Санкт-Петербургский государственный университет

Золотов Борис Алексеевич

Выпускная квалификационная работа

«Algorithms for Incremental Voronoi Diagrams»

Уровень образования: магистратура
Направление: 01.04.01 «Математика»
Основная образовательная программа: ВМ.5832.2019
Профиль (при наличии): нет

Научный руководитель:
Ph. D. in Informatics,
доцент факультета МКН СПбГУ
Арсеньева Елена Александровна

Рецензент:
Professor, Algorithms Research Group
Department of Computer Science
Université libre de Bruxelles
Иаконо Джон

Санкт-Петербург
2021 год

Contents

1 Introduction **3**

1.1 Voronoi diagrams . . . 4

1.1.1 Combinatorial Changes to the Voronoi Diagram and the Flarb Operation . . 4

1.1.2 Cost of the Flarb . . . 6

1.2 The coin problem . . . 7

1.3 Gluings of squares . . . 8

1.3.1 Chen—Han algorithm for gluings of squares . . . 9

1.4 Main results . . . 9

2 Sublinear algorithm for incremental Voronoi diagrams **9**

2.1 Description of Data Structure . . . 10

2.2 Insertion of a Site . . . 11

2.3 Recognizing Cells With Changes . . . 11

2.3.1 Cell f is Big . . . 11

2.3.2 Cell f is Small . . . 12

2.4 Implementing Combinatorial Changes . . . 12

2.4.1 Processing Big Cells . . . 12

2.4.2 Processing Small Cells . . . 14

2.5 When Small Cells Become Big . . . 16

2.6 Correctness and Time Complexity . . . 16

2.7 Discussion and open problems . . . 17

3 Estimating the number of combinatorial changes in a Voronoi diagram with several point sites and one line **17**

3.1 Voronoi diagrams and Davenport – Schinzel sequences . . . 18

3.2 Tree representation of a Davenport – Schinzel sequence . . . 20

3.3 The order of relabels . . . 21

3.4 Inserts and length of a Davenport – Schinzel sequence . . . 22

3.5 Proof of amortized bound . . . 23

3.6 Discussion and open problems . . . 24

4 Bounds on the number of egde-to-edge gluings of squares **24**

4.1 Algorithm to classify edge-to-edge gluings of squares . . . 27

4.2 Discussion and open problems . . . 27

1 Introduction

Computational geometry studies computations on discrete geometric objects such as arrangements, diagrams, foldings and drawings. In particular one of its main interests is to design algorithms to construct such objects and data structures that store them efficiently.

The most crucial and defining concept applied in all those constructions is *distance:* whatever is studied, there is always some underlying metric that describes the object in question and defines its properties. It can be just the Euclidean metric, or some polyhedral metric, or any other metric that corresponds to a surface or a folding of a polygon.

In this thesis, we consider two different types of a distance metric. The Euclidean metric in $\mathbb{R}^2$ gives rise to the ordinary Voronoi diagrams, and we develop an algorithm that allows for fast update of a Voronoi diagram that is stored explicitly. We also consider polyhedral metrics of the spaces induced by gluing n congruent squares edge to edge, and we propose a way to classify all such spaces in time polynomial in n.

Even though there are many well-known algorithms that construct the Voronoi diagram for a given family of sites [9], the problem of making a Voronoi diagram dynamic, i. e. implementing changes to it when a new point site is inserted, has only drawn attention in the implicit case: maintaining implicit Voronoi diagram can be done in very little time [11, 12, 20]. Maintaining a Voronoi diagram explicitly has not yet been considered.

The worst one can expect is that when a new site is inserted, the number of updates in a Voronoi diagram (i. e. vertices and edges that change) is linear. This can happen for every insertion if we consider an embedded diagram and store the coordinates of all the vertices.

The situation improves if we consider *the graph* of the Voronoi diagram that is subject to combinatorial changes: no coordinates matter, it is just deletion or addition of edges that is performed. In Allen et al. [2] it was proved that when a new site is inserted to a Voronoi diagram, only $O(\sqrt{n})$ combinatorial changes happen to the graph of the diagram. This opened a possibility to find a sublinear algorithm that finds and implements combinatorial changes in an explicitly stored graph of the Voronoi diagram.

In this thesis we present the first algorithm sublinear in n (which is the number of sites) that does exactly that. The algorithm has running time of $\tilde{O}(n^{3/4})$, which still does not achieve the $O(\sqrt{n})$ bound on the number of changes, that is why the quest for a better algorithm is still open.

One can try to employ the well-known «divide and conquer» strategy to find such an algorithm: divide all the sites into two halves by a line and update only the half of the diagram that receives the new site. We show that adding a line as a site to the diagram still allows for a $O(\sqrt{n})$ upper bound on the number of combinatorial changes per insertion. This means that, theoretically, «divide and conquer» is a desirable approach.

Another type of metrics and distances that can be often seen in computational geometry is polyhedral metrics and geodesic distances. A question concerning them that has been standing for a long time already is the Alexandrov's problem of finding a convex polyhedron corresponding to a given polyhedral metric. There is almost no hope of solving this problem exactly for an arbitrary polyhedral metric, that is why several special cases are considered in literature [6, 8, 15, 16, 17].

A result we achieved in this thesis is a significant advance for an important special case: we present a polynomial-time algorithm to classify all the edge-to-edge gluing of at most n squares. It is a development of my bachelor's thesis making use of its results and an answer to a natural open question.

Our results open new directions for subsequent studies: it rises new questions that can be asked — like finding more efficient algorithms or enumerating gluings of other polygons, and suggests new possible methods that can be used to answer them.

This thesis is based in part on published articles and given talks: the part concerning sublinear explicit incremental planar Voronoi diagrams has been presented at JCDCG^3 [4] and published in Journal for Information Processing [5], and the part concerning gluings of squares has been accepted for SoCG Young Researchers Forum 2021 [22]. The latter is a development of the bachelor's thesis of the same author [26].

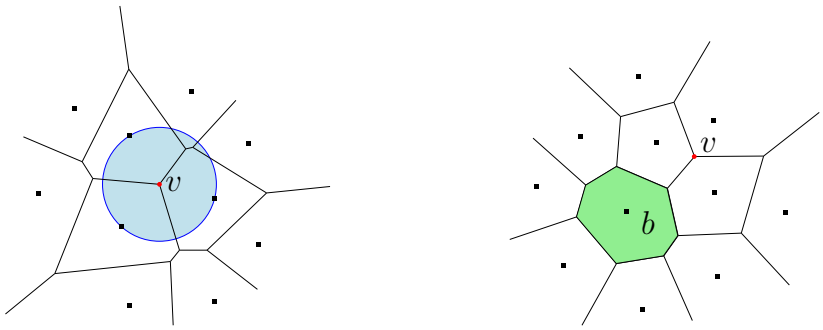
The rest of this section contains the main definitions and results that will be necessary for the main body of this thesis.

1.1 Voronoi diagrams

We begin with standard definitions related to Voronoi diagrams and their basic properties. A detailed treatment of Voronoi diagrams and their applications can be found in [10]. Let $S := \{s_1, s_2, \ldots, s_N\}$ be a set of N distinct points in $\mathbb{R}^2$; these points are called *sites*. Let $\mathrm{dist}(\cdot, \cdot)$ denote the Euclidean distance between two points in $\mathbb{R}^2$. We assume that the sites in S are in *general position*, that is, no four sites lie on a common circle.

Definition 1. The *Voronoi diagram* of S is the subdivision of $\mathbb{R}^2$ into N cells, called *Voronoi cells*, one cell for each site in S, such that a point q lies in the Voronoi cell of a site s_i if and only if $\mathrm{dist}(q, s_i) < \mathrm{dist}(q, s_j)$ for each $s_j \in S$ with $j \neq i$.

Let f_i denote the Voronoi cell of a site s_i. Edges of the Voronoi diagram, called *Voronoi edges,* are portions of bisectors between two sites which are the common boundary of the corresponding Voronoi cells. *Voronoi vertices* are points where at least three Voronoi cells meet. The *Voronoi circle* of a Voronoi vertex v is the circle passing through the sites whose cells are incident to v, see Figure 1a. Vertex v is the center of its Voronoi circle.



(a) (b)

Figure 1: A Voronoi diagram and (a) a Voronoi circle of vertex v and (b) paw v of Voronoi cell f.

Since the sites are in the general position, each Voronoi vertex has degree three. Each Voronoi edge is either a segment or a ray and the graph of the Voronoi diagram formed by its edges and vertices is planar and connected.

1.1.1 Combinatorial Changes to the Voronoi Diagram and the Flarb Operation

We now overview the definitions and results from Allen et al. [2] that we need to present our approach. In order to prove the $\Theta(N^{\frac{1}{2}})$ bound on the number of combinatorial changes caused by

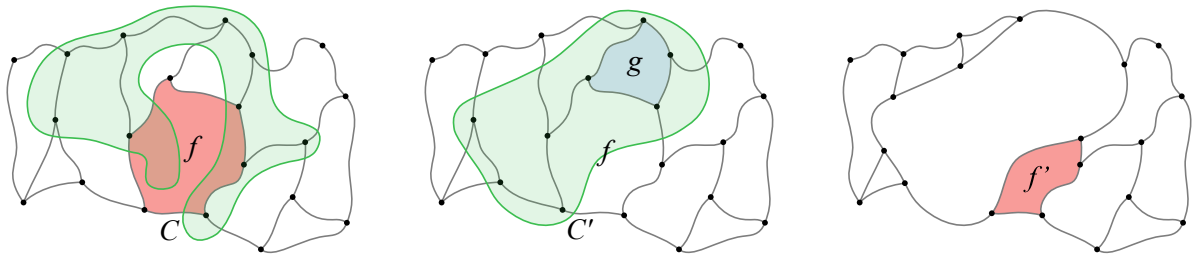
insertion of a site, a graph operation called *flarb* is introduced.

Let G be a planar 3-regular graph embedded in $\mathbb{R}^2$ without edge crossings (edges are not necessarily straight-line). Let $\mathcal{C}$ be a simple closed Jordan curve in $\mathbb{R}^2$.

