,1} = O(|\Delta_{0,1}|) = O(n_0)$. Note that for $i > 0$, Γ_i includes the time spent on unsuccessfully locating some query points in $\sigma_{<i}$. By Claim 7, for $i \in [1, m]$,

$$\begin{aligned}\Gamma_i &= O\Big(\sum_j D^{\sigma}_{\min}(\sigma_{i,j}) + \sum_j n_i \log n_i + |\sigma_{<i}| \log n_i + \sum_j |\sigma_{i,j}| \log\log n_i\Big)\\ &= O\Big(D^{\sigma}_{\min}(\sigma_i) + \sum_j n_i \log n_i + |\sigma_{<i}| \log n_i + |\sigma_i| \log\log n_i\Big).\end{aligned}$$

By Theorem 2, $\Gamma_0 = D_{0,1}(\sigma_0) + O\big(|\Delta_{0,1}|\big) = O\big(D^{\sigma}_{\min}(\sigma_0) + n_0 + |\sigma_0| \log\log n_0\big)$. Therefore,

$$\sum_{i=0}^{m} \Gamma_i = O\Big(\sum_{i=0}^{m} D^{\sigma}_{\min}(\sigma_i) + n_0 + \sum_{i=1}^{m}\sum_j n_i \log n_i + \sum_{i=0}^{m} |\sigma_i| \log\log n_i + \sum_{i=1}^{m} |\sigma_{<i}| \log n_i\Big).$$

Since $\Delta_{i,j}$ is constructed after answering $f(n_{i-1})$ new queries using $D_{i-1,*}$, the preprocessing time of $O(n_i \log n_i) = o(f(n_{i-1}))$ for constructing $\Delta_{i,j}$ and $D_{i,j}$ can be charged to these new queries. So $\sum_{i=1}^{m}\sum_j n_i \log n_i$ can be charged to the queries in σ, i.e., $\sum_{i=1}^{m}\sum_j n_i \log n_i = O(|\sigma|)$. We rewrite the term $\sum_{i=1}^{m} |\sigma_{<i}| \log n_i = \sum_{i=0}^{m-1}(|\sigma_i| \sum_{l=i+1}^{m} \log n_l)$.

▶ **Claim 8.** *For all $i \in [0, m-1]$, $\sum_{l=i+1}^{m} \log_2 n_l < 35 \log_2\log_2 n_i$.*

Consequently,

$$\begin{aligned}\sum_{i=0}^{m} \Gamma_i &= O\Big(\sum_{i=0}^{m} D^{\sigma}_{\min}(\sigma_i) + n + |\sigma| + \sum_{i=0}^{m} |\sigma_i| \log\log n_i + \sum_{i=0}^{m-1} |\sigma_i| \log\log n_i\Big)\\ &= O\Big(D^{\sigma}_{\min}(\sigma) + n + |\sigma| + \sum_{i=0}^{m} |\sigma_i| \log\log n_i\Big). \qquad (5)\end{aligned}$$

Define the following quantities:

$$\hat{\sigma}_{i,j} = \big\{q \in \sigma_{i,j} : q \text{ lies in some leaf node of } D^{\sigma}_{\min} \text{ at depth } \log_2 \log_2 n_i \text{ or less}\big\},$$
$$\hat{\sigma}_i = \bigcup_j \hat{\sigma}_{i,j}$$

Claim 9 below is analogous to Claim 5 in the proof of Lemma 4. It also has a similar proof.

▶ **Claim 9.** $|\hat{\sigma}_{i,j}| = O(\log^9 n_i + |\sigma_{i,j}|/\log n_i)$.

By Claim 9,

$$|\hat{\sigma}_i| = \sum_j |\hat{\sigma}_{i,j}| = O\Big(\sum_j \log^9 n_i + |\sigma_i|/\log n_i\Big). \tag{6}$$

If $|\sigma_i \backslash \hat{\sigma}_i| \geq |\hat{\sigma}_i|$, then

$$\begin{aligned} |\sigma_i| \log\log n_i &= |\sigma_i \backslash \hat{\sigma}_i| \log\log n_i + |\hat{\sigma}_i| \log\log n_i = O(|\sigma_i \backslash \hat{\sigma}_i| \log\log n_i) \\ &= O(D^{\sigma}_{\min}(\sigma_i)). \end{aligned}$$

In the last step above, we use the fact that $D^{\sigma}_{\min}(\sigma_i) = \Omega(|\sigma_i \backslash \hat{\sigma}_i| \log\log n_i)$, which is true because each query point in $\sigma_i \backslash \hat{\sigma}_i$ lies in a triangle (leaf node) in $D^{\sigma}_{\min}$ at depth greater than $\log_2 \log_2 n_i$. If $|\sigma_i \backslash \hat{\sigma}_i| < |\hat{\sigma}_i|$, then

$$\begin{aligned} |\sigma_i| \log\log n_i &= |\sigma_i \backslash \hat{\sigma}_i| \log\log n_i + |\hat{\sigma}_i| \log\log n_i = O(|\hat{\sigma}_i| \log\log n_i) \\ &= O\Big(\sum_j \log^9 n_i \log\log n_i + |\sigma_i|\Big). && (\because (6)) \\ &= O\Big(\sum_j n_i + |\sigma_i|\Big). \end{aligned}$$

