Continuous Voronoi Games on Graphs with Multiple Opponents

Tom Rijnbeek, BSc.
3657086
tom@tomrijnbeek.nl

August 16, 2015

Abstract

In this thesis, we study the puzzle game *Lines*—a commercial game developed by *Gamious*—which is a variant of the one-round Voronoi game on graphs. The game presents the player with a graph with coloured dots on the vertices or edges, corresponding with the moves of all opponents. The player is tasked with placing one or more dots in their own colour on the graph, such that when we calculate the Voronoi diagram using the coloured dots as sites, the union of player-coloured Voronoi cells is largest.

We develop an algorithm that calculates all winning placements in $O(|V||E||S| \cdot (|V| \log |V| + |E|))$ time for the simplified problem of placing one point, where $|V|$ is the number of vertices, $|E|$ the number of edges, and $|S|$ the number of existing sites. We then consider the generalised question of placing k sites and we present an algorithm that runs in $O(|V|^k |E|^{2k} k^{2k} (|\mathcal{C}| + k^2)^k)$ time, where $|\mathcal{C}|$ is the number of different colours of the sites.

Finally, we discuss further generalisations and extensions.

Contents

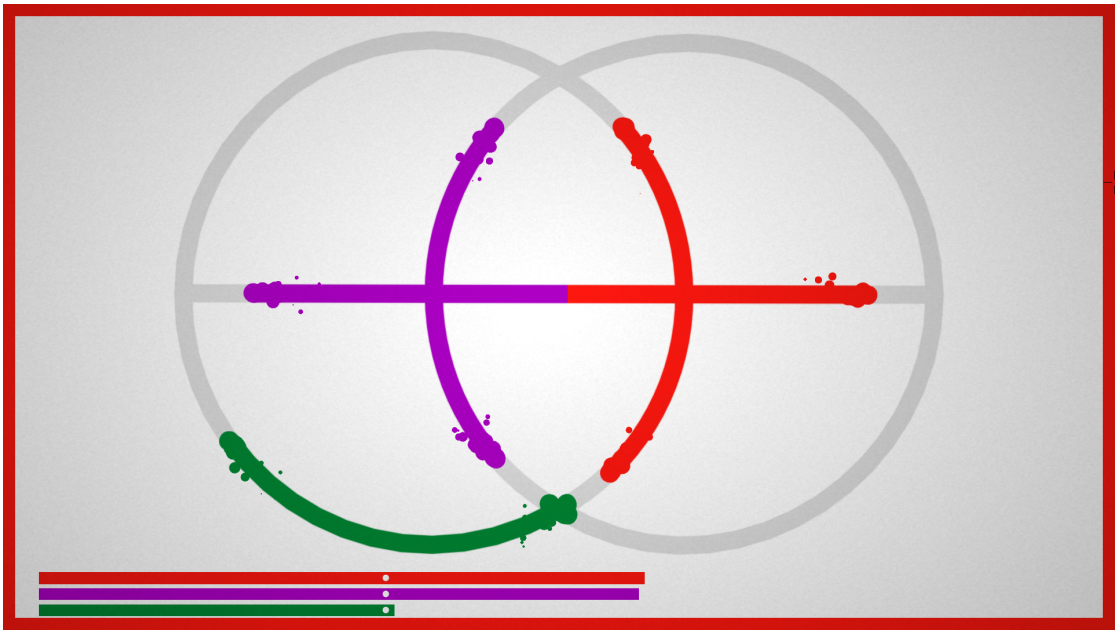


Figure 1: A screen capture of the game *Lines* by *Gamious*.

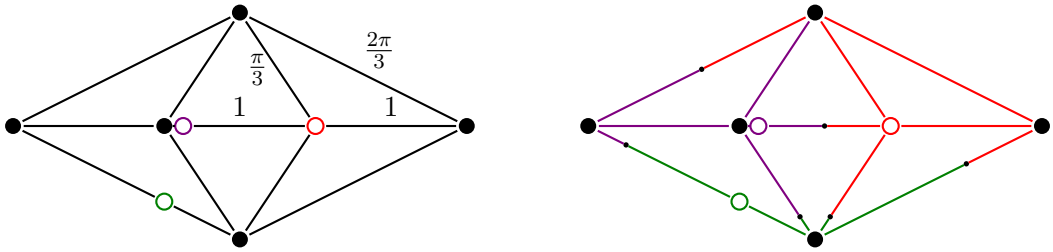
1. Introduction

The game *Lines* (see Figure 1) by *Gamious* is a single-player puzzle game in which the player has to solve a sequence of independent puzzles. Each puzzle consists of a network on which coloured dots (*sites*) are located. Each colour represents a party in the game: one colour is assigned to the player and the other colours are opponents. It is possible that the player colour is not yet present in the initial puzzle.

The puzzle consists of two phases: the modification phase and the simulation phase. Upon starting the simulation phase, all dots start expanding at constant speed in all directions along the network, covering the edges in their colour. The expansion always occurs along a discrete set of *fronts*. When a front arrives at an intersection, the front splits up into multiple fronts—one for each direction. When two fronts meet, expansion along that front is halted. The simulation phase ends when no front can move further, which is exactly when the entire network is coloured.

Figure 1 shows a game state in which the simulation phase is still ongoing. The active fronts can be identified by the particle effects. In the centre two fronts that met each other and stopped expanding can be seen, and in the bottom three green fronts can be found just after splitting at the intersection. Figure 2 shows a schematic representation of the network both at the start and at the end of the simulation phase.

The goal of the game is to have the player colour—shown as the colour of the screen border—cover a larger portion of the network than the other colours. In the example of Figure 1 the player will indeed win, since their colour (red) will cover the largest portion of the network, as seen in Figure 2b.



(a) Before simulation

(b) After simulation

Figure 2: Graph representation of the network from Figure 1. Filled circles represent vertices; open circles represent sites and can coincide with a vertex, in which case the vertex is not drawn. The colours of the edges represent which Voronoi cell they belong to and borders between coloured cells are marked with small dots. The numbers represent the lengths of the edges. This syntax is used throughout this thesis, unless otherwise specified.

To allow the player to influence the result of the game, they get access to different operations which are performed in the modification phase, before the simulation phase. The number of operations the player must perform of each kind is determined beforehand. The player has to figure out how to perform the operations such that their colour covers the largest portion in the modified network after the simulation phase. The four different operations are:

(a) add a new player-coloured site anywhere on the network;

(b) remove any existing coloured site from the network;

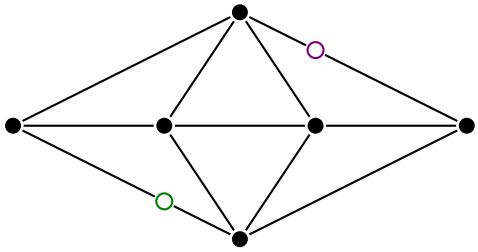
(c) make a cut in any connection of the network, which stops colours from expanding past it;

(d) add a new line segment connecting two existing points on the network.

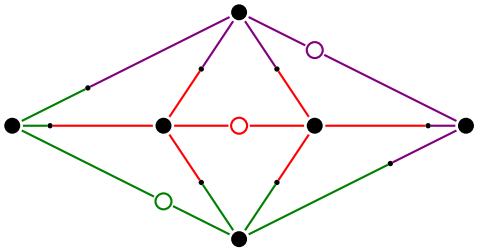
Figure 3 shows examples of puzzle instances and solutions for each of the four operations, where the set of allowed operations is exactly one operation of that type. However, instances of *Lines* can also allow more than one of the same operation, or even a combination of different operations.

The network on which *Lines* is played is in fact a graph and the completely coloured graph—that is, the state of the graph after the simulation phase ended—corresponds to the continuous Voronoi diagram on the graph using the coloured dots as sites. This game then closely resembles the Voronoi game [1], in which two players alternately place a facility in a geometric space in order to maximise their service area.

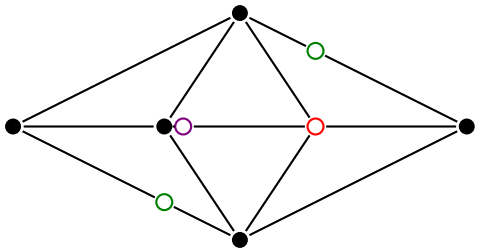
All existing research on Voronoi focusses on finding an optimal strategy for the player, specifically maximising the area covered by the player. With multiple opponents, however, maximising the covered area does not always result in a winning strategy. An example of this is shown in Figure 4: if the player gets to place a single site and wants



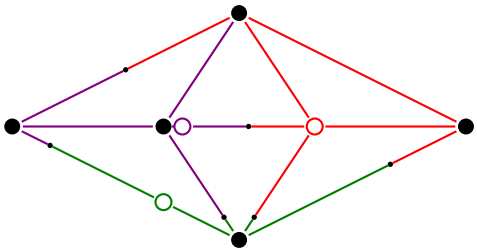
(a) Instance 1: place one point.



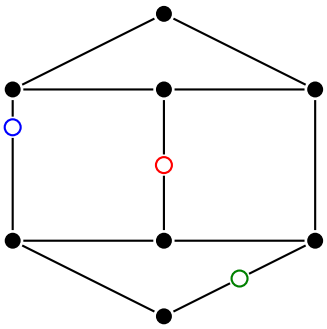
(b) Solution 1



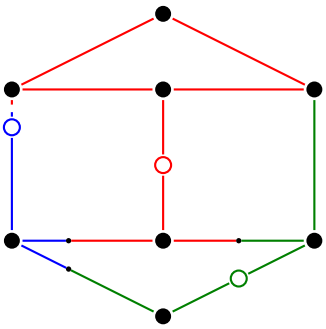
(c) Instance 2: remove one point.