Definition 2. Curve $\mathcal{C}$ is called *flarbable* for G if:

- the graph induced by vertices inside the interior of $\mathcal{C}$ is connected,
- $\mathcal{C}$ intersects each edge of G either at a single point or not at all,
- $\mathcal{C}$ passes through no vertex of G, and
- the intersection of $\mathcal{C}$ with each face of G is path-connected.

For example, curve $\mathcal{C}$ in Figure 2a is not flarbable since the intersection between its interior (shaded green) and the highlighted face (red) consists of two disconnected parts. In Figure 2b curve $\mathcal{C}'$ is flarbable.



(a) (b) (c)

Figure 2: Examples of (a) not flarbable curve (b) flarbable curve; (c) result of applying the flarb operation for curve $\mathcal{C}'$.

Given a graph G and a curve $\mathcal{C}$ flarbable for G, the *flarb* operation is, informally, removing part of G that is inside $\mathcal{C}$ and replacing it with $\mathcal{C}$. Formally, the flarb operation for G and $\mathcal{C}$ is defined as follows (see Figure 2b, Figure 2c):

- For each edge $e_i \in G$ that intersects $\mathcal{C}$ let u_i be its vertex lying inside $\mathcal{C}$ and v_i its vertex outside $\mathcal{C}$. Create a new vertex $w_i = \mathcal{C} \cap e_i$ and connect it to v_i along e_i.
- Connect consecutive vertices w_i along $\mathcal{C}$.
- Delete all the vertices and edges inside $\mathcal{C}$.

Let $\mathcal{G}(G, \mathcal{C})$ denote the graph obtained by applying the flarb operation to graph G and curve $\mathcal{C}$.

Lemma 1. *The following holds for graph $\mathcal{G}(G, \mathcal{C})$:*

(a) $\mathcal{G}(G, \mathcal{C})$ has at most two more vertices than G does;
(b) $\mathcal{G}(G, \mathcal{C})$ is a 3-regular planar graph;
(c) $\mathcal{G}(G, \mathcal{C})$ has at most one more face than G does.

Proof. Items (a) and (b) are proved in [2], Lemma 2.2. To prove (c) note that there is one new face bounded by the cycle added along $\mathcal{C}$ while performing the flarb. All the other faces of G are either deleted, left intact, or cropped by $\mathcal{C}$; these operations obviously do not increase the number of faces. □

Theorem 2 ([2])**.** *Let G be a graph of the Voronoi diagram of a set of $N-1$ sites $s_1 \dots s_{N-1}$. For any new site s_N there exists a flarbable curve $\mathcal{C}$ such that the graph of the Voronoi diagram of sites $s_1 \dots s_N$ is $\mathcal{G}(G, \mathcal{C})$.*

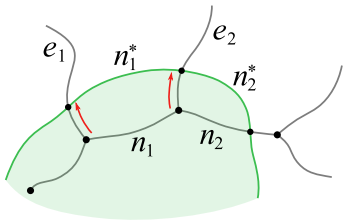


Figure 3: Edges n_1^* and n_2^* can be obtained without any links or cuts.

1.1.2 Cost of the Flarb

We want to analyze the number of structural changes that a graph undergoes when we apply the flarb operation to it. There are two basic combinatorial operations on graphs:

- *Link* is the addition of an edge between two non-adjacent vertices.
- *Cut* is the removal of an existing edge.

Other combinatorial operations, for example insertion of vertex of degree 2, are assumed to have no cost.

Definition 3. $\text{cost}(G, \mathcal{C})$ is the minimum number of links and cuts needed to transform G into $\mathcal{G}(G, \mathcal{C})$.

Note that sometimes there are less combinatorial changes needed than the number of edges intersected by $\mathcal{C}$. Consider edges e_1, e_2 of G crossed consecutively by $\mathcal{C}$ and edge n adjacent to them that reappears in $\mathcal{G}(G, \mathcal{C})$ as a part n^* of $\mathcal{C}$. Then n^* can be obtained without any links or cuts by lifting n along e_1 and e_2 until it coincides with n^* or (which is the same) shrinking e_1 and e_2 until their endpoints coincide with their intersections with $\mathcal{C}$ (see Figure 3). We will call it *preserving operation*.

Theorem 3 ([2])**.** *For a flarbable curve $\mathcal{C}$, it holds that*

$$\text{cost}(G, \mathcal{C}) \leq 12|\mathcal{S}(G, \mathcal{C})| + 3|\mathcal{B}(G, \mathcal{C})| + O(1).$$

Where

- $|\mathcal{B}(G, \mathcal{C})|$ *is the number of faces of G wholly contained inside $\mathcal{C}$ (g is such a face on Figure 2b).*
- $|\mathcal{S}(G, \mathcal{C})|$ *is the number of* shrinking *faces — i.e., the faces whose number of edges decreases when flarb operation is applied (face f is shrinking on Figure 2b–2c).*

The following upper bound can be used to evaluate the number of combinatorial changes needed to update the graph of a Voronoi diagram when a new site is inserted.

Theorem 4 ([2])**.** *Consider one insertion of a new site to a Voronoi diagram V.*

- *The number of cells of V undergoing combinatorial changes is $O(N^{\frac{1}{2}})$ amortized in a sequence of insertions;*
- *There are a constant number of combinatorial changes per cell;*
- *The cells of V with combinatorial changes form a connected region.*

Further in this thesis by *a change in cell* we always mean a combinatorial change, that is a *link* or a *cut*.

1.2 The coin problem

A detailed introduction into amortized analysis, including the definition of potential function and examples of estimates of it is given in [14]. Here we focus on one classical result concerning amortized analysis. It is rather folklore and is given as an exercise in several university courses; we show it here to establish its formal proof, because it is crucial for understanding of what we prove further.

Given n coins stored in n piles (some piles may be empty). A series of operations is executed on these piles. One operation consists of selecting a pile and distributing all its coins equally among other piles, one coin per pile. Piles that receive a coin and piles that do not can be chosen freely. An example of such operations can be seen in Figure 4. We assume the number of piles is equal to the number of coins so that these operations can be executed on any configuration of coins.

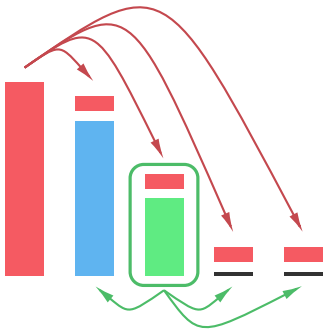


Figure 4: Two consecutive operations executed on the piles of coins: first, the coins of the leftmost (red) pile are distributed, afterwards the coins of the central (green) pile are distributed

Clearly during one operation at most n coins are distributed. However, during k consecutive operations the average number of coins disrtibuted per operation is $o(n)$. Informally, each operation makes the heights of the piles more uniform, making it impossible to have a large pile to distribute after each of the operations. This observation is formalized by the following theorem:

Theorem 5. *For $k \geq \sqrt{n}$, if k consecutive operations are executed, then the average number of coins distributed per operation is at most $3\sqrt{n}$.*

Proof. Denote piles by $p_1, \ldots, p_n$. We introduce a potential function that describes the configuration of piles and helps estimate the number of coins distributed during an operation executed on a pile:

$$\Phi = \sum_{j=1}^{n} \min\left\{\text{size}(p_j), \sqrt{n}\right\}.$$

Note that Φ is always at most n, since n is the sum of actual sizes of all the piles. Denote by T_i the number of coins distributed during the i-th operation, and by Φ_i the value of the potential after the i-th operation. Therefore, Φ_0 and Φ_k are the values of the potential before and after the execution of all the operations respectively.

We will now estimate $T_i + \Phi_{i-1} - \Phi_i$. To do so, note that the pile that is being distributed (without loss of generality, p_1) decreases in size to zero, and several other piles increase in size by 1. Distributing p_1 makes Φ decrease by at most $\sqrt{n}$, regardless of $\text{size}(p_1)$ being greater or less than $\sqrt{n}$, by definition of Φ. Figure 5 illustrates it: a change in size of a pile does not affect the potential if the size of the pile is greater than $\sqrt{n}$.

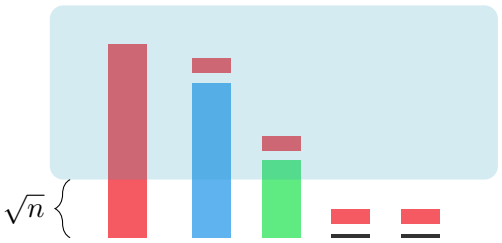


Figure 5: A removal or an addition of a coin only affects a pile if it has less than $\sqrt{n}$ coins

Several other piles receive one coin, the number of such piles is equal to T_i. However, some of these piles may not contribute to the change in the potential, again because their size is greater than $\sqrt{n}$. Note that the number of such large piles is at most $\sqrt{n}$. Thus, when the coins are being put into piles, there is at least $T_i - \sqrt{n}$ increase in potential. Combining these observations,

$$T_i + \Phi_{i-1} - \Phi_i \;\leq\; T_i + \sqrt{n} - \left(T_i - \sqrt{n}\right) \;=\; 2\sqrt{n}.$$

We can now sum up $T_i + \Phi_{i-1} - \Phi_i$ for each of the consecutive operations. Note that $\Phi_0 - \Phi_k$ is between $-n$ and n.

$$\sum_{i=1}^{k} \left(T_i + \Phi_{i-1} - \Phi_i\right) \;=\; \sum_{i=1}^{k} T_i + \Phi_0 - \Phi_k \;\leq\; 2\sqrt{n} \cdot k + n.$$

This means average T_i per operation is at most $2\sqrt{n} + \frac{n}{k} = 3\sqrt{n}$. □

Informally, this theorem means that if during an operation one pile loses many coins, and many distinct piles get at most one coin each, there can only be so much of such operations. We will rely on this observation further on.

1.3 Gluings of squares

Given a collection of 2D polygons, a *gluing* describes a closed surface by specifying how to glue each edge of these polygons onto another edge. We consider only proper gluings, where only segments of equal lengths can be glued together. The following theorem is crucial in that it establishes the connection between gluings and convex polyhedra:

Theorem 6 (Alexandrov, 1950, [1])**.** *If a gluing is homeomorphic to a sphere and the sum of angles at each of its vertices is at most* 360° *then there is a single convex polyhedron P that can be glued from this net.*

Note that the polygons of the gluing may be folded in order to glue the polyhedron.

