UTRECHT UNIVERSITY

MASTER THESIS
MSC COMPUTING SCIENCE

Where Do They Go?

A Geometric Seed Dispersal Model

Author:
C.P. (Tamara) Florijn
Student number:
5856442

Supervisors:
Frank Staals
Sarita de Berg
Second corrector:
Marc van Kreveld

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Faculty of Science

July 1, 2022

Abstract

C.P. (Tamara) Florijn

Where Do They Go?
A Geometric Seed Dispersal Model

Climate change forces plant species to migrate to other areas via seed dispersal, the process of seeds moving away from the parent plant. The objective of this study is to find a good geometric model of representation that allows us to answer the question: where do the seeds go?

The landscape is represented using a simple polygon with n vertices, and the initial source plants are modelled as a set of m sites. To model the influence of wind, we use a convex distance function based on a polygon with r vertices.

To answer queries such as 'What region is covered in plants at a given time?', we use a Voronoi diagram. We prove fundamental properties and design novel algorithms to compute geodesic Voronoi diagrams under a convex distance function. We prove a time complexity of $O\big(((n+m)\cdot r)\log((n+m)\cdot r)\log n\big)$. Our adaptation called the "lazy approach" improves that bound to $O\big((n+m\cdot r)\log(n+m\cdot r)\log n\big)$.

Keywords: Climate change, plant migration, seed dispersal, theoretical computer science, geometric algorithms, geodesic Voronoi diagrams, convex distance function

Contents

1 Introduction **1**
1.1 Ecological relevance . . . 1
1.2 Research questions . . . 2

2 Ecological background **4**
2.1 Overview . . . 4
2.1.1 Modelling spreading plants . . . 4
2.1.2 Habitat connectivity . . . 5
2.1.3 Other spreading models . . . 5
2.2 DIMO Model . . . 5

3 Geometric background **8**
3.1 Distance measures . . . 8
3.1.1 Convex distance function . . . 9
3.2 Shortest paths . . . 9
3.2.1 Geodesic distance . . . 9
3.2.2 Continuous Dijkstra . . . 10
3.3 Voronoi diagram . . . 11
3.3.1 Voronoi diagram under a convex distance function . . . 11
3.3.2 Abstract Voronoi diagram . . . 13
3.4 Geodesic Voronoi diagram . . . 13
3.4.1 Geodesic Voronoi diagram algorithm by Aronov . . . 14
3.4.2 Geodesic Voronoi diagram algorithm by Papadopoulou . . . 16
3.4.3 Optimal geodesic Voronoi diagram algorithm . . . 17
3.5 Minkowski sum . . . 17

4 Formalisation **19**
4.1 Basic problem based on DIMO . . . 19
4.2 Convex distance function . . . 21

5 Properties of geodesic cd-Voronoi diagrams **23**
5.1 Preliminaries . . . 23
5.2 Star-shaped Voronoi cell . . . 24
5.3 Additively weighted bisector . . . 26
5.4 Output complexity geodesic cd-bisector . . . 29
5.5 Geodesic cd-Voronoi diagram . . . 34

6 Algorithms **36**
6.1 Augmented cd-Voronoi diagram . . . 36
6.2 Additively weighted cd-bisector . . . 37
6.3 Geodesic cd-Voronoi diagram . . . 39
6.3.1 Lazy approach cd-Voronoi diagram . . . 44

7 Queries **49**
7.1 Distance to closest site 49
7.2 All points at given distance 49
7.3 All points at multiple distances 53
7.4 Queries under Euclidean distance measure 54

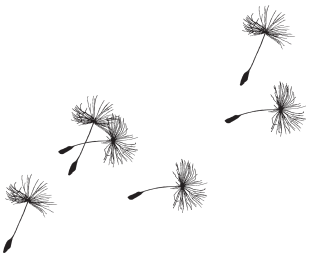
8 Extensions **55**
8.1 Germination delay 55
8.2 Habitat suitability 57
8.3 Obstacles 58
8.4 Anisotropic regions 58

9 Conclusion **60**
9.1 Future work 60

Bibliography **62**

A Research orientation **66**
A.1 Wieger Wamelink 66
A.2 Laurens Sparrius 67

Chapter 1



Introduction

Climate change forces plant species to migrate to other regions to survive, when their tolerance for environmental conditions such as temperature rise, droughts, and habitat fragmentation is exceeded. (Ozinga et al., 2009; Corlett, 2013). To shift their range, plants do not walk around but need to spread their seeds. This process is called *seed dispersal*. Seed dispersal is essential for all plant species to survive climate change (Thuiller et al., 2008). We will discuss this further in Section 1.1.

The main objective of this thesis is to design an appropriate geometric model to represent seed dispersal and to calculate where seeds disperse to in a given environment over a period of time. This includes designing and analyzing algorithms to solve geometric and ecological challenges. We will discuss our research questions in detail in Section 1.2.

As an orientation in the ecological field, we spoke to ecologist Wiemer Wamelink from Wageningen University & Research and ecologist Laurens Sparrius from the Dutch research institute FLORON, see Appendix A. Wamelink and his colleagues developed a seed dispersal model called DIMO (Wamelink et al., 2014). DIMO formed the basis of our geometrical approach to seed dispersal.

1.1 Ecological relevance

Because of climate change, the region of the earth with suitable climatic variables for a plant species shifts: Species either adapt, move or die. (Ozinga et al., 2009). To protect biodiversity, it is relevant to track and predict this process over time. Seed dispersal models provide deeper insight in plant spreading and range expansion. They predict the chances to survive climate change. This allows us to mitigate the damage done by climate change. The gained knowledge can be used to either i) prevent plant migration or ii) stimulate plant migration.

Invasive species, such as the Japanese knotweed, can cause serious issues. Generic models of seed dispersion can help focus the management and manipulation of these invasive species (Gosper et al., 2005; Büyüktahtakın and Haight, 2018). For example, well-placed barrier zones can stop the spreading (Gosper et al., 2005).

On the other hand, to maintain a diversity of plant species, stimulation of plant growth is more relevant than ever because of climate change, deforestation and other limitations of habitat. Plant species have limited migration speed that differs per species. Modelling seed dispersal can give more insight in the status of endangered species that cannot keep up with the pace: the tolerance for temperature changes is limited. As an example, McGuire et al. (2016) evaluated the connectivity of the US

landscape and calculated the effect that corridors would have on the plant movement capacity.

However, the application of dispersal models on ecologically relevant scales is currently problematic. As dispersal is a continuous process, it can be modelled continuously, using for example a single-parameter Pareto-distribution fitted to dispersal data (Treep et al., 2021) . However, current ecological models use grid-based landscape representations. This becomes computationally infeasible at the desired scale.

1.2 Research questions

As we mentioned before, we spoke to ecologist Wieger Wamelink from Wageningen University & Research. Wamelink developed a seed dispersal model called DIMO, short for DIspersal MOdel. DIMO simulates the dispersal of plants through a grid map over a period of time. The user can model barriers such as roads to indicate where seeds cannot disperse to. A more elaborate explanation on DIMO can be found in Section 2.2.

In our conversation with Wamelink, he elaborated on some of the challenges they faced with the construction and calculation of their ecological model DIMO. The recurrent problem was the long computation time.

Inspired by this conversation and directed by our literature review, we formulate our research questions. We focus on a geometrical representation instead of representations such as a graph or a grid model, because the problem is continuous in its essence. In this way, we can use the geometry and techniques from computational geometry to solve the problem. Our tasks are to:

1. Design a suitable geometrical model for seed dispersal on the basis of the ecological model DIMO.
2. Design algorithms to perform queries such as:
    a) At what time is a certain point covered in plants?
    b) What is the region that a plant occurs in after a given period of time?
3. What is the complexity of the output maps of the designed algorithms? And what is the time complexity of the algorithms? The complexity will be analyzed in terms of the size of the input, such as the number of input plants, the complexity of the vegetation maps and the requested number of time rounds.

We represent the environment as a simple polygon W with n vertices, and model the initial plant sources as a set P of m sites. To model the influence of wind, we use a convex distance function R based on a convex polygon with r vertices.

In Figure 1.1, we can see an example how plants spread in W under a convex distance function from time steps $t = 1$ to $t = 3$. We observe that the plant region that spreads from site p_3 has no overlap with the plant regions from other sites. On the other hand, we see points that are reached by both sites p_1 and p_2, at equal or different time steps. In these and more situations, we can use the information of a Voronoi diagram. A Voronoi diagram is a subdivision that denotes what region is closest to what site. For example, if we know that a point is closest to site p_3 and that p_3 does not reach the point, we know no other site will.

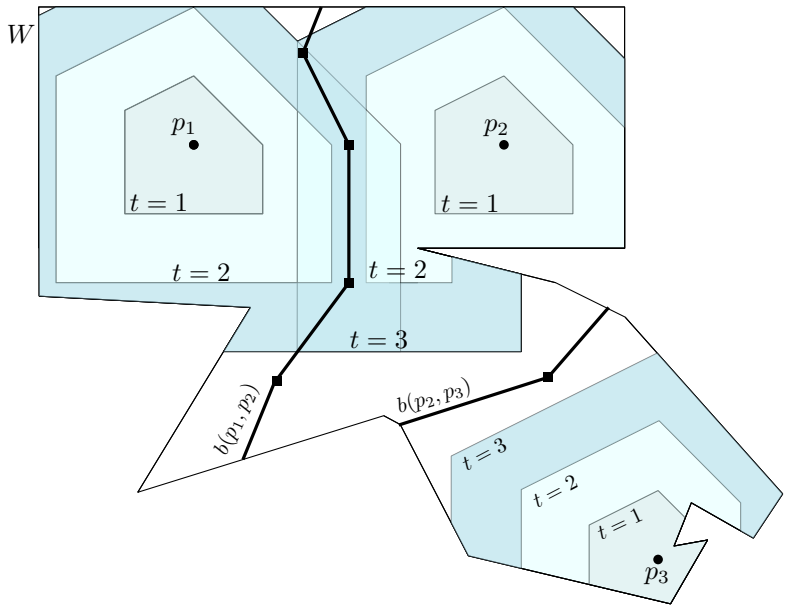


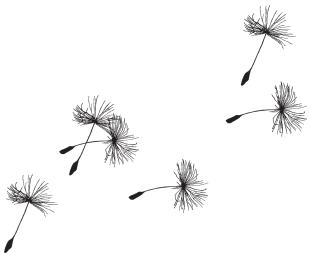
FIGURE 1.1: In this figure, we show an example how plants spread in polygon W under a convex distance function. Sites p_1, p_2 and p_3 are the initial source plants. The polygon with center p_1 in the form of an asymmetric house is the convex distance function. The boundaries of the regions reached by some site are indicated for time steps $t = 1, t = 2$ and $t = 3$. The bold polygonal line segments are indications of the bisectors $b(p_1, p_2)$ and $b(p_2, p_3)$.

Even though the time steps that divide the region are discrete, we will first solve the problem as if it were continuous by using a Voronoi diagram. Because we calculate the distances within a polygon, we use geodesic distances. We will use the term *geodesic cd-Voronoi diagram* to denote a geodesic Voronoi diagram under a convex distance function. After solving the problem using a continuous approach, we will discretise it again when we answer the queries.

Our approach will provide the ecological research community with a different way to model seed dispersal, allowing them to use geometrical techniques for a more efficient solution. The novel geometric methods contribute to the foundation of computational geometry to solve problems that could not be solved before.

In Chapter 2, we will elaborate on a collection of ecological models, and give a detailed description of the DIMO model. Chapter 3 provides an overview of the geometrical problems and techniques researched before. In Chapter 4, we formalise the ecological model into a geometrical model, and prove the output complexity. Since we solve the problem continuously and use a Voronoi diagram, Chapter 5 and 6 provide the fundamental proofs, the output complexity and algorithms to compute the geodesic cd-Voronoi diagram of a simple polygon. Chapter 7 describes the algorithm to perform queries once the geodesic cd-Voronoi diagram is known. We discuss observations about extensions of the problem in Chapter 8. Chapter 9 will conclude the study, where we discuss the implication of the results and future research.

Chapter 2



Ecological background

In this part, we will discuss a myriad of ecological models with different representations and goals. Then, we will give a detailed overview of the DIMO model.

2.1 Overview

Seeds can disperse via self-dispersal, such as the explosive mechanism of the plant species *Impatience capensis*, better known as jewelweed, or via external factors such as wind, water, animals and humans. Not all seeds travel the same distance; some rare events make the seeds dispersal very far. The *long distance dispersal* indicates the dispersal of 1% of the seeds that travelled the furthest. LDD is crucial for natural populations and spreading (Nathan, 2006).

So even within species, there is a large variation in distances over which individuals disperse. Ecologists express this variation in 'dispersal kernels', which represent the probability that an individual disperses over a specific distance. These kernels also quantify the potential of a species for LDD.

Ecologists have quantified dispersal kernels for plant seeds (Wamelink et al., 2014; Nathan and Muller-Landau, 2000; E. K. Klein et al., 2003; Treep et al., 2021), insects (Baguette, 2003), birds (Van Houtan et al., 2007) and mammals (Revilla and Wiegand, 2008). As dispersal is a continuous process, it can be modelled continuously, using for example a single-parameter Pareto-distribution fitted to dispersal data (Treep et al., 2021).

2.1.1 Modelling spreading plants

We will first shortly introduce the DIMO model that is developed by ecologist Wieger Wamelink, whom we spoke to, and his colleagues (Wamelink et al., 2014). DIMO simulates the dispersal and establishment of plants through a possibly fragmented landscape – grid map – over a period of time in steps of one year. Each run concerns one plant species with specific parameters. The model could be helpful in making or evaluating policy. In Section 2.2 we will give a more elaborate overview of the model.

Andújar et al. (2017) construct a geometrical model to predict the spreading of Johnsongrass (*Sorghum halepense*), an aggressively spreading weed, in maize fields in Central Spain. In contrast with the DIMO model, the authors use varying patch sizes and varying patch densities.

Just as the previous model, Somerville et al. (2020) investigate the prevention of weed spreading. Their review describes both static and spatio-temporal models. One of the problems they tackle is where to use spray to stop the spreading of weed. The challenge is to use enough spray without waste.

A study by Treep and his colleagues (Treep et al., 2021) proposes that seed dispersal in plants is a search strategy to find suitable habitat. To investigate their claim, they use a seed dispersal kernel, as we described in the previous section. The simulated landscapes, represented by grids, differ in patch size and distance between the patches. The researchers vary the parameters of a Pareto-distribution to model the scale from dominantly short-distance dispersal to non-local distribution with a high chance of LDD. Each time step, the number of seeds in each grid cell is updated using the rules given by the Pareto-distribution. As a result, they calculate for instance the success rate of finding habitat i.e. the fraction of seeds that lands on suitable habitat.

2.1.2 Habitat connectivity

Climate change could demand higher plant migration rates than possible, especially in regions that are disconnected because of for example roads. Dullinger et al. (2015) compare different levels of habitat fragmentation in their models to evaluate the migration rates of plants. They model a hypothetical landscape using a grid raster. Their results suggest to focus on the movement of species throughout the whole countryside instead of only protected areas, because of the importance of connectivity.

Albert et al. (2015) collect data to assess what plant traits affect seed dispersal. Their predictive models can help decide how to restore habitat connectivity. As an example, if mostly land animals instead of birds transport the seeds, then it is efficient to restore corridors for animals; in the other case it might not be. Note, this is not a geometric model, but a systematic literature review.

2.1.3 Other spreading models

Not only seed disperse, also other substances can spread over a region. Teggi et al. (2018) create a dispersion model for pollutants. The sources are modelled as polygons with corresponding emission rates. Whereas DIMO and other models use a representation of presence (1) or absence (0) of plants, the output of this model consists of a spatial map with the concentration of pollutants in the region.

2.2 DIMO Model

Wamelink and his colleagues (2014) developed a dispersal model called DIMO. With support of the researchers, we reviewed and ran the program. DIMO simulates the dispersal of plants on a 2D map over a period of time. Each run concerns one plant species with specific parameters. The output map shows the age of the plant per grid cell.

At the start of the model, the initial source locations of the plant are given in a 2D landscape, see for example Figure 2.1a. The region is represented by a homogeneous grid; each cell of 250m $\times$ 250m indicates the presence or absence of a species. A function defines the time of arrival of the dispersing seeds from the source locations. This

function depends on the dispersal by wind and by animals, possibly with hindrance of obstacles.

Figure 2.1b shows the plant age map after 10 years with a uniform wind, but the user can also use another function for the wind. The dispersal by wind is defined by meters per time unit per direction (given in degrees). The landscape can have different compartments for different wind functions. An example of the effect of wind is shown in Figure 2.1c. The dispersal by animals is given by a radius d in meters per time unit. Obstacles, either for animals or for the wind, limit the spreading. In the current version of DIMO, the region is split into compartments such that seeds cannot spread from one compartment to the other, see for example an output map with the influence of barriers in Figure 2.1f.

Not all regions are suitable for the germination and growing of plants, which is defined in the *habitat suitability map*, specific per plant species. This is different from a barrier, since the germination is limited, not the dispersal. Plants can only germinate in a suitable area.

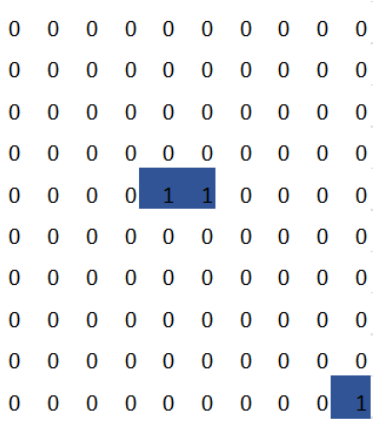
If a seed arrives on a location, the plants do not start growing immediately. The parameter *germination delay* denotes the time that a seed stays in the seed bank before it germinates. If the plant germinates, the plant should be counted as present. The parameter *the age of the first flowering* defines the time between germination and being a reproductive source plant itself. However, this parameter is not included in the current implementation version of DIMO. Figure 2.1d shows an example of a plant age map with germination delay. Since seeds cannot survive forever in the seed bank, the parameter *seed longevity* indicates the time a seed can survive dormant in the seed bank.

In short, for each plant, the input for DIMO consists of:

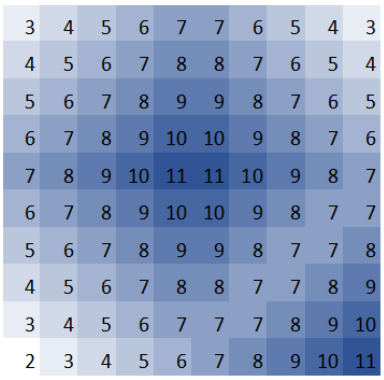
- A map of the landscape.
- A set of points, the source locations of the plant, including the age of the plant (Figure 2.1a).
- A set of points, the locations of the seeds of the plant, including the age of the seed.
- A habitat suitability map.
- A set of barriers, both for animals and for the wind (Figure 2.1e).
- Distance functions c_i for the wind speed and force for compartment i in the landscape.
- Parameters for the wind dispersal distance, animal dispersal distance, the seed bank longevity and the germination delay.

In DIMO, the output consists of:

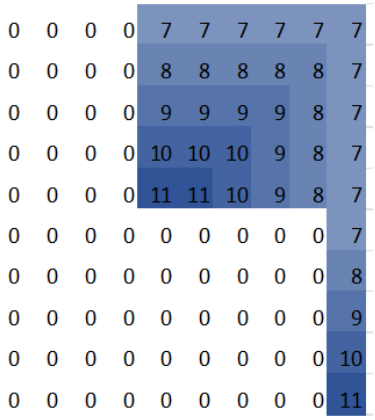
- A plant age map that gives the number of years the plant is present as reproductive plant (Figure 2.1b).
- Suitability map, only the suitable areas reached by the plant.
- Seed-bank-life map that denotes the number of years left that the seed can survive in the seed bank.
- Seed-distribution map: the points the plant reached in the last round.



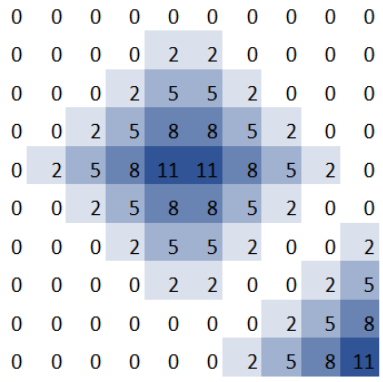
(A) The plant input map.