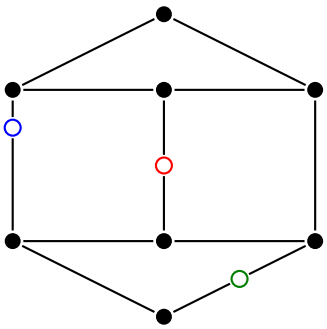
(d) Solution 2



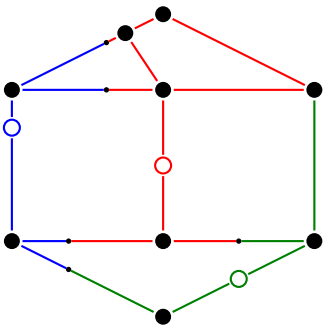
(e) Instance 3: make one cut.



(f) Solution 3

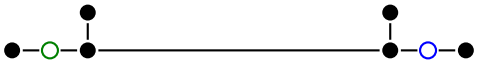


(g) Instance 4: add one segment.



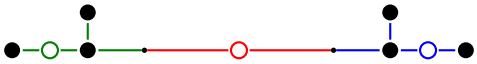
(h) Solution 4

Figure 3: Instances and solutions for the game Lines showing each of the four allowed operations.



(a) The unsolved instance.

(b) A solution in which the area covered in red is maximal, but does not cover a larger area than blue.



(c) A solution in which red covers a larger area than either colour.

Figure 4: An instance on which maximising the covered area does not result in a winning strategy if the player (red) is allowed to place a single point.

to cover a maximal portion of the graph, the site should be placed such that one of the vertical edges is covered. However, by doing so, the site on the opposite side will always cover a larger area than the player. The only way to win in this instance is to balance the area conquered from both opponents by placing the site in the middle and not getting the maximum area possible.

In this thesis we will therefore study the space of all possible winning moves by the player as opposed to determining the single optimal strategy, making this approach unique to existing research to Voronoi games. We will also primarily focus on the situation where the player has to place a number k of new sites. This turns the problem into a one-round Voronoi game with one or more opponents.

The analysis and methods in this thesis are unique, since it is the first time the problem of finding all solutions is considered as opposed to maximising the covered area. We thus introduce a new class of problems that can be studied in other variations of the Voronoi game as well.

Moreover, the Voronoi game with multiple opponents has never been studied before. Banik et al. [4] already analysed the Voronoi game with existing sites, but did not consider additional opponents owning these sites. The results of this thesis can not only be used to find all solutions in the *Lines* variant of the Voronoi game, but if we find a meaningful measure of an optimal solution with multiple opponents, the algorithm can be adapted to find the single optimal solution. This thesis therefore provides a meaningful starting point for the study of a more general Voronoi game where the number of players can be more than two.

We start by considering the simplified problem of placing one new site on a graph $G = (V, E)$ while ignoring the colouring of the sites S and present an algorithm that solves this problem in $O(|V||E||S| \cdot (|V| \log |V| + |E|))$ time (Section 3). Then we show that colouring the existing sites does not impact this running time (Section 4). To solve the problem for the placement of k sites, we first translate our solution for one site to the

placement of two sites and find a new running time of $O(|V|^2|E|^4|\mathcal{C}|\log|\mathcal{C}|)$ where $|\mathcal{C}|$ is the number of colours (Section 5). We then generalise our observations to the placement of an arbitrary number k of sites and present the final algorithm which calculates the winning placements in $O(|V|^k|E|^{2k}k^{2k}(|\mathcal{C}|+k^2)^k)$ time (Section 6). Further we discuss several extensions of this problem (Section 7) and show how the algorithm still applies if the sites to be placed have different colours (Section 7.2). Finally we briefly look at how *Lines* could be solved for the remaining operations and present simple procedures to find all winning edge snips in $O(|E|^2+|E||V|\log|V|)$ time and all winning k site removals in $O(|S|^k(|E|+|V|\log|V|))$ time (Section 8).

2. Related work

The Voronoi game has already been studied in many variations. The term is first introduced by Ahn et al. [1] as a geometric model of competitive facility location: two players $\mathcal{P}_1$ and $\mathcal{P}_2$ take turns placing facilities in a geometric space. After k turns the space is subdivided into the subspace for which a facility of $\mathcal{P}_1$ is closer than all facilities of $\mathcal{P}_2$ and the subspace for which the opposite holds. The player to which the largest subspace belongs wins the game.

The idea behind Voronoi games is not new, and spatial competitive facility location was already a broadly studied field before the introduction of Voronoi games. Extensive surveys on this subject have been made by Eiselt et al. [10, 11].

Ahn et al. focus on the Voronoi game on line segments and circles in which the players alternately place one facility k times. They show that on line segments and circles the second player has a winning strategy, but by allowing the placement of multiple facilities in a single round, where players can choose the number of facilities every round as long as they don't place more than k in total, $\mathcal{P}_1$ can ensure that the second player only wins by an arbitrarily small margin.

Since the problem of determining an optimal strategy for either player is hard to solve in planar domains, Cheong et al. [7] focus on the one-round Voronoi game, where first $\mathcal{P}_1$ places k facilities, followed by $\mathcal{P}_2$ placing k facilities as well. The authors show that if the geometric space is a square and k is large enough, there is always a winning strategy for $\mathcal{P}_2$. These results were generalised to rectangles with aspect ratio $\rho \leq 1$ by Fekete et al. [12], showing that $\mathcal{P}_2$ has a winning strategy if and only if $k \geq 3$ and $\rho > \frac{\sqrt{2}}{k}$ or $k = 2$ and $\rho > \frac{\sqrt{3}}{2}$. This paper is also to first to prove that maximising the area for $\mathcal{P}_2$ in a polygon with holes is NP-hard.

Teramoto et al. [16] translated the geometric game to a discrete variant in which the players alternately occupy vertices in a graph. The goal is to cover the majority of the remaining vertices according to the nearest neighbour rule. They analyse the best strategy on a complete k-ary tree and then continue to prove that finding an optimal strategy is NP-hard for the one-round Voronoi game on general graphs, and PSPACE-hard in the case of multiple rounds. The Voronoi game on path graphs is studied by Kiyomi et al. [13], who show that, except for some trivial cases, both players have a strategy that ensures a draw.

Another variation of the discrete Voronoi game is where the geometric space is continuous, but the space contains a finite number of *clients*. Every client is served by the closest facility. Instead of covering the largest area, the goal of the discrete Voronoi game is to maximise the number of clients served by the facilities of the player. Banik, Bhattacharya, and Das [3] have determined optimal strategies for both players in this discrete one-round Voronoi game where the geometric space is a line. If the sorted order of the clients along the line is known beforehand, these strategies can be calculated in $O(n)$ time for $\mathcal{P}_2$ assuming all the facilities of $\mathcal{P}_1$ are given, and $O(n^{k-c_k})$ time for $\mathcal{P}_1$, where n is the amount of clients and c_k a constant depending on k only. Polynomial algorithms to determine the optimal strategy for both players in a polygonal domain have been developed by Banik et al. [5]. In this case, the strategy for $\mathcal{P}_2$ can be calculated in $O(k + n(\log n + \log k))$ time if the placement of facilities for $\mathcal{P}_1$ is given.

In [4] Banik et al. consider the one-round discrete Voronoi game in two-dimensional Euclidean space with existing facilities and show that in this case the optimal placement for both players can be calculated in polynomial time as well: $O(n^2)$ time is required for to determine the strategy of $\mathcal{P}_2$, given the facility placement of $\mathcal{P}_2$; the optimal strategy for $\mathcal{P}_1$ can be found in $O(n^8)$ time.

Bandyapadhyay et al. [2] studied the continuous Voronoi game on graphs embedded in $\mathbb{R}^2$. They further introduce asymmetry by decoupling the number of facilities $\mathcal{P}_1$ and $\mathcal{P}_2$ are allowed to place. The authors develop a polynomial time algorithm to determine an optimal strategy for $\mathcal{P}_2$ on trees and prove that finding an optimal strategy on general graphs is NP-hard, but can be approximated in polynomial time.

3. One site

In this section we will formalise the problem of finding a winning site placement for the player. Then we will introduce the main components that we will use to construct the algorithm. Finally we will present the algorithm that can be used to solve the problem for the placement of one site.

3.1. Definitions

Let $G = (V, E)$ be an undirected, connected graph, consisting of vertices $v \in V$ and edges $e \in E$. Each edge e has a (finite) weight $w(e) > 0$. Each edge e corresponds to a bounded one-dimensional geometric space: a segment with length $w(e)$ of which the endpoints are exactly the two vertices the edge connects. We can specify points p on the graph by a pair (e, t), where e is an edge and $0 \leq t \leq w(e)$ the distance from a predetermined endpoint of the edge (e.g. given by an arbitrary well-order on V). For example, if we have an edge $\{u, v\}$ then $u = (\{u, v\}, 0)$ and $v = (\{u, v\}, w(e))$ or vice versa.

We will denote the set of all points on the edges of the graph with $\mathcal{E}$. In other words,

$$\mathcal{E} = \bigcup E.$$

Further let $S = \{s_1, ..., s_l\} \subseteq \mathcal{E}$ be a non-empty finite set of points (*sites*) lying on G.