There is no known exact algorithm for reconstructing the 3D polyhedron. It is known that the problem of reconstructing the polyhedron can be reduced to a system of partial differential equations [19]. Still this method does not produce the exact answer, and there is no known algorithm for it that works faster than pseudopolynomial in n, n being the number of vertices. Sometimes the coordinates of the polyhedron, even if its general shape is known, can not be expressed as closed formulas [18].

Enumerating all possible valid gluings is also not an easy task. Demaine et al. [15] showed that for any even n there is a polygon with n vertices that has $2^{\Omega(n)}$ gluings: that is a star with two

additional vertices on midpoints of edges half perimeter from one another. This is why it is also important to estimate the number of gluings of the collection of polygons under consideration.

Complete enumerations of gluings and the resulting polyhedra are only known for very specific cases such as the Latin cross [16], a single regular convex polygon [17], and a collection of regular pentagons [7, 8] glued edge-to-edge.

The case when the polygons to be glued together are all congruent regular k-gons, and the gluing is edge-to-edge, was studied recently for $k \geq 6$ [6]. The aim of Section 4 is to study the case of $k = 4$: namely, to *enumerate* all valid gluings of squares and *classify* them up to isomorphism.

1.3.1 Chen—Han algorithm for gluings of squares

It is shown [17] that polyhedra are isomorphic if the lengths of shortest geodesic paths between their vertices of nonzero curvature coincide. Thus, the problem of finding out if two gluings are isomorphic can be reduced to calculating the pairwise geodesic distances between vertices of a gluing. Algorithm we are using for this in Section 4.1 is the Chen—Han algorithm [13].

The idea of the algorithm is to project a cone of all possible paths from the source onto the polygons of the gluing. For n faces, this algorithm runs in $O(n^2)$ time. To apply it for arbitrary edge-to-edge gluings of squares, it has to be proven that the running time is preserved. We make use of a Lemma that was proved in the Bachelor's thesis.

Lemma 7 ([22, 26])**.** *If T is a square of the gluing and π is a geodesic shortest path between two vertices of the gluing then the intersection between π and T is of at most 5 segments.*

This lemma implies the following theorem.

Theorem 8. *The isomorphism between two edge-to-edge gluings of at most n squares can be tested in $O(n^2)$ time.*

1.4 Main results

In this thesis we:

1) Present a sublinear algorithm that handles insertion of a new point site to an explicitly stored graph of the Voronoi diagram: Theorem 11;

2) Prove the $O(\sqrt{n})$ bound on the number of combinatorial changes in the boundary of the cell of a single straight line in a Voronoi diagram: Theorem 15;

3) Prove upper (Theorem 22) and lower (Theorems 23, 24) bounds on the number of edge-to-edge gluings of at most n congruent squares.

2 Sublinear algorithm for incremental Voronoi diagrams

In this section we present our algorithm that handles insertion of a new site to the Voronoi diagram of several point sites. We start by describing the data structure in which the diagram is stored, and then proceed to implementing the changes in this data structure.

We will need three more definitions specific to our data structure.

Definition 4. The *size* of a Voronoi cell is the number of Voronoi edges constituting its boundary. We denote the size of cell f by $|f|$.

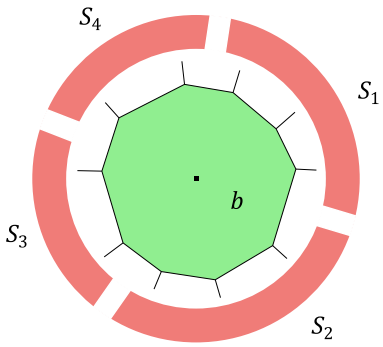


Figure 6: b is a big cell; each of data structures $S_1 \dots S_4$ is associated with a consecutive range of its paws and stores Voronoi circles of the relevant ones.

Definition 5. A Voronoi cell of a Voronoi diagram with N sites is called a *big cell* if it has size more than $N^{\frac{1}{4}}$. Otherwise it is called *small*.

Definition 6. The *paws* of Voronoi cell f are the Voronoi vertices that are connected to the boundary of f by an edge and are not themselves on the boundary of f, see Figure 1b. A paw is called *relevant* if it is not incident to a big cell.

2.1 Description of Data Structure

Our data structure consists of the following parts.

- The graph G_N of the Voronoi diagram represented by its adjacency list: for each Voronoi vertex v we store the list of all Voronoi vertices connected to v. Since the sites are in the general position, each list has length 3, therefore we can find and replace its elements in constant time. Thus any link or cut can be performed in constant time as well as insertion or deletion of a vertex of degree 2.
- For each vertex v its *data* D_v is stored. It is a list of the three sites that define the Voronoi circle of v, that is, the sites whose cells are incident to v.
- A dynamic nearest neighbor structure (DNN) [12] for the sites which supports insertion and deletion of sites and nearest neighbor queries in $\tilde{O}(1)$ amortized time.
- The *graph* Γ_N *of big cells* which is simply the dual graph to the subgraph of G_N formed by big cells. Vertices of Γ_N are big cells themselves and edges connect vertices corresponding to pairs of big cells that are adjacent. Graph Γ_N has $O(N^{\frac{3}{4}})$ edges, since it is a planar graph of at most $N^{\frac{3}{4}}$ vertices. For each pair of adjacent big cells b_1, b_2 we also store two Voronoi vertices they share. We store graph Γ_N as an adjacency list, where for each vertex, its edges are stored in a binary search tree ordered counterclockwise around the corresponding big cell. The vertices of Γ_N are stored in a binary search tree. This allows us to access any edge of Γ_N in $\tilde{O}(1)$ time.
- For each big cell b_i store a circular linked list of $\Theta(|b_i| \,/\, N^{\frac{1}{4}})$ data structures each associated with a consecutive range of $O(N^{\frac{1}{4}})$ paws of B_i, see Figure 6. Each structure stores the Voronoi circles of the relevant paws of b_i (recall that a paw is relevant if it is not incident to a big cell, see Definition 6).
  The collections of circles are stored using *dynamic circle-reporting structures (DCRs)* that are variants of the DNN structure constructed in [2]. DCRs support insertion and deletion of circles in time $\tilde{O}(1)$, and given a query point, report all k circles containing the point in time $\tilde{O}(k)$.

- For each big cell b_i a *yard tree* T_{b_i} supporting the following operations in $\tilde{O}(1)$ time:
  - for a specified continuous range $v_1 \dots v_m$ of vertices of b_i updating $D_{v_1} \dots D_{v_m}$, changing s_i to a given site s_j.
  - removing a continuous range of vertices from b_i and create a new cell with these vertices preserving their order *(split)*,
  - merging the trees that correspond to big cells b_i and b_j (when two cells are merged their common edge is deleted), so that the same operations can apply to the resulting tree.

  One can use link-cut trees [25] or a collection of red-black trees with two-way pointers for this purpose, see Cormen et al. [14] for details.
- For each cell f_i we need to store its size $|f_i|$.

2.2 Insertion of a Site

We aim to implement the update of graph G_{N-1} to become G_N when a *new site* s_N is added to the Voronoi diagram. Our goal is to quickly locate the affected cells that need combinatorial changes, and to avoid processing the other cells. When the cells that need changes are located, we implement these changes using the techniques of [2].

Let the cell of the new site s_N be called f_N. We denote the boundary of f_N by $\mathcal{C}_N$. According to Theorem 2, what we are about to perform is the *flarb* operation on graph G_{N-1} of the current Voronoi diagram and curve $\mathcal{C}_N$.

We first use the DNN structure to locate one Voronoi cell, call it f_{dnn}, that must change — the one whose site is the closest to newly added s_N. We then add s_N to the DNN. Finally we create the queue with all big cells of G_{N-1} and cell f_{dnn}. This whole procedure takes $\tilde{O}(1)$ time as the list of all big cells is already stored.

We then remove each cell f from the queue, process it, and add into the queue the small cells neighboring f with unprocessed changes. We do not have to add big cells neighboring f as all of them were already in the initial queue and thus will be processed. Figure 7 shows a pseudocode of this procedure.

2.3 Recognizing Cells With Changes

Let f be a cell with combinatorial changes. We can identify the neighboring cells of f that change using the following theorem:

Theorem 9 ([2])**.** *Let g be a cell adjacent to f. Let v_1, v_2 be the vertices of g that are paws of f. Cell g needs to undergo combinatorial changes if and only if the Voronoi circle of v_1 or v_2 encloses s_N.*

See Figure 8a for an example. Cell f is a cell with changes, n_1 and n_2 are its paws. The Voronoi circle of n_1 encloses the new site s_N and the one of n_2 does not. Therefore cells f_1 and f_2 need combinatorial changes as they are incident to vertex n_1, and cell f_3 does not need any changes.

We now consider separately the case when cell f is a big cell (Section 2.3.1) and the case when it is a small cell (Section 2.3.2).

2.3.1 Cell f is Big

We use DCRs of cell f: they return all the relevant paws of f whose Voronoi circles enclose s_N. Small cells that are incident to these paws and are adjacent to f need combinatorial changes

```
Q := queue of all big cells
Changed := empty array of cells
Q.append (f_dnn)
DNN.insert(s_N)
while not Q.empty do
    f := Q.dequeue
    Changed.append(f)
    if f is big then
        add f's small neighbors that need changes
            to Q using Section 2.3.1
    else                                (f is small)
        add f's neighbors that need changes
            to Q using Section 2.3.2
    end if
end while
implement changes in cells in Changed
    as described in Section 2.4
Some small cells have become big and some big have become small,
    fix corresponding data structures as described in Section 2.5
```

Figure 7: Pseudocode describing insertion of new site s_N

and thus have to be added to queue `Q`.