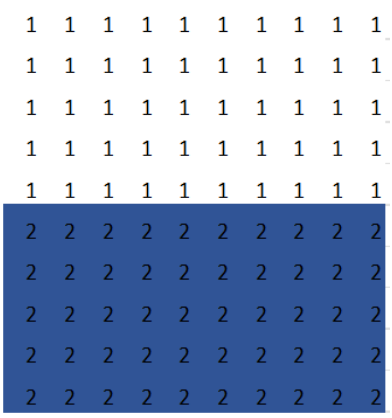
(B) The output plant age map (uniform wind).



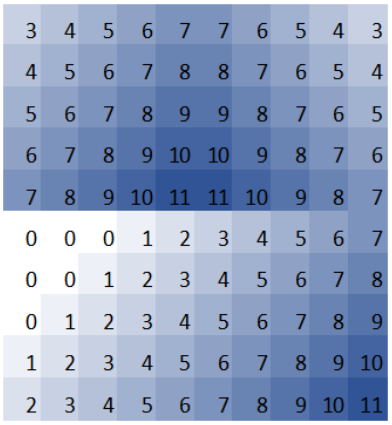
(C) The output plant age map (pluriform wind).



(D) The output map plant age (germination delay).

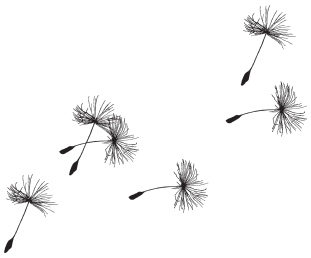


(E) The input map for barriers.



(F) The output plant age (barriers)

FIGURE 2.1: Visualisation of the DIMO model. The numbers show the age of the plant in the grid cell. (A) The plant input map shows the set of grids that contain the source plants, represented by "1". (B) The output plant age map with uniform wind. (C) The output plant age map with pluriform wind. The wind force is present in the direction North, North-East and East. (D) The output map plant age with a germination delay of 2. (E) The input map for barriers. Here, the region is divided into two regions, 1 and 2. The seeds cannot spread from region 1 to 2, and vice versa. (F) The output plant map when the barriers given in Figure 2.1e are used. Here, one can see that the seeds do not travel from compartment 1 to compartment 2.



Chapter 3

Geometric background

In the geometric background we will discuss geometrical problems that have been (partly) solved before. We will review shortest paths algorithms, including using other metrics than Euclidean distance. One of these metrics is the convex distance function, which we will inspect in more detail. Continuing, we will review Voronoi diagrams, Voronoi diagrams under a convex distance function and geodesic Voronoi diagrams. Finally, we will elaborate on Minkowski sums, because this operation corresponds closely with the way plant regions expand in one time step.

3.1 Distance measures

As we explained in Section 2.2, DIMO takes the force of the wind into account in the dispersal of seeds. We will discuss several distance measures and evaluate the suitability for our geometric seed dispersal model, specifically in the context of including the effect of wind.

The Euclidean distance is a popular distance measure and usually fits well in settings applied to the real world. Let $i = (i_x, i_y)$ and $j = (j_x, j_y)$ be two points in $\mathbb{R}^2$. Then the Euclidean distance between i and j is defined as

$$d(i,j) = \sqrt{(i_x - j_x)^2 + (i_y - j_y)^2}.$$

Another popular distance measure is the Manhattan distance. The Manhattan distance is defined as

$$d(i,j) = |i_x - j_x| + |i_y - j_y|.$$

The generalisation of the Euclidean and Manhattan distance is known as the Minkowski distance, given by

$$d_p(i,j) = \sqrt[p]{|i_x - j_x|^p + |i_y - j_y|^p}.$$

This measure is also called the L_p *norm*. The Manhattan distance measure corresponds with L_1 and the Euclidean distance measure corresponds with L_2.

The discussed measures are symmetric, positive and adhere to the triangle inequality. The symmetric property indicates that the distance from i to j is equal to the distance from j to i. The positive property means that the distance between two distinct points is positive, and the distance from any point to itself is 0. The triangle inequality states that the distance from i to j via k is at least as long as the shortest distance from i to j. The combination of these properties is known as a *metric*.

3.1.1 Convex distance function

As we said before, we would like to model the influence of wind. Because wind and headwind exist, we need to let go of the property of symmetry. Therefore we will look into a notion called the *convex distance function*, which exactly characterises distance functions satisfying the triangle inequality (Barequet et al., 2001).

In Figure 3.1, an example of a convex distance function based on a convex polygon is shown. From now on, we will always assume that a convex distance function is based on a polygon, unless stated otherwise. Let R be a convex polygon that contains the center O in its interior. The boundary of R denotes all the points with distance 1 from the center. To calculate the distance from a point p to a point a using a convex distance function, we translate R by placing O on site p, and shoot a ray from p through a. Let v be the intersection point between the ray and R. Then a convex distance function d_R is defined by

$$d_R(p,a) := \frac{||a-p||}{||v-p||},$$

where $||a-p||$ is the Euclidean distance from a to p. $d_R(p,a)$ is the factor that R must be contracted or stretched to touch a. (Ma, 2000).

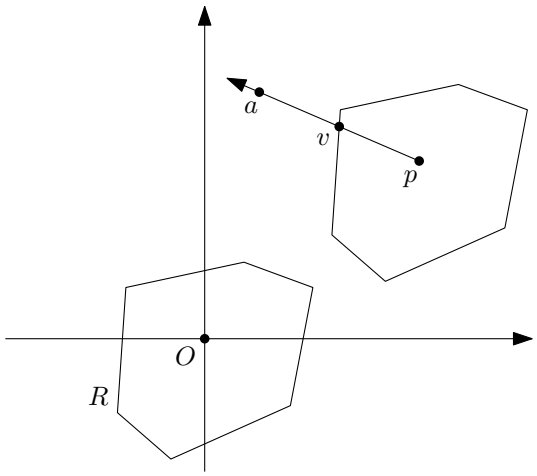


FIGURE 3.1: A convex distance function. (Ma, 2000).

3.2 Shortest paths

In this section, we will discuss articles that examine shortest path algorithms. First, we will discuss geodesic distances. Then we will elaborate on continuous Dijkstra, a technique that can be used to handle shortest paths in case of obstacles.

3.2.1 Geodesic distance

Given are two points p and q, that both lie in a polygon W. The geodesic distance between p and q is defined as the length of the shortest path from point p to q that is completely contained in W.

Geodesic distances have been studied extensively in the past. Problems in the geodesic setting that have been solved optimally regarding their time complexity include computing the shortest path tree of a point in a simple polygon (Guibas et al., 1986)

and an optimal algorithm for geodesic Voronoi diagrams in simple polygons (Oh, 2019). A more elaborate explanation on Voronoi diagrams can be found in Section 3.3.

Using the geodesic distance, Arge and Staals (2017) addressed a problem that has its roots in ecology or, even more specific, in the spreading of plants. We are given a threshold ϵ, two sets of m points in $\mathbb{R}^2$, a set of "red" points A, the plant locations, and a set of "blue" points B, the possible destination locations, in a simple polygon with n vertices. Each point $p \in A \cup B$ also has a real value p_v, representing for example temperature.

The question is to find, for every plant location (red point) a closest destination location (blue point), provided that the ecological value of the blue point differs at most ϵ from the ecological value of the red point. The algorithm they provide handles deletions and insertions in $O(\sqrt{m}\log^3 n)$ time and process geodesic nearest neighbour queries in $O(\sqrt{m}\log m \log^2 n)$, with a space usage of $O(m \log n + n)$.

3.2.2 Continuous Dijkstra

We will continue this overview of literature in the direction of problems where polygonal obstacles are present. Hershberger and Suri (1999) provide an optimal algorithm called continuous Dijkstra, to compute a planar map that encodes all shortest paths from a fixed source point to all other points in the plane. The map can process queries in $O(\log n)$ time, with n the number of vertices in the polygonal obstacles.

The algorithm uses an efficient wavefront propagation. It simulates a "bubble" expanding from one source point. The wavefront at time t consist of all the points that have shortest distance t to the source vertex v. The boundary of the wavefront is a set of circular arcs, as can be seen in Figure 3.2. The arrows in this figure show that the meeting point between these arcs is either a straight line or a hyperbola.

To simulate the wavefront, events are processed one by one. At these events, the topology of the wavefront might change. These events are categorised as *wavefront-wavefront collisions* or *wavefront-obstacle collisions*. Collisions between wavefronts can happen between arcs that are neighbours or non-neighbours. Collisions between non-neighbouring arcs are not easy to detect and process. Therefore, the authors introduced the concept of an *approximate wavefront*. Only in the second phase of the algorithm, the exact collision events are computed using Voronoi diagram techniques. Using an efficient data structure called a *quad-tree-style* subdivision, the edges and vertices of the obstacles can be saved and searched through efficiently.

The algorithms takes $O(n \log n)$ time and $O(n \log n)$ space. $O(n \log n)$ running time is optimal. Wang (2021) settled the open problem for the optimal space requirement by introducing an algorithm with $O(n \log n)$ time and $O(n)$ space complexity.

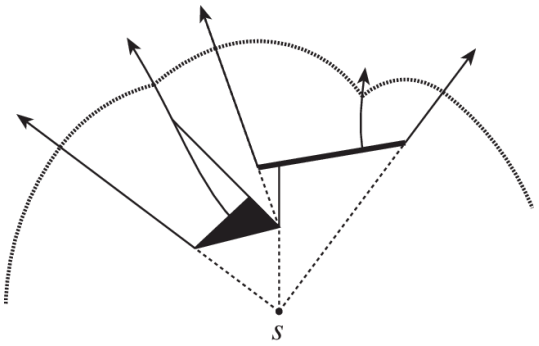


FIGURE 3.2: The wavefront of the algorithm continuous Dijkstra. (Hershberger and Suri, 1999).

3.3 Voronoi diagram

In this section, we will introduce Voronoi diagrams, Voronoi diagrams under a convex distance function, and shortly discuss abstract Voronoi diagrams. In the next section, we will discuss different algorithms to calculate geodesic Voronoi diagrams.

A Voronoi diagram partitions the plane with a given set P of sites into regions. Each $p \in P$ has a corresponding cell in the Voronoi diagram that contains all the points in the plane that are closer to p than any other site in P.

Fortune (1987) describes an optimal sweep line algorithm to compute the Voronoi diagram for a set of m point sites in Euclidean plane, commonly known as *Fortune's algorithm*. Fortune's algorithm takes $O(m \log m)$ time and $O(m)$ space.

3.3.1 Voronoi diagram under a convex distance function

Since we are interested in using different distance measures, we researched how to construct bisectors and Voronoi diagrams under a convex distance function.

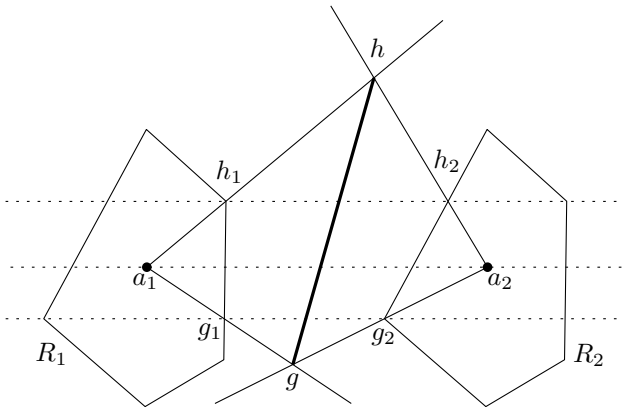


FIGURE 3.3: Construction of bisector between a_1 and a_2. (Ma, 2000).

The complete construction of the bisector using a sweep line costs $O(r)$ time if R is a polygon with r vertices. Since the bisector can have $r - 2$ segments, this running time is optimal (Ma, 2000). First, we will sketch how to construct one segment of the bisector, see Figure 3.3. This process can be repeated to form the complete bisector.

Given are two points a_1 and a_2 and assume w.l.o.g. that they are on a horizontal line. We will construct the bisector of these two points, see also Figure 3.3. Let h_1 be a vertex of R_1 and let h_2 be the intersection point of the horizontal line through h_1, and R_2. Find the intersection point h of the line through $\overline{a_1h_1}$ and the line through $\overline{a_2h_2}$. Point h is then on the bisector. Choose vertex g_2 of R_2 in such a way that the area between the horizontal line through h_1 and the horizontal line through g_2 does not contain any vertices. Then, find the intersection between the horizontal line through g_2 and R_1, g_1. Then, find intersection point g for the line through $\overline{a_1g_1}$ and $\overline{a_2g_2}$. In the case of Figure 3.3, point h_1 is a vertex of R_1 and g_2 is a vertex of R_2, but this is not necessarily so. The line segment $\overline{hg}$ is part of the bisector between a_1a_2.

In the degenerate case that one of the edges of R is parallel to a_1a_2, the bisector is not a line but a non-bounded region, see Figure 3.4. Therefore, we introduce the concept of a chosen bisector, based on lexicographical rules, and pick one of the edges of the non-bounded region as the bisector.

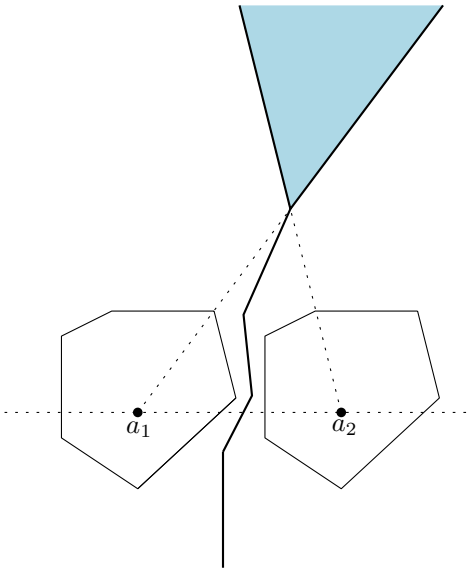


FIGURE 3.4: In the degenerate case, the bisector is not a line but a non-bounded region.(Ma, 2000).

Now that we know we can construct a bisector when using convex distance functions, the next question is how to efficiently compute the Voronoi diagram. Chew and Dyrsdale (1985) provide a divide-and-conquer algorithm that can compute the Voronoi diagram in $\Theta(m \log m)$ time, with m the number of sites, given that the intersection of two bisectors can be found in constant time. Dehne and R. Klein (1997) provide a sweep line algorithm that can handle abstract Voronoi diagrams for an arbitrary nice metric within (optimal) $O(m \log m)$, and includes symmetric convex distance functions (R. Klein, 1988). This approach can be generalised to handle arbitrary convex distance functions (Ma, 2000).

Anisotropic regions

In the previous section, we assumed that the complete region was associated with one convex distance function. In the anisotropic setting, different convex distance functions might be used for different subregions. If more than one convex distance function is allowed, the triangle inequality does not hold anymore, see also Section

8.4. The seed dispersal problem in the anisotropic setting is more difficult, and is usually approached by approximation instead of finding an exact solution. For example, Cheng et al. (2008) present an algorithm to approximate shortest paths in anisotropic regions.

3.3.2 Abstract Voronoi diagram

The abstract Voronoi diagram is a subdivision of the plane such that each cell defines the region closest to the point enclosed in that cell, but the distance measure is not necessarily Euclidean. There are requirements for the distance measure to make sure the Voronoi regions are path-connected and as a union fill the whole plane.

Klein (1988) introduces the notion of abstract Voronoi diagrams and provide a divide-and-conquer approach to compute abstract Voronoi diagrams. It was required that two bisecting curves under the given distance measure intersect only finitely often, to make sure the abstract Voronoi diagram is a finite graph. Among these distance measures are symmetric distance functions.

Klein et al. (2009) revisit abstract Voronoi diagrams and gives a new axiom system that works without the requirement that two bisecting curves may intersect only finitely often. It turns out that the other two axioms, that the Voronoi regions are path-connected and as a union fill the whole plane, are strong enough to design proofs and design an algorithm to compute the abstract Voronoi diagram for a larger class of concrete Voronoi diagrams. In particular, all convex distance functions are included now. The abstract Voronoi diagram can be computed in $O(m \log m)$ time, assuming that the computation of a bisector could be carried out in constant time.

3.4 Geodesic Voronoi diagram

Algorithms for abstract Voronoi diagrams can be used to compute Voronoi diagrams in a polygon under a geodesic distance measure. In the time complexity analysis of abstract Voronoi diagrams, we assumed the computation of a bisector takes constant time. The computation time of geodesic bisector, however, is dependent on n, the number of vertices. Therefore, the computation time of the abstract Voronoi diagram algorithm can be very long if we want to compute geodesic Voronoi diagrams. Therefore, the algorithm would need structural adaptations to suit geodesic distances.

In this section, we will elaborate on multiple algorithms specifically designed to compute geodesic Voronoi diagrams. First, we will give some concepts as defined in Aronov (1989) and elaborate on the output complexity.

Definitions

Given a simple polygon W with n vertices and set P of m sites in W. Let $Vor_W(P)$ denote the Voronoi diagram of the set of sites P. $V(s, W)$ is the Voronoi cell of site s in polygon W.

For any two points i and j, we define $\gamma(i, j)$ as the geodesic shortest path from i to j. Then we call the last vertex before j on $\gamma(i, j)$ the *anchor* of j.

Let e be an edge of the shortest path map of site s, and let v be the furthest endpoint of e from s. Then, extend e in increasing distance from s. The part of the extension of e that lies in the interior of W is called *the extension segment of e, emanating from v*.

The augmented geodesic Voronoi diagram, $Vor^*_W(P)$, is $Vor_W(P)$ augmented with the union of the extension segments from each site $s \in P$, but only within the Voronoi cell of s. All points in a region of $Vor^*_W(P)$ have the same anchor with respect to the site of the Voronoi cell they belong in.

A vertex in three or more Voronoi cells or on the boundary of W and in two Voronoi cells is called a *Voronoi vertex*. A *Voronoi edge* has breakpoints exactly at the intersection with extension segments, called *e-breakpoints*.

Output complexity

Geodesic Voronoi diagrams have size complexity $O(n+m)$, with n the number of vertices in W and m the number of sites, which was proved by Aronov (1989).

The complexity can be explained as follows. If we only look at the Voronoi edges and vertices, and ignore e-breakpoints, the Voronoi diagram is by construction a planar graph with vertices of degree 3 and above. Euler's formula shows the complexity of the map is linear in the number of faces. Since each Voronoi cell is connected, there are m faces for m sites, so there are $O(m)$ Voronoi vertices and Voronoi edges.

W has n edges, so we add $O(n)$ to the complexity of the output map. A Voronoi edge has e-breakpoints exactly at the intersection with extension segments. These extension segments correspond one-to-one with the vertices of W, so the number of e-breakpoints is bounded by n.

Thus, the total output complexity of a geodesic Voronoi diagram is $O(n+m)$.

3.4.1 Geodesic Voronoi diagram algorithm by Aronov

In this section, we will describe the divide-and-conquer algorithm to construct a geodesic Voronoi diagram for a Euclidean distance metric, as given by Aronov (1989).

Overview algorithm

In the preprocessing phase we triangulate W and compute a balanced decomposition of the triangulation tree. With the triangulation tree, we can cut up the polygon recursively in two parts in constant time per cut, such that each part has at least a quarter of the vertices.

In the base case of the recursive divide-and-conquer algorithm, if P consists of only one site s, then $Vor_W(P)$ is a single cell $V(s,W) = W$. If W is a triangle, then the Euclidean Voronoi diagram can be computed in $O(m \log m)$ time and truncated to W in linear time.

In the recursive step, we divide W in two subpolygons W_L and W_R, and divide P in P_L and P_R, such that $P_L \subset W_L$ and $P_R \subset W_R$. We recursively compute $Vor^*_{W_L}(P_L)$ and $Vor^*_{W_R}(P_R)$.

At last, we extend $Vor^*_{W_L}(P_L)$ to $Vor^*_W(P_L)$, and $Vor^*_{W_R}(P_R)$ to $Vor^*_W(P_R)$ in $O((n+m)\log(n+m))$ time. Then, we compute $Vor^*_W(P)$ by merging $Vor^*_W(P_L)$ and $Vor^*_W(P_R)$ in $O(n+m)$ time.

Extension phase

In the extension phase, we cut the polygon W in two subpolygons W_1 and W_2, separated by a chord e of the triangulation, and we are given $Vor^*_{W_1}(P_L)$. The output of this subroutine is $Vor^*_W(P_L)$.

We process all the triangles Δ in W_2, starting at the triangle adjacent to e. We perform a sweep line that is parallel to edge f, through which we enter Δ, and end at the edges of Δ, as is depicted in Figure 3.5. We add (potential) intersections between neighbours, either bisector-bisector or bisector-extension segment, as events in the event list. At each event, a region disappears and we add potential intersections between the new neighbours to the event list of the appropriate triangle.