Definition 3.1. For any pair of points $p, q \subseteq \mathcal{E}$ on a graph G we define $d_G(p, q)$ (or $d(p, q)$ for short if G is clear from the context) to be the shortest path distance between p and q on G.

Definition 3.2. The *Voronoi cell* of a point s $\mathrm{Vor}_{G,S}(s)$ (or $\mathrm{Vor}(s)$ if G and S are clear from the context) on G is defined as

$$\mathrm{Vor}_{G,S}(s) := \left\{p \in \mathcal{E} \mid \forall s' \in S : d_G(p, s) \leq d_G(p, s')\right\}.$$

The *Voronoi diagram* $\mathcal{V}_G(S)$ of S on G is then defined as

$$\mathcal{V}_G(S) := \left\{\mathrm{Vor}_{G,S}(s) \mid s \in S\right\}.$$

From Definition 3.2 it is clear that points which have two (or more) sites at equal distance that are closest to them also belong to both of their Voronoi cells. We call these points *boundary points* or *boundaries* for short. Figure 5 shows how we can have an infinite number of boundary points (purple). We will henceforth assume $d(v, s) \neq d(v, s')$ for each vertex v and pair of sites $s, s' \in S$ as this is sufficient to guarantee there are only a finite number of boundary points (see also Subsection 3.7).

Finally, we will see that it is often convenient to consider the sites as vertices. Therefore we will henceforth assume that each site $s \in S$ coincides with one of the vertices of G, thus $S \subseteq V \subseteq \mathcal{E}$. If this is not the case, we can trivially construct an equivalent graph with additional vertices to fulfil this requirement.

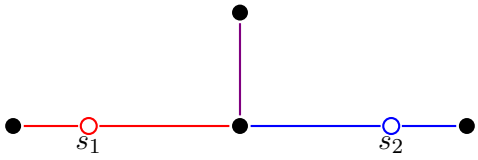


Figure 5: An example of a graph with an infinite number of boundary points (purple).

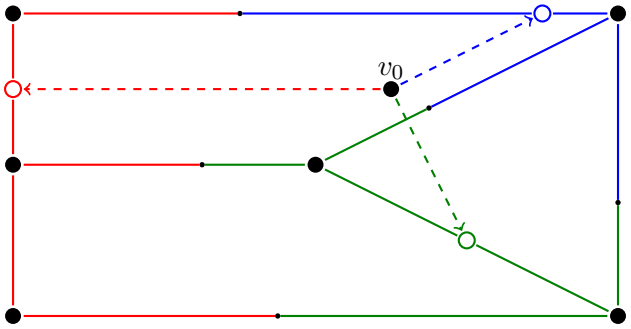


Figure 6: Illustration of how the Voronoi diagram is calculated from the extended shortest path tree.

3.2. Calculating a Voronoi diagram

In a paper on generalised network Voronoi diagrams, Okabe et al. [15] introduced a method to calculate Voronoi diagrams on graphs embedded in Euclidean space. This technique generalises to arbitrary graphs trivially. Let G be a graph and $S \subseteq V$ a set of sites. Add a new dummy vertex v_0 and connect it using arcs with weight 0 to all sites in S. It is also possible to use any other non-negative weight, as long as all arcs have equal length.

Now we calculate the extended shortest path tree (ESPT) for the dummy vertex v_0. If v_0 and its adjacent arcs are removed, several subtrees will remain, each containing exactly one element from S: the root of the subtree. These subtrees correspond to the Voronoi cells of the contained site. An example of the result of this algorithm can be found in Figure 6.

3.3. Problem definition

In this thesis we focus on adding a new point x on a graph G such that the corresponding Voronoi cell $\mathrm{Vor}(x)$ in the Voronoi diagram $\mathcal{V}_G(S \cup \{x\})$ is pairwise larger than each other Voronoi cell. If this condition is satisfied, we call x a *winning point* in (G, S).

Definition 3.3. Let G and S be defined as before. A point $x \in \mathcal{E}$ is called a *winning point* if $\forall s \in S : |\mathrm{Vor}(x)| \geq |\mathrm{Vor}(s)|$.

The size of a Voronoi cell is the sum of the size of the edge segments it contains according to the one-dimensional Lebesgue-measure or, in other words, the sum of the

lengths of the edge segments calculated proportionally from the total edge length. Remark that by requiring a finite number of boundaries, the total area of the boundaries by this definition is zero.

The problem we will be looking at is the following: given a graph G and a predefined set of sites $S = \{s_1, ..., s_l\} \subseteq V$, create a subdivision of $\mathcal{E}$ into two sets: the set of points x where x is a winning point and the remainder of $\mathcal{E}$.

Definition 3.4. The function $A : \mathcal{E} \to \mathbb{R}^{>0}$ gives for each $x \in \mathcal{E}$ the size of the Voronoi cell of x in $\mathcal{V}_G(S \cup \{x\})$:

$$A(x) := |\mathrm{Vor}_{G,(S\cup\{x\})}(x)|.$$

Recall that we wanted to prevent infinitely many boundary points, so if there exist $v \in V$ and $s \in S$ such that $d(x, v) = d(s, v)$ the value of $A(x)$ will be undefined. We will identify points x for which this is the case in the next section.

To compare the sizes of $\mathrm{Vor}(x)$ with the sizes of the remaining Voronoi cells their sizes also have to be known. We can assume their original sizes are known. The newly placed point *conquers* areas from the original Voronoi cells. We therefore define the family of functions $\bar{A}_j$ to be the size of the conquered area:

Definition 3.5. Let $S = \{s_1, ..., s_l\}$ be the set of sites.

$$\bar{A}_j(x) = |\mathrm{Vor}_{G,S}(s_j)| - |\mathrm{Vor}_{G,(S\cup\{x\})}(s_j)| \text{ for } 1 \leq j \leq l.$$

It is easy to see that the following relation between the functions A and $\bar{A}_j$ holds:

$$A = \sum_{j=1}^{l} \bar{A}_j. \tag{1}$$

The problem can now be reformulated into the following: find the set $W \subseteq \mathcal{E}$ of all points x for which

$$\forall s_j \in S : A(x) \geq |\mathrm{Vor}_{G,S}(s_j)| - \bar{A}_j(x)$$

which is exactly the set of winning points in (G, S).

3.4. Critical points

In this section we will look at the structure of the functions A and $\bar{A}_j$. In particular, we will prove the following lemma:

Lemma 3.1. *Let $G = (V, E)$ be a graph and let $S \subseteq V$ be a set of sites.*

(a) A is a piecewise linear function;

(b) $\bar{A}_j$ is a piecewise linear function for all $1 \leq j \leq l$.

The remainder of this section is dedicated to finding the set Γ of *critical points* on which A and $\bar{A}_j$ are not linear and showing the linearity of these functions on $\mathcal{E} \setminus \Gamma$.

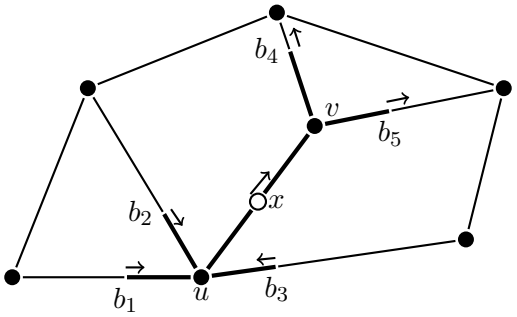


Figure 7: Example of the movement of boundaries when there is no interaction with other graph elements.

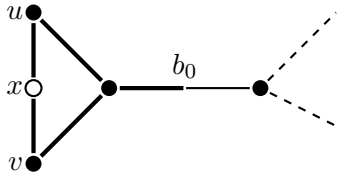


Figure 8: A boundary b_0 with two distinct shortest paths through both endpoints of the edge on which x is lying, making x a type I critical point.

Proof. First remark that (a) follows directly from (b) and equation (1), because a finite sum of piecewise linear functions is piecewise linear. The remainder of this proof is therefore aimed at proving (b).

Let us consider the Voronoi diagram $\mathcal{V}_G(S \cup \{x\})$ after adding x as a site. Figure 7 shows an example of a how such a Voronoi cell might look. Remark that such a Voronoi cell is a finite union of shortest paths from x to the boundaries b_i of its Voronoi cell. Since x lies on the edge $\{u, v\}$, we can divide the paths into paths that go through u and paths that go through v.

For now assume that all the boundaries lie on the interior of an edge. Imagine x moving along the edge with a certain distance ϵ towards v. All the boundaries that are linked to a path through u will then move in the direction of x with a distance of $\frac{\epsilon}{2}$, decreasing the size of the Voronoi cell by $\frac{\epsilon}{2}$, while all the boundaries linked to a path through v will move away from x with a distance of $\frac{\epsilon}{2}$, causing the Voronoi to grow by $\frac{\epsilon}{2}$ (see Figure 7).

Before we can conclude the Voronoi size changes in a linear fashion, we have to consider the fact that there might exist a boundary with two (or more) distinct shortest paths associated with it. If both paths go through either u or v, the boundary will behave no different, but if one path goes through u and one goes through v, we find that the Voronoi cell will always grow along this boundary if x is moved in either direction. Since the functions A and $\bar{A}_j$ cannot be linear in points for which such a pair of paths exist, we have identified a subset of Γ which we will call Γ_1 or *type I critical points*:

(a) Before (b) After

Figure 9: An example of a vertex flipping owners.

Definition 3.6. $\Gamma_1 \subseteq \mathcal{E}$ is the set of points x for which the Voronoi cell $\text{Vor}_{G,(S\cup\{x\})}(x)$ has a boundary with two equally long shortest paths going through the two different endpoints of the edge on which x is situated (see Figure 8).