Two cells are to be considered separately: those that are neighboring f through an edge that is crossed by $\mathcal{C}_N$. Denote them by f_{Left} and f_{Right}, see Figure 8b. If they are small, we check whether the Voronoi circles of at most four paws of f incident to them (call these paws $p_1 \dots p_4$) enclose s_N, and, if yes, add the corresponding cell to the queue. To find Voronoi circles of these paws we get the data $D_{p_1} \dots D_{p_4}$ from the structures T_{b_i} associated with big cells adjacent to f_{Left} and f_{Right}, which requires $\tilde{O}(1)$ time.

2.3.2 Cell f is Small

We can look at every paw n_i of f and identify those, whose Voronoi circle encloses s_N. This requires $\tilde{O}(N^{\frac{1}{4}})$ time in total. We add to `Q` small cells adjacent to f that are incident to these paws as they need changes according to Theorem 9.

2.4 Implementing Combinatorial Changes

In this section we describe how to implement combinatorial changes in a cell f which lies in `Changed`. We again consider separately the case when f is big (Section 2.4.1) and the case when f is small (Section 2.4.2).

2.4.1 Processing Big Cells

Processing a big cell f consists of the following four steps:

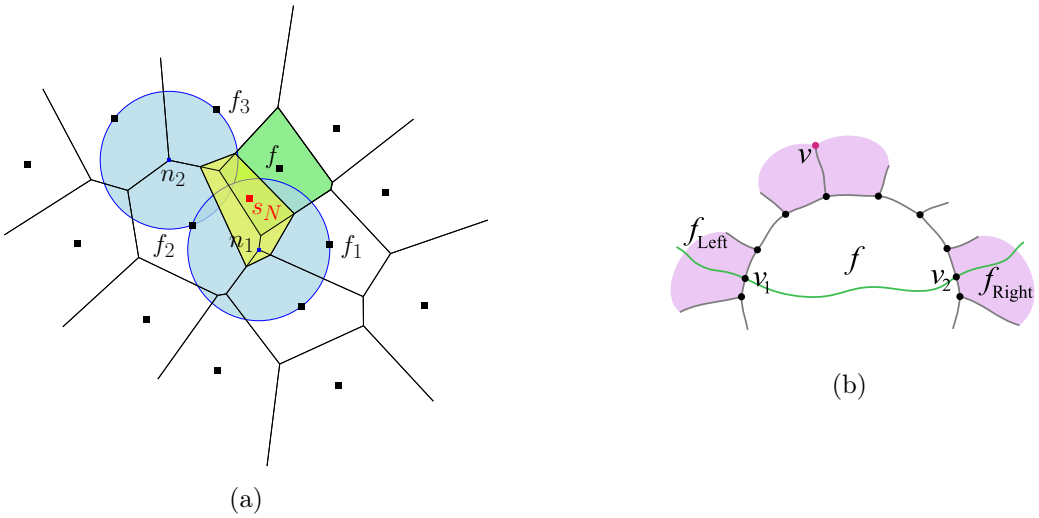


Figure 8: Identifying Voronoi cells that need changes. (a) Voronoi circle of vertex n_1 encloses s_N, and the circle of n_2 does not. (b) v is a paw of big cell f returned by a DCR. Highlighted are the cells that are to be added to the queue.

Updating the Vertices We update a continuous range of f's vertices $v_1 \dots v_k$ — we need to change their data $D_{v_1} \dots D_{v_k}$ to indicate that these vertices are now incident to the cell of s_N and not the cell of s. This can be done in $\tilde{O}(1)$ time using the yard tree T_f.

Updating the Graph of the Voronoi Diagram The first thing to do is a *link* along $\mathcal{C}_N$ creating two vertices: vertex v_1 incident to f, f_N, f_{Left}, and vertex v_2 incident to f, f_N, f_{Right} (see Figure 8b). After this *link* the part of f inside $\mathcal{C}_N$ becomes a part of the new cell f_N — luckily, all vertices of this part are already updated during the previous step.

There may be some big cells adjacent to f that are already processed, creating other parts of the new cell. We have to join these parts together by cutting edges of f that have portions inside $\mathcal{C}_N$ and are incident to already processed big cells. Finding these edges in a straightforward way could be slow as f can have a really large number of edges inside $\mathcal{C}_N$ and we do not have enough time to look at each of them individually. Luckily, graph Γ_{N-1} contains the information about edges shared by big cells. Thus we can in $\tilde{O}(1)$ time find and delete edges incident to both f and already processed big cells inside $\mathcal{C}_N$. The edges shared by f and other big cells inside $\mathcal{C}_N$ will be deleted when these other big cells will be processed.

Updating the Graph of Big Cells The two operations we just carried out — split of f by the new edge v_1v_2 and joining of some cells that are parts of f_N — can change the set of big cells and add or cut some connections between them. However, Γ_{N-1} can be updated accordingly in $\tilde{O}(1)$ time when such an operation is executed. It can be done as follows:

While undergoing a split, vertex f falls apart into two vertices: f' and $f_N^{(f)}$ (the latter represents a part of the new cell). They share newly created edge v_1v_2 of the Voronoi diagram. Note that the cells adjacent to $f_N^{(f)}$ form a continuous range of cells that were adjacent to f.

Thus we need $\tilde{O}(1)$ time to cut a continuous range from the binary search tree of cells adjacent to f, $\tilde{O}(1)$ time to add a new edge between f' and $f_N^{(f)}$ to their binary search trees and $\tilde{O}(1)$ time to re-balance the binary search tree of all big cells.

Joining can be also done in $\tilde{O}(1)$ time. When two cells f_1, f_2 are joined we remove a node

corresponding to f_1 from binary search tree of f_2 and vice versa, this takes $\tilde{O}(1)$ time. We then join the trees of f_1 and f_2 also in $\tilde{O}(1)$ time since cells that were adjacent to f_1 form a continuous range of cells adjacent to the new one.

Fixing Data Structures There are two data structures associated with f that have to be considered:

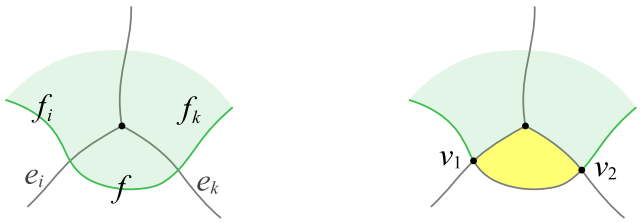
- T_f can be updated in $\tilde{O}(1)$ time the same way we did with the graph of big cells.
- DCRs of relevant paws: when big cells are joined or split, most of DCR-s stay intact. The only DCRs that need to be rebuilt are those whose range contains the endpoints of the edge that is either cut or added. Rebuilding of a DCR takes $\tilde{O}(N^{\frac{1}{4}})$ time since at most $O(N^{\frac{1}{4}})$ circles are stored there.

2.4.2 Processing Small Cells

A small cell is different from a big cell in that we can consider every edge of it, and it will take us $\tilde{O}(N^{\frac{1}{4}})$ time. We will implement the combinatorial changes in f, and after this we update the DCRs of neighboring big cells.

We can in time $\tilde{O}(N^{\frac{1}{4}})$ distinguish whether f has one, two, or more vertices inside the new cell (if they exist). Below we describe these three cases separately.

One Vertex Inside the New Cell See Figure 9a. Let f_i, f_k be the cells adjacent to f that share an edge with f inside $\mathcal{C}_N$. Let those edges be called e_i, e_k respectively.



(a) (b)

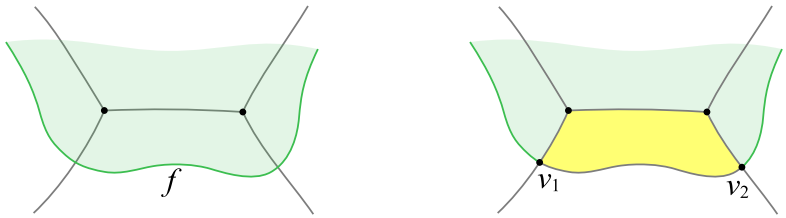
Figure 9: Processing a cell with one vertex inside the new cell

It is certain that neither f_i nor f_k have been processed yet: if f_i is processed then there would be a vertex $v = \mathcal{C}_N \cap e_i$. Then we have to create the face in the graph that is separated from f, f_i, f_k, bounded by $\mathcal{C}_N$, and is a part of the new cell f_N.

To do so, we perform a *link* operation inside f along $\mathcal{C}_N$: we create new vertices v_1 on e_i, v_2 on e_k and add an edge v_1v_2 to G_{N-1}, see Figure 9b. v_1 is incident to the cells of sites s, s_i and s_N; v_2 is incident to the cells of s, s_j and s_N.

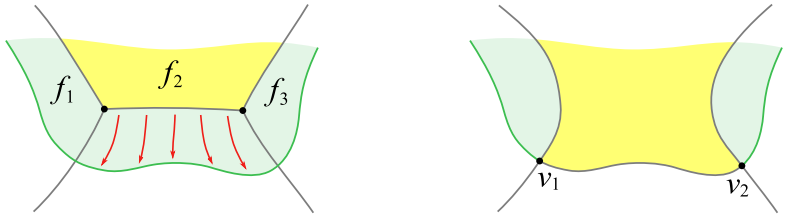
Two Vertices Inside the New Cell We check whether cells adjacent to f that share an edge with it inside $\mathcal{C}_N$ have been already processed. If not (see Figure 10a), we perform a *link* operation inside f similarly to the previous paragraph, see Figure 10b.

Otherwise let us denote three faces sharing an edge with cell f inside curve $\mathcal{C}_N$ by f_1, f_2, f_3, see Figure 11a.



(a) (b)

Figure 10: Processing a cell with two vertices inside the new cell when no surrounding cells are yet processed



(a) (b)

Figure 11: Processing a cell with two vertices inside the new cell when there are processed neighboring cells — no structural changes are needed

Lemma 10. *It is only f_2 that can have been already processed.*

Proof. Suppose f_1 is processed. It must then have an edge along $\mathcal{C}_N$. It implies that there is a vertex where $\mathcal{C}_N$ meets the common edge of f_1 and f. This vertex becomes the third vertex of f inside $\mathcal{C}_N$. However, f has only two such vertices, which is a contradiction. □

If f_2 is processed and is part of f_N then the data D_{v_1}, D_{v_2} of its vertices was updated when we were processing it. Therefore f does not need to undergo any combinatorial changes, the common edge of f and f_N can be obtained by preserving operation which was described in Section 1.1.1, see Figure 3. Thus no links and cuts are required, see Figure 11b. This completes implementing changes in f.

