

Incremental Voronoi diagrams

Sarah R. Allen^{*1}, Luis Barba², John Iacono³, and
Stefan Langerman^{†4}

- 1 Computer Science Department, Carnegie Mellon University, Pittsburgh, USA
srallen@cs.cmu.edu
- 2 Département d’Informatique, Université Libre de Bruxelles, Brussels, Belgium;
and
School of Computer Science, Carleton University, Ottawa, Canada
lbarbaf1@ulb.ac.be
- 3 Tandon School of Engineering, New York University, New York, USA
iacono@nyu.edu
- 4 Département d’Informatique, Université Libre de Bruxelles, Brussels, Belgium
stefan.langerman@ulb.ac.be

Abstract

We study the amortized number of combinatorial changes (edge insertions and removals) needed to update the graph structure of the Voronoi diagram $\text{VD}(S)$ (and several variants thereof) of a set S of n sites in the plane as sites are added to the set. To that effect, we define a general update operation for planar graphs that can be used to model the incremental construction of several variants of Voronoi diagrams as well as the incremental construction of an intersection of halfspaces in \mathbb{R}^3 . We show that the amortized number of edge insertions and removals needed to add a new site to the Voronoi diagram is $O(\sqrt{n})$. A matching $\Omega(\sqrt{n})$ combinatorial lower bound is shown, even in the case where the graph representing the Voronoi diagram is a tree. This contrasts with the $O(\log n)$ upper bound of Aronov et al. (2006) for farthest-point Voronoi diagrams in the special case where points are inserted in clockwise order along their convex hull.

We then present a semi-dynamic data structure that maintains the Voronoi diagram of a set S of n sites in convex position. This data structure supports the insertion of a new site p (and hence the addition of its Voronoi cell) and finds the asymptotically minimal number K of edge insertions and removals needed to obtain the diagram of $S \cup \{p\}$ from the diagram of S , in time $O(K \text{ polylog } n)$ worst case, which is $O(\sqrt{n} \text{ polylog } n)$ amortized by the aforementioned combinatorial result.

The most distinctive feature of this data structure is that the graph of the Voronoi diagram is maintained explicitly at all times and can be retrieved and traversed in the natural way; this contrasts with other known data structures supporting nearest neighbor queries. Our data structure supports general search operations on the current Voronoi diagram, which can, for example, be used to perform point location queries in the cells of the current Voronoi diagram in $O(\log n)$ time, or to determine whether two given sites are neighbors in the Delaunay triangulation.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Voronoi diagrams, dynamic data structures, Delaunay triangulation

Digital Object Identifier 10.4230/LIPIcs.SoCG.2016.15

* Supported by NSF grants CCF-0747250, CCF-1116594, and the Graduate Research Fellowship Program under Grant No. DGE-1252522.

† Directeur de Recherches du F.R.S.-FNRS.



© Sarah R. Allen, Luis Barba, John Iacono, and Stefan Langerman;
licensed under Creative Commons License CC-BY

32nd International Symposium on Computational Geometry (SoCG 2016).

Editors: Sándor Fekete and Anna Lubiw; Article No. 15; pp. 15:1–15:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Let S be a set of n sites in the plane. The graph structures of the Voronoi diagram $\text{VD}(S)$ and its dual the Delaunay triangulation $\text{DT}(S)$ capture much of the proximity information of that set. They contain the nearest neighbor graph, the minimum spanning tree, and the Gabriel graph of S , and have countless applications in computational geometry, shape reconstruction, computational biology, and machine learning.

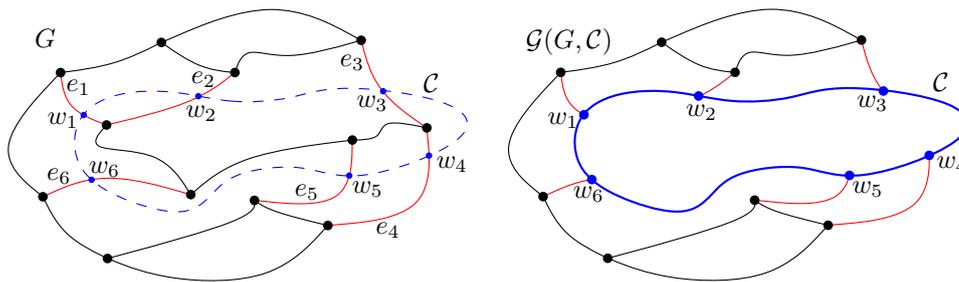
One of the most popular algorithms for constructing a Voronoi diagram inserts sites in random order, incrementally updating the diagram [8]. In that case, backward analysis shows that the expected number of changed edges in $\text{VD}(S)$ is constant, offering some hope that an efficient dynamic (or at least semi-dynamic) data structure for maintaining $\text{VD}(S)$ could exist. These hopes, however, are rapidly squashed, as it is easy to construct examples where the complexity of each successively added face is $\Omega(n)$, and thus each insertion changes the position of $\Omega(n)$ vertices and edges of $\text{VD}(S)$. The goal of this paper is to show that despite this worst-case behavior, the amortized number of *structural changes* to the graph of the Voronoi diagram of S , i.e., the minimum number of edge insertions and deletions needed to update $\text{VD}(S)$ throughout a sequence of site insertions to S , is much smaller.

This might come as a surprise in light of the fact that the number of combinatorial changes (usually modeled as *flips*) to the Delaunay triangulation of S upon the insertion of a point can be $\Omega(n)$ with each insertion, even when the sites are in convex position and are added in clockwise order. (Note that in that case the Voronoi diagram of S is a tree and the standard flip operation is a rotation in the tree.)

To overcome this worst-case behavior, Aronov et al. [2] studied what happens in this specific case (points in convex position added in clockwise order) if the rotation operation is replaced by the more elementary link (add an edge) and cut (delete an edge) operations in the tree. They show that, in that model, it is possible to reconfigure the tree after each site insertion while performing $O(\log n)$ links and cuts, amortized; however their proof is existential and no algorithm is provided to find those links and cuts. Pettie [13] shows both an alternate proof of that fact using forbidden 0-1 matrices and a matching lower bound.

One important application of Voronoi diagrams is to solve *nearest-neighbor* (or *farthest-neighbor*) queries: given a point in the plane, find the site nearest (or farthest) to this point. In the static case, this is done by preprocessing the (nearest or farthest point) Voronoi diagram to answer point-location queries in $O(\log n)$ time. Without the need to maintain $\text{VD}(S)$ explicitly, the problem of nearest neighbor queries is a *decomposable search problem* and can be made semi-dynamic using the standard dynamization techniques of Bentley and Saxe [4]. The best incremental data structure supporting nearest-neighbor queries performs queries and insertions in $O(\log^2 n / \log \log n)$ time [7, 12]. Recently, Chan [6] developed a randomized data structure supporting nearest-neighbor queries in $O(\log^2 n)$ time, insertions in $O(\log^3 n)$ expected amortized time, and deletions in $O(\log^6 n)$ expected amortized time.

Flarbs. In the mid-1980's it was observed that a number of variants of Voronoi diagrams and Delaunay triangulations using different metrics (Euclidean distance, L_p norms, convex distance functions) or different kinds of sites (points, segments, circles) could all be handled using similar techniques. To formalize this, several abstract frameworks were defined, such as the one of Edelsbrunner and Seidel [9] and the two variants of abstract Voronoi diagrams of Klein [11, 10]. In this paper we define a new abstract framework to deal with Voronoi diagrams constructed incrementally by inserting new sites.



■ **Figure 1** The flarb operation on a graph G induced by a flarbable curve \mathcal{C} , produces a graph $\mathcal{G}(G, \mathcal{C})$ with 2 more vertices. Fleeq-edges crossed by \mathcal{C} are shown in red.