It is simple to see that Γ_1 is finite: if there is a point $x \in \Gamma_1$ and if we look at the two shortest paths between x and the boundary that causes x to be in Γ_1, we find that along the paths must be at least one vertex with two equally long shortest paths to x through the different endpoints of the edge on which x lies, namely the vertex the paths visit last before the boundary. For every vertex there can be at most a finite number of points on the graph with two equally long shortest paths trough the different endpoints of its edge: at most one on every edge (see also Section 3.4.1 for a more detailed analysis).

If we now assume that $x \notin \Gamma_1$ and still assume that all boundaries lie on the interior of an edge, it is trivial to see that moving x along its edge changes the size of the Voronoi cell linearly. Since all the boundaries move linearly along their edge, the remaining Voronoi cells also change in size linearly, hence we conclude that if for a given x all boundaries lie on the interior of an edge and $x \notin \Gamma_1$, the functions A and $\bar{A}_j$ are linear in x.

The only case that remains is when one or more of the boundaries coincide with a vertex. Recall that we do not allow a placement for x such that a boundary coincides with a vertex to avoid the problem with infinite boundaries, so the functions A and $\bar{A}_j$ have a discontinuity at this point.

Even if we would allow such a placement, we would find that the functions are not necessarily linear in this situation. The number of boundaries the Voronoi cell $\text{Vor}_{G,(S\cup\{x\})}(x)$ will be different depending on the direction in which x is moved. Further, if x is moved towards the disputed vertex, not only the vertex will be conquered by x, but also the entire section of the graph that originally had a shorter path to another site s_j through the disputed vertex, i.e. the entire subtree with the vertex as root in the ESPT used to calculate the Voronoi diagram (see Figure 9).

We will call the set of points for which a boundary coincides with a vertex Γ_2 or *type II critical points*.

Definition 3.7. $\Gamma_2 \subseteq \mathcal{E}$ is the set of points x for which one of the boundaries of the Voronoi cell $\text{Vor}_{G,(S\cup\{x\})}(x)$ coincides with a vertex.

Again it is easy to see that Γ_2 is finite as well: if we consider the extended shortest path tree with some vertex v, each point on the graph will be contained in the tree somewhere. Each path to one of the leaves of the tree will contain at most one point

Figure 10: Two boundaries on a single edge.

with an equal shortest distance to v, thus possibly causing a boundary to lie in v. Therefore we conclude that for each vertex v we have a finite number of points that can cause a boundary to lie in v and thus Γ_2 must be of finite size as well (see also Section 3.4.2).

Finally we can conclude that the functions A and $\bar{A}_j$ behave linearly on $\mathcal{E}$ with the exception of *critical points* in $\Gamma = \Gamma_1 \cup \Gamma_2$. Γ is the union of two finite sets and thus finite itself. Hence the functions $\bar{A}_j$ are piecewise linear. As argued before it also immediately follows that A is piecewise linear. □

As we now have a distinction of all possible points of the functions A and $\bar{A}_j$ in which they are not linear, we can estimate how many of these points or analogously how many linear components the functions have on their entire domain. First we will introduce a useful lemma:

Lemma 3.2. *Let $G = (V, E)$ and $S \subseteq V$ be defined as before. For every $x \in \mathcal{E}$, every edge $e \in E$ will contain at most one boundary.*

Proof. Let $e \in E$ and assume there is more than one boundary on e. Let b_1 and b_2 be two neighbouring boundaries (see Figure 10). Now consider an arbitrary point p that lies on e between b_1 and b_2. This point p must have a site s closer than the two sites which own the outer two Voronoi cells. However, any shortest path from p to s must pass through one of the other Voronoi cells. Hence p can never be closer to s than to the site that owns the Voronoi cell the shortest path passes through. Therefore it is impossible that there is an area between the two boundaries that belongs to another Voronoi cell. As having this area follows directly from having two boundaries on one edge, our original assumption that this is possible must be false[1].

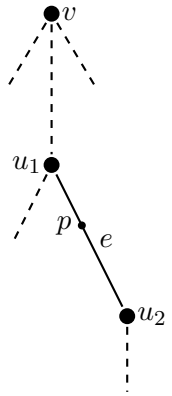
□

We will discuss both types of critical points in detail below.

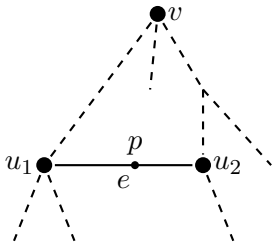
3.4.1. Type I

Remember that for type I critical points we have a boundary such that there are two shortest paths with equal length through different endpoints of the edge on which x lies. It trivially follows that both these paths must have at least one vertex they pass through in common: the endpoint of the edge on which the boundary lies the paths both pass through. We can therefore define the superset $\Gamma_1' \supseteq \Gamma_1$ of all points $x \in \mathcal{E}$ with two shortest paths to any vertex $v \in V$ with equal length through different endpoints of the

[1] Note that from this proof it immediately follows that Voronoi cells are star-shaped with respect to their site, as stated in property NL1 from [14].



(a) e is in the shortest path tree of v.



(b) e is not in the shortest path tree of v.

Figure 11: Illustration of the situations as sketched in the proof of Lemma 3.3.

edge on which x lies. An upper bound for this set will immediately give us an upper bound for the number of type I critical points.

To deduce an upper bound, we evaluate for each $v \in V$ how many points on the graph have two shortest paths to v in such a way that they could be a critical point by considering the following lemma:

Lemma 3.3. *Let $v \in V$ be a vertex and $e = \{u_1, u_2\} \in E$ an edge on a connected graph $G = (V, E)$. Then there is a point p on the interior of e such that the shortest path from p via u_1 to v and the shortest path from p via u_2 to v have equal length if and only if e is not in the shortest path tree with v as root.*

Proof. First let us assume there is such a point p of which the two shortest paths via respectively u_1 and u_2 to v are equally long. To find a contradiction, assume e lies on the shortest path tree. Without loss of generality let u_1 be the parent of u_2 in the shortest path tree (see Figure 11a).

We know that the length of the shortest path from u_2 to v is $d(u_2, v) = d(u_1, v) + w(e)$. Hence the length of the shortest path from p to v via u_2 will be $d(p, u_2) + d(u_2, v) = d(p, u_2) + d(u_1, v) + w(e)$. The length of the shortest path from p via u_1 to v will be $d(p, u_1) + d(u_1, v)$. Both paths have to be equally long, so we find $d(p, u_1) + d(u_1, v) = d(p, u_2) + d(u_1, v) + w(e) \Rightarrow d(p, u_1) - d(p, u_2) = w(e)$. As p lies on the interior of e, we know that $d(p, u_1) > 0$ and $d(p, u_2) > 0$, while we also know $d(p, u_1) + d(p, u_2) = w(e)$. It is thus impossible that the difference between $d(p, u_1)$ and $d(p, u_2)$ equals the length of the edge, resulting in a contradiction.

Now let e be an edge that does not lie on the shortest path tree (see Figure 11b). We know that $|d(u_1, v) - d(u_2, v)| < w(e)$. Surely if the opposite holds and we assume (without loss of generality) $d(u_1, v) < d(u_2, v)$, we find that $d(u_1, v) + w(e) < d(u_2, v)$ must also hold, which leads to a contradiction as we can then find a shorter path from v to u_2.

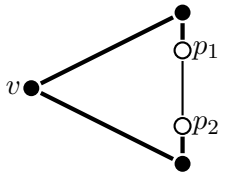


Figure 12: An example of an instance with two points with a distance d_0 from v on a single edge.

From $|d(u_1, v) - d(u_2, v)| < w(e)$ it follows that we can find a point p on e such that $d(p, u_2) - d(p, u_1) = d(u_1, v) - d(u_2, v) \Rightarrow d(p, u_1) + d(u_1, v) = d(p, u_2) + d(u_2, v)$, providing two shortest paths (one through u_1 and one through u_2) with equal length. □

From this lemma it follows that for every vertex, every edge contains at most one critical point. From here we can trivially deduce an upper bound of $|V||E|$ points in Γ_1', and therefore an asymptotic upper bound of $O(|V||E|)$ type I critical points. This is a tight upper bound, as will be shown at the end of the next section.

3.4.2. Type II

The second set of points to consider are points x for which one of the boundaries of the new Voronoi cell $\text{Vor}_{G,(S\cup\{x\})}(x)$ coincides with a vertex. Remark that this is equivalent to saying that there exists a vertex v such that $\min_{s\in S} d(s, v) = d(x, v)$.

The following lemma will help us find an upper bound on the number of type II critical points.

Lemma 3.4. *Let $G = (V, E)$ be a connected graph as before, and let further $v \in V$ be a vertex and $d_0 \in \mathbb{R}^{>0}$ a positive number. On every edge there are at most two points $p_1 \neq p_2$ such that $d(v, p_1) = d(v, p_2) = d_0$, and this is a tight upper bound.*

Proof. To get a contradiction, assume there are three different points p_1, p_2, p_3 on an edge e such that $d(v, p_1) = d(v, p_2) = d(v, p_3) = d_0$ and without loss of generality let p_1 and p_3 be the two outer points. The shortest path from p_2 to v will then either go through p_1 or p_3. Assume without loss of generality the former. As $d(v, p_1) = d(v, p_2) = d_0$, we find that $d(p_1, p_2) = 0$. The points have to be distinct, so this can only happen if $w(e) = 0$. Since we assume positive edge weights for every edge, we find a contradiction and conclude there can be at most two points on an edge with distance d_0 to v.