More Vertices Inside the New Cell Again we check whether any of the cells adjacent to f have been processed already. If not, it is enough to perform one *link* creating two new vertices v_1, v_2 and to update the data D_{v_j} of all the vertices of cell f between v_1 and v_2: now they are incident to the cell of s_N, see Figure 12a.

If some cells sharing an edge with f inside $\mathcal{C}_N$ are already processed and represent a part of new cell, then for each processed cell f' adjacent to f we also perform a *cut* removing their common edge e and then remove vertices incident to this edge that now have degree 2, see Figure 12b.

Updating the DCRs of Big Neighbors of f The last step is that for every vertex v of f whose list of adjacent cells has changed during update of G_{N-1} we find all big cells for which v is a paw (there are at most three such cells, since v has degree 3), recalculate the Voronoi circle of v, and update Voronoi circle of v in DCRs of those cells which takes $\tilde{O}(1)$ time.

(a) (b)

Figure 12: Processing a cell with three or more vertices inside the new cell. (a) No adjacent faces have been processed yet (b) Adjacent face f' has been processed

2.5 When Small Cells Become Big

When the size of a cell crosses the threshold of $N^{\frac{1}{4}}$, it can be easily identified since we store all the sizes. If a big cell b is split into a number of cells and one of them is small, or if N becomes greater than $|b|^4$, we delete all the structures associated with it, including DCRs and the structure T_f. We also remove from Γ_{N-1} the vertex corresponding to b.

The other way around, a new big cell can appear in the diagram when:

- the new cell f_N intersects many of the old cells and has more than $N^{\frac{1}{4}}$ vertices, or
- a cell f_k with $N^{\frac{1}{4}} - 1$ vertices has one vertex inside f_N and gets one additional vertex while being processed, see Figure 9.

New cell f_N inherits the portion of its DCRs from its parts that previously were parts of big cells. The number of circles of paws of previously small cells that need to be added to DCRs can be bounded from above by the size of a small cell times the number of cells that undergo changes — that is,

$$N^{\frac{1}{2}} \cdot N^{\frac{1}{4}} = N^{\frac{3}{4}}.$$

The structure T_{f_N} is inherited in part by f_N from big cells that intersect curve $\mathcal{C}_N$. The number of vertices that have to be added to T_{f_N} after that is bounded from above by the number of combinatorial changes in current insertion.

The cell f_k still has size $|f_k| \leq 2N^{\frac{1}{4}}$, so yard tree T_{f_k} can be built in $\tilde{O}(N^{\frac{1}{4}})$: it only takes time polylogarithmic in the size of the cell to add each vertex.

2.6 Correctness and Time Complexity

Theorem 11. *Inserting a new site s_N to our data structure and updating it requires $\tilde{O}(N^{\frac{3}{4}})$ amortized time.*

Proof. Let s be the number of small cells that change, and $b_1, b_2, \ldots, b_{|B|}$ be the big cells. Let ℓ_i be the number of circles returned by the DCR structures of b_i.

All the operations on a small cell take $\tilde{O}(N^{\frac{1}{4}})$ time. For all the big cells together the operations on updating the graph structure and the graph of big cells require $\tilde{O}(N^{\frac{3}{4}})$ total time. The number of DCR-s that have to be rebuilt is bounded from above by the number of changes in the graph.

Finally, the amortized time complexity is

$$\tilde{O}\left(sN^{\frac{1}{4}} + \sum_{i=1}^{|B|}\left(\left\lceil \frac{|b_i|}{N^{\frac{1}{4}}} \right\rceil + \ell_i\right) + N^{\frac{3}{4}} + sN^{\frac{1}{4}}\right).$$

Since s is $O(N^{\frac{1}{2}})$ amortized [2], $\sum_{i=1}^{|B|} |b_i| \leq N$, $|B| \leq N^{\frac{3}{4}}$, and $\sum_{i=1}^{|B|} \ell_i \leq sN^{\frac{1}{4}}$, this is simply $\tilde{O}(N^{\frac{3}{4}})$ amortized. □

The problem of maintaining the convex hull of a set of points in $\mathbb{R}^3$ subject to point insertion can also be solved using our data structure. Namely, consider the dual problem of maintaining the intersection of a set of halfspaces. The two blocks of our data structure that are specific for Voronoi diagrams, translate in this setting as follows. To find the first face affected by the insertion (or report that it does not exist) we need to find the vertex extreme in the direction normal to the plane being inserted; if it is affected, then all the three incident faces are affected. We check whether a vertex is affected by determinimg the above/below relation between this vertex and the plane baing inserted. Thus Chan's structure [12] is again enough for our needs.

2.7 Discussion and open problems

We presented the algorithm for the insertion of a site with the running time of $\tilde{O}(N^{\frac{3}{4}})$. However, this result still does not match the theoretical lower bound of $\Theta(N^{\frac{1}{2}})$. The question remains if it is possible to present an algorithm with a runtime of $\tilde{O}(N^{\frac{1}{2}})$. In the following section we show some preparations for one of possible approaches to such an algorithm.

3 Estimating the number of combinatorial changes in a Voronoi diagram with several point sites and one line

In this section, we consider a Voronoi diagram whose sites are a straight line ℓ and several points to one side from ℓ. Without loss of generality, ℓ is horizontal and all the sites are above ℓ. We also assume that the sites are *in general position:*

- no three lie on a common line,
- no four lie on a common circle,
- x-coordinates and y-coordinates of no two sites coincide.

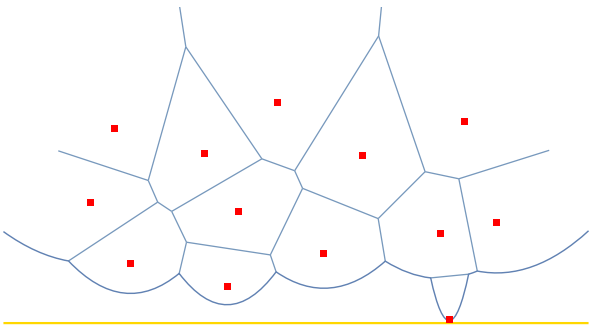


Figure 13: A Voronoi diagram for several point sites and a straight line

An example of such diagram can be seen in Figure 13. Insertion of a new point site is considered, and we want to estimate the number of combinatorial changes that happen in the diagram. We are mainly interested in the changes that concern *the beach line:*

Definition 7. *The beach line* is the border of the cell of ℓ. It is a curve comprised of parabolas whose focal points are point sites.

We know from [3] that for a Voronoi diagram of n point sites the amortized number of combinatorial changes per insertion is $O(\sqrt{n})$. We aim at proving a similar estimate for the case when one site is a line.

3.1 Voronoi diagrams and Davenport – Schinzel sequences

When dynamic Voronoi diagrams are studied, it is necessary to find the operations that can describe what happens to the Voronoi diagram when a new site is inserted. When all the sites are points, those operations are links and cuts. In this section, we describe an operation that can be used to express transformations in the beach line. The operation is called *relabel.*

We view the beach line as a sequence of sites which are the focal points of the corresponding parabolas. One site can enter this sequence several times.

Definition 8. Let s be a site of the Voronoi diagram, and s_{new} be the site that is inserted. Let I be a segment or a ray of ℓ. Then relabel(s, s_{new}, I) is replacing every occurence of s with s_{new} in the beach line above I.

A relabel can be applied for any two elements of a sequence and a segment, however in this paper we will only deal with relabels that arise from building the beach line adding parabolas one-by-one. An example of a relabel can be seen in Figure 14: letters s are replaced with letters s' inside segment I.

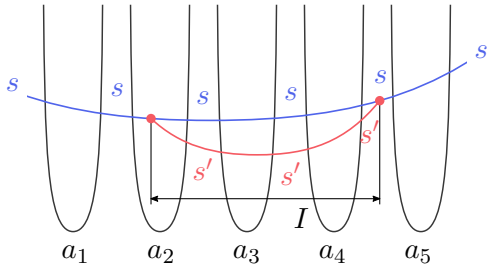


Figure 14: relabel(s, s', I)

When a point site s_{new} is inserted into a Voronoi diagram, several edges of the graph of the diagram that belonged to a cell of another site, now belong to the cell of s_{new}. However, this can be accounted for by a single link or cut. Similarly, when the parabola of s_{new} appears in several places in the beach line instead of s, this can be accounted for by a single relabel.

An example can be seen in Figure 15. Site s_{new} is inserted; old beach line is shown orange, and new beach line is shown black. Three parabolas are drawn fully: those correspond to s_1, s_2, and s_{new}. The change in the beach line of the Voronoi diagram caused by the insertion can be expressed as two relabels: relabel$(s_{\text{new}}, s_1, I_1)$ and relabel$(s_{\text{new}}, s_2, I_2)$ — the parabola of s_{new} is closer to ℓ than the parabolas of s_1 and s_2, that is why occurences of s_1 and s_2 are replaced with s_{new}. Note that the first relabel affects two occurences of s_1 in the beach line above I_1.

Let us consider the sequence of sites corresponding to the parabolas of the beach line in their respective order, denote it by S. Sequence S has one important property.

Definition 9 ([24])**.** A sequence is called a Davenport – Schinzel sequence of order 2 if it has no subsequences of the form $\dots a \dots b \dots a \dots b \dots$, and no two consecutive letters of it are equal. Davenport – Schinzel sequences of no other orders than 2 are considered in this paper, so we will further omit writing «of order 2».