Let G be a 3-regular embedded plane graph with n vertices¹. We seek to bound the number of edge removals and insertions needed to implement the following operation, hereafter referred to as a *flarb*²: Given a simple closed curve \mathcal{C} in the plane whose interior intersects G in a connected component, split both \mathcal{C} and all the edges that it crosses at the point of intersection, remove every edge and vertex that lies in the interior of \mathcal{C} , and add each curve in the subdivision of \mathcal{C} as a new edge; see Figure 1. This operation can be used to represent the insertion of new cells in different types of Voronoi diagrams. It can also be used to represent the changes to the 1-skeleton of a polyhedron in \mathbb{R}^3 after it is intersected with a halfspace.

Results. We show that the amortized *cost* of a flarb operation, where the combinatorial cost is defined to be the minimum number of edge insertions and removals needed to perform it, is $O(\sqrt{n})$. We also show a matching lower bound: some sequences of flarbs require $\Omega(\sqrt{n})$ links and cuts per flarb, even when the graph is a tree (or more precisely a Halin graph – a tree with all leaves connected by a cycle to make it 3-regular). This contrasts with the $O(\log n)$ upper bound of Aronov et al. [2] for the Voronoi diagram of points in convex position (also a tree) when points are added in clockwise order.

We complement these combinatorial bounds with an algorithmic result. We present an output-sensitive data structure that maintains the nearest- or farthest-point Voronoi diagram of a set S of n points in convex position as new points are added to S . Upon an insertion, the data structure finds the minimum number K (up to within a constant factor) of edge insertions and deletions necessary to update the Voronoi diagram of S . The running time of each insertion is $O(K \log^7 n)$, and by our combinatorial bounds, $K = O(\sqrt{n})$. This solves the open problem posed by Aronov et al. [2].

The distinguishing feature of this data structure is that it explicitly maintains the graph structure of the Voronoi diagram after every insertion, a property that is not provided by any nearest neighbor data structure that uses decomposable searching problem techniques. Further, the data structure also maintains the Voronoi diagram in a *grappa tree* [2], a variant of the link-cut trees of Sleator and Tarjan [14] that allows a powerful query operation called *oracle-search*. Roughly speaking, the oracle-search query has access to an oracle specifying a vertex to find. Given an edge of the tree, the oracle determines which of the two subtrees attached to its endpoints contains that vertex. Grappa trees use $O(\log n)$ time and oracle

¹ While the introduction used n for the number of sites in S , the combinatorial part of this article uses n for the number of vertices in the Voronoi diagram. By Euler's formula, those two values are asymptotically equivalent, up to a constant factor.

² Although the last two authors are honored by the flattering renaming of the flarb operation in the literature [13], this paper uses original terminology.

calls to find the sought vertex. A grappa tree is in some sense a dynamic version of the centroid decomposition for trees, which is used in many algorithms for searching in Voronoi diagrams. Using this structure, it is possible to solve a number of problems for the set S at any moment during the incremental construction, for example:

- Report whether two sites p and q are connected by a Delaunay edge in $O(\log n)$ time.
- Given a point q , find the Voronoi cell containing q in $O(\log n)$ time. This not only gives the nearest neighbor of q , but a pointer to the explicit description of its cell.
- Find the smallest disk enclosing S centered on a query segment $[pq]$, in $O(\log n)$ time [5].
- Find the smallest disk enclosing S , centered on a query circle C , in $O(\log n)$ time [3].
- Given a convex polygon P (counterclockwise array of its m vertices), find the smallest disk enclosing S and excluding P in $O(\log n + \log m)$ time [1].

The combinatorial bound for Voronoi diagrams also has direct algorithmic consequences, the most important being that it is possible to store all versions of this graph throughout a sequence of insertions using persistence in $O(n^{3/2})$ space. Since the entire structure of the graph is stored for each version, this provides a foundation for many applications that, for instance would require searching the sequence of insertions for the moment during which a specific event occurred.

Outline. The main approach used to bound the combinatorial cost of a flarb is to examine how the complexity of the faces changes. Notice that faces whose size remains the same do not require edge insertions and deletions. The other faces either grow or shrink, and a careful counting argument reveals that the cost of a flarb is at most the number faces that shrink (or disappear) upon execution of the flarb (Section 2). By using a potential function that sums the sizes of all faces, the combinatorial cost of shrinking faces is paid for by the reduction of their potential. To avoid incurring a high increase in potential for a large new face, the potential of each face is capped at \sqrt{n} . Then at most $O(\sqrt{n})$ large faces can shrink without changing potential and are accounted for separately (Section 3). The matching $\Omega(\sqrt{n})$ lower bound is presented in Section 4, and Section 5 presents the data structure for performing flarbs for the Voronoi diagrams of points in convex position.

2 The flarb operation

In this section we formalize the flarb operation that models the insertion of new sites in Voronoi diagrams, and present a preliminary analysis of the cost of a flarb.

Let $G = (V, E)$ be a planar 3-regular graph embedded in \mathbb{R}^2 (not-necessarily with a straight-line embedding). Let \mathcal{C} be a simple closed Jordan curve in the plane. Define $\text{IN}(\mathcal{C})$ to be the set of vertices of G that lie in the interior of \mathcal{C} and let $\text{EX}(\mathcal{C}) = V \setminus \text{IN}(\mathcal{C})$. We say that \mathcal{C} is *flarbable* for G if the following conditions hold: (1) the graph induced by $\text{IN}(\mathcal{C})$ is connected, (2) \mathcal{C} intersects each edge of G either at a single point or not at all, (3) \mathcal{C} passes through no vertex of G , and (4) the intersection of \mathcal{C} with each face of G is path-connected.

In the case where the graph G is clear from context, we simply say that \mathcal{C} is flarbable. The *fleeq* of \mathcal{C} is the circular sequence $\mathcal{E}_{\mathcal{C}} = e_1, \dots, e_k$ of edges in E that are crossed by \mathcal{C} ; we call the edges in $\mathcal{E}_{\mathcal{C}}$ *fleeq-edges*. A face whose interior is crossed by \mathcal{C} is called a \mathcal{C} -*face*. We assume without loss of generality that \mathcal{C} is oriented clockwise and that the edges in $\mathcal{E}_{\mathcal{C}}$ are ordered according to their intersection with \mathcal{C} . Given a flarbable curve \mathcal{C} on G , we present the following definition.

► **Definition 1.** For a planar graph G and a curve \mathcal{C} that is flarbable for G , we define a *flarb* operation $\mathcal{F}(G, \mathcal{E}_{\mathcal{C}})$ which produces a new 3-connected graph $\mathcal{G}(G, \mathcal{C})$ as follows (see Figure 1 for a depiction): (1) For each edge $e_i = (u_i, v_i)$ in $\mathcal{E}_{\mathcal{C}}$ such that $u_i \in \text{IN}(\mathcal{C})$ and $v_i \in \text{EX}(\mathcal{C})$, create a new vertex $w_i = \mathcal{C} \cap e_i$ and connect it to v_i along e_i . (2) For each pair e_i, e_{i+1} of successive edges in $\mathcal{E}_{\mathcal{C}}$, create a new edge (w_i, w_{i+1}) between them along \mathcal{C} . We call (w_i, w_{i+1}) a \mathcal{C} -edge (all indices are taken modulo k). (3) Delete all vertices of $\text{IN}(\mathcal{C})$ along with their incident edges.

Note that at most two new vertices are created. Since each newly created vertex has degree three and all remaining vertices are unaffected, the new graph is 3-regular. In other words, the flarb operation $\mathcal{F}(G, \mathcal{E}_{\mathcal{C}})$ creates a cycle along \mathcal{C} and removes the portion of the graph enclosed by \mathcal{C} . Note that for any point set in general position (no four points lie on the same circle), its Voronoi diagram is a 3-regular planar graph, assuming we use the line at infinity to join the endpoints of its unbounded edges in clockwise order. Therefore, a flarb can be used to represent the changes to the Voronoi diagram upon insertion of a new site.

► **Observation 2.** Given a set S of points in general position, let $\mathcal{V}(S)$ be the graph of the Voronoi diagram of S . For a new point q , there exists some curve \mathcal{C}_S^q such that $\mathcal{G}(\mathcal{V}(S), \mathcal{C}_S^q) = \mathcal{V}(S \cup \{q\})$; namely, \mathcal{C}_S^q is the boundary of the Voronoi cell of q in $\mathcal{V}(S \cup \{q\})$.