Figure 12 shows an example of two points on the same edge equidistant to a vertex v, proving that the upper bound is tight.

□

If we now replace d_0 by the distance from v to its closest site, the upper bound we found can be immediately translated to the upper bound of $|\Gamma_2|$: for every vertex we have at most $2 \cdot |E|$ type II critical points, so we find an overall upper bound of $2|V||E|$ or $O(|V||E|)$ points.

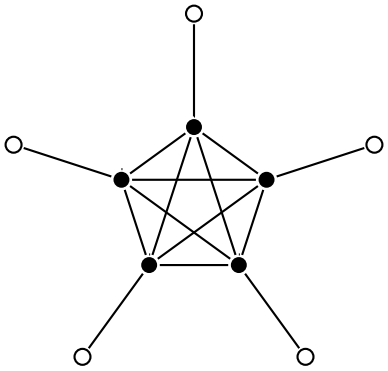


Figure 13: An illustration of an example graph that achieves the tight upper bound for type I critical points (edge lengths have been omitted for clarity).

This upper bound is also a tight upper bound. Consider a graph that consists of a clique with unit length edges with an additional edge from every vertex to a new degree one vertex with a unique length strictly between one and two unit lengths. These additional edges have a site on their endpoint. See Figure 13 for an example with a clique of size five. Each vertex v will generate one type I critical points and two type II critical points on every edge except the edges adjacent to v. This gives $O(|V||E|)$ critical points of both types.

3.5. Calculating critical points

Once we identify all critical points, we can calculate the values of A and $\bar{A}_j$ in each of these critical points and the vertices of G. To achieve this, we can use the linearity of these functions on edge segments between vertices and critical points.

Definition 3.8. An *elementary segment* is a maximal connected subset of an edge not containing a vertex or critical point.

In this section we will discuss how the critical points are evaluated.

3.5.1. Type I

We can simplify the calculation of type I critical points by calculating its superset Γ_1'. Lemma 3.3 tells us where to find these points already: we know that all edges that do not lie on the shortest path tree with v as its root contain a point from which we can find two different shortest paths to v with equal length through the different endpoints of the edge. We can then compare the ancestors of the endpoints of the edge to check if the paths are disjoint, to make sure we don't calculate points doubly, since the two paths might have more than just the final edge segment in common, and thus have more than two vertices with equal distance through both endpoints to the critical points. If we calculate this shortest path tree, we can find all critical points for a vertex v easily by checking all edges against the shortest path tree.

The procedure is described in Algorithm 1. In this algorithm $p_v(u)$ denotes the parent of u in the shortest path tree of v.

Algorithm 1

```
function CalculateTypeIPoints(G = (V, E))
    Γ1 ← ∅
    for all v ∈ V do
        CalculateShortestPathTree(G, v)
        for all {u1, u2} ∈ E do
            if InSameComponent(v, u1) & p_v(u1) ≠ u2 & p_v(u2) ≠ u1 then
                t ← 0.5 + 0.5 * [d(v, u1) − d(v, u2)]/w({u1, u2})
                Γ1 ← Γ1 ∪ {({u1, u2}, t)}
            end if
        end for
    end for
    return Γ1
end function
```

Calculating the shortest path tree SPT_v for a given vertex v can be done using Dijkstra's algorithm [8] in $O(|V| \log |V| + |E|)$ time. We then iterate through all edges and check whether the edge lies in the shortest path tree by checking if the two endpoints are each other's parent in SPT_v. This is a constant time lookup. We then calculate the exact location of the critical point on the edge if necessary in constant time.

The entire loop therefore takes $O(|E|)$ time, giving a total running time of $O(|V| \log |V| + |E|)$ time for one vertex, and $O(|V|^2 \log |V| + |V||E|)$ time for the entire graph.

3.5.2. Type II

Next, we calculate all points that lie at an equal distance from a given vertex v as its closest site. Given a shortest path tree, we can find the closest site to v in $O(|S|)$ time, giving us a total running time of $O(|V| \log |V| + |E| + |S|)$ time. Note that while we could mark the closest site during the execution of Dijkstra's algorithm, the worst-case running time cannot be improved as in some obscure cases we might repeatedly find a closer site. Since we assumed $S \subseteq V$, we can ignore the $|S|$ term in the running time.

If the original Voronoi diagram $\mathcal{V}(S)$ is given, the closest site for a given vertex v can be found trivially in constant time. We can compute the Voronoi diagram through the extended shortest path tree which will give us the closest site and distance to it for each vertex.

Given a shortest path tree with v as root we can easily find points with a given distance from v by iterating over all edges, checking the distance to v from each of the endpoints and determining whether there is a point lying on the edge with the specified distance (see Algorithm 2). This gives us a running time of $O(|E|)$ for one vertex if the original Voronoi diagram is given, and $O(|V||E|)$ time for the entire graph.

Algorithm 2

```
function CalculateTypeIIPoints(G = (V, E))
    Γ_2 ← ∅
    for all v ∈ V do
        CalculateShortestPathTree(G, v)
        s ← closest site to v
        for all {u_1, u_2} ∈ E do
            if d(u_1, v) < d(s, v) < d(u_1, v) + w({u_1, u_2}) & p_v(u_1) ≠ p_s(u_1) then
                t ← [d(s, v) − d(u_1, v)]/w(e)
                Γ_2 ← Γ_2 ∪ {({u_1, u_2}, t)}
            end if
            if d(u_2, v) < d(s, v) < d(u_2, v) + w({u_1, u_2}) & p_v(u_2) ≠ p_s(u_2) then
                t ← 1 − [d(s, v) − d(u_2, v)]/w(e)
                Γ_2 ← Γ_2 ∪ {({u_1, u_2}, t)}
            end if
        end for
    end for
    return P
end function
```

Together with the type I critical points and the sites, we come to a total time requirement of $O(|V|^2 \log |V| + |V||E|)$ time to calculate all critical points.

3.6. Calculating win areas

We now know the locations of the critical points, but we still have to calculate the values of A and $\bar{A}_j$ for these points. Recall that A and $\bar{A}_j$ are not defined for type II critical points and sites. If we approach these points along the graph the function will approach a different value for either side. We can calculate this limit and use the two obtained values for interpolation on the elementary segments.

Given the original Voronoi diagram $\mathcal{V}(S)$ and for each vertex the closest distance to its closest site, we can calculate the new Voronoi diagram $\mathcal{V}(S \cup \{x\})$ by expanding a shortest path tree from x where we only add vertices to the tree if the distance to x is smaller than the distance to their current closest site.

Now we only have to resolve new boundaries on the edges. If we look at the shortest path tree of x, we notice that every edge contained in this tree will be fully added to the new Voronoi cell. The edges adjacent to the shortest path tree (i.e. adjacent to the leaves of the tree) will partly belong to the new Voronoi cell, since we know the two endpoints of those edges belong to different Voronoi cells, thus there must be a boundary somewhere along the edge. All the remaining edges can not be in the Voronoi cell, since neither of their endpoints are in the shortest path three.

While expanding the shortest path tree, we incrementally calculate the values of A and $\bar{A}_j$ by keeping track of the areas we cover. From Lemma 3.2 we remember that each

edge in G contains at most one boundary in the original Voronoi diagram, hence we only have to do a constant amount of work for each edge we visit. Therefore the total time required to calculate each function is $O(|V|\log|V| + |E|)$ time, giving a total running time $O(|S|(|V|\log|V| + |E|))$ time to calculate the values of A and $\bar{A}_j$ for a given point.

This calculation has to be repeated for every vertex and critical point. We can then iterate through all elementary segments and find the areas for which $A \geq \max_{s_j \in S} \mathrm{Vor}(s_j) - \bar{A}_j$ in $|S|$ time. Combining this gives us an algorithm that calculates the winning areas in $O(|V||E||S| \cdot (|V|\log|V| + |E|))$ time.

Theorem 3.5. *Let $G = (V, E)$ be a graph and S a set of sites as before. The set W of winning points can be calculated in $O(|V||E||S| \cdot (|V|\log|V| + |E|))$ time.*

3.7. Generalisations

Recall that we forbid placements x for which there exists a vertex v and site s with $d(x, v) = d(s, v)$, since this can cause an infinite number of points equidistant to x and s as shown before in Figure 5. As opposed to forbidding these placements, we can also define the behaviour for points equidistant to two sites: since we have already introduced an arbitrary ordering $s_1, ..., s_l$, we can say that points with equal distances to s_j and $s_{j'}$ with $j < j'$ fall to s_j. The placement x can be inserted in this order arbitrarily. It is easy to see that this behaviour does not invalidate any result discussed in this section.

We have also so far required the graph G to be connected. However, it is easy to see that the algorithm easily translates to graphs with multiple components.

4. Colours

We will now look at the generalisation of the problem where multiple sites can be grouped into colours.

In our new problem, every site is assigned to a colour. In other words, we have a partition of the set of sites S into a set of colours $\mathcal{C} = \{C_1, ..., C_r\}$.

Now the problem can be altered to require the new site x to create a Voronoi diagram that is larger than the sum of the sizes of Voronoi cells of every colour. In other words, find the set $W \subseteq \mathcal{E}$ of all points x for which

$$\forall C_q \in \mathcal{C} : |\mathrm{Vor}(x)| \geq \sum_{s_j \in C_q} |\mathrm{Vor}(s_j)|.$$

If $x \in W$, we call x a *winning point with respect to colours*.