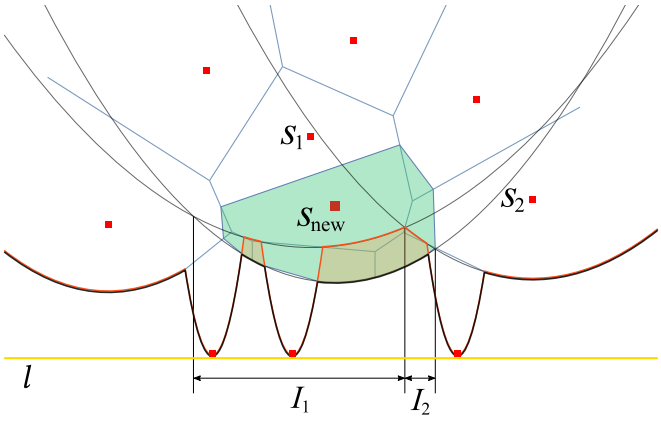


Figure 15: The change that occurs in the beach line when a new site is inserted can be expressed as two relabels

Lemma 12. *Sequence S of sites corresponding to the parabolas of the beach line is a Davenport – Schinzel sequence.*

The inverse also holds and is stated by the following Lemma.

Lemma 13 ([24])**.** *For every Davenport – Schinzel sequence S such that its first and last letters are equal, there exists such a configuration of point sites that the sequence corresponding to the beach line of the Voronoi diagram of this configuration is S.*

We will further only consider Davenport – Schinzel sequences with equal first and last letters. After all, each sequence can be made such a sequence by adding a letter ϵ to its front and to its back.

We can consider relabel as an operation on an arbitrary Davenport – Schinzel sequence, replacing each occurence of one letter with another letter inside a given segment. Note that we can construct any Davenport – Schinzel sequence using only relabels.

Definition 10. Let S be a Davenport – Schinzel sequence over the alphabet $\{s_1, \ldots, s_n\}$. Define $\text{Rel}(S)$ as the shortest sequence of relabel operations such that

1) S can be obtained by applying the relabels in $\text{Rel}(S)$ consecutively to the sequence s_1 of length 1,

2) if $i < j$, all relabels with s_i as the second argument occur in $\text{Rel}(S)$ before all relabels with s_j as the second argument.

Definition 11. Let S be a Davenport – Schinzel sequence over the alphabet $\{s_1, \ldots, s_n\}$. $\text{Rel}_j(S)$ is a subsequence of $\text{Rel}(S)$ where the second argument is equal to s_j. S_j is the Davenport – Schinzel sequence obtained by applying $\text{Rel}_2(S)$, …, $\text{Rel}_j(S)$ consequtively to the sequence s_1 of length one.

Lemma 14. *For each Davenport – Schinzel sequence S there exists* $\text{Rel}(S)$.

Proof. Consider the configuration of sites and parabolas corresponding to S. Such a configuration exists by Lemma 13. Add the parabolas of this configuration to it one-by-one and note that every addition of a parabola can be expressed as a set of relabels. All the properties that have to hold for $\text{Rel}(S)$ also hold for the sequence we constructed. □

We will be proving the following Theorem:

Theorem 15. *For any Davenport – Schinzel sequence S over the alphabet $\{s_1, \ldots, s_n\}$ the length of* $\mathrm{Rel}(S)$ *is* $O(n\sqrt{n})$.

This Theorem means that the average number of relabels per addition of a letter is $O(\sqrt{n})$.

3.2 Tree representation of a Davenport – Schinzel sequence

Over the history of the studies of Davenport – Schinzel sequences, several ways to connect them to trees and other data structures were suggested [21, 23]. In this thesis, we consider another natural yet new approach.

Consider two letters a, b in a Davenport – Schinzel sequence S. Since no alternations of the form $\ldots a \ldots b \ldots a \ldots b \ldots$ are allowed, two cases are possible:

1) all occurences of a are located in S before all occurences of b (or vice versa),

2) all occurences of b are located in S between two consecutive occurences of a (or vice versa).

All the other configurations yield at least one anternation of which there can be none in a Davenport – Schinzel sequence.

Definition 12. We say that letter a of a Davenport – Schinzel sequence S *surrounds* letter b if all occurences of b are located in S between two consecutive occurences of a.

Lemma 16. *There exist no letters $a_1, \ldots a_m$ in a Davenport – Schinzel sequence S such that*

- a_1 *surrounds* a_2,
- a_2 *surrounds* a_3,

  $\ldots$
- a_{m-1} *surrounds* a_m,
- a_m *surrounds* a_1.

Proof. The condition above would mean that all occurences of a_1 are located between two consecutive occurences of a_m, and all occurences of a_m are located between two consecutive occurences of a_1. We arrive at a contradiction. □

Let us connect by an oriented edge each letter to each letter it surrounds. Lemma 16 means that the graph we obtain is acyclic. That means we can perform a topological sort on it and find direct descendants of each vertex a: those surrounded by a but not surrounded by any other vertex surrounded by a.

What we obtain by connecting each vertex to each its direct descendant is a tree. An example of a tree corresponding to a Davenport – Schinzel sequence and a configuration of sites and parabolas corresponding to it can be seen in Figure 16.

The correspondence between Davenport – Schinzel sequence and trees defined this way is not one-to-one: the same tree corresponds to the sequences $axaya$ and $axya$. However, this can be fixed by adding walls inside nodes, and in this thesis we will be using the same notation for letters in the sequence and nodes in the tree, applying notions such as «number of occurences» and «number a children» to a single name. Define the size of a vertex in a tree:

Definition 13. For letter a in a Davenport – Schinzel sequence S we denote by $\mathrm{size}(a)$ the number of children of a in the tree corresponding to S plus one.

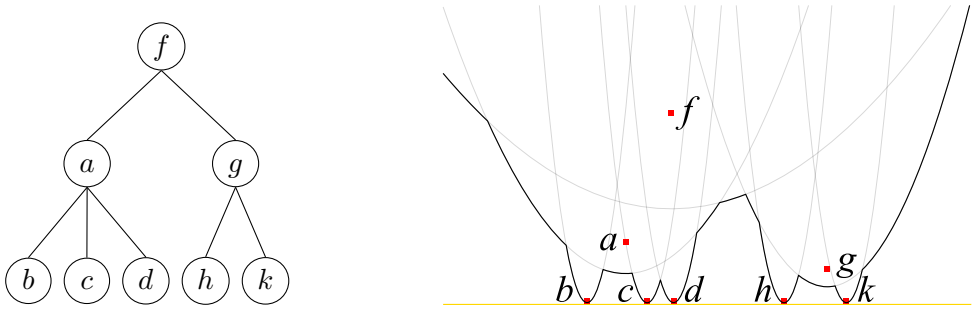


Figure 16: (a) A tree and (b) a configuration of sites and parabolas that correspond to a Davenport – Schinzel sequence $fabacdafghgkgf$

Lemma 17. *For a Davenport – Schinzel sequence S over the alphabet $\{s_1, \ldots, s_n\}$,*

$$2n - 1 \;\geq\; \sum_{i=1}^{n} \text{size}(s_i) \;\geq\; \text{length}(S).$$

Proof. To prove the first inequality, let us give one coin to the root of the tree, and two coins to every other letter. In total we have given at most $2n - 1$ coins. Let each letter pass one of its coins to its parent in the tree. Now each letter has one coin given to it initially, and one coin from each of its children. This makes the total number of coins be equal to $\sum \text{size}(s_i)$.

For the second inequality, note that the number of occurences of a letter is at most the number of its children plus one, since there can not be two equal letters in a row, and two occurences of a letter a must be separated by at least one child of a. □

3.3 The order of relabels

Now that we have connected Davenport – Schinzel sequences to trees, we will consider changes of the tree corresponding to a sequence when a relabel is applied to it. This will help us visualize the operations, define and estimate the potential function, and better illustrate the changes in a sequence.

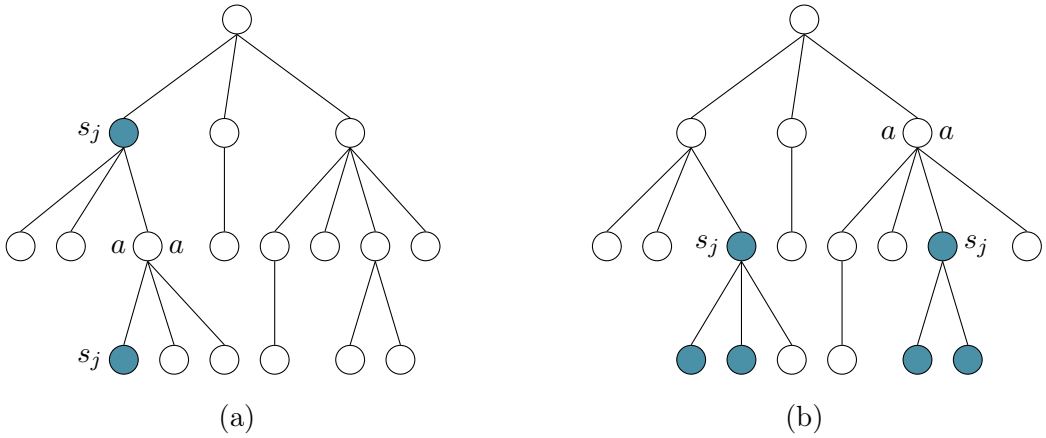


Figure 17: (a) Nodes that are relabelled must form a contiguous subtree, (b) The roots of such subtrees must be children of a single vertex

Lemma 18. *The set of letters s_i such that* $\text{relabel}(s_i, s_j, \cdot) \in \text{Rel}_j(S)$ *is a union of several contiguous subtrees whose roots are children of a single vertex R_j in the tree corresponding to S_{j-1}.*

Proof. Consider two vertices that are relabelled. If one of them is an indirect descendant of the other, then the vertices between them must also be relabelled. Otherwise there is a non-relabelled vertex that surrounds the lower relabelled vertex. This yields an alternation $s_j a s_j a$ in S_j, see Figure 17a. Thus, the vertices that are relabelled form contiguous subtrees.