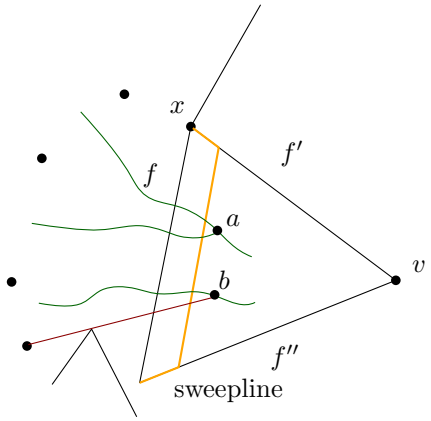


FIGURE 3.5: Triangle Δ, which we enter through edge f, whose remaining edges are f' and f''. The vertex shared by f' and f'' is v. Point a denotes an intersection point between two bisector curves, and point b denotes an intersection point between a bisector edge and an extension segment.

Merge phase

In this phase of the algorithm, we are given two sets of sites P_L and P_R (with a chord of the polygon separating them), and $Vor^*_W(P_L)$ and $Vor^*_W(P_R)$. To compute the merge of $Vor^*_W(P_L)$ and $Vor^*_W(P_R)$, we need to compute the bisector between P_L and P_R, $b(P_L, P_R)$.

To find the points of $b(P_L, P_R)$ on the boundary of W, we divide the boundary in $O(n+m)$ line segments, such that each segment lies in one region with one corresponding site and anchor point of set P_L and and one corresponding site and anchor point of set P_R. We can find in linear time what points on the bisector have the same distance to the closest point in P_L and the closest point in P_R.

Given a point of $b(P_L, P_R)$, we know the first two sites and anchor points that define the bisector. The anchor points only change when the bisector intersects with an extension segment. After that, we can trace the bisector with the new anchor points further in the same way.

Time complexity

Since the extension phase takes $O((n+m)\log(n+m))$ time, and the merge phase takes $O(n+m)$ time, we can use recurrence inequality bounds to find a time complexity of

$$O((n+m)\log(n+m)\log n).$$

3.4.2 Geodesic Voronoi diagram algorithm by Papadopoulou

In this section, we will shortly discuss the geodesic Voronoi diagram algorithm by Papadopoulou and Lee (1998). They construct a divide-and-conquer algorithm and combine it with sweep techniques. This improves the time complexity to $O((n+m)\log(n+m))$. Similar to Aronov (1989), the described algorithm computes the augmented geodesic Voronoi diagram.

First, we triangulate the polygon W. The dual graph of the triangulated polygon is a tree, and we select one triangle of degree one as its root. The diagonal d of each triangle Δ partitions the polygon in two subpolygons, one "below" Δ and one "above" Δ.

We use the conventions as pictured in Figure 3.6. $\Delta(d)$ and $\Delta'(d)$ are the triangles adjacent to diagonal d, such that $\Delta'(d)$ ("above" d) is on the path from the root to $\Delta(d)$ ("below" d).

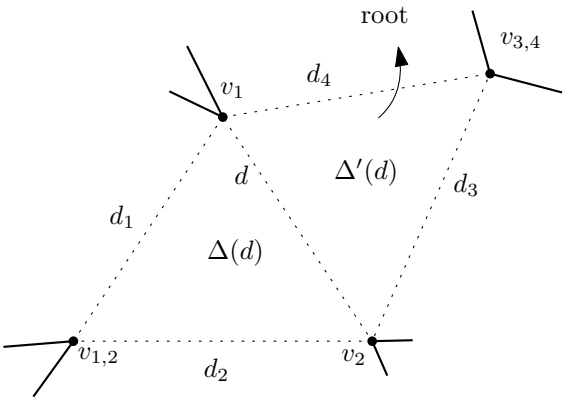


FIGURE 3.6: Conventions as used in Papadopoulou and Lee (1998).

In the first phase, we sweep the polygon in postorder traversal of the tree. For each triangle $\Delta(d)$ of the root diagonal of d, we compute the geodesic Voronoi diagram of all sites "below" d. We store the information in a subdivision of complexity $O(n+m)$.

In the second phase, we sweep the polygon in preorder traversal of the tree. For each triangle $\Delta(d)$ of the root diagonal of d, we compute the geodesic Voronoi diagram of all sites "above" d, and again store the information in a subdivision of complexity $O(n+m)$.

In the final phase, we merge the two subdivisions inside each triangle to obtain the geodesic Voronoi diagram. The Voronoi diagram of all sites "below" d, truncated to $\Delta(d)$, and the Voronoi diagram of all sites "above" d truncated tot $\Delta(d)$, is exactly the Voronoi diagram of all sites in W, truncated to $\Delta(d)$.

The merging phase takes $O(n+m)$ time in total. The sweeping phases take $O((n+m)\log(n+m))$ time, since $\Omega(n+m)$ events can occur in the worst-case scenario. Therefore, the total time complexity is $O((n+m)\log(n+m))$.

3.4.3 Optimal geodesic Voronoi diagram algorithm

In recent years, a myriad of techniques were presented that improved the run time of the algorithm.

Oh and Ahn (2020) present a sweep line algorithm with a time complexity of $O(n + m\log m\log^2 n)$. They compute the adjacency information of the Voronoi cells, instead of computing the whole Voronoi structure during the sweep line. This reduces the number of events. Whereas the algorithm of Papadopoulou and Lee (1998) could have $\Omega(n+m)$ events, only $O(m)$ events occur in this algorithm.

Liu (2020) improves the running time even further to $O(n + m(\log m + \log^2 n))$ time. They sweep the polygon using the method introduced by Papadopoulou and Lee (1998), and combine it with the technique of computing adjacency information as Oh and Ahn (2020) presented.

At last, Oh (2019) proves that it was possible to compute the geodesic Voronoi diagram optimally in $O(n + m\log m)$. They compute a balanced geodesic triangulation of simple polygon W, and sweep the polygon twice, cell by cell. In each cell, they compute the Voronoi diagram of all the sites that lie in the region swept so far in $O(n + m\log m)$ time. The merge phase that is similar to Papadopoulou and Lee (1998) takes $O(n+m)$ time, which results in a total (optimal) time complexity of $O(n + m\log m)$.

3.5 Minkowski sum

In the seed dispersal problem, seeds spread themselves in discrete time steps. Suppose the shape of the wind force is given by a polygon R. That means that every plant from round t will spread seeds around itself in the form of polygon R. We can compute the region of plants in round $t+1$, say B_{t+1}, by taking the Minkowski sum of the region of plants in round t, say B_t, with polygon R.

We can visualise the Minkowski sum as sliding polygon R along the boundary of B_t. An example can be seen in Figure 3.7. The formal definition of the Minkowski sum of sets $S_1 \subset \mathbb{R}^2$ and $S_2 \subset \mathbb{R}^2$ as given in de Berg et al. (2008) is

$$S_1 \oplus S_2 := \{p + q : p \in S_1, q \in S_2.\}$$

Since polygons are planar sets, the definition also applies to polygons. The Minkowski sum of two convex polygons with n_1 and n_2 vertices is convex itself and can be computed in $O(n_1 + n_2)$ time. If one of the polygons is convex and the other one is non-convex, then the time complexity is $O(n_1 \cdot n_2)$. If both are non-convex, then the running time is $O({n_1}^2 n_2^2)$. (de Berg et al., 2008).

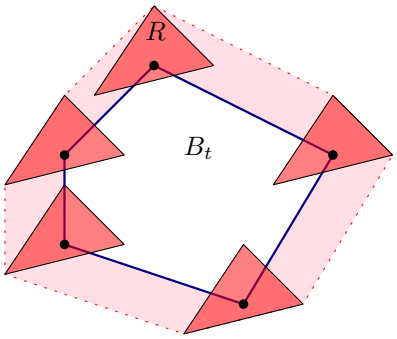
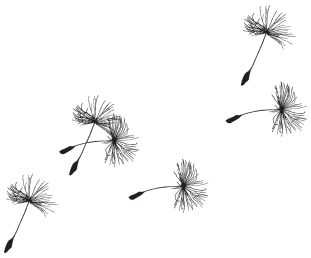


FIGURE 3.7: Visualisation of a Minkowski sum of a polygon B_t and R.

Chapter 4



Formalisation

An important part of solving the problem is formalising the ecological model into a geometrical model. In this chapter, we give the problem statement and output complexity of the basic problem based on DIMO and its extension with a convex distance function.

4.1 Basic problem based on DIMO

Given are a region W, which we will represent as a simple polygon with n vertices, and a set P with m sites, initial source plants. At $t = 0$, the whole region W, except for all sites $p \in P$, does not contain plants. Assume all sites p have age 1 at time $t = 0$. Each time step t, each point in the region with plants at time t, we call it B_t, disperse their seeds within a dispersal radius d. That is, the plant region B_{t+1} is the Minkowski sum of B_t and a disk with radius d. The region with seeds will be reproductive plants and spread seeds themselves after one time step. We want to answer the following question: at time 1 to t, what is the plant region B_t, and what age do the plants have? The output map is a subdivision of W, where each region has a corresponding age with value $[0, t + 1]$. Note, points that are unreachable will have value 0.

Approach

The region with plants that originally came from plant p_i can overlap with the region of plant p_j. But since all sites grow at equal speed with no obstacles, we can compute the geodesic Voronoi diagram, and solve the problem as if it were continuous for each region of the Voronoi diagram separately. The complete output map is the union of all cells of the Voronoi diagram.

If we for now assume that each Voronoi cell is convex, the problem is the same as growing a disk with radius d each step. The radius of the disk with plants is $t \cdot d$ at time t starting with time $t = 0$. For example, this means that B_1 is a disk with radius d and age 1 with a center site p (p has age 2), given that the region with plants did not yet intersect with the boundary of the Voronoi cell of p. The output consists of concentric circles, with site p in the center, intersected with its Voronoi cell.

Instead, we know that Voronoi cells can be non-convex. Then, concentric circles of different radii can appear. An example is presented in Figure 4.1. We will discuss how to solve this problem in more detail in Chapter 7.

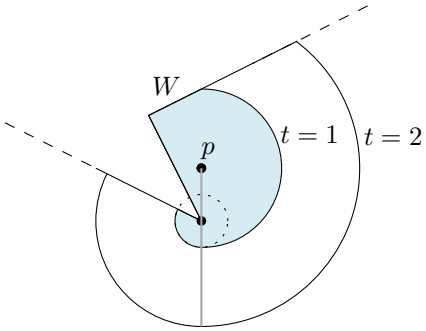


FIGURE 4.1: If world W is not convex, then concentric circles of a different radius than d can appear.

Complexity output map

The output map is a subdivision of W, where each region has a corresponding age with value $[0, t+1]$. In the output map, we will not make a distinction between plants that originate from different origins.

As we mentioned before, we can solve this variation by computing the geodesic Voronoi diagram of m source plants. A geodesic Voronoi diagram has complexity $O(n+m)$ (Aronov, 1989). Since the disks are convex, each disk can intersect a Voronoi edge of a Voronoi cell at most twice. Each Voronoi cell edge can be intersected at most four times, since it borders two Voronoi cells. Disks within a Voronoi cell cannot intersect each other. Therefore, one time step has complexity $O(n+m)$. In total, the output complexity is $O((n+m) \cdot t)$.

In Figure 4.2, we show a construction that has complexity $\Omega(m \cdot t)$ for any t and m, since all $m \cdot t$ disks intersect with an edge of W, with just a constant number of edges. Moreover, for any given t and n, we can construct a star-shaped polygon as in Figure 4.3 that has complexity $\Omega(n \cdot t)$ with a constant number of sites. If we combine these figures and change one of arms of Figure 4.3 into the construction of Figure 4.2, we create a construction with output complexity $\Omega(m \cdot t + n \cdot t) = \Omega((n+m) \cdot t)$, which is equal to the upper bound. Therefore, the worst-case output complexity is $\Theta((n+m) \cdot t)$.

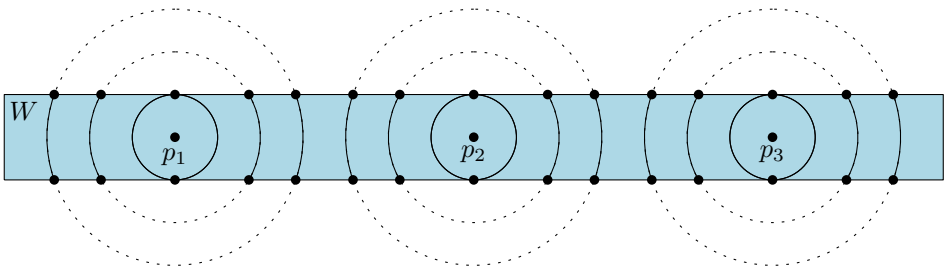


FIGURE 4.2: In this figure, the disks grow from sites p_1, p_2 and p_3. Every time step, the disk intersects with the edges of W twice.

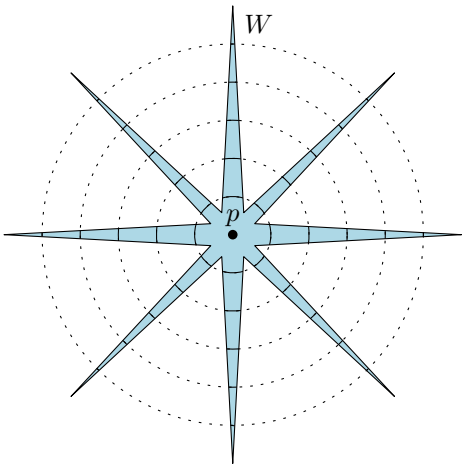


FIGURE 4.3: In this figure, the disks grow from site p. Every time step, the disk intersects with every edge of W twice.

4.2 Convex distance function

Given are a simple n-gon W, a set P of m sites, and convex distance function R based on a convex polygon of r vertices. At $t = 0$, the whole region W, except for $p \in P$, does not contain plants. Assume all sites p have age 1 at time $t = 0$.

Each time step t, all points in B_t disperse their seeds within a convex polygon R with r vertices. That is, the plant region B_{t+1} is the Minkowski sum of B_t and R. The region with seeds will be reproductive plants and spread seeds themselves after one time step. We want to answer the following question: at time 1 to t, what is the plant region B_t, and what age do the plants have? The output map is a subdivision of W, where each region has a corresponding age with value $[0, t + 1]$.

Approach

The formal definition of the Minkowski sum of sets $S_1 \subset \mathbb{R}^2$ and $S_2 \subset \mathbb{R}^2$ as given in de Berg et al. (2008) is

$$S_1 \oplus S_2 := p + q : p \in S_1, q \in S_2.$$

If $p = (p_x, p_y)$ and $q = (q_x, q_y)$, then

$$p + q := (p_x + q_x, p_y + q_y)$$

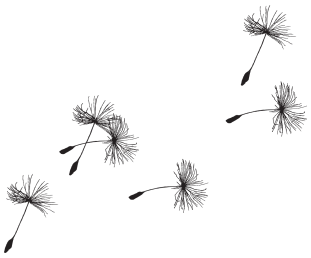
Suppose, we only look at one site s. At time $t = 1$, B_1 induced by s is R with center s. Then, B_2 is $B_1 + R = R + R$. Therefore, in this case, $S_1 = S_2$ and $p = q$, so

$$p + q = 2p := (p_x + p_x, p_y + p_y) = (2p_x, 2p_y)$$

In round t, if S_2 denotes the region with plants and S_1 denotes the convex distance function R, then $S_2 = t \cdot S_1$. Then $q = t \cdot p$, so

$$p + q = \big(p_x + t \cdot p_x, p_y + t \cdot p_y\big) = \big((t + 1) \cdot p_x, (t + 1) \cdot p_y\big).$$

We solve the specific discrete problem of computing $B_{1,2,\ldots,t}$ by first solving the problem as if it were continuous. As long as we consider a simple polygon without obstacles or inhabitable regions, B_t corresponds to all points at distance t to the closest site in W. Therefore, we need to compute the geodesic Voronoi diagram under R. To the best of our knowledge, there is no algorithm yet to compute the geodesic Voronoi diagram under a convex distance function. We will provide fundamental properties for the output complexity in Chapter 5, and give an algorithm to compute the geodesic Voronoi diagram under a convex distance function in Chapter 6. In Chapter 7, we discretize the problem again and answer the query as we formulated here.



Chapter 5

Properties of geodesic cd-Voronoi diagrams

In this section, we will provide the fundamental theorems that prove the output complexity of the cd-Voronoi diagram. We first prove that the Voronoi cell in a cd-Voronoi diagram is star-shaped. We will show that the output complexity of an additively weighted cd-bisector is $O(r)$. Then, we will prove the output complexity of the geodesic cd-bisector, and end with a proof of the output complexity of a geodesic cd-Voronoi diagram. We start with the preliminaries.

5.1 Preliminaries

Let P be a set of m sites in a simple polygon W with n vertices. We assume the general position, that is no vertex of W is equidistant from two distinct sites of P and no point of W is equidistant from four distinct sites of P. This was also assumed in previous work on geodesic Voronoi diagrams (Aronov, 1989; Papadopoulou and Lee, 1998; Oh, 2019).

Let R be a convex distance function defined by a convex r-gon. We will use *cd-polygon* to refer to this r-gon. The boundary R denotes distance 1 to its center c. We assume that the boundary of the cd-polygon does not intersect c. W.l.o.g., we define distance 1 in such a way that when placing R at each site in P and vertex of W, then these polygons do not overlap.

Note, even though we use the term 'convex distance function', we always mean that the function is based on a convex polygon with r vertices.

A *cd-edge* is an edge in the cd-polygon, a *cd-vertex* is a vertex of the cd-polygon. If Θ_1 is a cd-vertex, $\overline{\Theta_1}$ denotes the cd-edge with one endpoint Θ_1, and the other endpoint in clockwise order w.r.t. Θ_1. The *anchor cd-edge* of a piece of bisector e is the cd-edge of site s or t that defines e.

$\hat{d}(x,y)$ is the shortest distance between x and y for a convex distance function R. The *cd-distance* is the length of the path for a convex distance function R. If path $\gamma(s,t)$ is *cd-shorter* than $\gamma'(s,t)$, it means that in the context of a convex distance function R, $\gamma(s,t)$ is shorter than $\gamma'(s,t)$.

$\gamma(x,y)$ is the shortest path from x to y in the Euclidean metric, which is also a shortest path with the convex distance function. A *shortest cd-path* is a shortest path for a convex distance function.

5.2 Star-shaped Voronoi cell

In this section, we will prove that Voronoi cells in geodesic Voronoi diagrams under a convex distance function are star-shaped.

Lemma 5.2.1. *Given R, the straight line segment between two points s and t is a shortest path.*

Proof. Convex distance functions adhere to the triangle inequality (Barequet et al., 2001). Therefore, the straight line segment is a shortest path. □

Lemma 5.2.2. *For any points s and t, there exist a geodesic shortest cd-path that is polygonal.*

Proof. Convex distance functions adhere to triangle inequality, so any cd-shortest path from s to t that is not a polygonal path, can be made into a polygonal path with the same cd-distance. □

Lemma 5.2.3. *Suppose the shortest Euclidean geodesic path $\gamma(s,t)$ from s to t has Euclidean geodesic distance $d(s,t)$. Then, the path $\gamma(s,t)$ is also a shortest path under any convex distance function, that is, no other path is cd-shorter.*

Proof. Suppose there is a cd-shortest path $\eta(s,t)$ that is different from and cd-shorter than the Euclidean path $\gamma(s,t)$. That means that there is a subpath on $\eta(s,t)$ that is different from $\gamma(s,t)$. Because of Lemma 5.2.2, we can assume that $\eta(s,t)$ is polygonal. If there are multiple subpaths that are cd-shorter and different from $\gamma(s,t)$, w.l.o.g., we consider the subpath with the shortest Euclidean distance. Let y be the first vertex on the subpath that is on $\eta(s,t)$ but not in $\gamma(s,t)$, let x be the last point in $\eta(s,t)$ before y that is also in $\gamma(s,t)$, and let z be the first point in $\eta(s,t)$ after y that is also in $\gamma(s,t)$. Let w be the next vertex after y in $\eta(s,t)$, as shown in Figure 5.1.

Because the polygon W is simple, the region between $\gamma(s,t)$ (black) and $\eta(s,t)$ (red) is empty. Still, it could be the case that the direct path from x to w intersects with $\gamma(s,t)$. Therefore, suppose there is a point x' on the line segment $\overline{xy}$ with ϵ distance from y, and a point w' on the segment $\overline{yw}$ with ϵ distance from y. Since y is not on $\gamma(s,t)$, $\overline{x'w'}$ will not intersect $\gamma(s,t)$, nor will it intersect the polygon boundary.