We can calculate this set W using the same algorithm as described in the previous section. The only changes that have to be made are the comparisons for each elementary segment. We now have to compare the function A against a finite sum of functions $|\mathrm{Vor}(s_j)| - A_j$, which are linear on the elementary segment we are currently considering, making the sum linear as well. This does not change the complexity of the algorithm in any way, hence we can trivially extend the algorithm to include colours.

Furthermore, the problem without colours can be seen as special case of the more generic problem: namely if we would colour each site differently, we would have our original problem back.

It is also possible to have an instance where some of the existing sites are assigned to the player. That is, the player already starts with some portion of the graph covered in their own colour. Since we evaluate the size of each of the Voronoi cells, it is easy to make a small adjustment to the algorithm that adds the areas from the existing player sites to the covered area before comparing against the remaining colours.

Theorem 4.1. *Let $G = (V, E)$ be a graph, S a set of sites, and $\mathcal{C}$ a set of colours as before. The set W of winning points with respect to colours can be calculated in $O(|V||E||\mathcal{C}| \cdot (|V| \log |V| + |E|))$ time.*

5. Two sites

After introducing colours for the existing sites, the next extension of the algorithm is to allow the placement of multiple points $x_1, ..., x_k$ for which we add up the covered areas. We thus look for the set $W^k \subseteq \underbrace{\mathcal{E} \times ... \times \mathcal{E}}_{k \text{ times}}$ of all sets X with $|X| = k$ for which

$$\forall C_q \in \mathcal{C} : \sum_{x_i \in X} |\text{Vor}(x_i)| \geq \sum_{s_j \in C_q} |\text{Vor}(s_j)|. \tag{2}$$

Observe that elements of W^k behave exactly like a colour. This problem could therefore be considered as adding k points that each have the same colour. We are then looking for a placement of k points with a new colour such that the portion of the graph that is coloured in this new colour is larger than each of the portions coloured by other colours.

If $X \in W^k$ is a placement fulfilling the conditions posed above, we call X a *winning k-set*.

To simplify the problem we will initially fix $k = 2$. In the next section we will discuss how we can generalise the problem to the placement of an arbitrary number of points.

5.1. Inclusion-exclusion principle

If we are given the first of the two points, we can easily find the winning positions for the second point using a variation of the previously discussed algorithm. However, there are infinitely many placements possible for the first point. Hence we would have to solve the sub-problem of placing the second point in a parametrised fashion.

We can also use a different approach based on the inclusion-exclusion principle. Instead of looking at the Voronoi cell $\text{Vor}(x_i)$ of a point $x_i \in X = \{x_1, x_2\}$ in the Voronoi diagram $\mathcal{V}(S \cup X)$, we can look at the Voronoi cell in $\mathcal{V}(S \cup \{x_i\})$. Ideally, adding up the sizes of both these Voronoi cells for $x_i \in X$ yields the size of the total covered area in the complete Voronoi diagram $\mathcal{V}(S \cup X)$. Unfortunately it is possible that the two points x_1 and x_2 partly cover the same area in the graph, which would invalidate the sum as size of the total covered area.

To offset this we have to subtract all the areas that are covered twice. The following functions will give us exactly this area for two points:

Definition 5.1.

$$B(x_1, x_2) = |\text{Vor}_{G,S\cup\{x_1\}}(x_1) \cap \text{Vor}_{G,S\cup\{x_2\}}(x_2)|.$$

$$\bar{B}_j(x_1, x_2) = |\text{Vor}_{G,S\cup\{x_1\}}(x_1) \cap \text{Vor}_{G,S\cup\{x_2\}}(x_2) \cap \text{Vor}_{G,S}(s_j)|.$$

The function B will give us the size of the overlapping area of the two Voronoi cells of x_1 and x_2 if they were added to the graph separately. Similarly to the family of functions $\bar{A}_j$ we have defined before, the family of functions $\bar{B}_j$ gives us for each j the size of the

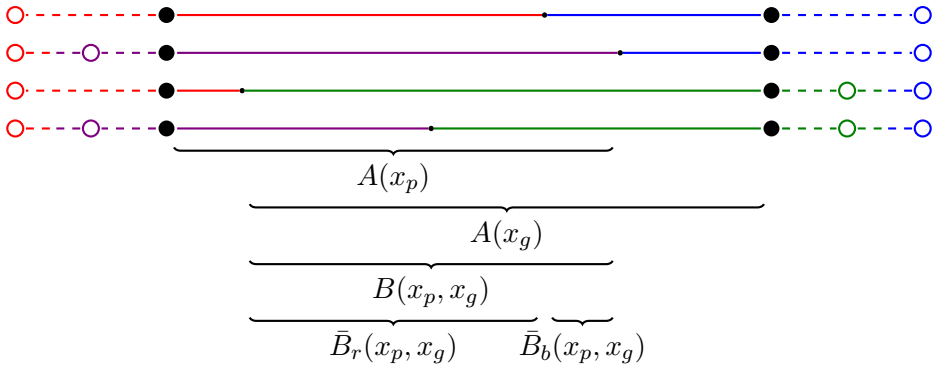


Figure 14: An illustration of the functions A, B and $\bar{B}_j$ on a single edge. Lengths of dashed edges are not drawn proportionally.

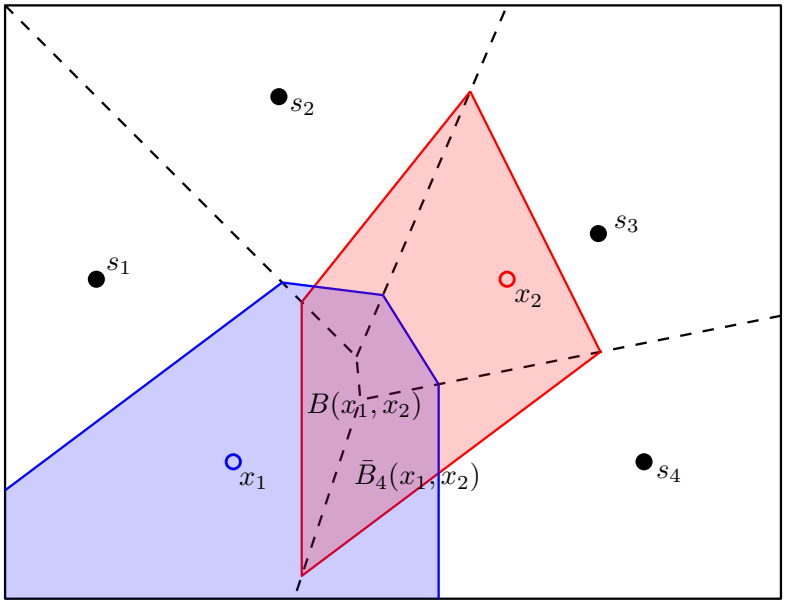


Figure 15: An illustration of the functions B and $\bar{B}_j$ in a Euclidean setting. Remark that $\bar{B}_4(x_1, x_2) \subseteq B(x_1, x_2)$.

area we have counted as conquered from the point s_j twice (see Figures 14 and 15). Like before (see (1)) the following relation between B and $\bar{B}_j$ holds:

$$B = \sum_{j=1}^{l} \bar{B}_j, \tag{3}$$

Using these functions we can rewrite the problem as defined in (2) for $k = 2$ in terms of A and B: find the set W^2 of all pairs $X = \{x_1, x_2\}$ for which

$$\forall C_q \in \mathcal{C} : A(x_1) + A(x_2) - B(x_1, x_2) \geq \sum_{s_j \in C_q} \left[|\mathrm{Vor}_{G,S}(s_j)| - \bar{A}_j(x_1) - \bar{A}_j(x_2) + \bar{B}_j(x_1, x_2) \right].$$

With our algorithm so far, only the functions B and $\bar{B}_j$ are not yet known. We will see that these functions have a structure comparable to the functions A and $\bar{A}_j$, which allows us to evaluate them in a similar manner.

5.2. Calculating intersection sizes

We will consider the behaviour of the functions B and $\bar{B}_j$ on elementary segments. That is, let x_1 and x_2 move along two elementary segments parametrised by λ and μ respectively. Now let us look at what can happen on a single edge $e = \{u, v\}$ in G.

Observe that the following cases are exhaustive: it is impossible that a boundary of either Voronoi cell only lies on the edge for a subspace of the parameter space spanned by λ and μ as x_1 and x_2 are not passing through any critical points.

(i) For all λ and μ, there is at least one Voronoi cell of $\mathrm{Vor}(x_1)$; $\mathrm{Vor}(x_2)$ which does not intersect with e. The size of the intersecting area on e is zero.

(ii) For all λ and μ, e is both in the interior of $\mathrm{Vor}(x_1)$ and $\mathrm{Vor}(x_2)$. The size of the intersection area on e is the total length of the edge.

(iii) For all λ and μ, e is in the interior of one of the cells (say $\mathrm{Vor}(x_1)$) and partly in the interior of the other ($\mathrm{Vor}(x_2)$). As x_2 moves, the boundary of $\mathrm{Vor}(x_2)$ moves along the edge, hence the size of the intersection is on this edge is linear in μ.

(iv) For all λ and μ, e contains a boundary of both $\mathrm{Vor}(x_1)$ and $\mathrm{Vor}(x_2)$ and there are no λ and μ for which the two boundaries coincide. The intersection between $\mathrm{Vor}(x_1)$ and $\mathrm{Vor}(x_2)$ can never be empty, since that would imply a situation as shown in Figure 16b. Adding both x_1 and x_2 would result in an area between the two boundaries on e of these Voronoi cells that is not covered by either which is not possible by the same argument as used in the proof of Lemma 3.2. The intersection is thus non-empty for all λ and μ, and the size of the intersection is linear in λ and μ.