If two roots of such subtrees are not children of a single vertex, there is a non-relabelled ancestor of one of them that is not an ancestor of the other, call it a. This yields an alternation $s_j a s_j a$ in S_j, see Figure 17b. This is clearly a contradiction. □

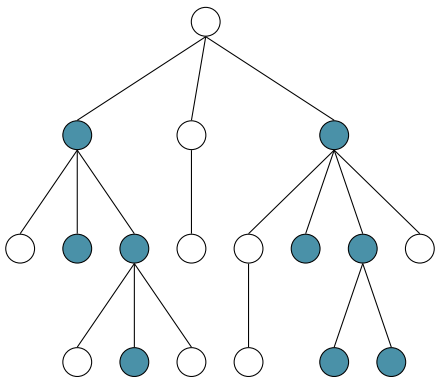


Figure 18: A possible set of vertices that are relabelled

An example of a possible set of vertices that can be relabeled in $\text{Rel}_j(S)$ (i. e. that satisfies the condition of Lemma 18) can be seen in Figure 18.

Without loss of generality, we assume that relabels in $\text{Rel}_j(S)$ are applied from top of the tree down, level by level, from left to right. This order of relabels helps us make sure we avoid any alternations on intermediate steps when not all relabels in $\text{Rel}_j(S)$ are yet applied, since at any step the set of vertices that have been relabelled satisfies the condition of Lemma 18. Also this order is convenient for our proof of the $n\sqrt{n}$ bound.

Lemma 19. *When relabels in* $\text{Rel}_j(S)$ *are applied, only two nodes of the tree can increase in size: s_j and R_j.*

Proof. When relabels are applied, some occurences of some letters are replaced with s_j. Suppose a child b of a vertex a is not a child of a anymore. It means a relabel has been applied to a or another descendant of a.

If the relabel is applied to a, then b can find itself between two occurences of s_j or between an s_j and an a. In the first case b is now a child of s_j, and in the second case it is a child of R_j, since it is neither surrounded by s_j nor by a, and R_j is the parent of s_j.

If the relabel is not applied to a, than the only other letter that can happen to surround b is s_j: a was the closest to surround b, and s_j is the only letter of which there can be new occurences. □

3.4 Inserts and length of a Davenport – Schinzel sequence

One would want to say that when $\text{relabel}(s_i, s_j, I)$ is applied, $\text{size}(s_i)$ decreases: s_j either steals some children from s_i or wholly replaces the only occurence of s_i. However, there is one case when a relabel does not reduce $\text{size}(s_i)$.

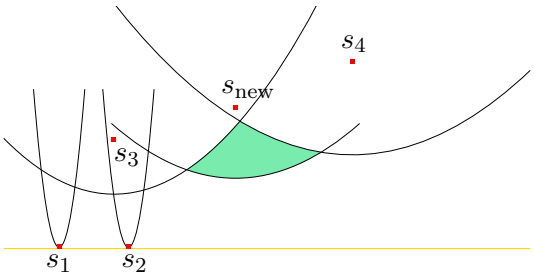


Figure 19: An example of an insert. A new occurence of s_{new} appears, yet size(s_3) does not decrease. This can be expressed as a relabel of s_3 by s_{new}.

Definition 14. An insert is a relabel that does not reduce the size of the node it is applied to.

An example of an insert can be seen in Figure 19. We will denote by Insert_i and Relabel_i the numbers of relabels in Rel_i that are and are not inserts, respectively. We can not use the argument that inserts make sizes of many distinct nodes decrease, that is why we will try to take the length of the Davenport – Schinzel sequence into consideration. For this, we will describe what happens to the length of a sequence and to the sizes of nodes in the tree corresponding to it when several inserts and relabels are applied.

Lemma 20. *1) When an insert is applied to a Davenport – Schinzel sequence, the length of the sequence increases by 1.*

2) When a relabel that is not an insert is applied to a Davenport – Schinzel sequence, the length of the sequence may decrease, but by at most 2.

Proof of 1). If relabel(s_i, s_j, I) is an insert, it can in fact be expressed as an addition of one occurence of s_j to the Davenport – Schinzel sequence. Note that s_j can only appear once inside I, since otherwise it would steal children from s_i inside I. Furthermore, s_j can not replace a single whole occurence of s_i, since the children of s_i that are incident to that occurence will become children of s_j.

Proof of 2). If the length of the sequence decreases when relabel(s_i, s_j, I) is applied, it means that some occurences of s_j that appear as a result of the relabel are adjacent to some existing occurences of s_j. Since we are performing relabels from the top of the tree down, this can only happen to the two extreme occurences of s_i if they are being relabelled: they should be adjacent to some occurences of s_j, since s_j can either surround or have a common parent with s_i at the moment of relabel(s_i, s_j, I). □

Corollary 21. *The increase in length of the Davenport – Schinzel sequence after all the operations from* $\text{Rel}_j(S)$ *is applied is at least* $\text{Insert}_i - 2 \cdot \text{Relabel}_i$.

3.5 Proof of amortized bound

Proof of Theorem 15. Define the potential function for a Davenport – Schinzel sequence S over the alphabet $\{s_1, \ldots, s_n\}$ as

$$\Phi = 3 \cdot \sum_{i=1}^{n} \min\left\{\text{size}(s_i), 2\sqrt{n}\right\} - \text{length}(S).$$

We are going to estimate $|\mathrm{Rel}_j(S)| + \Phi_j - \Phi_{j-1}$. We denote by $\mathrm{size}_j(s_i)$ the size of the vertex s_i in the tree corresponding to the sequence S_j.

$$\begin{aligned}|\mathrm{Rel}_j(S)| + \Phi_j - \Phi_{j-1} \;=\; & \mathrm{Insert}_j + \mathrm{Relabel}_j + \\ & + \; 3 \cdot \sum_{i=1}^{n} \min\left\{\mathrm{size}_j(s_i), 2\sqrt{n}\right\} \; - \; 3 \cdot \sum_{i=1}^{n} \min\left\{\mathrm{size}_{j-1}(s_i), 2\sqrt{n}\right\} \; - \qquad (*) \\ & - \; (\mathrm{length}(S_j) \; - \; \mathrm{length}(S_{j-1})) \qquad (**)\end{aligned}$$

By Lemma 19 we can estimate $(*)$ from above by $2\sqrt{n} - (\mathrm{Relabel}_j - \sqrt{n})$: only s_j and R_j can increase in size, and all the relabels that are not inserts reduce the size of the vertex that they are applied to (which are distinct for all distinct inserts). However, at most $\sqrt{n}$ relabels applied to the vertices whose size is at least $2\sqrt{n}$ (due to Lemma 17, there can only be $\sqrt{n}$ such vertices) do not reduce the value of the potential.

By Corollary 21, we can estimate $(**)$ from above by $-\mathrm{Insert}_j + 2\,\mathrm{Relabel}_j$. The final estimate for the whole sum is

$$\begin{aligned}|\mathrm{Rel}_j(S)| + \Phi_j - \Phi_{j-1} \;\leq\; & \mathrm{Insert}_j \; + \; \mathrm{Relabel}_j + \\ & + \; 9\sqrt{n} \; - \; 3\,\mathrm{Relabel}_j \; - \\ & - \; \mathrm{Insert}_j \; + 2\,\mathrm{Relabel}_j \; = \\ = \; & 9\sqrt{n}.\end{aligned}$$

$\Phi_n - \Phi_1$ is bounded by $3n$, and all the other values of the potential cancel out. This means that $|\mathrm{Rel}(S)| = O(n\sqrt{n})$. □

3.6 Discussion and open problems

After the upper bound of $O(\sqrt{n})$ is shown, a natural question is if there exists a matching lower bound: is there actually a Davenport – Schinzel sequence the construction of which requires $\Omega(n\sqrt{n})$ relabels. Finsing such an example or proving a tghter upper bound is an interesting quest.

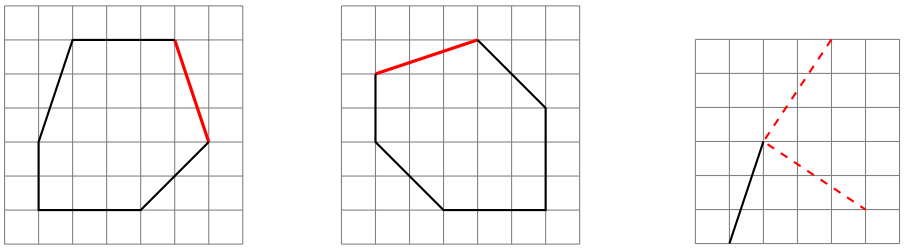
4 Bounds on the number of egde-to-edge gluings of squares

In this section, we prove that the number of edge-to-edge gluings of n squares is polynomial in n. This result allows to develop a polynomial algorithm to list all the gluings.

Theorem 22. *There are $O\left(n^{36}\right)$ edge-to-edge gluings of at most n squares that correspond to convex polyhedra.*

Proof. Triangulate the polyhedron corresponding to the gluing and draw its faces on the square grid. By Gauss—Bonnet theorem, the polyhedron has no more than 8 vertices, and thus at most 18 edges. An edge shared by two faces must have the same lengths of x- and y-projections on the drawings of these faces, see Figures 20a, 20b.

Count the number of sets of triangles satisfying this restriction and taking up at most n squares. For each edge, we can pick lengths of its x- and y-projections. Since the edge is a part of a flat face, all the squares that intersect the edge are distinct. There is at most n of them, which yields that both projections are at most n, so there is at most n^2 ways to choose the edge.



(a) (b) (c)

Figure 20: (a), (b) Highlighted edge has the same lengths of projections on the drawings of two faces. (c) Two ways to place an edge with given projections that preserve convexity of the face.

Once the projections of the edges are known, let us draw the faces on the grid. At every vertex, there is at most two ways to place the next edge, such that the convexity of the face is preserved, those differ by $\frac{\pi}{2}$, see Figure 20c for an example. This adds at most $2^{2 \cdot 18}$ ways to draw the faces once the edges are known, which gives the total of at most $(2n)^{36}$ gluings. □

Theorem 23. *There are $\Omega\left(n^3\right)$ edge-to-edge gluings of at most n squares that correspond to convex polyhedra.*

Proof. To prove the theorem, we construct a series of such gluings. These gluings correspond to doubly-covered octagons, the octagons being obtained by cutting edges of a rectangle with sides no longer than $\frac{\sqrt{n}}{2}$, one at least twice as long as the other, see Figure 21.