Then, because of triangle inequality, we can shortcut vertex y and let the path go from x' to w' directly, which is cd-shorter or just as short, and has a (strictly) shorter Euclidean distance. This is in contradiction with our assumption that the path from x to z via y and w and possibly other vertices was a shortest cd-path that had the shortest Euclidean distance.

Therefore, there cannot be a cd-shortest path $\eta(s,t)$ that is different from and cd-shorter than the Euclidean path $\gamma(s,t)$. □

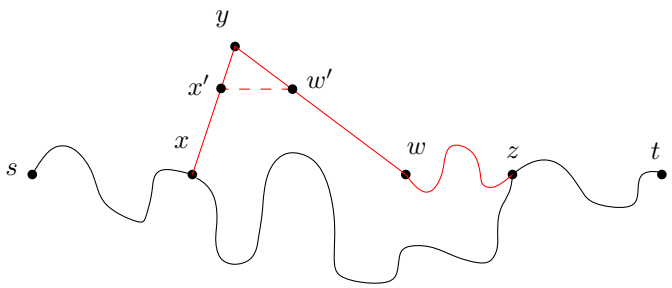


FIGURE 5.1: Here, the black path $sxzt$ denotes the shortest Euclidean path, and the red path $sxywz$ denotes the shortest cd-path that is different from the Euclidean path.

Definition 5.2.1. As defined by Aronov (1989), the *shortest path tree* of W from site s is the union of all the shortest paths from s to vertices of W.

Because of Lemma 5.2.3, we know that a shortest cd-path between two points is the shortest Euclidean path. Therefore, we will define the shortest path tree under a convex distance function as follows:

Definition 5.2.2. The shortest path tree in a simple polygon under a convex distance function R is defined as the Euclidean shortest path tree.

Similar to Ma (2000), we will define a bisector and a Voronoi diagram for a convex distance function R in such a way that many properties of the Euclidean bisector and Voronoi persist.

As seen in Figure 3.4, the bisector may not be a polygonal line but may contain a two-dimensional region. Therefore, we introduce the concept of a chosen bisector, based on lexicographical rules, and pick one of the edges of the region as the bisector.

Definition 5.2.3. Let $a \prec b$ denote that point a precedes b in the lexicographical ordering, which is the case if $a_x < b_x$, or $a_x = b_x$ and $a_y < b_y$. Then, the region of a with respect to b, $D(a,b)$, is defined as the set $\{p \in P : \hat{d}(a,p) \leq \hat{d}(b,p)\}$, and $D(b,a)$ its complement, within polygon W. The boundary between $D(a,b)$ and $D(b,a)$ is the *chosen bisector* $b(a,b)$.

Definition 5.2.4. Similar to Ma (2000), we define a *star-shaped* region A with center site c for a convex distance function so that any cd-shortest path from c to any point y in A is completely contained within A. Note, the center vertex is not necessarily the Euclidean center.

Definition 5.2.5. In line with Ma (2000), let $P = \{a_1, ..., a_n\}$ be a set of sites. We call

$$V(a_i, P) = \bigcap_{j \neq i} In(D(a_i, a_j)),$$

the Voronoi region of a_i. With slight abuse of notation, we will write V_{a_i} if the context is clear. *In* denotes the interior of the set.

Theorem 5.2.4 (Star-shaped)**.** *Under any convex distance function, Voronoi cells in the geodesic Voronoi diagram are star-shaped with respect to their center.*

Proof. For a visual representation, see Figure 5.2. Suppose we have a site c and a site d with corresponding Voronoi cells, respectively V_c and V_d. Let q be a point in V_c and

suppose a shortest cd-path $\gamma(c,q)$ from c to q is not entirely contained in V_c. Let p be a point in V_d that is on $\gamma(c,q)$.

First, we consider the situation that $d \prec c$. $\hat{d}(c,q) = \hat{d}(c,p) + \hat{d}(p,q)$, and $\hat{d}(d,q) \leq \hat{d}(d,p) + \hat{d}(p,q)$, since the shortest path from d to q does not necessarily contain p. Since p is in V_d, $\hat{d}(d,p) \leq \hat{d}(c,p)$. Therefore, $\hat{d}(d,q) \leq \hat{d}(c,q)$, which contradicts with the assumption that q is in V_c, since q is closer or just as close to point d, while points in V_c only contain points strictly closer to c.

Second, we consider the situation that $c \prec d$. $\hat{d}(c,q) = \hat{d}(c,p) + \hat{d}(p,q)$, and $\hat{d}(d,q) \leq \hat{d}(d,p) + \hat{d}(p,q)$. Since p is in V_d, $\hat{d}(d,p) < \hat{d}(c,p)$. Therefore, $\hat{d}(d,q) < \hat{d}(c,q)$, which also contradicts with the assumption that q is in V_c, since q is strictly closer to d, while points in V_c only contain points closer to c or at the same distance from d.

Thus, under any convex distance function, Voronoi cells in the geodesic Voronoi diagram are star-shaped with respect to their center site.

□

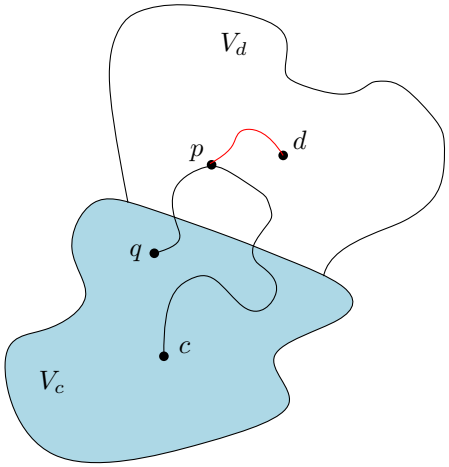


FIGURE 5.2: V_c, the blue region, denotes the Voronoi cell of point c, V_d is the Voronoi cell of point d. The path d to p is given in red.

5.3 Additively weighted bisector

To compute the geodesic cd-Voronoi diagram, we need to compute a geodesic cd-bisector. First, we will discuss the differences between a cd-bisector and a geodesic cd-bisector, and show that we need a construction for an additively weighted bisector to compute the geodesic cd-bisector. Then, we will argue that the additively weighted bisector consists of connected straight line segments. At last, we will prove that the complexity of the bisector is $O(r)$, as long as the anchor points do not change.

Difference cd-bisector and geodesic cd-bisector

In this section, we will discuss the differences between a cd-bisector and an geodesic bisector, and why the construction for a cd-bisector fails in the case that one of the two sites has an additive benefit.

Let s and t be two sites in W, and let v be a vertex of W. Suppose we know that vertex v is the anchor point (from s) of a piece of the bisector between s and t. Then, $\hat{d}(s,t) = \hat{d}(s,v) + \hat{d}(v,t)$.

In this case, the classic construction of a bisector does not work, see Figure 5.3 for a visual representation. Suppose the bisector between s and t intersects the line segment $\overline{vt}$. Let Θ_1 be a cd-vertex with respect to v, and let Θ_2 be the intersection with the horizontal line through Θ_1 and the cd-polygon R with center t. Classically, we would shoot a line through v and Θ_1, and t and Θ_2. The intersection point, u, of the two lines is on the bisector.

The reason it works in the classic version, is the equality of the ratio's

$$\frac{|\overline{v\Theta_1}|}{|\overline{vu}|} = \frac{|\overline{t\Theta_2}|}{|\overline{tu}|}.$$

If one of the two has an additive benefit, we cannot use the method that both polygons should be enlarged with the same factor. Then, one of the two is assumed to have a larger enlargement factor than it should have. Instead, we need to use a construction for an *additively weighted bisector*, where v has an additive benefit (the cd-distance from s to v) in comparison to t.

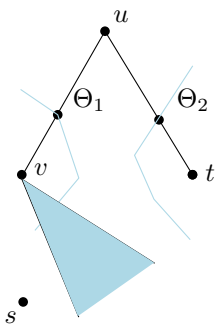


FIGURE 5.3: u is not necessarily on the bisector between v and t.

Output complexity additively weighted bisector

Theorem 5.3.1 (Additively weighted bisector)**.** *The additively weighted bisector consists of $O(r)$ connected straight line segments.*

Proof. To prove that the additively weighted bisector consists of straight line segments, we will construct an equation for each piece of the bisector, and show that this equation is linear. In the following sections, we will assume we already know the anchor cd-edge of the piece of bisector we construct. After that, we will elaborate on finding the starting cd-edge.

Suppose we are given two sites s and t, and two cd-edges $\overline{\Theta_1}$ and $\overline{\Theta_2}$ for respectively site s and point t, and we are given that $\overline{\Theta_1}$ and $\overline{\Theta_2}$ are anchor cd-edges.

We extend these cd-edges to lines and draw them in a coordinate system. W.l.o.g. suppose that s is on the origin of the coordinate system, and t is on the x-axis, see a visual representation in Figure 5.4. We will name the lines $\ell_s(f)$ and $\ell_t(f)$ for respectively site s and t, for a given translation factor f. At time step 0, the line $\ell_s(0)$ corresponds with the line through cd-edge $\overline{\Theta_1}$. Then, we draw the next time step $\ell_s(1)$.

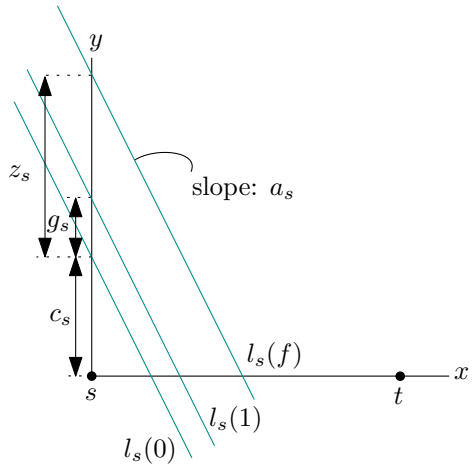


FIGURE 5.4: Visual representation of the lines through a cd-edge of s in a non-vertical situation.

If we look at a later time step, the cd-polygon expands. So $l_s(f)$ and $l_t(f)$ translate up or down as f varies – in other words, move further from s or t respectively. (x,y) lies on the bisector if and only if (x,y) lies on $\ell_s(f)$ and $\ell_t(f)$ for the same enlargement factor f. At that point, the distance to s and t is equal. If we look at an arbitrary enlargement factor (so an arbitrary f), we can create a mathematical equation using the variables f, x and y, and other variables we define here.

$$a_s = \begin{cases} 1, & \text{if } \ell_s(f) \text{ is vertical} \\ \text{slope of } \ell_s(f), & \text{otherwise} \end{cases}$$

$$b_s = \begin{cases} 0, & \text{if } \ell_s(f) \text{ is vertical} \\ 1, & \text{otherwise} \end{cases}$$

$$c_s = \begin{cases} \text{x-intercept of } \ell_s(0) & \text{if } \ell_s(f) \text{ is vertical} \\ \text{y-intercept of } \ell_s(0), & \text{otherwise} \end{cases}$$

$$g_s = \begin{cases} \text{x-intercept of } \ell_s(1) - c_s & \text{if } \ell_s(f) \text{ is vertical} \\ \text{y-intercept of } \ell_s(1) - c_s, & \text{otherwise} \end{cases}$$

Note that g_s has a positive value by definition.

To define f in the coordinate system, we need an extra variable z_s.

$$z_s = \begin{cases} \text{x-intercept of } \ell_s(f) - c_s & \text{if } \ell_s(f) \text{ is vertical} \\ \text{y-intercept of } \ell_s(f) - c_s, & \text{otherwise} \end{cases}$$

$$f_s = \frac{z_s}{g_s}$$

The equation for $l_s(f)$ is then given by

$$-a_s \cdot x + b_s \cdot y - c_s - f \cdot g_s = 0.$$

If we define the variables a_t, b_t, c_t and g_t alike for $l_t(f)$ we obtain the following formula

$$-a_t \cdot x + b_t \cdot y - c_t - f \cdot g_t = 0.$$

For the bisector, we are interested in the situation that both lines have the same factor f, and the same x and y coordinate, which is a part of the bisector.

We rewrite the formula's. If the anchor cd-edges are unchanged, the equation for the bisector piece is

$$\begin{aligned}
f = \frac{-a_s \cdot x + b_s \cdot y - c_s}{g_s} &= \frac{-a_t \cdot x + b_t \cdot y - c_t}{g_t} \\
&\Rightarrow \\
\tfrac{1}{g_s} \cdot (-a_s \cdot x + b_s \cdot y - c_s) &= \tfrac{1}{g_t} \cdot (-a_t \cdot x + b_t \cdot y - c_t) \\
&\Rightarrow \\
\tfrac{1}{g_s} \cdot (-a_s \cdot x + b_s \cdot y - c_s) + \tfrac{1}{g_t} \cdot (a_t \cdot x - b_t \cdot y + c_t) &= 0 \\
&= \\
(\tfrac{1}{g_t} \cdot a_t - \tfrac{1}{g_s} \cdot a_s) \cdot x + (\tfrac{1}{g_s} \cdot b_s - \tfrac{1}{g_t} \cdot b_t) \cdot y + \tfrac{1}{g_t} \cdot c_t - \tfrac{1}{g_s} \cdot c_s &= 0
\end{aligned}$$

Since $g_s, g_t, a_s, a_t, b_s, b_t, c_s$ and c_t are all constants, we know this equation is a straight line. Therefore, the additively weighted bisector consists of connected straight line segments.

Degenerate cases If g_s or g_t is 0, then the formula is not defined, but this does not occur since g_s and g_t are positive.

The equation for the bisector changes exactly at an intersection with a line from the center through a cd-vertex, and the equation is linear. Therefore, the complexity of an additively weighted bisector is $O(r)$. □

5.4 Output complexity geodesic cd-bisector

In this section, we will prove that de output complexity of a geodesic cd-bisector is $O(n + r)$.

First, we will repeat a definition as defined by Aronov (1989). Let e be an edge of the shortest path map with respect to s, and let v be the furthest endpoint of e from s. Then, extend e in increasing distance from s. The part of the extension of e that lies in the interior of W is called *the extension segment of e, emanating from v*.

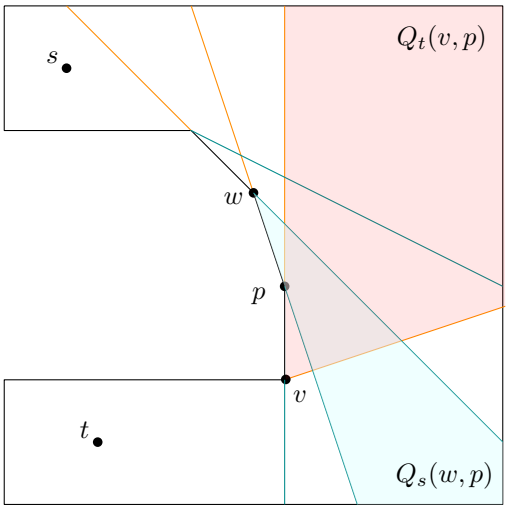


FIGURE 5.5: $Q_t(v,p)$ (red), is the region between the extension segment emanating from v and the extension segment emanating from p with respect to t. $Q_s(w,p)$ (blue), is the region between the extension segment emanating from w and the extension segment emanating from p with respect to s.

In Figure 5.5, we see a schematic representation. Let $Q_t(v,p)$ be the region between the extension segment emanating from v and the extension segment from neighboring vertex p, such v is the anchor-point from all points in $Q_t(v,p)$. Let $Q_s(w,p)$ be the region between the extension segment emanating from w, and the extension segment from neighboring vertex p, such all points in $Q_s(w,p)$ have anchor point w. We will call these regions *cones*. Note, the cones associated with a specific source point cover the entire polygon and have no overlap. So, in this case with two source points s and t, each point in the polygon W is associated with two cones, one from s and one from t.

As proven by Ma (2000), under any convex distance function, the bisector is a polygonal path with complexity $O(r)$.

Definition 5.4.1. Breakpoints induced by the vertices of the convex distance function are called *cd-breakpoints*. Breakpoints induced by extension segments are called *e-breakpoints*.

We first prove that the geodesic cd-bisector has complexity $O(r \cdot n)$, and then improve this result to $O(r+n)$.

Lemma 5.4.1. *Given R, the geodesic bisector between two points s and t has complexity $O(r \cdot n)$.*

Proof. As proven by Aronov (1989), a geodesic Voronoi edge in the Euclidean metric that is a part of the bisector between s and t has breakpoints at exactly the intersection points with extension segments. Within this region, no e-breakpoints can occur, only cd-breakpoints.

Since the bisector is connected, the bisector intersects the boundary of polygon W at two points. Between the intersection points with the boundary, the bisector intersects at most n extension segments.

Lemma 5.4.2. *The bisector can intersect each extension segment at most once, so it intersects the boundary of each cone at most twice.*

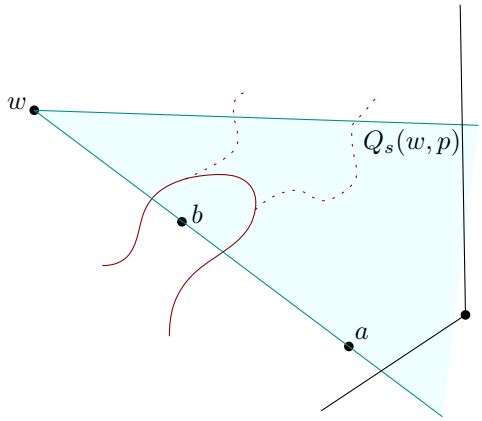


FIGURE 5.6: The bisector intersects the boundary of $Q_s(w,p)$ more than twice.

Proof. We consider cone $Q_s(w,p)$. Suppose the bisector intersects the boundary of $Q_s(w,p)$ more than twice, so it intersects an extension segment at least twice. Both variations (two intersections and more than two intersections) are drawn in Figure 5.6 in the context of Figure 5.5. Let a be a point on one of the extension segments of $Q_s(w,p)$, such that the line segment from w to a intersects the bisector twice. By definition of $Q_s(w,p)$, the shortest path from s to a passes through w. Since the Euclidean path is a shortest cd-path, we will consider the shortest path to be the straight line segment from w to a.

Vertex w is either in V_s or in V_t. First, suppose w and thus also a are in V_s. Then, the shortest path from s to a is not entirely contained within V_s, which is in contradiction with Theorem 5.2.4.

Second, suppose w and thus a are in V_t. Then, let b be a point on $\overline{wa} \cap V_s$. The shortest path from s to b goes via w, by definition of $Q_s(w,p)$. Since w is not in V_s but in V_t, the shortest path from s to b is not entirely contained in V_s, which is in contradiction with Theorem 5.2.4. Therefore, it is not possible that the bisector intersects an extension segment more than once, so it intersects the boundary of each cone at most twice. □

As proven by Aronov (1989), there are $O(n)$ extension segments, in total $O(n)$ e-breakpoints occur. So, we can split the bisector in $O(n)$ pieces. Within this piece, all breakpoints are cd-breakpoints. We will now consider a piece of the bisector within two extension segments, as shown in Figure 5.5.

We see the situation between two extension segments is correspondent with an additively weighted bisector (that is, either two extension segments from one cone, or one extension segment from a cone from site s and one extension segment from a cone from site t).

Because of Lemma 5.3.1, we know that the complexity of this piece of bisector is $O(r)$. Since the $O(n)$ extension segments cut the bisector in n pieces, and each piece has $O(r)$ cd-breakpoints, the total complexity is $O(r \cdot n)$.

□

Theorem 5.4.3 (Geodesic bisector)**.** *Under any convex distance function, the geodesic bisector between site a and b has complexity* $O(r+n)$.

Proof. Let $\Theta_1 ... \Theta_r$ be the vertices of convex distance function R, see Figure 5.7.

We will define $Q_s(w, \Theta_i)$ as the line segment from w with angle θ_i within cone $Q_s(w, p)$, with the first intersection with the boundary of W as endpoint. Because of the definition of $Q_s(w, p)$, the shortest path from s to a point on $Q_s(w, \Theta_i)$ passes w last.

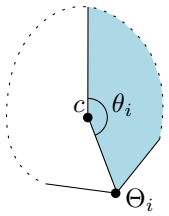


FIGURE 5.7: The convex distance function R, with vertex Θ_i at angle θ_i with respect to the half line from c up.

Lemma 5.4.4. *Θ_i can induce at most one cd-breakpoint with respect to s.*

Proof. For this proof, we will discuss the two situations that the bisector might have more than one cd-breakpoint induced by Θ_i. First, we will discuss that the bisector cannot intersect $Q_s(w, \Theta_i)$ more than once. Then, we will discuss that the bisector cannot intersect both $Q_s(w, \Theta_i)$ and $Q_s(v, \Theta_i)$, with v another vertex of polygon W.