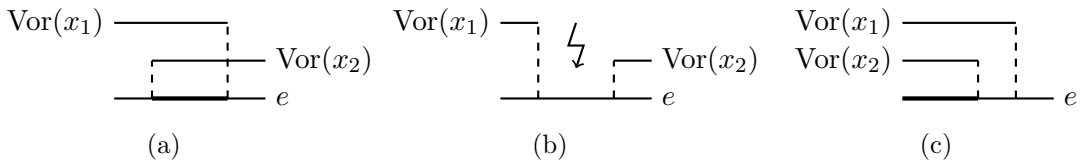


Figure 16: The different possibilities of how two Voronoi cells partly overlap with an edge. The Voronoi cells are displayed above the edge for clarity.

(v) For all λ and μ, e contains a boundary of both Vor(x_1) and Vor(x_2) and there is at least one choice of λ and μ for which the boundaries coincide. We can distinguish the case where the Voronoi cells cover segments on different sides of the edge (Figure 16a), or the same side (Figure 16c).

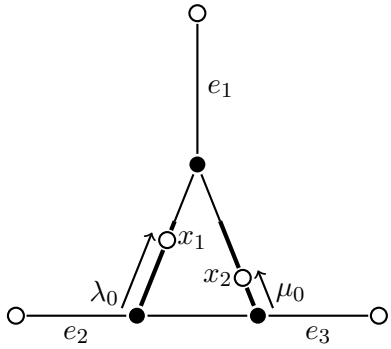
In the first case, since we never have an empty segment between the two cells, the borders must coincide for some choice of λ and μ at the boundary of their respective domain. Since we assume elementary segments to be open segments, this case is thus identical to the previous case, in which the size of the intersection is linear in λ and μ.

In the latter case, we might have an infinite number of choices of λ, μ for which the boundaries coincide: if we have such a placement of x_1 and x_2 for which the boundaries coincide, we can move both x_1 and x_2 with distance ϵ along their respective segment—since elementary segments are open—to find another placement for which the boundaries coincide. More precisely, we find that the boundaries coincide for every choice of λ and μ according to a linear relation between the two parameters. The size of the intersection area on this edge is therefore linear in λ and μ except for a zero-area subset—a line segment that will always have a slope of $\pm 45°$—of choices.

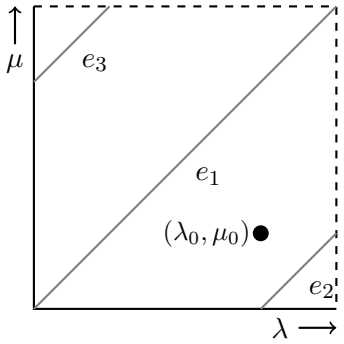
If we look at the parameter space Λ of the function B spanned by λ and μ, we can divide the space in two subspaces in which the contribution of e to the intersection area is linear. The boundary between these two subspaces is given exactly by the points (λ, μ) for which the boundaries of the Voronoi cells coincide as described in the last case above. These points will lie along a line inside the parameter space.

Each of the edges in G might cause such a split in the parameter space for its contribution to the total size of the intersection of the Voronoi cells. The total size of the intersection is the sum of all contributions, so we find that the size is piecewise linear in λ and μ. There is at most one line along which the parameter space is split for each edge, hence the parameter space is subdivided in $O(|E|^2)$ sections on which the function B is linear in λ and μ (see Figures 17 and 18).

If x_1 and x_2 lie on the same critical segment, we have to exclude the solutions for which $\lambda = \mu$ and thus $x_1 = x_2$. In this case we can split Λ once more by the line $\lambda = \mu$. Since this will at most double the total number of cells, the number of cells in the parameter space does not asymptotically increase. Remark that it is also possible to only consider the part of the parameter space for which $\lambda < \mu$ (or vice versa) since the functions are



(a) The instance with two elementary segments marked.



(b) The parameter space of B of the marked elementary segments.

Figure 17: Example of the parameter space of B in λ and μ. The grey lines mark the segments along which the function is not differentiable.

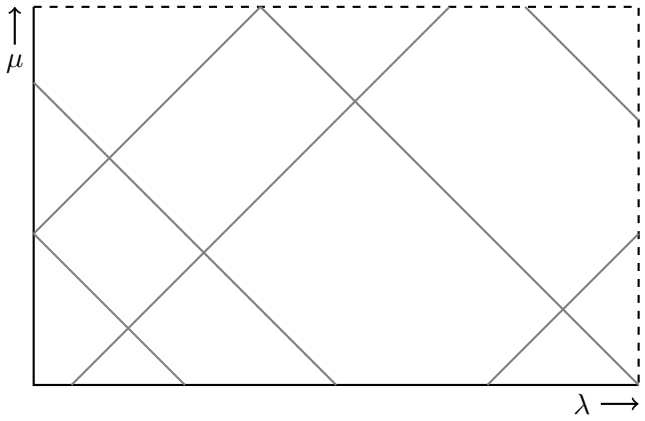


Figure 18: The generic structure of the parameter space of B in λ and μ. The grey lines mark the segments along which the function is not differentiable.

symmetric in λ and μ.

So far we have determined that the function B is piecewise linear, but we also need to evaluate the values of the functions $\bar{B}_j$. If the edge e was contained in a single Voronoi cell in the original Voronoi diagram we do not have to worry about additional breaks in the functions $\bar{B}_j$ because the contribution of the edge to $\bar{B}_j$ will be either zero or equal to its contribution to B.

In the other case e must contain two Voronoi cells and thus a boundary in the original Voronoi diagram. All of the new Voronoi cells that cover a portion of e must contain this boundary. In particular, this means that for all λ and μ the boundary is contained in the intersection of all Voronoi cells on e and there are no λ and μ for which the boundaries of the new Voronoi cells coincide with the boundary of the original Voronoi diagram. While the intersection might contain more than one Voronoi cell on an edge, the functions $\bar{B}_j$ cannot have a break unless the total function B has a break. The functions $\bar{B}_j$ are therefore piecewise linear, and linear in the cells we have already constructed in the parameter space Λ.

We can also express the functions A and $\bar{A}_j$ in the parameters λ and μ. Furthermore these functions will be linear on the elementary segment we are currently considering. We thus find that we can calculate the total covered size and the size of each original Voronoi cell in $\mathcal{V}(S \cup \{x_1, x_2\})$ using these functions for every placement of x_1 and x_2 along the selected segments. To find all winning 2-sets on this pair of elementary segments we have to compare the size functions in each of the cells in the parameter space.

5.3. Comparing colours

We have now divided our domain Λ into $O(|E|^2)$ areas in which the covered area for every colour is a linear function in λ and μ. To find all λ and μ—and therefore placements of x_1 and x_2 on the elementary segments we are currently considering—such that $\{x_1, x_2\}$ is a winning 2-set, we have to compare the size of the area covered by x_1 and x_2 to the area covered by the colours.

The total area covered by x_1 and x_2 can be expressed as

$$A(x_1) + A(x_2) - B(x_1, x_2)$$

and the total area covered by a colour $C_q \in \mathcal{C}$ after adding x_1 and x_2 can be expressed as

$$\sum_{s_j \in C_q} |\mathrm{Vor}_{G,S}(s_j)| - \bar{A}_j(x_1) - \bar{A}_j(x_2) + \bar{B}_j(x_1, x_2).$$

The sizes of the original Voronoi cells are given, and all the functions $A, \bar{A}_j, B$, and $\bar{B}_j$ are linear in λ and μ in each cell of the parameter space. Hence, if we pairwise compare the covered area to each of the areas of the remaining colour, we find a set of linear equations. Each of these equations represents a half-plane in the infinitely extended parameter space spanned by λ and μ in which the covered area is larger than the respective colour's area (see Figure 19). The intersection of these half-planes thus provides the choices for λ and μ for which $\{x_1, x_2\}$ is a winning 2-set.

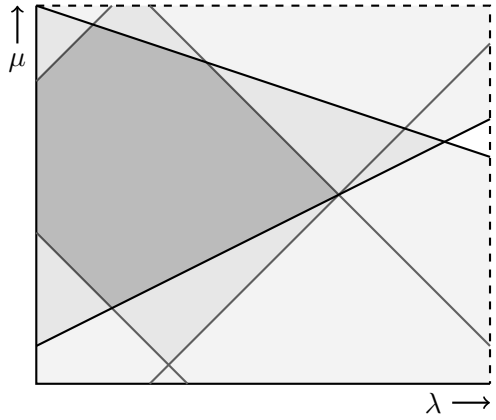


Figure 19: Example of how the set of winning pairs might look within a single cell in the parameter space. Gray lines represent the cell boundaries, black lines represent the borders of the half-planes in which the player's colour covers a larger portion of the graph than another colour.

We have $|\mathcal{C}|$ colours to compare against, so for each cell we have to calculate the intersection of $|\mathcal{C}|$ half-planes, which can be done in $O(|\mathcal{C}| \log |\mathcal{C}|)$ time [6]. Since we only want to limit the solutions to a single cell we add additional half-planes that represent the boundaries of the cell. As there will be at most eight (the axis-aligned parameter space boundaries and four diagonal boundaries), this does not make the running time asymptotically worse.