More generally, convex polytopes defined by the intersection of halfspaces in \mathbb{R}^3 behave similarly: the intersection of a new halfspace with a convex polytope modifies the structure of its 1-skeleton by adding a new face. This structural change can be obtained performing a flarb operation in which the flarbable curve consists of the boundary of the new face.

► **Definition 3.** Given a \mathcal{C} -face f of G , the *modified face* of f is the face f' of $\mathcal{G}(G, \mathcal{C})$ that coincides with f outside of \mathcal{C} . In other words, f' is the face that remains from f after performing the flarb $\mathcal{F}(G, \mathcal{E}_{\mathcal{C}})$. We say that a \mathcal{C} -face f is *preserved* (by the flarb $\mathcal{F}(G, \mathcal{E}_{\mathcal{C}})$) if $|f| = |f'|$. Moreover, we say that each edge in a preserved face is *preserved* (by $\mathcal{F}(G, \mathcal{E}_{\mathcal{C}})$). Denote by $\mathcal{P}(G, \mathcal{C})$ the set of faces preserved by $\mathcal{F}(G, \mathcal{E}_{\mathcal{C}})$ and let $\mathcal{B}(G, \mathcal{C})$ be the set of faces wholly contained in the interior of \mathcal{C} .

Since a preserved \mathcal{C} -face bounded by two fleeq-edges e_i and e_{i+1} has the same size before and after the flarb, there must be an edge e of G connecting e_i with e_{i+1} which is replaced by a \mathcal{C} -edge e^* after the flarb. In this case, we say that the edge e *reappears* as e^* .

The following auxiliary lemma will help us bound the number of operations needed to produce the graph $\mathcal{G}(G, \mathcal{C})$, and follows directly from the Euler characteristic of connected planar graphs:

► **Lemma 4.** Let H be a connected planar graph with vertices of degree either 1, 2 or 3. For each $i \in \{1, 2, 3\}$, let δ_i be the number of vertices of H with degree i . Then, H has exactly $2\delta_1 + \delta_2 + 3F_H - 3$ edges, where F_H is the number of bounded faces of H .

Given a 3-regular graph $G = (V, E)$ and a flarbable curve \mathcal{C} we want to analyze the number of structural changes that G needs to undergo to perform $\mathcal{F}(G, \mathcal{E}_{\mathcal{C}})$. To this end, we define the combinatorial *cost* of $\mathcal{F}(G, \mathcal{E}_{\mathcal{C}})$, denoted by $\text{COST}(G, \mathcal{C})$, to be the minimum number of links and cuts needed to transform G into $\mathcal{G}(G, \mathcal{C})$ (note that an algorithm may not implement the flarb operation exactly as described in Definition 1). We assume that any other operation has no cost and is hence not included in the cost of the flarb.

Consider the fleeq $\mathcal{E}_{\mathcal{C}} = e_1, \dots, e_k$ and the \mathcal{C} -edges created by $\mathcal{F}(G, \mathcal{E}_{\mathcal{C}})$. Let e be an edge adjacent to some e_i and e_{i+1} that reappears as the \mathcal{C} -edge e^* . Notice that we can obtain e^* without any links or cuts to G : simply shrink e_i and e_{i+1} so that their endpoints in $\text{IN}(\mathcal{C})$

now coincide with their intersections with \mathcal{C} . Then modify e to coincide with the portion of \mathcal{C} connecting the new endpoints of e_i and e_{i+1} . Using this *preserving operation*, we obtain the \mathcal{C} -edge e^* with no cost to the flarb. Intuitively, preserved edges are cost-free in a flarb while non-preserved edges have a nonzero cost. This notion is formalized as follows.

► **Lemma 5.** *For a flarbable curve \mathcal{C} , $(|\mathcal{E}_{\mathcal{C}}| + |\mathcal{B}(G, \mathcal{C})| - |\mathcal{P}(G, \mathcal{C})|)/2 \leq \text{COST}(G, \mathcal{C})$ and $\text{COST}(G, \mathcal{C}) \leq 4|\mathcal{E}_{\mathcal{C}}| + 3|\mathcal{B}(G, \mathcal{C})| - 4|\mathcal{P}(G, \mathcal{C})|$.*

3 The combinatorial upper bound

In this section, we define a potential function to bound the amortized cost of each operation in a sequence of flarb operations. For a 3-regular embedded planar graph $G = (V, E)$, we define two potential functions: a local potential function μ to measure the potential of each face, and a global potential function Φ to measure the potential of the whole graph.

► **Definition 6.** Let F be the set of faces of a 3-regular embedded planar graph $G = (V, E)$. For each face $f \in F$, let $\mu(f) = \min\{\lceil \sqrt{|V|} \rceil, |f|\}$, where $|f|$ is the number of edges on the boundary of f . The potential $\Phi(G)$ of G is defined as follows: $\Phi(G) = \lambda \sum_{f \in F} \mu(f)$, for some sufficiently large positive constant λ to be defined later.

Recall that the potential $\mu(f)$ of a \mathcal{C} -face f remains unchanged as long as $|f|, |f'| \geq \sqrt{|V|}$, where f' is the modified face of f after the flarb. Since there is no change in potential that we can use within large \mathcal{C} -faces, we exclude them from our analysis and focus only on smaller \mathcal{C} -faces. However, by excluding these large faces, we effectively split \mathcal{C} into smaller curves that cross only \mathcal{C} -faces with less than $\sqrt{|V|}$ edges. We formalize this notion in the following section.

3.1 Flarbable sub-curves

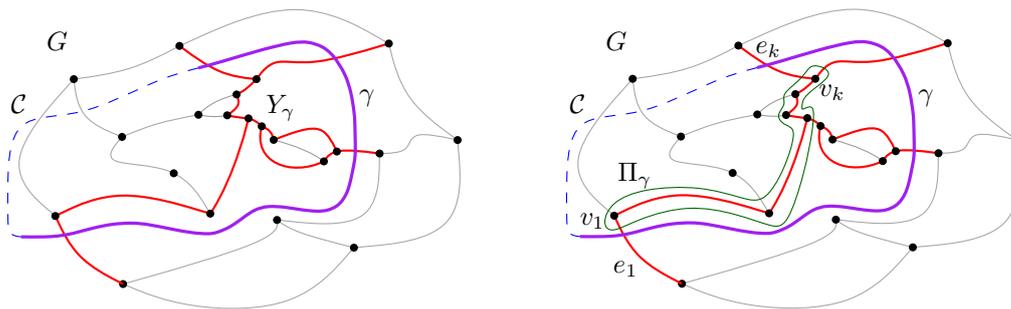
Given a flarbable curve \mathcal{C} , a (connected) curve $\gamma \subseteq \mathcal{C}$ is a *flarbable sub-curve*. Let $\epsilon_\gamma = e_1, \dots, e_k$ (or simply ϵ) be the set of fleeq-edges intersected by γ given in order of intersection after orienting γ arbitrarily. We call ϵ the *subfleeq* induced by γ . We say that a face is a γ -face if two of its fleeq-edges are crossed by γ (if γ has an endpoint in the interior of this face, it is not a γ -face).

Consider the set of all edges of G intersected or enclosed by \mathcal{C} that bound some γ -face. Since $\mathcal{E}_{\mathcal{C}}$ is flarbable, these edges induce a connected subgraph Y_γ of G with $|\epsilon| = k$ leaves (vertices of degree 1), namely the endpoints outside of \mathcal{C} of each fleeq-edge in ϵ ; see Figure 2. Notice that Y_γ may consist of some bounded faces contained in the interior of \mathcal{C} . Let H_γ be the set of bounded faces of Y_γ and let δ_2 be the number of vertices of degree 2 of Y_γ . Since Y_γ consists of k vertices of degree 1, Lemma 4 implies the following result.