Combining the two cases above and the fact that $D^{\sigma}_{\min}(\sigma_i) = \Omega(|\sigma_i|)$, we obtain $|\sigma_i| \log\log n_i = O\big(D^{\sigma}_{\min}(\sigma_i) + \sum_j n_i\big)$. Substituting this equation into (5) gives

$$\sum_{i=0}^{m} \Gamma_i = O\Big(D^{\sigma}_{\min}(\sigma) + n + |\sigma| + \sum_{i=0}^{m} D^{\sigma}_{\min}(\sigma_i) + \sum_{i=0}^{m} \sum_j n_i\Big).$$

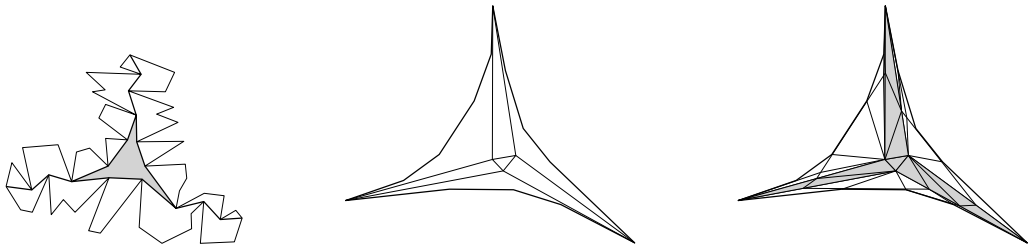
We have shown previously that $\sum_{i=1}^{m} \sum_j n_i \log n_i = O(|\sigma|)$. Note that $\sum_j n_0 = n_0 = n$ as there are only one triangulation and one structure at level 0. Also, $\sum_{i=0}^{m} D^{\sigma}_{\min}(\sigma_i) = D^{\sigma}_{\min}(\sigma)$ and $D^{\sigma}_{\min}(\sigma) = \Omega(|\sigma|)$. Therefore, by Lemma 3(ii), $\sum_{i=0}^{m} \Gamma_i = O\big(D^{\sigma}_{\min}(\sigma)+n\big) = O(\mathrm{OPT}+n)$.

To bound the size of our data structure, observe that $\Delta_{i,j}$ and $D_{i,j}$ have $O(n_i)$ size and $(\Delta_{i,j}, D_{i,j})$ replace $(\Delta_{i,j-1}, D_{i,j-1})$. Therefore, the total size is $O(\sum_{i=0}^{m} n_i)$. Since $n_i = O(f(n_{i-1})/\log n_{i-1}) = O(\log^5 n_{i-1})$, it is clear that $n_i = O(n/2^{i-1})$ by an inductive argument. As a result, the total size is $O(\sum_{i=0}^{m} n_i) = O(\sum_{i=0}^{m} n/2^{i-1}) = O(n)$. ◀

4 Planar connected subdivision

We describe a point location structure for a connected subdivision S with n vertices. It uses $O(n)$ space and processes any online query sequence σ in $O(|\sigma| \log\log n + n + \mathrm{OPT})$ time.

We need a *balanced geodesic triangulation* of a simple polygon P [7]. Let k be the number of vertices of P. Pick three vertices $v_1, v_{k/3}, v_{2k/3}$ of P (which divide the boundary of P into chains of roughly equal sizes). The three geodesic paths among $v_1, v_{k/3}, v_{2k/3}$ bound the



(a) (b) (c)

Figure 5 (a) Kite and the geodesic triangle inside (shown shaded). (b) Divide into triangles and boomerangs. (c) Triangulation.

so-called *kite*. Refer to Figure 5(a) for an example. The part of the kite with an non-empty interior is a *geodesic triangle* τ, whose boundary consists of three reflex chains. Next, compute the geodesic paths from v_1 and $v_{k/3}$ to $v_{k/6}$, the middle vertex in the chain between v_1 and $v_{k/3}$. This creates another kite joining v_1, $v_{k/6}$ and $v_{k/3}$ and hence another geodesic triangle τ' inside this kite. The same process is repeated to other parts of P recursively. In the end, we obtain a balanced geodesic triangulation, which can be computed in $O(|P|)$ time [7].