Lemma 5.4.5. *The bisector intersects $Q_s(w, \Theta_i)$ at most once.*

Proof. For the sake of contradiction, assume the bisector intersects $Q_s(w, \Theta_i)$ more than once. For a visual representation, see Figure 5.8.

The proof is very similar to the proof of Lemma 5.4.2, so we will omit the details here. □

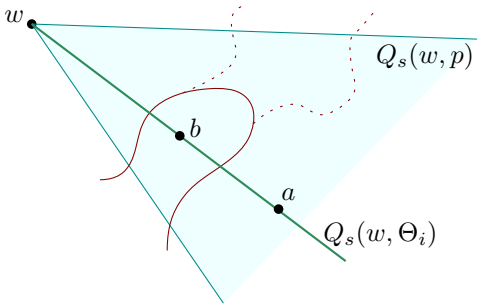


FIGURE 5.8: The bisector (red) intersects $Q_s(w, \Theta_i)$ twice, which is not possible.

Lemma 5.4.6. *The bisector cannot intersect both $Q_s(w, \Theta_i)$ and $Q_s(v, \Theta_i)$, with v another vertex of polygon W.*

Proof. Assume that $Q_s(w, \Theta_i)$ and $Q_s(v, \Theta_i)$, are intersected by the bisector. Because of Lemma 5.4.5, we know that the bisector cannot intersect the line segment twice.

First, we consider the situation that the Euclidean shortest path from s to w goes via v. Figure 5.9 shows a visual representation. Let p_v be the intersection of the bisector with the line segment $Q_s(v,\Theta_i)$. Let p_w be the intersection of the bisector with the line segment $Q_s(w,\Theta_i)$. Since the bisector is connected, p_v and p_w are connected via a path. Suppose $w \in V_s$, then v is not in V_s. That is in contradiction with Theorem 5.2.4, since the shortest path from s to w goes via v. Suppose $v \in V_s$. Since the bisector intersects the boundary of cone $Q_s(v,\Theta_i)$ and v is on the shortest path from s to w, w is not in V_s. Let a be a point on $Q_s(w,\Theta_i)$ such that the line segment $\overline{wa}$ intersects p_w. a in the same Voronoi region as v, so $a \in V_s$. But the shortest path from s to a goes via w, and $w \notin V_s$. That is in contradiction with Theorem 5.2.4. So, this situation leads to a contradiction.

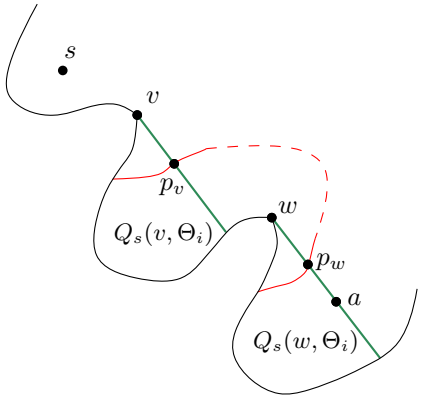


FIGURE 5.9: The bisector (red) intersects $Q_s(w,\Theta_i)$ and $Q_s(v,\Theta_i)$, if v is on the subpath from w to v.

Second, we consider the other situation that the Euclidean shortest path from s to w shares a subpath with the Euclidean shortest path from s to v, with u their last shared vertex. Figure 5.10 shows a visual representation. Let p_v be the intersection of the bisector with $Q_s(v,\Theta_i)$. Let p_w be the intersection of the bisector with $Q_s(w,\Theta_i)$. Since the bisector is connected, p_v and p_w are connected via a path. $\overline{vuw}$ divides the polygon into two subpolygons, one region containing p_v, the other containing p_w. Therefore we know that the bisector intersects either $\overline{uv}$ or $\overline{uw}$. W.l.o.g., assume it intersects $\overline{uv}$. Suppose $v \in V_s$. Since the bisector intersects $\overline{uv}$, $u \notin V_s$. This is in contradiction with Theorem 5.2.4, since u is on the shortest path from s to v. Suppose $v \notin V_s$. Let a be a point on $Q_s,(v,\Theta_i)$, such that $\overline{va}$ intersects p_v. But the shortest path from s to a goes via v, and $v \notin V_s$. That is in contradiction with Theorem 5.2.4. So, this situation also leads to a contradiction. □

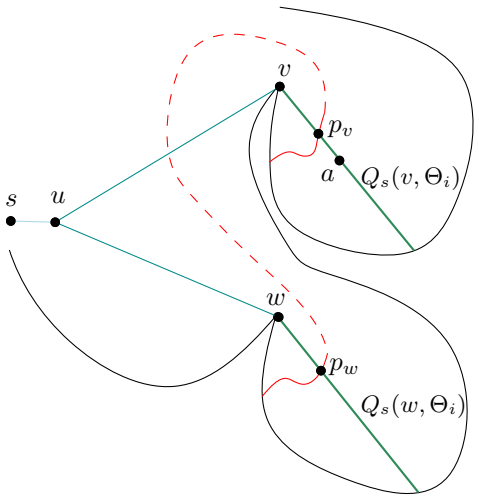


FIGURE 5.10: The bisector (red) intersects $Q_s(w,\Theta_i)$ and $Q_s(v,\Theta_i)$, if the shortest path from s to v and s to w share a subpath.

Now we discussed all situations, and they all lead to a contradiction, we can conclude that Θ_i can induce no more than one cd-breakpoint with respect to s. □

Since each vertex of the convex distance function can induce at most one cd-breakpoint with respect to a fixed source site, in total there are $O(r)$ cd-breakpoints. Because we know the number of e-breakpoints is $O(n)$, the total complexity is $O(n + r)$.

□

5.5 Geodesic cd-Voronoi diagram

Lemma 5.5.1. *A geodesic cd-Voronoi diagram for m sites consists of m Voronoi regions.*

Proof. The Voronoi cells are star-shaped, as proven in Lemma 5.2.4. Therefore, the Voronoi diagram consists of m Voronoi regions. □

Lemma 5.5.2. *The geodesic cd-Voronoi diagram has complexity $O(m(r + n))$.*

Proof. Recall, a Voronoi vertex is a vertex in the diagram that is in three or more Voronoi cells, or on the boundary of W and in two or more Voronoi cells. A Voronoi edge is a union of line segments, and does not contain Voronoi vertices. Note that cd-breakpoints and e-breakpoints have degree 2.

First we only look at the Voronoi vertices and edges. A Voronoi vertex has degree ≥ 3. Knowing that there are m Voronoi cells, with use of Euler's formula, it follows that $O(m)$ Voronoi edges exist in the Voronoi diagram.

Each Voronoi edge is a part of the bisector between two Voronoi regions, and consists of straight line segments, with a complexity of $O(r + n)$ as proven in Theorem 5.4.3. The complete Voronoi diagram of m sites then consists of $O(m(r + n))$ line segments and vertices.

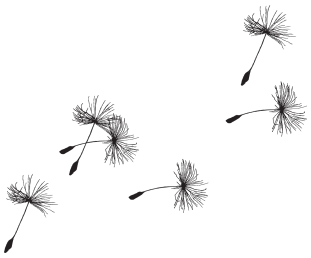
□

We can make the bound slightly tighter.

Theorem 5.5.3 (Geodesic cd-Voronoi diagram)**.** *The geodesic cd-Voronoi diagram has complexity* $O(m \cdot r + n)$.

Proof. In total, $O(m)$ Voronoi edges occur in the Voronoi diagram. Each Voronoi edge has complexity $O(r + n)$, and consists straight of line segments, with $O(r)$ cd-breakpoints and $O(n)$ e-breakpoints (Theorem 5.4.3). As Aronov (1989) proved, the total number of e-breakpoints in the Voronoi diagram is bounded above by the number of vertices of W, because the extension segments cannot intersect the boundary of the Voronoi cell at more than one point, and each extension segment is in one-to-one correspondence with the vertex it emanates from.

Thus, the complete Voronoi diagram of m sites consists of of $O(m \cdot r + n)$ line segments and vertices. □



Chapter 6

Algorithms

In this chapter, we will discuss our designed algorithms to compute an augmented cd-Voronoi diagram, an additively weighted cd-bisector, and a geodesic cd-Voronoi diagram.

6.1 Augmented cd-Voronoi diagram

The augmented geodesic Voronoi diagram is the geodesic Voronoi diagram augmented with the union of the extension segments from each site $s \in P$, but only within the Voronoi cell of s.

In the context of convex distance functions, we add *cd-extension segments* to the augmented Voronoi diagram. Cd-extension segments either emanate from sites or from vertices of W, see Figure 6.1. Cd-extension segments that emanate from a site s are defined by s as one endpoint, intersect a cd-vertex of R with center s, and are contained in the interior of W, so the other endpoint lies on the boundary of W. Cd-extension segments that emanate from a vertex v of W, are defined by v as one endpoint, intersect a cd-vertex of R with center v, and are contained in the interior of W. Next to that, a cd-extension segment that emanates from v should be in the interior of the region with anchor v, in other words, it is contained in the cone enclosed by the extension segment that emanates from v and boundary of W. Note that cd-extension segments do not intersect each other and do not intersect with extension segments.

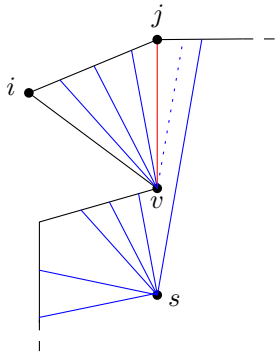


FIGURE 6.1: In this figure, red segments denote extension segments, and blue segments denote cd-extension segments. The dotted blue line segment is an invalid cd-extension segment, since it emanates from a vertex v, but it does not have anchor v. Note, for brevity, only those cd-extension segments are drawn that intersect the visible part of the boundary of the polygon.

The augmented cd-Voronoi diagram is the union of the cd-Voronoi diagram, extension segments from each site $s \in P$ within the Voronoi cell of s, and cd-extension segments that emanate from each site s and each vertex v of the boundary of W. With a total of $O(n)$ extension segments, $O(m)$ cd-extensions for each extension segment and for each site, the total complexity of an augmented cd-Voronoi diagram is $O((m+n) \cdot r)$. Note that this complexity is higher than the $O(n + m \cdot r)$ complexity of the cd-Voronoi diagram.

Construction

In this section, we give the algorithm for an augmented cd-Voronoi diagram, in the basic case that P consitsts of one site.

If P consists of only one site s, we can compute the extension segments in linear time by using the shortest path map (Guibas et al., 1986), and store the anchor points of each vertex, such that we can access the anchor point of a vertex in constant time. To calculate the cd-extension segments, we will perform a rotational sweep in clockwise order over the edges of W extended with the extension segments.

For each edge $\overline{ij}$, with vertices i and j:

1. Retrieve the anchor point v of $\overline{ij}$ in constant time. Note, the interior of $\overline{ij}$ has one anchor point, that can also be site s.
2. Let α_i be the angle of vertex i relative to anchor point v, and let α_j be the angle of vertex j relative to anchor point v.
3. Add the cd-extension segments that emanate from v and have an angle in the interval $[\alpha_i, \alpha_j]$. The other endpoint intersects with $\overline{ij}$.

For each edge of W, we spend $O(r)$ time, since the search for the first cd-extension segment in step 3 takes $O(\log r)$ time, and $O(r)$ cd-extension segments are added. So, the construction of the augmented cd-Voronoi diagram takes $O(n \cdot r)$ time for one site in W.

6.2 Additively weighted cd-bisector

As we investigated in Section 5.3, an additively weighted bisector consists of $O(r)$ cd-breakpoints. In this section, we will elaborate on the algorithm to compute the bisector. Since we can construct the equation of the bisector in between two extension segments, we can trace it once we have a starting point. We assume the anchor points do not change. The anchor cd-edges change exactly at the intersection with cd-extension segments. Then, a new equation should be formed to calculate the next bisector segment. In this way, we can trace the bisector. Therefore, we will provide multiple methods to find the starting point here.

Find starting point

In the classic version, we know that the cd-vertex at the top induces a part of the bisector, and we can draw the horizontal line to obtain the relevant cd-polygon point from the other site. This is not the case for the additive variant.

We know that the geodesic bisector is connected, and that it divides the polygon into two regions, with s and t included in their corresponding region. Therefore, we can

find the intersection point between the bisector and the shortest path between the two sites (which is a straight line segment). We need to look up the corresponding cd-edges that define the bisector piece, and continue the bisector construction.

Time complexity

The output complexity of an additively weighted bisector is $O(r)$ as proven in Theorem 5.3.1.

The starting point can be found in $O(\log r)$ time, by first using binary search to find the cd-edge of s that intersects $\overline{st}$, and then using binary search to find the cd-edge of t that intersects $\overline{st}$. In constant time, we can construct the linear equations and find the intersection point of the bisector with $\overline{st}$.

We can find the next cd-breakpoint of the bisector in constant time. Since the bisector has $O(r)$ cd-breakpoints, the construction of the bisector takes $O(r)$ time.

Bisector construction

In Section 5.3, we calculated the bisector using equations, but we can also geometrically construct the bisector. We will write ℓ_{XY} to denote the line through point X and Y.

Consider point s, with corresponding cd-edge $\overline{\Theta_0}$ with cd-vertices Θ_0 en Θ_1, and consider point t with corresponding cd-edge $\overline{\Theta_2}$ with cd-vertices Θ_2 en Θ_3. See Figure 6.2. Suppose we are given that the half-line from s through Θ_1 intersects a segment of the bisector that is induced by the cd-edges $\overline{\Theta_0}$ and $\overline{\Theta_2}$.

We draw the next step in time for both s and t, which is respectively $\overline{\Theta'_0}$ from Θ'_0 to Θ'_1, and $\overline{\Theta'_2}$ from Θ'_2 to Θ'_3. Draw line $\ell_{\Theta_0\Theta_1}$ and line $\ell_{\Theta_2\Theta_3}$, with intersection point p. Draw line $\ell_{\Theta'_0\Theta'_1}$ and line $\ell_{\Theta'_2\Theta'_3}$, with intersection point q. We draw a line b through the intersection points p and q of these two lines. The intersection point between b and $\ell_{s\Theta_1}$ is on the bisector.

Degenerate case: If the cd-edges are parallel, there is no intersection point. In that case, we can easily find the intersection point using the equations.

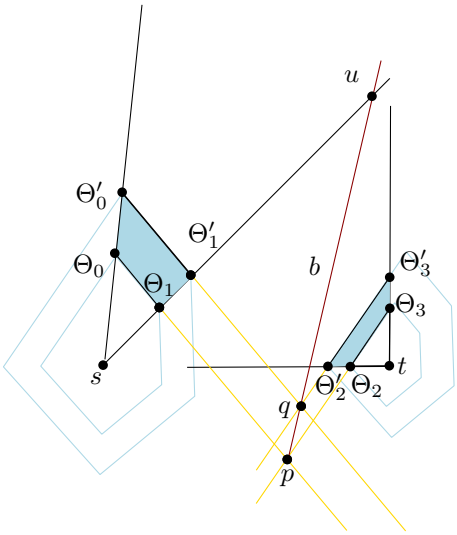


FIGURE 6.2: u is on the bisector.

6.3 Geodesic cd-Voronoi diagram

If we want to use a convex distance function instead of the Euclidean distance measure, we need to alter the algorithm for a geodesic Voronoi diagram. In this section, we will give the adapted version for a geodesic cd-Voronoi diagram by Aronov (1989) as described in Section 3.4.1.

Overview algorithm

Let $Vor_W(P)$ denote the geodesic cd-Voronoi diagram of the set of sites P, and let $Vor^*_W(P)$ be $Vor_W(P)$ augmented with the union of the extension segments from each site $s \in P$, and cd-extension segments from each site s and each vertex v of the boundary of W, but only within the Voronoi cell of s. $V(s, W)$ is the Voronoi cell of site s in polygon W.

Preprocessing

We triangulate W in $O(n)$ time (Chazelle, 1991). We preprocess the triangulated polygon for point location queries in $O(n)$ time and for each site we query its corresponding triangle, in $O(\log n)$ time per query (Edelsbrunner et al., 1986), with a total of $O(n + m \log n)$ time. Then we compute a balanced decomposition of the triangulation tree in $O(n)$ time (Guibas et al., 1986), such that we can cut up the polygon recursively in two parts in constant time per cut, where each part has at least a quarter of the vertices.

Recursive algorithm

We are given a (sub)polygon W' with $n' \leq n$ vertices with a set of sites $P' \subseteq P$ of size $m' \leq m$.

i. In the base case of the recursive divide-and-conquer algorithm, if P' consists of only one site s, then $Vor_{W'}(P')$ is a single cell $V(s, W') = W'$. We can compute the extension segments in $Vor^*_{W'}(P')$ in linear time by using the shortest path map (Guibas et al., 1986), and we can add the cd-extension segments in $O(n' \cdot r)$ time with the construction algorithm as given in the previous section; otherwise

ii. If W' is a triangle, then the cd-Voronoi diagram can be computed in $O(rm' \log m')$ time (Ma, 2000) and truncated to W' in linear time. Add $O(r)$ cd-extension segments per site s, and truncate them to the corresponding Voronoi cell of s; otherwise

iii. We divide W' in two subpolygons W_L and W_R, and divide P' in P_L and P_R, such that $P_L \subset W_L$ and $P_R \subset W_R$, using the balanced decomposition as described above. This can be done in constant time (Aronov, 1989). Then, we recursively compute $Vor^*_{W_L}(P_L)$ and $Vor^*_{W_R}(P_R)$.

iv. Then, we extend $Vor^*_{W_L}(P_L)$ to $Vor^*_{W'}(P_L)$, and $Vor^*_{W_R}(P_R)$ to $Vor^*_{W'}(P_R)$ in $O\big(((n' + m') \cdot r) \log((n' + m') \cdot r)\big)$ time.

v. At last, we compute $Vor^*_{W'}(P')$ by merging $Vor^*_{W'}(P_L)$ and $Vor^*_{W'}(P_R)$ in $O((n' + m') \cdot r')$ time.

Preprocessing takes $O(n + m \log n)$ time. The recursive part splits into two subproblems of size $(\alpha n, m_1)$ and $((1 - \alpha)n, m_2)$, with $m_1 + m_2 = m$, and $\alpha \in [\frac{1}{4}, \frac{3}{4}]$. r is fixed

throughout the algorithm. Note, step ii. occurs $O(m)$ times, since P' consists of at least one site in that step.

Therefore, using recurrence inequality bounds, the time complexity of calculating the cd-Voronoi diagram is

$$O\big(((n+m)\cdot r)\log((n+m)\cdot r)\log n\big).$$

Extension phase

We cut the polygon W' in two subpolygons W_1 and W_2, separated by a chord e of the triangulation, and we are given $Vor^*_{W_1}(P_1)$. Let $W_1 \cup W_2$ have n' vertices, and P_1 m' sites. The output of this subroutine is $Vor^*_W(P_1)$.

1. i. First, we extract the ordered list L of the regions in $Vor^*_{W_1}(P_1)$ that are adjacent to e, and convert this list to a search tree T.

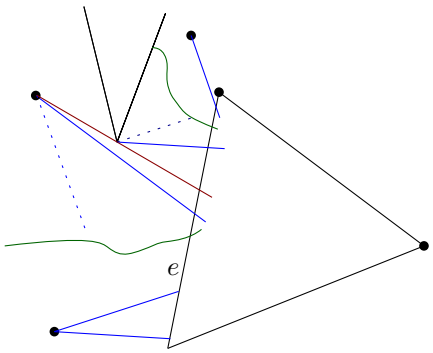


FIGURE 6.3: In this figure, the green segments denote bisector edges, red segments denote extension segments, and blue segments denote cd-extension segments. The dotted blue cd-extension segments are not included in $Vor^*_{W_1}(P)$, because they do not intersect the triangle between two bisector edges. Their corresponding regions have disappeared already.

   ii. For each pair of adjacent edges in L, either two adjacent bisector edges, a bisector edge and an extension segment or a bisector edge and a cd-extension segment, we compute the intersection point, without checking its feasibility. Note, neighbouring extension segments or cd-extension segments cannot intersect. We perform a point location query to determine the triangle it lands in and add the intersection point to the bucket of the corresponding triangle.

2. We process all the triangles in W_2, starting at the triangle adjacent to e, followed by its children, and so on. We enter triangle Δ through edge f, whose remaining edges are f' and f''. Let v be the vertex shared by f' and f''. For a visual representation, see Figure 6.4.

i. Let x be the common endpoint of f and f'. Locate the anchor y of the region that x belongs to. Consider the ray from x directed away from y. If this ray intersects or overlaps with f, create a new region between the new extension segment and f'. Then, determine the cd-extension segments in the new region with anchor point x, and create new regions sandwiched between the ray and the cd-extension segment, and between the cd-extension segments. See a visual reference in Figure 6.4.