► **Corollary 7.** *The graph Y_γ consists of exactly $2k + \delta_2 + 3|H_\gamma| - 3$ edges.*

Recall that a \mathcal{C} -face f is preserved if its corresponding modified face f' in $\mathcal{G}(G, \mathcal{C})$ has the same number of edges, i.e., $|f'| = |f|$. We say that f is *augmented* if $|f'| = |f| + 1$ and we call f *shrinking* if $|f'| < |f|$. Notice that these are all the possible cases as f gains at most one new edge during the flarb, namely the \mathcal{C} -edge crossing this face.

In the context of a particular flarbable sub-curve γ , let a_γ, s_γ and p_γ be the number of augmented, shrinking and preserved γ -faces, respectively (or simply a, s and p if γ is clear from the context). We further differentiate among the s shrinking γ -faces. A shrinking γ -face is *interior* if it contains no vertex of degree 2 of Y_γ and does not share an edge with



■ **Figure 2** Left: A flarbable sub-curves γ is contained in a flarbable curve \mathcal{C} . The graph Y_γ is the union of all edges bounding a γ -face. Right: The path Π_γ connects the endpoints of the first and last fleeq-edges crossed by γ by going along the boundary of the outer-face of Y_γ .

an augmenting face. Let s_a be the number of shrinking γ -faces that share an edge with an augmented face, let s_b be the number of shrinking γ -faces not adjacent to an augmented face that have a vertex of degree 2 of Y_γ , and let s_c be the number of interior shrinking γ -faces. Therefore, $s = s_a + s_b + s_c$ is the total number of shrinking γ -faces.

Since each augmented face has at most two edges and because there are a augmented faces, we know that $s_a \leq 2a$. Let v_1 and v_k be the endpoints of the edges e_1 and e_k that lie inside \mathcal{C} . Let Π_γ be the unique path connecting v_1 and v_k in Y_γ that traverses along the boundary of the outer face of Y_γ and stays in the interior of \mathcal{C} ; see Figure 2.

Notice that Π_γ contains all the edges of γ -faces that may bound a \mathcal{C} -face that is not crossed by γ . In the end, we aim to have bounds on the number of edges that will be removed from the γ -faces during the flarb, but some of these edges may be double counted if they are shared with other \mathcal{C} -faces. Therefore, we aim to bound the length of Π_γ and count precisely these possible double-counted edges.

► **Lemma 8.** *The path Π_γ has length at most $k + 3|H_\gamma| + \delta_2 - a - s_c$.*

3.2 How much do faces shrink in a flarb?

In order to analyze the effect of the flarb operations on flarbable sub-curves, we think of each edge as consisting of two *half-edges*, each adjacent to one of the two faces incident to this edge. For a given edge, the algorithm may delete its half-edges during two separate flarbs of different flarbable sub-curves.

We define the operation $\mathcal{F}(G, \gamma)$ to be the operation which executes steps 1 and 2 of the flarb on the flarbable sub-curve γ and then deletes each half-edge with both endpoints in $\text{IN}(\mathcal{C})$ adjacent to a γ -face. Since $\mathcal{F}(G, \gamma)$ removes and adds half-edges, we are interested in bounding the net balance of half-edges throughout the flarb. To do this, we measure the change in size of a face during the flarb.

Recall that a, s and p are the number of augmented, shrinking and preserved γ -faces, respectively. The following result provides a bound on the total “shrinkage” of the faces crossed by a given flarbable sub-curve.

► **Theorem 9.** *Given a flarbable curve \mathcal{C} on G and a flarbable sub-curve γ crossing the fleeq-edges $\epsilon = e_1, \dots, e_k$, let f_1, \dots, f_k be the sequence of γ -faces and let f'_1, \dots, f'_k be their corresponding modified faces after the flarb $\mathcal{F}(G, \gamma)$. Then, $\sum_{i=1}^k (|f_i| - |f'_i|) \geq s/2$.*

Proof. Recall that no successive γ -faces can both be augmented unless $\mathcal{E}_\mathcal{C}$ consists of three edges incident to a single vertex. In this case, at most 3 γ -faces can be augmented, so

$\sum_{i=1}^k (|f_i| - |f'_i|) = 3$ and the result holds trivially; hence, we assume from now on that no two successive faces are both augmented.

Let Δ be the number of half-edges removed during $\mathcal{F}(G, \gamma)$. Notice that to count how much a face f_i shrinks when becoming f'_i after the flarb, we need to count the number of half-edges of f_i that are deleted and the number that are added in f'_i . Since exactly one half-edge is added in each f'_i , we know that $\sum_{i=1}^k (|f_i| - |f'_i|) = \Delta - k$. We claim that $\Delta \geq k + s/2$. If this claim is true, then $\sum_{i=1}^k (|f_i| - |f'_i|) \geq s/2$ as stated in the theorem. In the remainder of this proof, we show this bound on Δ .

Let $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$ be the subgraph of Y_{γ} obtained by removing its k fleeq-edges. Therefore, we know that $|E_{\mathcal{T}}| = k + 3|H_{\gamma}| + \delta_2 - 3$ by Corollary 7. To have a precise counting of Δ , notice that for some edges of \mathcal{T} , $\mathcal{F}(G, \gamma)$ removes only one of their half-edges and for others it will remove both of them. Since the fleeq-edges are present in each of the faces f_1, \dots, f_k before and after the flarb, we get that $\Delta = 2|E_{\mathcal{T}}| - S_{\mathcal{T}}$, where $S_{\mathcal{T}}$ denotes the number of edges in \mathcal{T} with only one half-edge incident to a face of f_1, \dots, f_k .

Note that the edges of $S_{\mathcal{T}}$ are exactly the edges on the path Π_{γ} bounded in Lemma 8. Therefore, $S_{\mathcal{T}} \leq k + 3|H_{\gamma}| + \delta_2 - a - s_c$. By using this bound, we get $\Delta \geq 2(k + 3|H_{\gamma}| + \delta_2 - 3) - (k + 3|H_{\gamma}| + \delta_2 - a - s_c) = k + 3|H_{\gamma}| + \delta_2 + a + s_c - 6$. Since each shrinking γ -face accounted for by s_b has a vertex of degree 2 in Y_{γ} , we know that $\delta_2 \geq s_b$. Moreover, $s_a \leq 2a$ as each shrinking γ -face can be adjacent to at most two augmenting γ -faces. Thus, since $s = s_a + s_b + s_c$, we get that $\Delta \geq k + 3|H_{\gamma}| + s_a/2 + s_b + s_c \geq k + s/2$, where s is the number of shrinking γ -faces proving the claimed bound on Δ . \blacktriangleleft

3.3 Flarbable sequences

Let $\mathcal{G}^0 = G$. A sequence of curves $\mathcal{C} = \mathcal{C}_1, \dots, \mathcal{C}_k$ is *flarbable* if for each $i \in [k]$, \mathcal{C}_i is a flarbable on $\mathcal{G}^i = \mathcal{G}(\mathcal{G}^{i-1}, \mathcal{C}_i)$. As a notational shorthand, let \mathcal{F}^i denote the flarb operation $\mathcal{F}(\mathcal{G}^{i-1}, \mathcal{C}_i)$ when \mathcal{C} is a flarbable sequence for G .

► **Theorem 10.** *For a 3-regular planar graph $G = (V, E)$ and some flarbable sequence $\mathcal{C} = \mathcal{C}_1, \dots, \mathcal{C}_N$ of flarbable fleeqs, for all $i \in [N]$, $\text{COST}(\mathcal{G}^{i-1}, \mathcal{C}_i) + \Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) \leq O(\sqrt{|V_i|})$, where V_i is the set of vertices of \mathcal{G}^i .*

Proof. Partition \mathcal{C}_i into smaller curves $\gamma_1, \dots, \gamma_h$ such that for all $j \in [h]$, γ_j is a maximal curve contained in \mathcal{C}_i that does not intersect the interior of a face with more than $\sqrt{|V_i|}$ edges. Since there can be at most $\sqrt{|V_i|}$ faces of size $\sqrt{|V_i|}$, we know that $h \leq \sqrt{|V_i|}$. Let ϵ_j be the subfleeq containing each fleeq-edge crossed by γ_j . Let a_j, s_j and p_j be the number of augmented, shrinking and preserved γ_j -faces, respectively. Notice that $|\epsilon_j| = a_j + s_j + p_j + 1$. Moreover, since each augmented face is adjacent to a shrinking face, we know that $a_j \leq s_j + 1$. Therefore, $|\epsilon_j| \leq 2s_j + p_j + 2$.