We simulate the decomposition as sketched in Section 2 using balanced geodesic triangulations. Let $\text{conv}(S)$ denote the convex hull of S. We divide the exterior face of $\text{conv}(S)$ into triangles as described in Section 2. Each region r inside $\text{conv}(S)$ is a simple polygon. We first compute a balanced geodesic triangulation $\tilde{\Delta}_r$ of r. We triangulate each geodesic triangle τ in $\tilde{\Delta}_r$ as follows. We shoot two rays inward from each vertex of τ. They intercept each other and form four triangles. Figure 5(b) shows an example. These four triangles are surrounded by three boomerangs. The boomerangs are triangulated as described in Section 3.1, and this process places $O(\log n)$ vertices on the boundaries of three of the triangles in the middle. We connect these vertices to triangulate these three triangles. Figure 5(c) shows an example. This complete the triangulation of τ. The triangulations of all geodesic triangles in $\tilde{\Delta}_r$ form the triangulation Δ_r. Any triangle that lies in r intersects $O(\log^2 n)$ triangles in τ, implying that any triangle that lies in r intersects $O(\log^3 n)$ triangles in Δ_r. The collection of all triangles obtained above form the triangulation Δ_S. It takes $O(n)$ time to compute Δ_S. We apply Theorem 1 to Δ_S to obtain a point location structure D_S.

Define $D^\sigma_{\min}$ for S as in Section 3.1.3. A leaf of $D^\sigma_{\min}$ corresponds to a triangle t that lies in a region r of S, so t intersects $O(\log^3 n)$ in Δ_S. It means that we can expand the leaf nodes of $D^\sigma_{\min}$ into linear decision subtrees of height $O(\log\log n)$ so that the expanded linear decision tree D' takes $D^\sigma_{\min}(\sigma) + O(|\sigma|\log\log n)$ time to locate the query points in σ in Δ_S. The total processing time by D_S (including the preprocessing time) is $D_S(\sigma) + O(n)$, which by Theorem 1 is $O(D'(\sigma) + n) = O(D^\sigma_{\min}(\sigma) + n + |\sigma|\log\log n) = O(\text{OPT} + n + |\sigma|\log\log n)$.

▶ **Theorem 10.** *Let S be a planar connected subdivision with n vertices. There is a point-line comparison based data structure that uses $O(n)$ space and processes any online sequence σ of point location queries in S in $O(|\sigma|\log\log n + n + \text{OPT})$ time, where* OPT *is the minimum time needed by any point location linear decision tree for S to process σ. The time bound includes the $O(n)$ preprocessing time.*

5 Conclusion

The performance of our data structure is asymptotically optimal when compared with static point location linear decision trees. It is an open problem to obtain optimal performance when

compared with linear decision trees that may reorganize themselves. This open problem may be difficult as it is related to the dynamic optimality conjecture by Sleator and Tarjan [17], which conjectures that the performance of a splay tree is no more than $O(n)$ plus a constant times the time required by any binary search tree algorithm. The dynamic optimality conjecture is still open after over thirty years.

References

1 P. Afshani, J. Barbay, and T. Chan. Instance optimal geometric algorithms. *Proc. 50th Annu. IEEE Symp. Found. Computer Sci.*, 2009, 129–138.

2 U. Adamy and R. Seidel. On the exact worst case query complexity of planar point location. *Proc. 9th Annu. ACM-SIAM Symp. Discrete Alg.*, 1998, 609–618.

3 S. Arya, T. Malamatos, and D. M. Mount. A simple entropy-based algorithm for planar point location. *ACM Trans. Alg.*, vol. 3, no. 2, 2007, article 17.

4 S. Arya, T. Malamatos, D. Mount, and K. Wong. Optimal expected-case planar point location. *SIAM J. Comput.*, 37 (2007), 584–610.

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

6 B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6 (1991), 485–524.

7 B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12 (1994), 54–68.

8 S.-W. Cheng and M.-K. Lau. Adaptive Point Location in Planar Convex Subdivisions. Preliminary version appeared in *Proc. 26th Int'l Symp. Algorithms and Computation*, 2015, 14-22. The full version is to appear in *Int'l J. Comput. Geom. Theory and Appl.*

9 S. Collette, V. Dujmović, J. Iacono, S. Langerman, and P. Morin. Entropy, triangulation, and point location in planar subdivisions. *ACM Trans. Alg.*, vol. 8, no. 3, 2012, article 29.

10 D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra – a unified approach, *Proc. 17th Int'l Colloq. Automata, Languages and Programming*, 1990, 400–413.

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

12 J. Iacono. Expected asymptotically optimal planar point location. *Comput. Geom.: Theory and Appl.*, 29 (2004), 19–22.

13 J. Iacono and W. Mulzer. A static optimality transformation with applications to planar point location. *Int'l J. Comput. Geom. Appl.*, 22 (2012), 3270–340.

14 D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12 (1983), 28–35.

15 N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Comm. ACM*, 29 (1986), 669–679.

16 C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 1948.

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