   Do the same for the other endpoint of f.

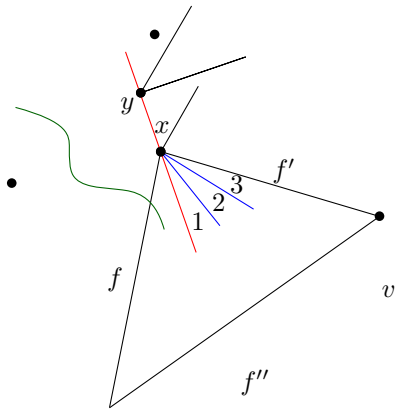


FIGURE 6.4: In this figure, the green segments denote bisector edges, red segments denote extension segments, and blue segments denote cd-extension segments. We add here the new regions 1, 2, and 3, sandwiched between the extension segment through y and x, and f'.

ii. We construct a priority queue Q for triangle Δ for all intersections in its bucket. We perform a sweep line that is parallel with f and ends at the edges of Δ, and handle the events in increasing distance from f.

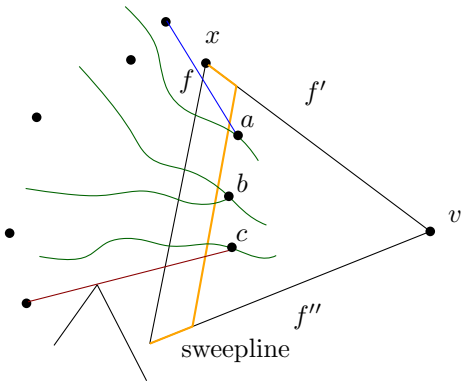


FIGURE 6.5: In this figure, the green segments denote bisector edges, red segments denote extension segments, and blue segments denote cd-extension segments. a is an intersection point between a bisector edge and a cd-extension segment, b is an intersection point between two bisector edges, and c is an intersection between a bisector edge and an extension segment.

iii. We can encounter three types of intersection points, see Figure 6.5.

1) Bisector-bisector intersections: If the intersection point p is valid, that is an intersection of two edges that currently bound a region together in T, a region in the Voronoi diagram disappears and a new edge appears. This event corresponds with a Voronoi vertex. The new edge has an updated anchor point and anchor cd-edge. We compute two intersections with the new pairs of adjacent edges (that can be bisector-bisector, bisector-extension segment or bisector-cd-extension segment), and locate the corresponding triangle of the intersection point. If they belong to the current Δ, they are added to Q, otherwise they are added to the bucket of the corresponding triangle. If p is non-valid, we ignore and delete it.

2) Bisector-extension segment intersections: If the intersection point p is valid, a region in T disappears and a new edge appears. This event corresponds with an e-breakpoint. The new edge has an updated anchor point. We compute two intersections with the new pairs of adjacent edges (that can be bisector-bisector, bisector-extension segment or bisector-cd-extension segment), and locate the corresponding triangle of the intersection point. If they belong to the current Δ, they are added to Q, otherwise they are added to the bucket of the corresponding triangle. If p is non-valid, we ignore and delete it.

3) Bisector-cd-extension segment intersections: If the intersection point p is valid, a region in T disappears and a new edge appears. This event corresponds with a cd-breakpoint. The new edge has an updated anchor cd-edge. We compute two intersections with the new pairs of adjacent edges (that can be bisector-bisector, bisector-extension segment or bisector-cd-extension segment), and locate the corresponding triangle of the intersection point. If they belong to the current Δ, they are added to Q, otherwise they are added to the bucket of the corresponding triangle. If p is non-valid, we ignore and delete it.

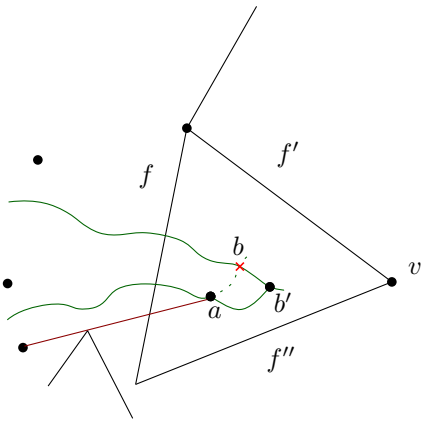


FIGURE 6.6: In this figure, the green segments denote bisector edges and red segment denotes an extension segment. a is an intersection point between a bisector edge and an extension segment. This intersection point caused the previously calculated intersection b to be invalid. b' is the new intersection point that is added to the queue.

iv. If we finish triangle Δ, the trees for f' and f'' are produced by conceptually splitting the tree at v.

We leave out the details of degenerate cases here for brevity.

Time complexity extension phase

We will discuss the time complexity for the extension algorithm that computes the augmented cd-Voronoi diagram. The output complexity is $O((n' + m') \cdot r)$.

- Step 1.i: We extract the list of the regions that are adjacent to the separating chord e, which takes linear time.
- Step 1.ii: We perform point location queries that take $O(\log n')$ per query. Since we only add bisector-bisector, bisector-extension segment and bisector-cd-extension segment intersection, we add $O(m')$ intersections.

For the substeps in step 2, we will bound the time per candidate intersection. First, we will discuss the number of candidate intersections.

All $O((n' + m') \cdot r)$ vertices that are included in the outputted augmented cd-Voronoi diagram have been added as candidate intersection. We also add intersections that later prove to be invalid. Each intersection (bisector-bisector, bisector-extension segment and bisector-cd-extension segment) will cause two candidates to become invalid, namely the candidate intersections with their neighbours. Therefore, the total number of candidate intersections is $O((n' + m') \cdot r)$.

Since their are $O((n' + m') \cdot r)$ candidate intersections, the total size of all the queues and trees is bounded by $O((n' + m') \cdot r)$.

- Step 2.i.: We spend $O(\log((n' + m') \cdot r))$ time per tree addition, since $O(|T|) = O((n' + m') \cdot r)$. The total number of tree additions is bounded by the number of added regions, which is $O((n' + m') \cdot r)$.
- Step 2.ii.: We construct a queue, and notice that further queue operations take $O(\log((n' + m') \cdot r))$ time, since no intersection appears in more than one queue.
- Step 2.iii.: tree deletion and tree addition takes $O(\log((n' + m') \cdot r))$ time per candidate intersection. Point location queries take $O(\log n')$ time per candidate intersection.
- Step 2.iv: Splitting and producing the trees takes $O(\log((n' + m') \cdot r))$ time.

In total, since the total number of candidate intersections is $O((n' + m') \cdot r)$ and the time spent per candidate intersection is $O(\log((n' + m') \cdot r)$, the extension phase has time complexity

$$O\big(((n' + m') \cdot r) \log((n' + m') \cdot r)\big).$$

Merge phase

In this phase of the algorithm, we are given two sets of sites P_L and P_R (with a chord of the polygon separating them), and $Vor^*_W(P_L)$ and $Vor^*_W(P_R)$.

Suppose $s \in P_L$. Let $V(W, s)$ denote the Voronoi cell of s in $Vor_W(P_L)$ and $V'(W, s)$ the Voronoi cell of s in $Vor_W(P_L \cup P_R)$. Let $H(P_L, P_R)$ be the region in W with all points

closer to a site in P_L than a site in P_R. Let $b(P_L, P_R)$ be the bisector between P_L and P_R. Then,

$$V'(W,s) = V(W,s) \cap \big(H(P_L,P_R) \cup b(P_L,P_R)\big)$$

To find the points of intersection between $b(P_L, P_R)$ and the boundary of W, we divide the boundary in $O((n+m)\cdot r)$ segments, such that each segment has one corresponding site, anchor point and anchor cd-edge for both sets P_L and P_R. Then, we can calculate the distance from the boundary segment to the closest site in P_L and P_R easily. We can find in linear time what points on the bisector have the same distance to the closest site in P_L and the closest site in P_R.

$b(P_L, P_R)$ consists of pieces of bisector between two sites. Given a point of $b(P_L, P_R)$, we can trace the bisector of P_L and P_R, since we know the first two sites, anchor points and anchor cd-edges that define the bisector. The anchor points only change when the bisector intersects with an extension segment, and the anchor cd-edges only change when encountering a cd-breakpoint. After such an event, we can trace the bisector with the new anchor points or anchor cd-edges further in the same way, where each step takes constant time.

In conclusion, the time complexity of the merge phase is

$$O((n'+m')\cdot r').$$

To summarize the whole section, the total time to compute the augmented cd-Voronoi diagram is

$$O\big(((n+m)\cdot r)\log((n+m)\cdot r)\log n\big).$$

6.3.1 Lazy approach cd-Voronoi diagram

As mentioned before, the complexity of the augmented cd-Voronoi diagram is higher than the cd-Voronoi diagram without the cd-extension segments. We do not need to construct all cd-extension segments to construct the Voronoi diagram. In this section, we will discuss a "lazy approach" to calculate the geodesic cd-Voronoi diagram, based on the adapted version of Aronov's algorithm given in the previous section.

If we look again at Figure 6.4, we see that we add three new regions induced by cd-extension segments. We observe that the green bisector segment can never intersect blue cd-extension segment 1 before intersecting the red extension segment. And again, the bisector segment can only intersect the second cd-extension segment once it intersected the first cd-extension segment.

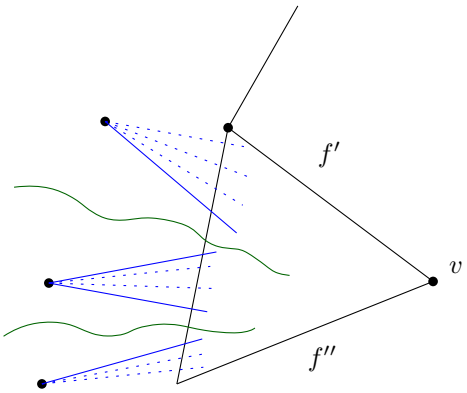


FIGURE 6.7: In this figure, the green segments denote bisector edges and blue segments denote cd-extension segments. The dotted cd-extension segments cannot intersect the bisector edge before the neighbouring cd-extension segment intersect the bisector edge.

The general observation is that bisector edges can only intersect cd-extension segments that are neighbours. If we look at Figure 6.7, we see two green bisector edges, and blue cd-extension segments. Only the cd-extension segments that are direct neighbours of a bisector edge need to be computed. The other (dotted) cd-extension segments that are enclosed by other cd-extension segments can never intersect with a bisector segment before the neighbouring cd-extension segments intersect the bisector segment. Therefore, we only need to calculate two cd-extension segments per bisector edge.

We will assume we can obtain the neighbours of a cd-edge in constant time, so given an angle α_i, we can obtain angle α_{i-1} and α_{i+1} in constant time, for example using a linked list (Sundell and Tsigas, 2008). Also, we will assume that we store the anchor cd-edge of a bisector edge during the algorithm. Combining both, we can obtain the neighbouring cd-extension segments of a bisector edge in constant time.

Overview

The lazy approach requires some adjustments to the algorithm. In the recursive algorithm, we do not have to add cd-extension segments in step i, so it takes linear time in the number of regions, which is $O(n' + m'r)$. In step ii., we only add two neighbouring cd-extensions per bisector edge that intersects the boundary of the triangle, such that no cd-extension segment intersects the boundary closer to the bisector edge, see Figure 6.8. In combination with the computation of the cd-Voronoi diagram, this takes $O(rm' \log m')$ time.

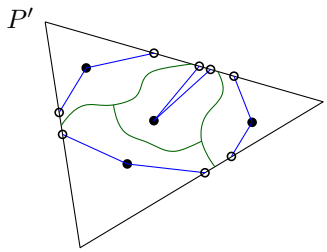


FIGURE 6.8: In this figure, the green segments denote bisector edges and blue segments denote cd-extension segments. Per bisector edge, we add only two cd-extension segments, and calculate the intersections with the triangle P' (black circles).

As we will discuss in detail further, the extension phase will take $O((n' + m' \cdot r) \log(n' + m' \cdot r))$ time, and the merge phase takes $O(n' \log r + m' \log^2 r)$. time. The preprocessing phase still takes $O(n + m \log n)$ time.

Therefore, using recurrence inequality bounds, the time complexity of calculating the cd-Voronoi diagram using the lazy approach is

$$O\big((n + m \cdot r) \log(n + m \cdot r) \log n\big).$$

Note that the original running time was $O\big((n \cdot r + m \cdot r) \log(n \cdot r + m \cdot r) \log n\big)$.

Extension phase

In the extension phase, we can leave step 1 unchanged. In step 2.i., we only add a new region induced by the extension segment, and do not add new regions induced by cd-extension segments. Step 2.ii. does not need to be changed.

The greatest changes will take place in step 2.iii, where we handle the three types of intersection points. We only write down the differences between the original version and the lazy approach.

1) Bisector-bisector intersections: No adaptation required.
2) Bisector-extension segment intersections: This event corresponds with an e-breakpoint. The new edge has an updated anchor point. We will add the next neighbouring cd-extension segment that is closest to the extension segment and emanates from the new anchor point. We create a new region sandwiched between the extension segment and the cd-extension segment. Next, we will compute two intersections for both the new bisector edge and its neighbours (that can be bisector-bisector, bisector-extension segment or bisector-cd-extension segment), and the cd-extension segment and its neighbours (only bisector-cd-extension segment). Just like the original version, we do the appropriate point location queries.
3) Bisector-cd-extension segment intersections: This event corresponds with a cd-breakpoint. The new edge has an updated anchor cd-edge. We will add the next neighbouring cd-extension segment that is closest to the cd-extension segment and emanates from the current anchor point. We create a new region sandwiched between the two cd-extension segments. Next, we will compute two intersections for both the new bisector edge and its neighbours (that

can be bisector-bisector, bisector-extension segment or bisector-cd-extension segment), and the cd-extension segment and its neighbours (only bisector-cd-extension segment). Just like the original version, we perform the appropriate point location queries.

Step 2.iv. is unchanged.

Complexity extension phase lazy approach

We will discuss the time complexity for the extension algorithm that computes the cd-Voronoi diagram, using the lazy approach. Note, the output complexity is $O(n' + m' \cdot r)$ instead of the output complexity of $O((n' + m') \cdot r)$ in the original version.

We will discuss the number of candidate intersections. All $O(n' + m' \cdot r)$ vertices that are included in the outputted cd-Voronoi diagram have been added as candidate intersection point. Each intersection (bisector-bisector, bisector-extension segment and bisector-cd-extension segment) still causes two candidates to become invalid, namely the candidate intersections with their neighbours. Therefore, the total number of candidate intersections is $O(n' + m' \cdot r)$. This is an improvement in comparison to the $O((n' + m') \cdot r)$ candidate intersections in the original version.

Since their are $O(n' + m' \cdot r)$ candidate intersections, the total size of all the queues and tree is bounded by $O(n' + m' \cdot r)$. Therefore, all queue maintenance operations and tree additions and deletions have a bound of $O(\log(n' + m' \cdot r))$ time.

The rest of the analysis of the complexity is similar to the previous algorithm. We can conclude, in total, since the total number of candidate intersections is $O(n' + m' \cdot r)$ and the time spent per candidate intersection is $O(\log(n' + m' \cdot r))$, the extension phase takes

$$O\big((n' + m' \cdot r)\log(n' + m' \cdot r)\big)$$

time.

Merge phase

We are given two sets of sites P_L and P_R, and $Vor^*_W(P_L)$ and $Vor^*_W(P_R)$, note only with extension segments.

1. To find the points of intersection of $b(P_L, P_R)$ and the boundary of W, we divide the boundary in $O(n + m)$ pieces, such that each piece is associated with a unique closest site from P_L and corresponding anchor point and a unique closest site from P_R and corresponding anchor point. A piece can have multiple anchor cd-edges.

2. For the $O(n+m)$ endpoints of the pieces, we calculate the distance to the closest site s and t in respectively P_L and P_R. The anchor point for the closest site is known. We only need to find the corresponding anchor cd-edge in $O(\log r)$ time for both sites s and t. Then we can access the distance to the closest site s and t in constant time.
3. We trace the boundary of W. For each piece, with endpoints i and j:
   (a) If i and j are both closer to s or t, then the bisector does not intersect $\overline{ij}$, because the bisector intersects $\overline{ij}$ at most once.
   (b) If s is closer to i and t is closer to j, or the other way around, then we know that the bisector intersects $\overline{ij}$.
4. In case 3(b), we need to compute the intersection point between the bisector of s and t and $\overline{ij}$. Let a_s and a_t be the anchor points of respectively s and t, which is fixed for all points on $\overline{ij}$. We perform a binary search on the cd-extension segments that emanate from a_s and intersect $\overline{ij}$. For each cd-extension segment, we want to find the corresponding cd-edge from a_t that induce a piece of the bisector. We perform a binary search on the cd-edges from t, and compute the equations as constructed in Section 6.2 in constant time. If we found the appropriate pair of cd-edges in $O(\log r)$ time, we test if the piece of bisector intersects with $\overline{ij}$. It takes $O(\log r)$ tests to find the intersection. The double binary search therefore takes $O(\log^2 r)$ time.

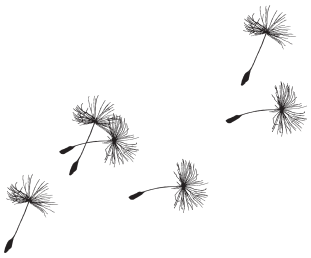
Step 1 takes $O(n'+m')$ time. Step 2 takes $O(\log r)$ time and occurs $O(n'+m')$ times, so takes a total of $O((n'+m')\log r)$ time. In step 3, we can trace the boundary of the polygon in $O(n'+m')$ time. How often do we need to perform the double binary search in step 4? The number of intersections of $b(P_L, P_R)$ with the boundary, is bounded by the degree 1 vertices in the Voronoi diagram. In other words, we need to know how many intersections occur between Voronoi edges and the boundary of the polygon. Recall that a Voronoi edge is a part of the bisector between two Voronoi regions that consists of straight line segments, joined by cd-breakpoints and e-breakpoints. As proven in Lemma 5.5.3, a geodesic cd-Voronoi diagram has in total $O(m)$ Voronoi edges. Therefore, we perform the double binary search $O(m')$ times. The double binary search takes $O(\log^2 r)$ time, so step 4 takes a total of $O(m' \log^2 r)$ time.

So, the time complexity of the merge phase in the lazy approach is $O(n' \log r + m' \log^2 r)$.

In summary, the total time complexity of calculating the cd-Voronoi diagram using the lazy approach is

$$O\big((n + m \cdot r)\log(n + m \cdot r)\log n\big).$$

Chapter 7



Queries

In the previous chapter, we discussed how to construct a geodesic Voronoi diagram under a convex distance function based on an r-gon. In this chapter, we will discuss some queries we can perform, given that we calculated the cd-Voronoi diagram of a set of sites P in polygon W.

7.1 Distance to closest site

Suppose, we are given the cd-Voronoi diagram of P in W. The first query we discuss is, given a point p in W, what is the time that p is covered? This question is equivalent to the distance to p from site s that is closest to p. To find the closest site s, we need to build a data structure that supports point location queries. With $O(mr+n)$ edges in the cd-Voronoi diagram, it would take $O((mr+n)\log(mr+n))$ time to build such a data structure (Mulmuley, 1990). A query takes $O(\log(mr+n))$ time. Once the closest site is found, we can retrieve the anchor point a of p in constant time. It takes $O(\log r)$ time to look up the corresponding anchor cd-edge. Then, we can calculate the convex distance $\hat{d}(s,p) = \hat{d}(s,a) + \hat{d}(a,p)$ in constant time.

Thus, the time complexity of preprocessing is $O((mr+n)\log(mr+n))$, and the time complexity of a query is $O(\log(mr+n))$.

The time complexity does not improve if we assume the augmented cd-Voronoi diagram is given as input. We do not have to spend $O(\log r)$ to look up the corresponding anchor cd-edge. Unfortunately, since the complexity of an augmented cd-Voronoi diagram is $O((n+m)\cdot r)$, preprocessing takes longer, and the total query time is $O(\log((n+m)\cdot r))$.

7.2 All points at given distance

The second query we discuss is, given a time $t > 0$, what region B_t is covered by plants? B_t corresponds with all points in W that have distance at most t to some site in P. We first prove the output complexity, and then provide an algorithm to compute the query under a convex distance function. In the next section, we will mention the required adjustments to perform the query under a Euclidean distance measure.

Output map complexity

In this section, we will give the output complexity of B_t. We first prove the lower bound and then the upper bound of B_t for some t.