Let \mathcal{L}_i be the set of \mathcal{C}_i -faces with at least $\sqrt{|V_i|}$ edges and let ω_i be the set of all faces of \mathcal{G}^{i-1} completely enclosed in the interior of \mathcal{C}_i .

First, we upper bound $\text{COST}(\mathcal{G}^{i-1}, \mathcal{C}_i)$. By Lemma 5, we know that $\text{COST}(\mathcal{G}^{i-1}, \mathcal{C}_i) \leq 4|\mathcal{E}_{\mathcal{C}_i}| + 3|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)| - 4|\mathcal{P}(\mathcal{G}^{i-1}, \mathcal{C}_i)| = 4\sum_{j=1}^h |\epsilon_j| + 3|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)| - 4|\mathcal{P}(\mathcal{G}^{i-1}, \mathcal{C}_i)| \leq 4\sum_{j=1}^h (2s_j + p_j + 2) + 3|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)| - 4|\mathcal{P}(\mathcal{G}^{i-1}, \mathcal{C}_i)|$. Because each preserved face is crossed by exactly one flarbable sub-curve, $\sum_{j=1}^h p_j = |\mathcal{P}(\mathcal{G}^{i-1}, \mathcal{C}_i)|$. Therefore, $\text{COST}(\mathcal{G}^{i-1}, \mathcal{C}_i) \leq 4\sum_{j=1}^h (2s_j + 2) + 3|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)| = 8h + 8\sum_{j=1}^h s_j + 3|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)|$. Since $h \leq \sqrt{|V_i|}$, we conclude that

$$\text{cost}[\mathcal{G}^{i-1}, \mathcal{C}_i] \leq 8\sqrt{|V_i|} + 8\sum_{j=1}^h s_j + 3|\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)|. \quad (1)$$

Next, we upper bound the change in potential $\Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1})$. Given a flarbable curve or sub-curve γ , let $\mathcal{A}(\gamma)$ denote the set of γ -faces. Recall that for a γ -face $f \in \mathcal{A}(\gamma)$, f' is the modified face of f . Also, let f_n be the new face created by \mathcal{F}^i , i.e., the face of \mathcal{G}^i bounded by \mathcal{C}_i . Recall that for each face $f \in \mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)$, f disappears and there is a drop in potential of $\mu(f) \geq 1$. Using this, we can break up the summation so that

$$\begin{aligned} \Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) &= \mu(f_n) + \lambda \sum_{f \in \mathcal{A}(\mathcal{C}_i)} (\mu(f') - \mu(f)) - \lambda \sum_{f \in \mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)} \mu(f) \\ &\leq \mu(f_n) + \lambda \sum_{f \in \mathcal{A}(\mathcal{C}_i)} (\mu(f') - \mu(f)) - \lambda |\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)|. \end{aligned}$$

We now break up the first summation by independently considering the large faces in \mathcal{L}_i and the remaining smaller faces which are crossed by some flarbable sub-curve. We get that $\Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) \leq \mu(f_n) + \lambda \sum_{j=1}^h \left(\sum_{f \in \mathcal{A}(\gamma_j)} (\mu(f') - \mu(f)) \right) + \lambda \sum_{f \in \mathcal{L}_i} (\mu(f') - \mu(f)) - \lambda |\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)|$.

Since each face can gain at most one edge, in particular we know that $\mu(f') - \mu(f) \leq 1$ for each $f \in \mathcal{L}_i$. Moreover, $\mu(f_n) \leq \sqrt{|V_i|}$ by the definition of μ . Thus, $\Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) \leq \sqrt{|V_i|} + \lambda \sum_{j=1}^h \left(\sum_{f \in \mathcal{A}(\gamma_j)} (\mu(f') - \mu(f)) \right) + \lambda |\mathcal{L}_i| - \lambda |\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)|$.

Note that $\mu(f) = |f|$ for each face $f \in \mathcal{A}(\gamma_j)$, $1 \leq j \leq h$. Thus, applying Theorem 9 to the first summation, we get $\Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) \leq \sqrt{|V_i|} - \frac{\lambda}{2} \sum_{j=1}^h s_j + \lambda |\mathcal{L}_i| - \lambda |\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)|$. Since there can be at most $\sqrt{|V_i|}$ faces of size $\sqrt{|V_i|}$, we know that $|\mathcal{L}_i| \leq \sqrt{|V_i|}$. Therefore, $\Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) \leq (\lambda + 1)\sqrt{|V_i|} - \frac{\lambda}{2} \sum_{j=1}^h s_j - \lambda |\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)|$.

Putting this together with (1), we get that $\text{COST}(\mathcal{G}^{i-1}, \mathcal{C}_i) + \Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) \leq (\lambda + 9)\sqrt{|V_i|} + (8 - \frac{\lambda}{2}) \sum_{j=1}^h s_j + (3 - \lambda) |\mathcal{B}(\mathcal{G}^{i-1}, \mathcal{C}_i)|$. By letting λ be a sufficiently large constant (namely $\lambda = 16$), we get that $\text{COST}(\mathcal{G}^{i-1}, \mathcal{C}_i) + \Phi(\mathcal{G}^i) - \Phi(\mathcal{G}^{i-1}) = O(\sqrt{|V_i|})$. ◀

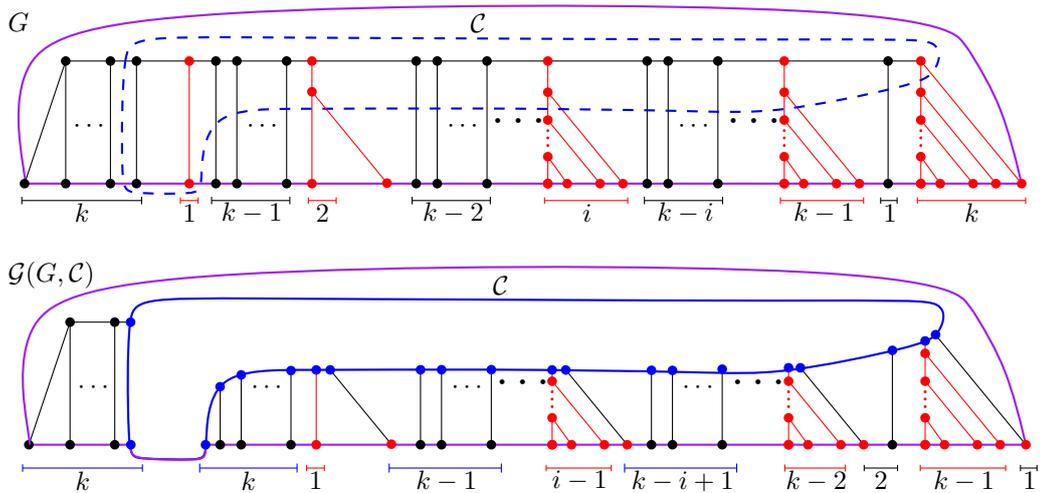
► **Corollary 11.** *Let G be a 3-regular plane graph with ν vertices. For a sequence $\mathcal{C} = \mathcal{C}_1, \dots, \mathcal{C}_N$ of flarbable fleeqs for graph $G = (V, E)$ where $\nu = |V|$, $\sum_{i=1}^N \text{COST}(\mathcal{G}^{i-1}, \mathcal{C}_i) = O(\nu + N\sqrt{\nu + N})$.*

Proof. Using the result of Theorem 10, we can write $\sum_{i=1}^N \text{COST}(\mathcal{G}^{i-1}, \mathcal{C}_i) + \Phi(\mathcal{G}^N) - \Phi(G) = O(N\sqrt{|V_i|})$. Because $\Phi(G) = \lambda \sum_{f \in F} \mu(f)$, we know that $\Phi(G) = O(\nu)$. Analogously, since each flarb operation adds at most 2 vertices, we know that the number of vertices in \mathcal{G}^N is $O(\nu + N)$ which, in turn, implies that $\Phi(\mathcal{G}^N) = O(\nu + N)$. Therefore, $\sum_{i=1}^N \text{COST}(\mathcal{G}^{i-1}, \mathcal{C}_i) = O(N\sqrt{|V_i|} + \Phi(G) - \Phi(\mathcal{G}^N)) = O(\nu + N\sqrt{\nu + N})$. ◀

4 The lower bound

In Section 4, we present an example of a 3-regular Halin graph G with ν vertices – a tree with all leaves connected by a cycle to make it 3-regular – and a corresponding flarb operation with cost $\Omega(\sqrt{\nu})$ that yields a graph isomorphic to G . Because this sequence can be repeated, the amortized cost of a flarb is $\Theta(\sqrt{\nu})$.