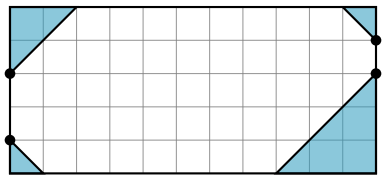


Figure 21: An example of an octagon produced by cutting angles of a rectangle

Pick width and height of the rectangle. Denote width by a, pick it so that $0 < a \leq \frac{1}{2}\sqrt{n}$. Denote height by b, pick it so that $0 < b \leq \frac{a}{2}$. An octagon is defined by its vertices on the vertical edges of the rectangle (those are highlighted in Figure 21). There are $\binom{b}{2}^2 \sim \frac{1}{4}b^4$ ways to choose these vertices.

To sum up, the number of octagons obtained by cutting edges of a rectangle is

$$\Omega\left(\sum_{a=1}^{\frac{\sqrt{n}}{2}}\sum_{b=1}^{\frac{a}{2}} b^4\right) = \Omega\left(\sum_{a=1}^{\frac{\sqrt{n}}{2}} a^5\right) = \Omega\left(n^3\right).$$

□

Theorem 24. *The bound of Theorem 23 is tight: there are $O(n^3)$ doubly covered convex polygons that can be glued from n squares.*

Proof. Edges of a doubly covered polygon glued from squares can only have four directions: vertical, horizontal, or inclined by $\frac{\pi}{2}$ or $\frac{3\pi}{2}$. Thus any doubly covered polygon glued from squares is an octagon cut from a rectangle. Some edges of the octagon, however, may have zero length.

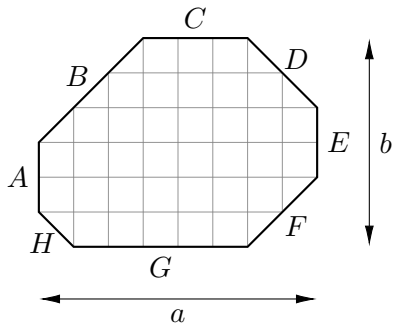


Figure 22: An octagon cut from rectangle and its dimensions

Let us denote the numbers of squares traversed by the edges by $A, \ldots, H$, as shown in Figure 22. The width of the circumscribed rectangle is denoted by a, and the height of the circumscribed rectangle is denoted by b. We can think that $a \leq b$. Note that

$$\begin{cases} A + B + H = D + E + F = b \\ B + C + D = F + G + H = a \\ b \leq a \\ a \cdot b \leq \frac{n}{2} \end{cases}$$

Six variables out of these ten: a, A, B, D, F, H — define other four. This means, given values of a, A, B, D, F, H, the octagon is either defined uniquely or non-existent:

$$\begin{aligned} b &= A + B + H \\ E &= A + B + H - D - F \\ C &= a - B - D \\ G &= a - H - F \end{aligned}$$

Let us now count the ways to pick a, A, B, D, F, H.

First case $1 \leq b \leq a \leq \sqrt{n}$. In this case all the variables $A, \ldots, H$ are at most $\sqrt{n}$, which yields the number of ways to pick six variables does not exceed $(\sqrt{n})^6 = n^3$.

Second case $a \geq \sqrt{n}$, $b \leq \frac{n}{a}$. In this case A, B, D, F, H are at most $\frac{n}{a}$ since they all contribute to the height of the octagon. The number of ways to pick six variables is hence equal to

$$\sum_{a=\sqrt{n}}^{n} \left(\frac{n}{a}\right)^5.$$

We now split and estimate this sum. Assume n is a power of 2 or consider the closest to n power of 2 from above.

$$\sum_{a=\sqrt{n}}^{n} \left(\frac{n}{a}\right)^5 \leq \sum_{i=\frac{\log n}{2}}^{\log n} \sum_{a=2^{i-1}}^{2^i} \left(\frac{n}{a}\right)^5 \leq$$

$$\leq \sum_{i=\frac{\log n}{2}}^{\log n} \left(\frac{n}{2^{i-1}}\right)^5 \cdot 2^i = \sum_{i=\frac{\log n}{2}}^{\log n} 2n \cdot \left(\frac{n}{2^{i-1}}\right)^4 =$$

$$= \sum_{i=0}^{\frac{\log n}{2}} 2n \cdot \left(\frac{\sqrt{n}}{2^{i-1}}\right)^4 = 2n^3 \cdot \sum_{i=0}^{\frac{\log n}{2}} \frac{16}{2^{4i}} = O(n^3).$$

□

We implemented an algorithm that enumerates all the gluings of at most n squares for a given graph structure of a convex polyhedron. It showed that one gluing can admit several ways to cut itself into flat polygons, see Figure 23. Thus it can appear in the list several times.

(a) (b)

Figure 23: Doubly covered parallelogram can be cut into two flat quadrilaterals in two ways, the latter consisting of its faces

4.1 Algorithm to classify edge-to-edge gluings of squares

The algorithm consists of the following steps:

1) Generate the list of all edge-to-edge gluings of at most n squares, denote it $L(n)$. Due to Theorem 22, this step takes polynomial time.

2) For each gluing in $L(n)$, generate matrix of pairwise distances between its vertices. Due to Theorem 8, this step takes $O(n^3)$ time per gluing.

3) Unicalize the list of matrices up to homothety and permutation of rows and columns, leave only corresponding elements of $L(n)$. Since the matrices are of at most 8 rows and 8 columns, it takes polynomial time to remove duplicates from the list.

The output of this algorithm is the list of all non-isomorphic edge-to-edge gluings of at most n squares.

4.2 Discussion and open problems

The cornerstone of the technique we have been using is the possibility to draw a face of a polyhedron glued from squares on a planar grid. It allows us to estimate the number of valid gluings. The same technique can seemingly be applied for the cases of regular hexagons and triangles, since these polygons also tile the plane.

References

[1] Alexandr Alexandrov. *Convex Polyhedra*. Springer-Verlag, Berlin, 2005.

[2] Sarah R. Allen, Luis Barba, John Iacono, and Stefan Langerman. Incremental voronoi diagrams. *Discrete & Computational Geometry*, 58(4):822–848, 2017.

[3] Sarah R. Allen, Luis Barba, John Iacono, and Stefan Langerman. Incremental voronoi diagrams. *Discrete & Computational Geometry*, 58(4):822–848, 2017.

[4] Elena Arseneva, John Iacono, Greg Koumoutsos, Stefan Langerman, and Boris Zolotov. Sublinear Explicit Incremental Planar Voronoi Diagrams. In *The 22-nd Japan Conference on Discrete and Computational Geometry, Graphs, and Games*, pages 33–34, September 2019.

[5] Elena Arseneva, John Iacono, Grigorios Koumoutsos, Stefan Langerman, and Boris Zolotov. Sublinear Explicit Incremental Planar Voronoi Diagrams. *Journal of Information Processing*, 28:766–774, 2020.

[6] Elena Arseneva and Stefan Langerman. Which Convex Polyhedra Can Be Made by Gluing Regular Hexagons? *Graphs and Combinatorics*, page 1–7, 2019.

[7] Elena Arseneva, Stefan Langerman, and Boris Zolotov. A Complete List of All Convex Polyhedra Made by Gluing Regular Pentagons. In *The 22-nd Japan Conference on Discrete and Computational Geometry, Graphs, and Games*, pages 33–34, September 2019.

[8] Elena Arseneva, Stefan Langerman, and Boris Zolotov. A Complete List of All Convex Polyhedra Made by Gluing Regular Pentagons. *Journal of Information Processing*, 28:791–799, 2020.

[9] Franz Aurenhammer and Rolf Klein. Voronoi diagrams. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 201–290. North Holland / Elsevier, 2000.

[10] Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams And Delaunay Triangulations*. World Scientific Publishing Co. Pte. Ltd., 2013.

[11] Timothy M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. *J. ACM*, 57(3):16:1–16:15, 2010.

[12] Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. In *35th International Symposium on Computational Geometry, SoCG 2019*, pages 24:1–24:13, 2019.

[13] Jindong Chen and Yijie Han. Shortest paths on a polyhedron. In *6-th annual symposium on Computational geometry*, page 360–369, Berkley, California, USA, June 1990. SCG '90.

[14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. Massachusetts Institute of Technology, third edition, 2009.

[15] Erik Demaine, Martin Demaine, Anna Lubiw, and Joseph O'Rourke. Enumerating foldings and unfoldings between polygons and polytopes. *Graphs and Combinatorics*, 18(1):93–104, 2002.

[16] Erik Demaine, Martin Demaine, Anna Lubiw, Joseph O'Rourke, and Irena Pashchenko. Metamorphosis of the cube. In *Proc. SOCG*, pages 409–410. ACM, 1999.

[17] Erik Demaine and Joseph O'Rourke. *Geometric folding algorithms*. Cambridge University Press, 2007.

[18] David Eppstein, Michael J Bannister, William E Devanny, and Michael T Goodrich. The Galois complexity of graph drawing: Why numerical solutions are ubiquitous for force-directed, spectral, and circle packing drawings. In *International Symposium on Graph Drawing*, pages 149–161. Springer, 2014.

[19] Daniel M Kane, Gregory N Price, and Erik D Demaine. A Pseudopolynomial Algorithm for Alexandrov's Theorem. In *WADS*, pages 435–446. Springer, 2009.

[20] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 2495–2504, 2017.

[21] Martin Klazar. Combinatorial aspects of Davenport – Schinzel sequences. *Discrete Mathematics*, 165/166:431–445, 1997.

[22] Stefan Langerman, Nicolas Potvin, and Boris Zolotov. Enumerating All Convex Polyhedra Glued from Squares in Polynomial Time. In *CG Week Young Researchers Forum 2021*, pages 61–64, June 2021. Based on the project prepared during an exchange semester at ULB.

[23] Seth Pettie. Splay Trees, Davenport – Schinzel Sequences, and the Deque Conjecture. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 08 2007.

[24] Micha Sharir and Pankaj K. Agarwal. *Davenport – Schinzel sequences and their geometric applications*. Cambridge Academic, 1995.

[25] Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

[26] Boris Zolotov. Algorithmic Aspects of Alexandrov's Uniqueness Theorem. *Bachelor's thesis at the Faculty of Mathematics and Computer Sciences, SPBU*, pages 1–31, 2019.