All m sites can induce a region of $\Omega(r)$ complexity, as is clear in the case that B_t consists of m disconnected regions. Furthermore, all n vertices can induce $\Omega(r)$ cd-breakpoints, as can be seen in Figure 7.1.

Therefore, the worst-case complexity is $\Omega((n+m)\cdot r)$.

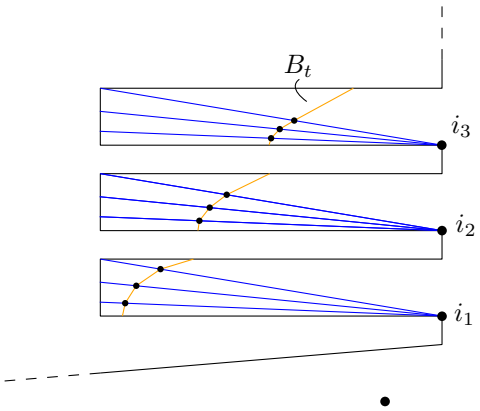


FIGURE 7.1: The blue segments denote cd-extension segments. i_1, i_2 and i_3 are anchor points. In this construction, it can be seen that B_t can have $O(r)$ cd-breakpoints per anchor point.

We will prove that the number of intersections between regions induced by different sites is bounded by the number of edges in the augmented cd-Voronoi diagram. These intersections occur at the boundary of the Voronoi cell, since the boundary of a Voronoi cell consists of all the points that are equally distant to the sites of the corresponding Voronoi cells.

Lemma 7.2.1. *Let i and j be two neighbouring vertices of the Voronoi cell of site s. Then the boundary of B_t intersects $\overline{ij}$ at most once.*

Proof. Let Δ be the triangle enclosed by vertices i, j, the anchor point a_i of i and the anchor point a_j of j. The vertices of Δ are i, j, and either a_i or a_j. All points within this triangle have the same anchor point and the anchor cd-edge, since they only change at the intersection with extension segments and cd-extension segments. Therefore, the equation of the distance from s to a point p is the same for all $p \in \Delta$. The equation of B_t is all points p such that $\hat{d}(s,p) = t$, which is a linear equation. Therefore, the piece of B_t within Δ is a straight line segment, and intersects $\overline{ij}$ at most once. □

Because of Lemma 7.2.1, B_t has $O((n+m)\cdot r)$ intersections with the edges of the Voronoi cells. B_t has breakpoints exactly at the intersection with cd-extension segments and extension segments. Therefore, the upper bound of the complexity of B_t is $O((n+m)\cdot r)$. Therefore, the complexity of B_t is

$$\Theta((n+m)\cdot r).$$

Overview algorithm

We are given the augmented cd-Voronoi diagram of sites P in W. Note that this includes both extension segments and cd-extension segments.

For each site s in P, we perform a rotational sweep on the vertices of its Voronoi cell, $V(s)$, to test whether the vertex should be included in B_t or not. Let i and j be two

neighbouring vertices of $V(s)$, and let $\gamma(s,i)$ be the shortest path from s to i. We first explain how to find a starting point.

Find starting point

Case i: We encounter a vertex i with $\hat{d}(s,i) \leq t$ and a vertex j with $\hat{d}(s,j) > t$. Then the boundary of B_t intersects $\overline{ij}$ exactly once, see Lemma 7.2.1. Since all points on the edge $\overline{ij}$ have the same anchor point and the anchor cd-edge, we can create the equation of the line segment of B_t within that cone in constant time, such that for all points p on the line, $\hat{d}(s,p) = t$. Then, we can calculate the intersection with $\overline{ij}$ in constant time. Note: If we first encounter a vertex i with $\hat{d}(s,i) > t$, followed by a vertex j with $\hat{d}(s,j) \leq t$, the procedure is similar; otherwise

Case ii: The first two vertices we encounter are neighbouring vertices i and j of $V(s)$ with $\hat{d}(s,i) \leq t$ and $\hat{d}(s,j) \leq t$. Then B_t does not intersect $\overline{ij}$, thus $\overline{ij}$ is fully contained in B_t, see Lemma 7.2.1, and see Figure 7.2. We continue the rotational sweep until we find a situation as described in case i. If there is no such situation, $V(s)$ is entirely contained in B_t, as can be seen in Figure 7.3. None of the edges of $V(s)$ will appear in the output; otherwise

Case iii: We find two vertices i and j with $\hat{d}(s,i) > t$ and $\hat{d}(s,j) > t$. To find a starting point, we will trace the shortest path from s to i, $\gamma(s,i)$, to find the intersection between B_t and $\gamma(s,i)$. $\gamma(s,i)$ intersects the boundary of B_t at most once, see Lemma 7.2.2. It is possible that B_t does not intersect the boundary of $V(s)$, see Figure 7.4. We trace back the vertices on $\gamma(s,i)$, until we find a vertex k with $\hat{d}(s,k) \leq t$. Let k be the anchor of vertex m that is also on $\gamma(s,i)$. k is included in B_t, but m is not. Then we find the intersection between $\overline{km}$ and B_t, which is the point at distance t from s, in constant time.

Rotational sweep

1. Once a starting point is known, we can trace the boundary of B_t. The boundary of B_t consists all points p such that $\hat{d}(s,p) = t$. As long as the anchor point and anchor cd-edge stay the same, the piece of the B_t is a straight line segment, that changes exactly at the intersection with extension segments and cd-extension segments.
2. If B_t intersects the boundary of W again, we continue the rotational sweep line, looking for the next starting point.
3. If we finish the rotational sweep of $V(s)$, by either finishing all vertices of $V(s)$ or closing the region of B_t around s, we know there is no other disconnected region of B_t within $V(s)$ because of Theorem 7.2.2. Therefore, we can continue to the Voronoi cell of the next site.

.

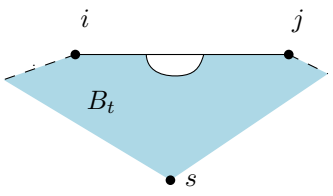


FIGURE 7.2: B_t intersects $\overline{ij}$ at most once. If vertex i and j are contained in B_t, then the complete edge $\overline{ij}$ is contained in B_t.

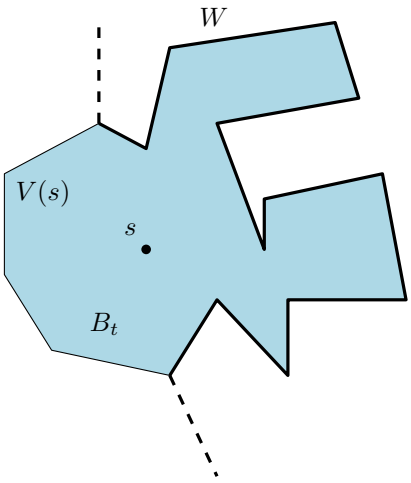


FIGURE 7.3: In this situation, all vertices of $V(s)$ are contained in B_t. None of the edges of $V(s)$ will appear in the output, since it is in the middle of B_t.

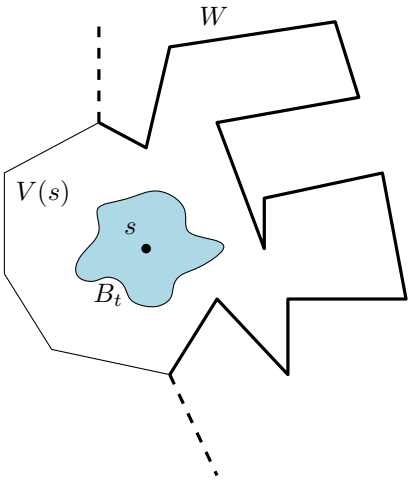


FIGURE 7.4: In this situation, none of the vertices of $V(s)$ are contained in B_t.

Lemma 7.2.2. *Let i be a vertex of the Voronoi cell of s. The boundary of B_t intersects the shortest path from s to i at most once.*

Proof. Following the shortest path to i away from s, the distance is a monotone increasing function. Therefore, there is at most one point p on the path from s to i such that $\hat{d}(s,p) = t$ and that p lies on the boundary of B_t.

If $\hat{d}(s,i) \leq t$, then, i is part of B_t, so the boundary of B_t does not intersect the shortest path from s to i. If $\hat{d}(s,i) > t$, s is contained in B_t since $t > 0$, so the boundary of B_t intersects the shortest path from s to i exactly once.

□

Time complexity

Theorem 7.2.3. *Given the augmented cd-Voronoi diagram, the optimal time complexity to compute B_t is $O((n+m)\cdot r)$.*

Proof. In the algorithm to find the starting point, case i. occurs $O((n+m)\cdot r)$ times, and takes constant time per step. Also case ii. occurs $O((n+m)\cdot r)$ times, the number of edges in the augmented Voronoi diagram. Case ii. itself takes constant time.

Case iii, tracing the shortest path, could take $O((n+m)\cdot r)$ time, since that many vertices exist in the augmented Voronoi diagram. A vertex is shared by at most three Voronoi cells, which means a vertex can occur at most three times in a trace. Therefore, all traces together take $O((n+m)\cdot r)$ time.

Now we will discuss the time complexity of the rotational sweep. In step 1., as long as the anchor point and anchor cd-edge stay the same, constructing a piece takes constant time. These change exactly at the intersection with extension segments and cd-extension segments, that occurs in total $O((n+m)\cdot r)$ times. Therefore, step 1 stakes $O((n+m)\cdot r)$ time. Step 2 takes constant time, and occurs $O((n+m)\cdot r)$ times.

Therefore, the time complexity of the algorithm is $O((n+m)\cdot r)$. Since the worst-case output complexity is $\Omega((n+m)\cdot r)$, we can conclude the time complexity is optimal. □

Different input

If the input consists of the cd-Voronoi diagram without extension segments and cd-extension segments, we can easily augment the cd-Voronoi diagram using the algorithm as given in Section 6.1. Since that would take $O((n+m)\cdot r)$ time, the time complexity of the query algorithm would not change, and is thus still optimal.

7.3 All points at multiple distances

In this section, we will return to the problem as formulated in Section 4.2. We assume that we are given the augmented cd-Voronoi diagram as input.

Given a time t, what is plant region $B_{1,2,\dots,t}$, and what age do the plants have? Assume all sites p have age 1 at time 0. The output map is a subdivision $\mathcal{SD}_t$ of W, where each region has a corresponding age with value $[0, t+1]$.

B_i corresponds with the points at distance $i-1 < x \leq i$ from some site in W. In the output subdivision, plant p has age $t+1$, and the points in $B_t \setminus B_{t-1}$ have age 1.

Let B_i have age value $t+1-i$. W has default value 0. We will define B_0 as the empty set. Then,

$$\mathcal{SD}_t = (W \setminus B_t) \cup \big(\bigcup_{i=0}^{t} B_i \setminus B_{i-1}\big).$$

Note that the boundaries do not intersect. Therefore, the optimal time complexity to compute $\mathcal{SD}_t$ is

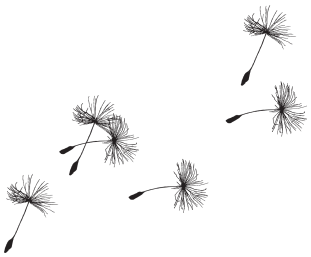
$$O(t\cdot((n+m)\cdot r)).$$

7.4 Queries under Euclidean distance measure

We can adjust the algorithm for the query to compute all points at one or multiple distances to suit a Euclidean distance measure. We assume we are given the augmented geodesic Voronoi diagram of P in W.

In the augmented geodesic Voronoi diagram under a Euclidean distance measure, each region has it own unique closest site and anchor point. In contrast with a convex distance function, pieces of B_t within a region of the augmented geodesic Voronoi diagram do not have to be straight line segments, but can also consists of arcs.

Even so, we can easily trace the boundary of B_t for all points p from some site s at Euclidean distance $d(s, p) = t$, and compute the distance in constant time. The closest site and anchor point, and thus the analytic expression, change at exactly the intersection with extension segments. The formula to compute $\mathcal{SD}_t$ does not have to change.



Chapter 8

Extensions

In this chapter, we will share observations on several extensions of the basic problem based on DIMO, namely germination delay, habitat suitability, obstacles and anisotropic regions.

8.1 Germination delay

In the DIMO model, plants first spread their seeds. After a certain time g, the germination delay, seeds have to grown plants and can start spreading seeds themselves. All points at a distance t from plants in the previous round will be covered in seeds.

The wave propagation on the other hand, only spreads from the initial source point. The first time step, all points at distance d are covered in seeds. After g time steps, the seeds have grown to plants. In the next time steps, the points at distance $t \cdot r$ from the source plant will be covered in seeds.

As one can see in Figure 8.1, the model of DIMO is not the same as a wave propagation if the germination delay is non-zero.

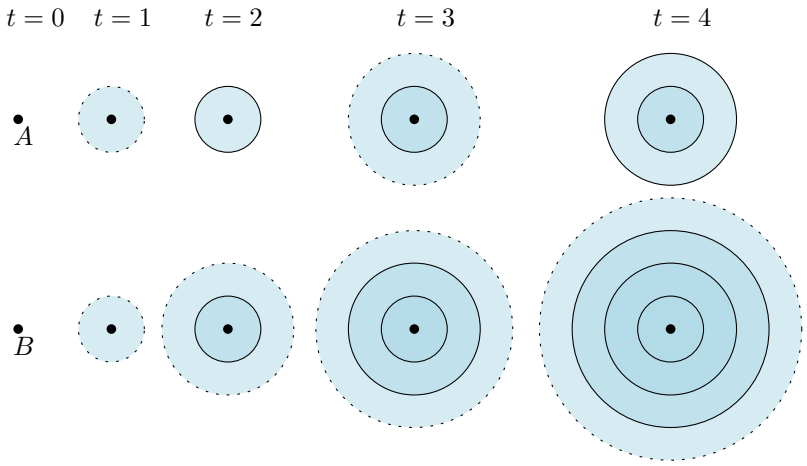


FIGURE 8.1: The plants from point A expand according to the model of DIMO, and the plants from point B expand as a wave propagation. In this figure, the dotted line represents the border of a region with seeds. A continuous line represents a region where plants are present. The germination delay is 1.

We will provide equations to clarify the difference between a plant that spreads as a wave, and a plant that spreads according to the model of DIMO.

Let d be the radius of the circle that a plant can spread each time step. Let $A_p(t)$ the radius of the circle that plants cover in time step t, and let A_s be the radius of the circle that seeds cover in time step t.

Wave propagation

Suppose the plants spread as a wave, and the germination delay is 0, that means that region of seeds is equal to the region of plants.

$$A_p(t) = d \cdot t,$$

$$A_s(t) = d \cdot t.$$

Suppose the germination delay is 1, as is also the case in Figure 8.1. Then,

$$A_p(t) = d \cdot (t-1) = dt - r,$$

$$A_s(t) = d \cdot t.$$

Suppose the germination delay is denoted by g. Then,

$$A_p(t) = \max(d \cdot (t-g), 0) = \max(dt - dg, 0),$$
$$A_s(t) = d \cdot t.$$

DIMO model

Suppose the plants spread according to the DIMO model, and the germination delay is 0, so that region of seeds is equal to the region of plants.

$$A_p(t) = d \cdot t,$$

$$A_s(t) = d \cdot t.$$

These formula's are the same as in the situation with the wave propagation. This changes if we suppose the germination delay is 1, as is also the case in Figure 8.1. Then,

$$A_p(t) = d \cdot \left\lfloor \frac{t}{2} \right\rfloor$$

$$A_s(t) = d \cdot \left\lceil \frac{t}{2} \right\rceil.$$

Suppose the germination delay is denoted by g. Then,

$$A_p(t) = d \cdot \left\lfloor \frac{t}{g+1} \right\rfloor$$

$$A_s(t) = d \cdot \left\lceil \frac{t}{g+1} \right\rceil$$

So, if we add germination delay to the model, the wave propagation and the DIMO model are not the same anymore. Still, we can use the formula to calculate the region reached according to the DIMO model.

8.2 Habitat suitability

In DIMO, the user can add a habitat suitability map as an input to the model and appoint regions that are not suitable for plants to grow. In some cases, seeds can "fly" over the inhabitable region in one step, as shown in Figure 8.2 (left), while in other cases B_t intersects the inhabitable region, as shown in Figure 8.2 (right).

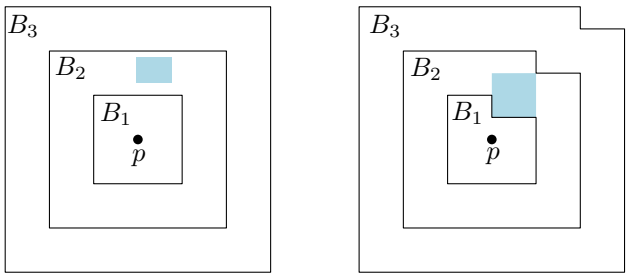


FIGURE 8.2: Here, the filled, blue region is not suitable for plant growth.

Moreover, in the case of inhabitable regions, the boundary of B_t does not necessarily have to be at distance t from the closest site, nor does it have to be at geodesic distance t from the closest site.

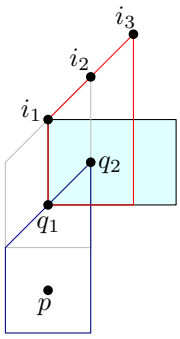


FIGURE 8.3: Here, the filled, blue region is not suitable for plant growth. The red polygon with center p in the form of a half house is the cd-polygon. i_1 is the highest point at the geodesic distance 2 from p. i_2 is the highest point at distance 2 from p, as the DIMO model would treat inhabitable regions. i_3 is the highest point at distance 2 from p if we would ignore the inhabitable region.

In Figure 8.3, we see a clear situation that B_t is not at distance t, ignoring inhabitable regions, and not at geodesic distance t. Suppose we want to calculate the region at distance 2 from a site p. Suppose we only look at the point at distance 2 that is highest above p. First, we draw R with center p. In the model of DIMO, seeds can "fly" over inhabitable regions. So, we draw R with center q_1. Point i_2 is then the highest point at distance 2 from p.

Classic distance on the contrary, would place R with center q_2 for the second time step. The top vertex then would be i_3. Geodesic distance treats obstacles as non-passable. Using R with center q_1, point i_2 would not be at distance 1. The highest point from point p at geodesic distance 2 would be i_1.

Thus, to deal with inhabitable regions, we cannot ignore the regions, and we cannot use geodesic distance.

8.3 Obstacles

Suppose, the region we consider is not a simple polygon, but contains obstacles. In that case, a shortest path in the Euclidean metric may not be a shortest path under a convex distance function based on a polygon, as can be seen in Figure 8.4. Though pq via s is shorter in the Euclidean metric, with a specific convex distance function (on the right) pq is shorter via r.

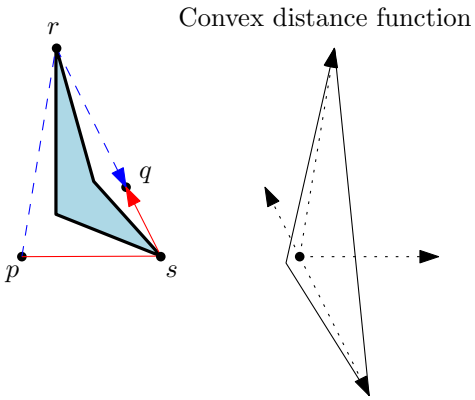


FIGURE 8.4: Shortest paths under a convex distance function may be different from the shortest path under a Euclidean metric.

8.4 Anisotropic regions

Triangle inequality does not hold in an anisotropic region, where each region has a different convex distance function. In Figure 8.5, the shortest distance from p to r in Euclidean metric is a straight line segment. Under the convex distance function the path from p to r via q is shorter than the direct path..

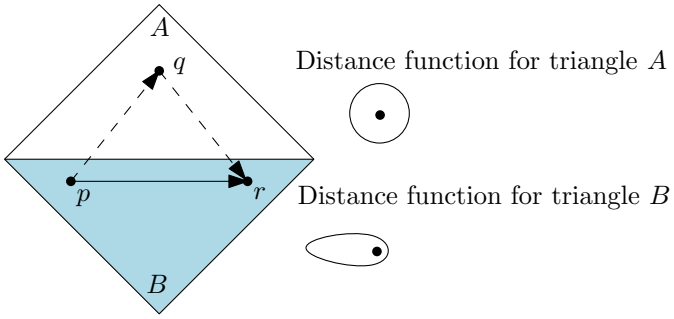
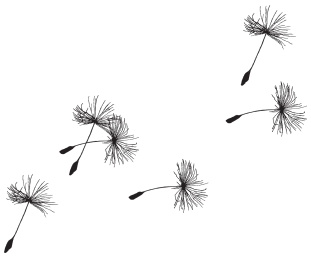


FIGURE 8.5: Triangle inequality does not hold in anisotropic regions.