Let $\nu = 2k(k + 1) - 2$ for some positive integer k . The construction of the 3-regular graph with ν vertices is depicted in Figure 3. In this graph, we show the existence of a flarbable curve \mathcal{C} (dashed in the figure) such that the flarb operation on G produces a graph $\mathcal{G}(G, \mathcal{C})$ isomorphic to G . Moreover, \mathcal{C} crosses at least k augmented \mathcal{C} -faces and k shrinking \mathcal{C} -faces. Therefore, $\text{COST}(G, \mathcal{C}) \geq k = \Omega(\sqrt{\nu})$ by Lemma 5. Since we end up with a graph that is isomorphic to the original, we can produce a new flarbable curve having



■ **Figure 3** A 3-regular graph G with $\nu = 2k(k + 1) - 2$ vertices. A flarbable curve \mathcal{C} induces a flarb such that $\mathcal{G}(G, \mathcal{C})$ is isomorphic with G .

the same effect. That is, there is a sequence of N flarbable curves $\mathcal{C}_1, \dots, \mathcal{C}_N$ such that $\sum_{i=1}^N \text{COST}(\mathcal{G}^{i-1}, \mathcal{C}_i) = \Omega(N\sqrt{\nu})$,

5 Computing the flarb

In this section, we describe a data structure to maintain the Voronoi diagram of a set S of n sites in convex position as new sites are added to S . Our structure allows us to find the edges of each preserved face and ignore them, thereby focusing only on necessary modifications to the combinatorial structure. The time we spend in these operations is then proportional to the number of non-preserved edges. Since this number is proportional to the cost of the flarb, our data structure supports site insertions in time that is almost optimal (up to a polylogarithmic factor).

Grappa trees. Grappa trees [2] are a modification of link-cut trees, a data structure introduced by Sleator and Tarjan [14] to maintain the combinatorial structure of trees. They support the creation of new isolated vertices, the *link* operation which adds an edge between two vertices in disjoint trees, and the *cut* operation which removes an edge, splitting a tree into two trees.

We use Grappa trees to maintain the combinatorial structure of the incrementally constructed Voronoi diagram $\mathcal{V}(S)$ of a set S of sites in convex position. Recall that each insertion defines a flarbable curve \mathcal{C} , namely the boundary of the Voronoi cell of the inserted site. Our algorithm performs this flarb operation in time $O(\text{COST}(\mathcal{V}(S), \mathcal{C}) \log^7 n)$, where n is the number of vertices inserted so far. That is, we obtain an algorithm whose running time depends on the minimum number of link and cut operations that the Voronoi diagram must undergo after each insertion. Moreover, this Voronoi diagram answers nearest neighbor queries in $O(\log n)$ time among other related queries.

A grappa tree, as introduced by Aronov et al. [2], is a data structure based on the worst-case version of the link-cut tree construction of Sleator and Tarjan [14]. This structure maintains a forest of fixed-topology rooted binary trees subject to many operations, including MAKE-TREE, LINK, CUT and EVERT (which changes the root of a tree) each in $O(\log n)$ worst-case time while using $O(n)$ space.

As in [2, 14], we decompose each rooted binary tree into a set of maximal vertex-disjoint downward paths, called *heavy paths*, connected by tree edges called *light edges*. Each heavy path is in turn represented by a biased binary tree whose leaf-nodes correspond to the vertices of the heavy path. Non-leaf nodes represent edges of this heavy path, ordered in the biased tree according to their depth along the path. Therefore, vertices that are higher (closer to the root) in the path correspond to leaves farther left in the biased tree. Each leaf node ℓ of a biased tree B represents an internal vertex v of the tree which has a unique light edge l_v adjacent to it. We keep a pointer from ℓ to this light edge. Note that the other endpoint of l_v is the root of another heavy path which in turn is represented by another biased tree, say B' . We merge these two biased trees by adding a pointer from ℓ to the root of B' . After merging all the biased trees in this way, we obtain the grappa tree of T . A node of the grappa tree that is non-leaf in its biased tree represents a heavy edge and has two children, whereas a node that is a leaf of its biased tree represents a vertex of the heavy path (and its unique adjacent light edge) and has only one child. By a suitable choice of paths and biasing, as described in [14], the grappa-tree has height $O(\log n)$.

In addition, grappa trees allow us to store left and right marks on each of its nodes, i.e., on each edge of T . To assign the mark of a node, grappa trees support the $O(\log n)$ -time operation $\text{LEFT-MARK}(T, v, m_l)$ which sets the mark m_l to every edge in the path from v to the root of T ($\text{RIGHT-MARK}(T, v, m_r)$ is defined analogously). In our setting, we use the marks of an edge e to keep track of the faces adjacent to this edge in a geometric embedding of T . Since T is rooted, we can differentiate between the left and the right faces adjacent to e . For a full description of the operations supported by grappa trees refer to [2].

► **Theorem 12** (Theorem 7 from [2]). *A grappa-tree maintains the combinatorial structure of a forest and supports each operation in $O(\log n)$ worst-case time per operation, where n is the total size of the trees affected by the operation.*

The Voronoi diagram. Let S be a set of n sites in convex position and let $\mathcal{V}(S)$ be the binary tree representing the Voronoi diagram of S . We store $\mathcal{V}(S)$ using a grappa tree. In addition, we assume that each edge of $\mathcal{V}(S)$ has two *face-markers*, its left and right markers which respectively store the site of S whose Voronoi region is adjacent to this edge. While a grappa tree stores only the topological structure of $\mathcal{V}(S)$, with the aid of the face-markers we can retrieve the geometric representation of $\mathcal{V}(S)$. Namely, for each vertex v of $\mathcal{V}(S)$, we can look at its adjacent edges and their face-markers to retrieve the point in the plane representing the location of v in the Voronoi diagram of S in $O(1)$ time. Therefore, we refer to v also as a point in the plane. Recall that each vertex v of $\mathcal{V}(S)$ is the center of a circle that passes through at least three sites of S , we call these sites the *definers* of v and we call this circle the *definer circle* of v .

► **Observation 13.** *Given a new site q in the plane such that $S' = S \cup \{q\}$ is in convex position, the vertices of $\mathcal{V}(S)$ that are closer to q than to any other point of S' are exactly the vertices whose definer circle encloses q .*

Let q be a new site such that $S' = S \cup \{q\}$ is in convex position. Let $\text{CELL}(q, S')$ be the Voronoi region of q in the Voronoi diagram of S' and let $\partial\text{CELL}(q, S')$ denote its boundary. Recall that we can think of $\mathcal{V}(S)$ as a Halin graph by connecting all its leaves by a cycle to make it 3-regular. While we do not explicitly use this cycle, we need it to make our definitions consistent. In this Halin graph, the curve $\partial\text{CELL}(q, S')$ can be made into a closed curve by going around the leaf of $\mathcal{V}(S)$ contained in $\text{CELL}(q, S')$; namely the point at infinity of the bisector between the two neighbors of q along the convex hull of S' . In this way,

$\partial\text{CELL}(q, S')$ becomes a flarbable curve. Therefore, we are interested in performing the flarb operation it induces which leads into a transformation of $\mathcal{V}(S)$ into $\mathcal{V}(S')$.

Heavy paths in Voronoi diagrams. Recall that for the grappa tree of $\mathcal{V}(S)$, we computed a heavy path decomposition of $\mathcal{V}(S)$. In this section, we first identify the portion of each of these heavy paths that lies inside $\text{CELL}(q, S')$. Once this is done, we test if any edge adjacent to an endpoint of these paths is preserved. Then within each heavy path, we use the biased trees built on it to further find whether there are non-preserved edges inside this heavy path. After identifying all the non-preserved edges, we remove them which results in a split of $\mathcal{V}(S)$ into a forest where each edge inside $\text{CELL}(q, S')$ is preserved. Finally, we show how to link back the disjoint components into the tree resulting from the flarb operation.

► **Observation 14.** *Given a 3-regular graph G and a flarbable curve \mathcal{C} , if we can test if a point is enclosed by \mathcal{C} in $O(1)$ time, then we can test if an edge is preserved in $O(1)$ time.*

As a first step, we find the heavy paths of $\mathcal{V}(S)$ whose roots lie in $\text{CELL}(q, S')$. Additionally, we find the portion of each of these heavy paths that lies inside $\text{CELL}(q, S')$.

Recall that there is a leaf ρ of $\mathcal{V}(S)$ that lies in $\text{CELL}(q, S')$ being the point at infinity of the bisector between the two neighbors of q along the convex hull of S' . As a first step, we root $\mathcal{V}(S)$ at ρ by calling $\text{Evert}(\rho)$. In this way, ρ becomes the root of $\mathcal{V}(S)$ and all the heavy paths have a *root* which is their endpoint closest to ρ .

Let R be the set the of roots of all heavy paths of $\mathcal{V}(S)$, and let $R_q = \{r \in R : r \in \text{CELL}(q, S')\}$. We focus now on computing the set R_q . By Observation 13, each root in R_q has a definer circle that contains q . We use a dynamic data structure that stores the definer circles of the roots in R and returns those circles containing a given query point in poly-logarithmic time [6]. Using this data structure, we can find a root of R_q whose definer contains q , remove it and repeat the process until finding all the roots in R_q . We obtain the following result.

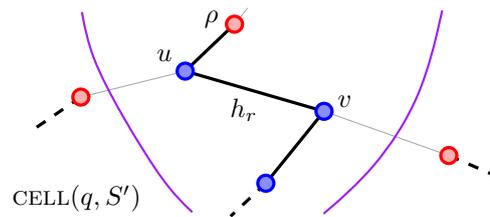
► **Lemma 15.** *We can compute each root in R_q in total $O(|R_q| \log^6 n)$ amortized time.*

Given a root $r \in R$, let h_r be the heavy path whose root is r . Because the portion of $\mathcal{V}(S)$ that lies inside $\text{CELL}(q, S')$ is a connected subtree, we know that, for each $r \in R_q$, the portion of the path h_r contained in $\text{CELL}(q, S')$ is also connected. In order to compute this connected subpath, we want to find the last vertex of h_r that lies inside of $\text{CELL}(q, S')$, or equivalently, the unique edge of h_r having exactly one endpoint in the interior of $\text{CELL}(q, S')$. We call such an edge the *q-transition edge* of h_r (or simply transition edge). Using the search properties of the biased tree representing the heavy path h_r , we obtain the following result.

► **Lemma 16.** *For a root $r \in R_q$, we can compute the transition edge of h_r in $O(\log n)$ time.*

Finding non-preserved edges. Let $\mathcal{V}_q(S)$ be the subtree induced by all the edges of $\mathcal{V}(S)$ that intersect $\text{CELL}(q, S')$. Now, we work towards showing how to identify each non-preserved edge of $\mathcal{V}_q(S)$ in the fleeq induced by $\partial\text{CELL}(q, S')$. For each root $r \in R_q$, we compute the transition edge e_r of h_r using Lemma 16 in $O(\log n)$ time per edge. Assume that w is the vertex of e_r that is closer to r (or is equal to r). We consider each edge adjacent to w and test whether or not it is preserved. Since each vertex of $\mathcal{V}_q(S)$ has access to its definers via the faces markers of its adjacent edges, we can test if this vertex lies in $\text{CELL}(q, S')$. Thus, by Observation 14, we can decide whether an edge of $\mathcal{V}_q(S)$ is preserved in $O(1)$ time.

We mark each non-preserved edge among them as *shadow*. Because we can test if an edge is preserved in $O(1)$ time, and since computing e_r takes $O(\log n)$ time by Lemma 16, this



■ **Figure 4** Path h_r contains two adjacent vertices u and v such that the light edge of u is a left edge while the light edge of v is a right edge. The edge uv cannot be preserved.

can be done in total $O(|R_q| \log n)$ time amortized. In addition, notice that if h_r contains two adjacent vertices u and v such that the light edge of u is a left edge while the light edge of v is a right edge (or vice versa), then the edge uv cannot be preserved; see Figure 4. In this case, we say that uv is a *bent edge*. We want to mark all the bent edges in $\mathcal{V}_q(S)$ as shadow, but first we need to identify them efficiently.

Note that it suffices to find all the bent edges of h_r for a given root $r \in R_q$, and then repeat this process for each root in R_q . To find the bent edges in h_r , we further extend the grappa-tree in such a way that the biased tree representing h_r allows us to search for bent edges in $O(\log n)$ time. This extension is described as follows. Recall that each leaf s_v of a biased tree corresponds to a vertex v of the heavy path and has a pointer to the unique light edge adjacent to v . Since each light edge is either left or right, we can extend the biased tree to allow us to search in $O(\log n)$ time for the first two consecutive leaves where a change in direction occurs. From there, standard techniques allow us to find the next change in direction in additional $O(\log n)$ time. Therefore, we can find all the bent edges of a heavy path h_r in $O(\log n)$ time per bent edge. After finding each bent edge in h_r , we mark it as a shadow edge.

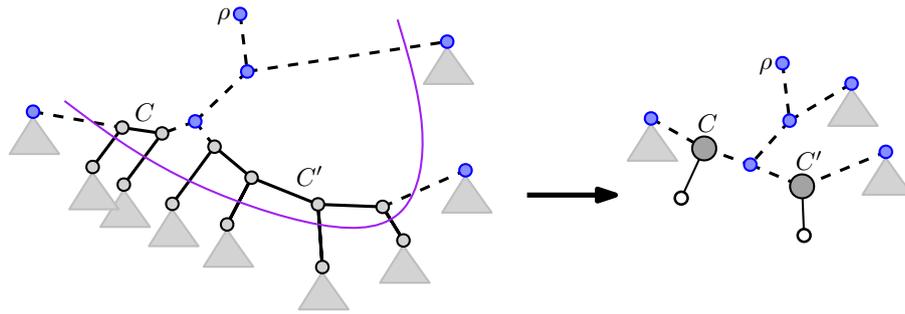
► **Lemma 17.** *An edge of $\mathcal{V}_q(S)$ is a preserved edge if and only if it was not marked as a shadow edge.*

Let σ be the number of shadow edges of $\mathcal{V}(S)$, which is equal to the number of non-preserved edges by Lemma 17. The following relates the size of R_q with the value of σ and the cost of the flarb.

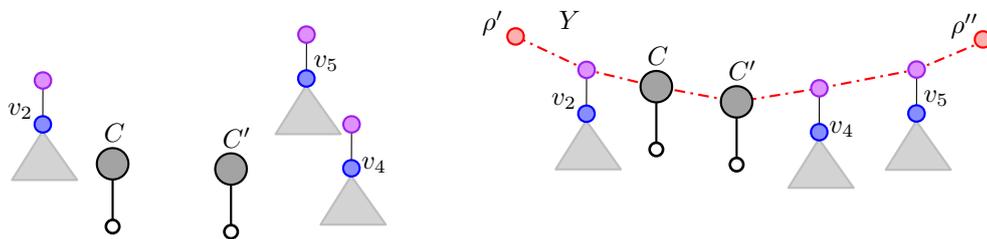
► **Lemma 18.** *It holds that $|R_q| = O(\sigma \log n)$. Moreover, $\sigma = \Theta(\text{COST}(\mathcal{V}(S), \partial \text{CELL}(q, S')))$.*

The compressed tree. Let \mathcal{F} be the forest obtained from $\mathcal{V}_q(S)$ by removing all the shadow edges (this is just for analysis purposes, so far no cut has been performed). Note that each connected components of \mathcal{F} consists only of preserved edges that intersect $\text{CELL}(q, S')$. Thus, each component inside $\text{CELL}(q, S')$ is a comb, with a path as *spine* and each child of a spine vertex pointing to the same side; see Figure 5. Thus, we have right and left combs, depending on whether the children of the spine are left or right children.