We will close this chapter with some general observations. If W contains inhabitable regions, we cannot solve the problem using geodesic distance techniques. If W contains obstacles or consists of anisotropic regions, we lose fundamental properties such as triangle inequality. Therefore, we cannot use the designed algorithms in these contexts. Germination delay on the other hand, is similar to the basic problem. The designed algorithm and queries can be adapted to suit a problem that involves germination delay.

Chapter 9



Conclusion

In September 2021, we had a conversation with ecologist Wieger Wamelink. He spoke about the computational challenges that arose when using seed dispersal model DIMO. Inspired to approach the ecological models differently, we used the geometry and techniques from computational geometry to solve the problem. In this last section, we will discuss our theoretical results, discuss their implications and limitations, and explore suggestions for future work.

In this study, we formalize the ecological problem in a geometric setting. We represent a landscape with a simple n-gon W and model the initial source plants as a set P of m sites. We use a convex distance function based on a convex polygon with r vertices to model the influence of wind.

To answer queries such as 'At what time is a given point covered in plants?' we use a Voronoi diagram. Using a convex distance function based on a convex r-gon, we prove the fundamental properties of a geodesic cd-Voronoi diagram. Because a Voronoi cell is star-shaped, we adapt techniques originally designed for geodesic Voronoi diagrams to suit contexts under a convex distance function. We design novel algorithms based on the algorithm by Aronov (1989), and prove a time complexity of $O\big(((n+m)\cdot r)\log((n+m)\cdot r)\log n\big)$. Our adaptation called the "lazy approach" improves that bound to $O\big((n+m\cdot r)\log(n+m\cdot r)\log n\big)$.

To the best of our knowledge, the algorithm presented here is the first algorithm to compute a geodesic Voronoi diagram under a convex distance function. The novel geometric techniques for convex distance functions contribute to the fundamental research in computational geometry. They can serve as inspiration and foundation for further research.

9.1 Future work

We can still make significant progress in time complexity. For example, in this thesis we create an algorithm based on Aronov's algorithm (1989). We expect to be able to apply many of the proved properties of cd-Voronoi diagrams to Papadopoulou's algorithm (1998) as well. For example, we can reuse the proof for star-shaped Voronoi cells, since this algorithm also uses the star-shaped property. Moreover, the extension phase that is part of Aronov's algorithm, is also part of Papadopoulou's algorithm, so our adaptation for a convex distance function can be reused.

We expect the challenge for an adaptation of Papadapoulou's algorithm again lies in the difference between the complexity of a cd-Voronoi diagram and an augmented cd-Voronoi diagram. If we divide the polygon in regions such that each region

has its own anchor point and anchor cd-edge, the complexity of the subdivision would gain a factor $O(nr + mr)$, while the geodesic cd-Voronoi diagram without cd-extension segments has output complexity $O(n + mr)$. But if we do not use the cd-extension segments to divide the polygon, we cannot use the knowledge that each region has its own distinct associated closest site, anchor point, and anchor cd-edge. This problem needs to be solved to in order to achieve a factor $O(n + mr)$ instead of $O(nr + mr)$ in the time complexity bound.

Further geometric challenges lie ahead to better approximate and encompass the ecological model. The landscape is more accurately represented if we could model obstacles as polygonal holes. In addition, we could add inhabitable regions where plants cannot grow but seeds can fly over. The current algorithms require structural changes to deal with polygonal holes. In accordance with the wishes of ecological scientists as touched upon by Wieger Wamelink in Appendix A, we could extend our research to design algorithms that can handle a dynamic obstacle map and a dynamic habitat suitability map. Note, the knowledge that landscapes change slowly could direct the research, for example asking whether we can efficiently modify obstacles if only a constant number of vertices moves in each time step.

We just mentioned how further computational geometry research could to help solving the limited representation of the landscape. If we look at the applicability for the ecological research community, there is still a gap to fulfill her needs and wishes. We should invest time and effort to implement the algorithms and make them easily accessible in accordance with Open Science (Vicente-Saez and Martinez-Fuentes, 2018).

Our research results and potential extensions allow the ecological research community to approach the seed dispersal problem differently, and use geometrical techniques to simulate plant dispersal more efficiently. The development of efficient dispersal models using a realistic representation of the landscape is important to track the effect of climate change on plant survival. Extending the collaboration between the scientific fields of ecology and computational geometry can create and apply novel fundamental methods to solve societal issues caused by climate change.

Bibliography

Albert, A., A. Mårell, M. Picard, and C. Baltzinger (2015). "Using basic plant traits to predict ungulate seed dispersal potential". In: *Ecography* 38, pp. 440–449. DOI: 10.1111/ecog.00709.

Andújar, D., X. Rodriguez, V. Rueda-Ayala, C. San Martín, A. Ribeiro, C. Fernández-Quintanilla, and J. Dorado (2017). "A Geometrical Model to Predict the Spatial Expansion of Sorghum Halepense in Maize Fields". In: *Gesunde Pflanzen* 69.2, pp. 73–81. DOI: 10.1007/s10343-017-0388-6.

Arge, L. and F. Staals (2017). *Dynamic Geodesic Nearest Neighbor Searching in a Simple Polygon*. http://arxiv.org/abs/1707.02961.

Aronov, B. (1989). "On the geodesic voronoi diagram of point sites in a simple polygon". In: *Algorithmica* 4, pp. 109–140. DOI: 10.1007/bf01553882.

Baguette, M. (2003). "Long distance dispersal and landscape occupancy in a metapopulation of the cranberry fritillary butterfly". In: *Ecography* 26.2. Publisher: Wiley Online Library, pp. 153–160. DOI: 10.1034/j.1600-0587.2003.03364.x.

Barequet, G., M. T. Dickerson, and M. T. Goodrich (2001). "Voronoi diagrams for convex polygon-offset distance functions". In: *Discrete & Computational Geometry* 25.2, pp. 271–291. DOI: 10.1007/s004540010081.

Büyüktahtakın, İ. E. and R. G. Haight (2018). "A review of operations research models in invasive species management: state of the art, challenges, and future directions". In: *Annals of Operations Research* 271.2, pp. 357–403. DOI: https://doi.org/10.1007/s10479-017-2670-5.

Chazelle, B. (1991). "Triangulating a simple polygon in linear time". In: *Discrete & Computational Geometry* 6.3. Publisher: Springer, pp. 485–524. DOI: https://doi.org/10.1007/bf02574703.

Cheng, S., H. Na, A. Vigneron, and Y. Wang (2008). "Approximate Shortest Paths in Anisotropic Regions". In: *SIAM Journal on Computing* 38.3, pp. 802–824. DOI: 10.1137/06067777X.

Chew, L. P. and R. L. S. Dyrsdale (1985). "Voronoi diagrams based on convex distance functions". In: *Proceedings of the first annual symposium on Computational geometry*. Association for Computing Machinery, pp. 235–244. DOI: https://doi.org/10.1145/323233.323264.

Corlett, R. T. (2013). "Will plant movements keep up with climate change?" In: *Trends in ecology and evolution* 28.8, p. 7. DOI: https://doi.org/10.1016/j.tree.2013.04.003.

"Computational Geometry" (2008). In: *Computational Geometry: Algorithms and Applications*. Ed. by M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. Springer, p. 297. DOI: 10.1007/978-3-540-77974-2_1.

Dehne, F. and R. Klein (1997). ""The Big Sweep": On the Power of the Wavefront Approach to Voronoi Diagrams". In: *Algorithmica* 17.1, pp. 19–32. DOI: https://doi.org/10.1007/bf02523236.

Dullinger, S., N. Dendoncker, A. Gattringer, M. Leitner, T. Mang, D. Moser, C. A. Mücher, C. Plutzar, M. Rounsevell, W. Willner, N. E. Zimmermann, and K. Hülber (2015). "Modelling the effect of habitat fragmentation on climate-driven migration of European forest understorey plants". In: *Diversity and Distributions* 21.12, pp. 1375–1387. DOI: 10.1111/ddi.12370.

Edelsbrunner, H., L. J. Guibas, and J. Stolfi (1986). "Optimal point location in a monotone subdivision". In: *SIAM Journal on Computing* 15.2, pp. 317–340. DOI: https://doi.org/10.1137/0215023.

Fortune, S. (1987). "A sweepline algorithm for Voronoi diagrams". In: *Algorithmica* 2.1, p. 153. DOI: https://doi.org/10.1007/bf01840357.

Gosper, C. R., C. D. Stansbury, and G. Vivian-Smith (2005). "Seed dispersal of fleshy-fruited invasive plants by birds: contributing factors and management options". In: *Diversity and Distributions* 11.6, pp. 549–558. DOI: https://doi.org/10.1111/j.1366-9516.2005.00195.x.

Guibas, L., J. Hershberger, D. Leven, M. Sharir, and R. Tarjan (1986). "Linear time algorithms for visibility and shortest path problems inside simple polygons". In: *Proceedings of the second annual symposium on Computational geometry - SCG '86*. the second annual symposium. ACM Press, pp. 1–13. DOI: 10.1145/10515.10516.

Hershberger, J. and S. Suri (1999). "An Optimal Algorithm for Euclidean Shortest Paths in the Plane". In: *SIAM Journal on Computing* 28.6, pp. 2215–2256. DOI: 10.1137/S0097539795289604.

Klein, E. K., C. Lavigne, X. Foueillassar, P.-H. Gouyon, and C. Larédo (2003). "Corn pollen dispersal: Quasi-mechanistic models and field experiments". In: *Ecological Monographs* 73.1, pp. 131–150. DOI: 10.1890/0012-9615(2003)073[0131:CPDQMM]2.0.CO;2.

Klein, R. (1988). "Abstract voronoi diagrams and their applications". In: *Computational Geometry and its Applications*. Ed. by H. Noltemeier. Springer, pp. 148–157. DOI: 10.1007/3-540-50335-8_31.

Klein, R., E. Langetepe, and Z. Nilforoushan (2009). "Abstract Voronoi diagrams revisited". In: *Computational Geometry* 42.9, pp. 885–902. DOI: 10.1016/j.comgeo.2009.03.002.

Liu, C.-H. (2020). "A Nearly Optimal Algorithm for the Geodesic Voronoi Diagram of Points in a Simple Polygon". In: *Algorithmica* 82.4, pp. 915–937. DOI: 10.1007/s00453-019-00624-2.

Ma, L. (2000). "Bisectors and Voronoi Diagrams for Convex Distance Functions". PhD thesis. University of Hagen.

McGuire, J. L., J. J. Lawler, B. H. McRae, T. A. Nuñez, and D. M. Theobald (2016). "Achieving climate connectivity in a fragmented landscape". In: *Proceedings of the National Academy of Sciences* 113.26, pp. 7195–7200. DOI: 10.1073/pnas.1602817113.

Mulmuley, K. (1990). "A fast planar partition algorithm, I". In: *Journal of Symbolic Computation* 10.3, pp. 253–280. DOI: https://doi.org/10.1016/s0747-7171(08)80064-8.

Nathan, R. (2006). "Long-Distance Dispersal of Plants". In: *Science* 313.5788, pp. 786–788. DOI: 10.1126/science.1124975.

Nathan, R. and H. C. Muller-Landau (2000). "Spatial patterns of seed dispersal, their determinants and consequences for recruitment". In: *Trends in ecology & evolution* 15.7. Publisher: Elsevier, pp. 278–285. DOI: https://doi.org/10.1016/s0169-5347(00)01874-7.

Oh, E. (2019). "Optimal Algorithm for Geodesic Nearest-point Voronoi Diagrams in Simple Polygons". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. Ed. by T. M. Chan. Society for Industrial and Applied Mathematics, pp. 391–409. DOI: https://doi.org/10.1137/1.9781611975482.25.

Oh, E. and H.-K. Ahn (2020). "Voronoi Diagrams for a Moderate-Sized Point-Set in a Simple Polygon". In: *Discrete & Computational Geometry* 63.2, pp. 418–454. DOI: 10.1007/s00454-019-00063-4.

Ozinga, W. A., C. Römermann, R. M. Bekker, A. Prinzing, W. L. M. Tamis, J. H. J. Schaminée, S. M. Hennekens, K. Thompson, P. Poschlod, M. Kleyer, J. P. Bakker, and J. M. van Groenendael (2009). "Dispersal failure contributes to plant losses in NW Europe". In: *Ecology Letters* 12.1, pp. 66–74. DOI: 10.1111/j.1461-0248.2008.01261.x.

Papadopoulou, E. and D. T. Lee (1998). "A New Approach for the Geodesic Voronoi Diagram of Points in a Simple Polygon and Other Restricted Polygonal Domains". In: *Algorithmica* 20.4, pp. 319–352. DOI: 10.1007/PL00009199.

Revilla, E. and T. Wiegand (2008). "Individual movement behavior, matrix heterogeneity, and the dynamics of spatially structured populations". In: *Proceedings of the National Academy of Sciences* 105.49, pp. 19120–19125. DOI: 10.1073/pnas.0801725105.

Somerville, G. J., M. Sønderskov, S. K. Mathiassen, and H. Metcalfe (2020). "Spatial Modelling of Within-Field Weed Populations; a Review". In: *Agronomy* 10.7. Number: 7 Publisher: Multidisciplinary Digital Publishing Institute, p. 1044. DOI: 10.3390/agronomy10071044.

Sundell, H. and P. Tsigas (2008). "Lock-free deques and doubly linked lists". In: *Journal of Parallel and Distributed Computing* 68.7, pp. 1008–1020. DOI: 10.1016/j.jpdc.2008.03.001.

Teggi, S., S. Costanzini, G. Ghermandi, C. Malagoli, and M. Vinceti (2018). "A GIS-based atmospheric dispersion model for pollutants emitted by complex source areas". In: *Science of The Total Environment* 610-611, pp. 175–190. DOI: 10.1016/j.scitotenv.2017.07.196.

Thuiller, W., C. Albert, M. B. Araújo, P. M. Berry, M. Cabeza, A. Guisan, T. Hickler, G. F. Midgley, J. Paterson, F. M. Schurr, M. T. Sykes, and N. E. Zimmermann (2008). "Predicting global change impacts on plant species' distributions: Future challenges". In: *Perspectives in Plant Ecology, Evolution and Systematics* 9.3, pp. 137–152. DOI: 10.1016/j.ppees.2007.09.004.

Treep, J., M. de Jager, F. Bartumeus, and M. B. Soons (2021). "Seed dispersal as a search strategy: dynamic and fragmented landscapes select for multi-scale movement strategies in plants". In: *Movement Ecology* 9.1, p. 4. DOI: 10.1186/s40462-020-00239-1.

Van Houtan, K. S., S. L. Pimm, J. M. Halley, R. O. Bierregaard, and T. E. Lovejoy (2007). "Dispersal of Amazonian birds in continuous and fragmented forest". In: *Ecology Letters* 10.3, pp. 219–229. DOI: 10.1111/j.1461-0248.2007.01004.x.

Vicente-Saez, R. and C. Martinez-Fuentes (2018). "Open Science now: A systematic literature review for an integrated definition". In: *Journal of Business Research* 88, pp. 428–436. DOI: 10.1016/j.jbusres.2017.12.043.

Wamelink, G. W. W., R. Jochem, W. Geertsema, A. H. Prins, W. A. Ozinga, J. van der Greft-van Rossum, J. Franke, A. H. Malinowska, A. H. Prins, D. C. J. van der Hoek, and C. J. Grashof-Bokdam (2014). "DIMO, a plant dispersal model". In: *Statutory Research Tasks Unit for Nature & the Environment (WOT Natuur & Milieu)*.

Wang, H. (2021). "Shortest Paths Among Obstacles in the Plane Revisited". In: *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*. Ed. by D. Marx, pp. 810–821. DOI: https://doi.org/10.1137/1.9781611976465.51.

Figures

All figures are self-made, except:

Figure 3.1: "A convex distance function". Adapted from Figure 1.1.1 in Ma, 2000.

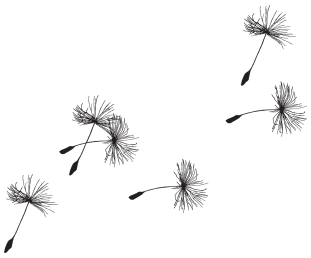
Figure 3.2: "The wavefront of the algorithm continuous Dijkstra". Original source: Figure 7 in Hershberger and Suri, 1999.

Figure 3.3 "Construction of bisector". Adapted from Figure 2.2.1.1 in Ma, 2000.

Figure 3.4: "The degenerate case". Adapted from Figure 2.1.3.1 in Ma, 2000.

Figure 3.6: "Conventions". Adapted from Figure 3 in Papadopoulou and Lee, 1998.

Appendix A



Research orientation

To orient on the ecological subject of seed dispersion, we contacted the ecologists Wieger Wamelink and Laurens Sparrius. Both took the time to explain in detail how their field is structured, how the seed dispersal models contribute to science, and what challenges lie ahead. In this chapter, we give the summaries of the conversations.

A.1 Wieger Wamelink

On September 14, 2021, we spoke to Wieger Wamelink, ecologist at Wageningen University & Research. Together with his colleagues, he designed a model for plant dispersion, called DIMO. We will discuss the three main themes of the conversation: the benefits, the technical challenges and the limitations of the seed dispersal model.

What is the benefit of knowing or predicting seed dispersion?

'In the past, the benefit came down to scientific interest to explain why plant species occur at certain places. Now, we have a new incentive, namely climate change. Each plant species has an optimum in environmental temperature. If the temperature rises two degrees Celcius, we will lose over two hundred species in the Netherlands. Species either adapt, they migrate or they go extinct. As an example, in the Netherlands the tree species Birch loses its leaves already in summer recent years, which we predicted with our model.'

'Sometimes barriers prevent the spreading of seeds. These barriers could be natural, such as rivers, or human-made, such as roads. To restore the flow of seeds, we search for ways to reconnect the nature reserves or forests. In Germany, we now have a project in a very fragmented area, in which we try to find the best ways to reconnect the area again.'

'We usually do not focus on trying to prevent invasive species from spreading. Usually humans are a large factor, which makes prevention difficult. It is outside the scope of our research.'

What are technical or geometric challenges you faced in building the model?

'We chose a grid model as a basis, mostly because we are used to it. Also, many input maps have a grid representation. Water maps, on the contrary, use a polygon representation. Converting the polygons to grid representation takes a lot of time.'

'Some plants can travel very far, such as an orchid. All the grid cells need to be calculated one by one which takes a lot of time. And time is the bottleneck, not space.'

'When we model barriers, we use line segments in the grid model. Sometimes the barriers do not connect well over the different grids, resulting in unwanted gaps that produce errors. This takes a lot of time to fix.'

'Using a 3D terrain would take too much time. Whether a barrier is high or low does matter, but we model that using a degree of permeability for barriers.'

Limitations: what would you hope for in a plant dispersion model?

'It would help us a lot if we could switch easily between the representation of polygons and grids. For example, if we could use the polygon representation to solve a sub-problem with the water card, then switch back efficiently to a grid model when we present it to outsiders.

'At this moment, we use the current vegetation map for both the past and the future. Updates occur frequently, such as a parameter update or an input map change. It would be handy if the model could do these updates dynamically. Note: the hundred maps are not random, there is a line of logical update in it.'

'It has always been a goal for me to compute the model from the Penultimate Glacial Period (the glacial age that occurred before the Last Glacial Period), but for now that takes too much time.'

A.2 Laurens Sparrius

On September 21, 2021, we spoke to ecologist Laurens Sparrius, acquainted with the Dutch research institute FLORON. FLORON maps the distribution of common plants in the Netherlands. The data is collected by hundreds of people, mostly volunteers.

In our conversation, Sparrius elaborated on the structure and the purpose of the organisation. The data in the databases of Floron describe plant observations. These consist of the name of the plant, a time stamp and a location.

The Floron Verspreidingsatlas Vaatplanten[1] presents this data in a visually attractive way to support easy use. A user can make a variety of requests, such as asking the system to provide a map for one specific plant over a longer period of time, or limit the region to see an overview of plants that grow there.

When we asked for geometric or technical challenges, Sparrius mentioned the challenge of showing the data clearly and efficiently. As an example, the location of the observations all have an inaccuracy. Some observations are accurate up to a few meters, while other have an uncertainty radius of a few hundred meters. This is shown as an octagon, but is stored as a centre point and a radius. Since these challenges lie in the context of data visualisation, we will not focus our research on this direction.

[1] `https://www.verspreidingsatlas.nl/1398`