Our objective in the long term is to cut all the shadow edges and link the remaining components in the appropriate order to complete the flarb. To this end, we would like to perform an Eulerian tour on the subtree $\mathcal{V}_q(S)$ to find the order in which the subtrees of $\mathcal{V}(S) \setminus \mathcal{V}_q(S)$ that hang from the leaves of $\mathcal{V}_q(S)$ appear along this tour. However, this may be too expensive as we want to perform this in time proportional to the number of shadow edges, and the size of $\mathcal{V}_q(S)$ can be way larger. To make this process efficient, we compress $\mathcal{V}_q(S)$ by contracting each comb of \mathcal{F} into a single super node. By performing an Eulerian



■ **Figure 5** Two combs of \mathcal{F} that are compressed into super nodes with their respective dummy leaves. An Eulerian tour around the compressed tree provides us with the order in which the trees hanging outside of $\text{CELL}(q, S')$ should be attached.



■ **Figure 6** Left: An anchor node is created for each isolated leaf of $\mathcal{V}_q(S)$ and attached as its parent. Other isolated nodes are ignored. Right: A super comb is created connecting two new leaves ρ' and ρ'' through a path. This path connects anchor and super nodes in the order retrieved by the Eulerian tour around the compressed tree.

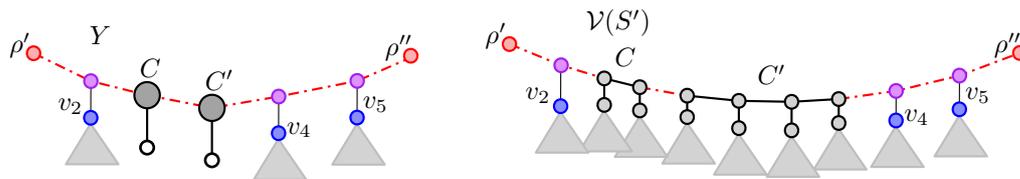
tour around this compressed tree, we obtain the order in which each component needs to be attached. We construct the compressed flarb and then we decompress it as follows.

Note that each comb has exactly two shadow edges that connect it with the rest of the tree. Thus, we contract the entire component containing the comb into a single super node and add a left or right dummy child to it depending on whether this comb was left or right, respectively; see Figure 5. After the compression, the shadow edges together with the super nodes and the dummy vertices form a tree, called the *compressed tree*, that has $O(\sigma)$ vertices and edges, where σ is the total number of shadow edges.

► **Lemma 19.** *We can compute the compressed tree in $O(\sigma \log \sigma)$ time.*

The compressed tree is then a binary tree where each super node has degree three and each edge is a shadow edge. We now perform an Eulerian tour around this compressed tree and retrieve the order in which the leaves of this tree are visited. Some leaves are dummy leaves and some of them are original leaves of $\mathcal{V}_q(S)$; see Figure 5.

Completing the flarb. We now proceed to remove each of the shadow edges which results in a (compressed) forest with $O(\sigma)$ components. Note that each of the original leaves of $\mathcal{V}_q(S)$ was connected with its parent via a shadow edge and hence it lies now as a single component in the resulting forest. For each of these original leaves of $\mathcal{V}_q(S)$, we create a new *anchor* node and link it as the parent of this leaf. Moreover, there could be internal vertices that become isolated. In particular this will be the case of the root ρ . These vertices are deleted and ignored for the rest of the process.



■ **Figure 7** The tree $\mathcal{V}(S')$ achieved after the decompression.

To complete the flarb, we create two new nodes ρ' and ρ'' which will be the two new leaves of the Voronoi diagram, one of them replacing ρ . Then, we construct a path with endpoints ρ and ρ' that connects the super nodes and the anchor nodes according to the traversal order of their leaves; see Figure 6. The resulting tree is a *super comb* Y , where each vertex on the spine is either a super node or an anchor node, and all the leaves are either dummy leaves or original leaves of $\mathcal{V}_q(S)$. Since we glued $O(\sigma)$ components into a tree, we need $O(\sigma)$ time.

We proceed now to decompress Y . To decompress a super node of Y that corresponds to a comb, we consider the two neighbors of the super node in Y and attach each of them to the ends of the spine of the comb. For an anchor node, we simply note that there is a component of $\mathcal{V}(S)$ hanging from its leaf; see Figure 7. In this way, we obtain all the edges that need linking. After the decompression, we end with the tree $\mathcal{V}(S')$ resulting from the flarb. Thus, the flarb operation of inserting q can be implemented with $O(\sigma)$ link and cuts.

Recall that any optimal algorithm needs to perform a cut for each edge that is not preserved. Since each non-preserved edge is shadow by Lemma 17, the optimal algorithm needs to perform at least $\Omega(\sigma)$ operations. Therefore, our algorithm is optimal and computes the flarb using $\Theta(\sigma)$ link and cuts. Moreover, by Lemmas 15 and 16 we can compute the flarb in $O(|R_q| \log^6 n + \sigma \log n)$ amortized time using $\Theta(\sigma)$ link and cuts. Since $|R_q| = O(\sigma \log n)$ and $\sigma = \Theta(\text{COST}(\mathcal{V}(S), \partial_{\text{CELL}}(q, S')))$ by Lemma 18, we obtain the following.

► **Theorem 20.** *The flarb operation corresponding to the insertion of q can be implemented with $O(K)$ link and cuts, where K is the cost of the flarb. Moreover, it can be implemented in $O(K \log^7 n)$ amortized time.*

References

- 1 Greg Aloupis, Luis Barba, and Stefan Langerman. Circle separability queries in logarithmic time. In *Proceedings of the 24th Canadian Conference on Computational Geometry, CCCG'12*, pages 121–125, August 2012.
- 2 Boris Aronov, Prosenjit Bose, Erik D Demaine, Joachim Gudmundsson, John Iacono, Stefan Langerman, and Michiel Smid. Data structures for halfplane proximity queries and incremental Voronoi diagrams. In *LATIN 2006: Theoretical Informatics*, pages 80–92. Springer, 2006.
- 3 Luis Barba. Disk constrained 1-center queries. In *Proceedings of the 24th Canadian Conference on Computational Geometry, CCCG'12*, pages 15–19, August 2012.
- 4 Jon Louis Bentley and James B Saxe. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.
- 5 P. Bose, S. Langerman, and S. Roy. Smallest enclosing circle centered on a query line segment. In *Proceedings of the 20th Canadian Conference on Computational Geometry (CCCG 2008)*, pages 167–170, 2008.
- 6 Timothy M Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. *Journal of the ACM (JACM)*, 57(3):16, 2010.

15:16 Incremental Voronoi diagrams

- 7 Yi-Jen Chiang and Roberto Tamassia. Dynamic algorithms in computational geometry. *Proceedings of the IEEE*, 80(9):1412–1434, 1992.
- 8 Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational geometry*. Springer, 2000.
- 9 Herbert Edelsbrunner and Raimund Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1(1):25–44, 1986.
- 10 Rolf Klein. *Concrete and Abstract Voronoi Diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer, 1989.
- 11 Rolf Klein, Elmar Langetepe, and Zahra Nilforoushan. Abstract voronoi diagrams revisited. *Computational Geometry*, 42(9):885–902, 2009.
- 12 Mark H Overmars. *The design of dynamic data structures*, volume 156. Springer Science & Business Media, 1983.
- 13 Seth Pettie. Applications of forbidden 0–1 matrices to search tree and path compression-based data structures. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1457–1467, 2010.
- 14 Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.