The total running time for calculating the winning area for one pair of elementary segments now sums up to $O(|E|^2|\mathcal{C}| \log |\mathcal{C}|)$ time. Therefore all winning combinations of points x_1 and x_2 can be found by calculating the winning placements given the choice of two elementary segments (of which there are $O(|V||E|)$) which takes a total of $O(|V|^2|E|^4|\mathcal{C}| \log |\mathcal{C}|)$ time.

Theorem 5.1. *Let $G = (V, E)$ be a graph, S a set of sites, and $\mathcal{C}$ a set of colours as before. The set W^2 of winning 2-sets can be calculated in $O(|V|^2|E|^4|\mathcal{C}| \log |\mathcal{C}|)$ time.*

6. k sites

In the previous section we looked at the generalisation of placing two points. Using similar concepts, the algorithm can also be extended to calculate the winning placements of an arbitrary number k of points. The algorithm follows the same structure as the algorithm for two points, but there are some complications, which make the problem harder to solve for an arbitrary number of points.

6.1. Generalised inclusion-exclusion principle

The algorithm will still heavily rely on the inclusion-exclusion principle. To make calculations using this principle possible, we introduced the functions B and $\bar{B}_j$ to represent the size of the intersections between two individually added Voronoi cells. If we extend the inclusion-exclusion principle we will need a function to represent the size of the intersection between any subset of the k placed points. Therefore we will redefine the functions A and $\bar{A}_j$ to be a function $\mathcal{P}(\mathcal{E}) \to \mathbb{R}$ as follows:

Definition 6.1.

$$A(X) = |\bigcap_{x_i \in X} \mathrm{Vor}_{G,S\cup\{x_i\}}(x_i)|.$$

$$\bar{A}_j(X) = |\bigcap_{x_i \in X} \mathrm{Vor}_{G,S\cup\{x_i\}}(x_i) \cap \mathrm{Vor}_{G,S}(s_j)|.$$

For clarity reasons (and with a slight abuse of notation), we will often remove the braces from the notation (i.e. $A(x_1, x_2)$ in fact means $A(\{x_1, x_2\})$). This means this new definition of A and $\bar{A}_j$ is fully compatible with the definition of B and $\bar{B}_j$ for two points, and also with the previous definition of A and $\bar{A}_j$ for one point.

The identities described in (1) and (3) also hold for the generalised definition. Recall that the sites are denoted by $s_1, ..., s_l$.

$$A = \sum_{j=1}^{l} \bar{A}_j. \qquad \text{(1 and 3 revisited)}$$

We can now again restate our problem in terms of functions: given $k \in \mathbb{N}$, find the set $W^k \subseteq \underbrace{\mathcal{E} \times ... \times \mathcal{E}}_{k \text{ times}}$ of all sets X with $|X| = k$ for which

$$\forall C_q \in \mathcal{C} : \sum_{0\neq J\subseteq X} \left[(-1)^{|J-1|} A(J)\right] \geq \sum_{s_j \in C_q} \left[|\mathrm{Vor}_{G,S}(s_j)| - \sum_{0\neq J\subseteq X} \left[(-1)^{|J-1|} \bar{A}_j(J)\right]\right].$$

6.2. Calculating intersection sizes

To calculate the total size of the conquered area using the inclusion-exclusion principle, we have to calculate the size of the intersection of each subset of individual Voronoi cells. In this section we will only describe how the intersection of all k cells is calculated. The intersection of the subsets can be solved in the same way using $k' < k$.

As before, we will be choosing an elementary segment for each x_i to be on. We can then parametrise the placement of x_i along this segment using a scalar λ_i. Given the elementary segments, we can define the functions A and $\bar{A}_j$ as functions of $\vec{\lambda} = (\lambda_1, ..., \lambda_k)$ on this subdomain.

We will consider the behaviour of the intersection of an individually added Voronoi cell $\mathrm{Vor}_{G,(S\cup\{x_i\})}(x_i)$ on a single edge. Again we know that—because we fixed the elementary segments on which the points $x_1, ..., x_k$ are placed—each of the Voronoi cells will always intersect the edge fully, partially, or not at all, for all choices of $\vec{\lambda}$. More precisely, if we consider the edge e to be an interval $[0, w(e)]$, the intersection of each Voronoi cell with the edge will be one of the following cases:

(i) $\emptyset$;

(ii) $[0, w(e)] = e$;

(iii) $[0, r_i(\lambda_i)]$, where r_i is a linear function of slope ± 1;

(iv) $[\ell_i(\lambda_i), w(e)]$, where ℓ_i is a linear function of slope ± 1 .

We will ignore the first two cases: if we have at least one Voronoi cell that does not intersect the edge at all, then the edge will never contain an intersection of all Voronoi cells; if a Voronoi cell fully contains the edge, the intersection of all Voronoi cells on this edge and the intersection of all Voronoi cells but this one is the same, so we can ignore it. Therefore we will assume that these cases do not occur (if they do, we can drop them and solve the problem for $k' < k$).

This leaves the last two cases. We can separate the points $x_1, ..., x_k$ into two sets: the set $\mathcal{L}$ of points x_i that have an interval $[0, r_i(\lambda_i)]$ as intersection from one side of the edge and the set $\mathcal{R}$ of points x_i that intersect a section $[\ell_i(\lambda_i), w(e)]$ of the edge from the other side. Remark that for any choice of $\vec{\lambda}$ we find that the intersection of all cells is given by

$$[\max_{x_i \in \mathcal{R}} \ell_i(\lambda_i), \min_{x_i \in \mathcal{L}} r_i(\lambda_i)].$$

We will define

$$r_{\min}(\vec{\lambda}) = \min_{x_i \in \mathcal{L}} r_i(\lambda_i);$$

$$\ell_{\max}(\vec{\lambda}) = \max_{x_i \in \mathcal{R}} \ell_i(\lambda_i).$$

Hence the intersection can also be expressed as

$$[\ell_{\max}(\vec{\lambda}), r_{\min}(\vec{\lambda})]$$

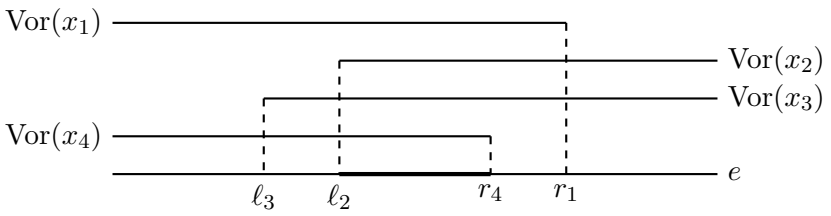


Figure 20: Example of the intersection of four Voronoi cells on a single edge. The Voronoi cells are displayed above the edge for clarity.

and the size of the intersection on e can be seen as a function

$$r_{\min} - \ell_{\max}.$$

See Figure 20 for an illustration of this terminology.

Recall that it is not possible to have $x_i \in \mathcal{L}$ and $x_{i'} \in \mathcal{R}$ such that $r_i < \ell_{i'}$, because the intersection between the two Voronoi cells on this edge would be empty, which is impossible as argued in the previous section: adding both points and thus Voronoi cells to the graph simultaneously would cause a neutral segment between the two Voronoi cells, which is impossible. By extension we find $r_{\min} \geq \ell_{\max}$. Hence if $\mathcal{L}$ and $\mathcal{R}$ are both non-empty, the intersection of all Voronoi cells will be non-empty.

Let us now consider the k-dimensional parameter space Λ spanned by $\lambda_1, ..., \lambda_k$. As before we would like to divide Λ into cells on which we can express the size of the intersection on a single edge as linear function. To achieve this, we divide Λ into cells on which $r_{\min}$ and $\ell_{\max}$ can be represented by a single linear function.

In the case of $k = 2$, we found that we needed at most two linear functions to express the intersection size in the case that both points belonged to $\mathcal{L}$ (or $\mathcal{R}$). The trivial extension to k points would be to consider a cell for each possible order for the points r_i, which would give us $|\mathcal{L}|!$ possibilities and thus $|\mathcal{L}|!$ cells in the parameter space. Additionally, we would find $|\mathcal{R}|!$ cells for the other side, giving a division of Λ in $|\mathcal{L}|! \cdot |\mathcal{R}|!$, or $O(k!)$, cells.

We can reduce this number by not considering all orders of points, but only divide the parameter space based on which of the points determines the boundary of the intersection. We can then divide Λ into $|\mathcal{L}|$ cells. These cells each correspond to one $x_i \in \mathcal{L}$ determining the right boundary of the intersection. To find these cells, we can calculate this cell for each $x_i \in \mathcal{L}$, which is a convex polytope determined by the intersection of half-spaces $\{r_i \leq r_{i'} : x_{i'} \in \mathcal{L}; i \neq i'\}$.

Since all the cells will be defined by these half-spaces, we can also look at the complete arrangement of hyperplanes $\{r_i = r_{i'} : x_i, x_{i'} \in \mathcal{L}; i \neq i'\}$. Each of the cells in this arrangement will be a subspace in which each of the functions r_i can be expressed by a single linear function.

A total of $\binom{|\mathcal{L}|}{2}$ hyperplanes are needed to divide Λ into cells on which the function $r_{\min}$ is linear. Analogously, $\binom{|\mathcal{R}|}{2}$ hyperplanes are needed for the function $\ell_{\max}$. This gives us a total of $\Theta(k^2)$ hyperplanes.

So for each edge e we need $\Theta(k^2)$ hyperplanes to separate the cells on which the function representing the size of the intersection is linear on e. Therefore, to divide the parameter space Λ into cells in which the total intersection size over all edges is linear, we need $\Theta(|E| \cdot k^2)$ hyperplanes. We can use these hyperplanes to calculate an arrangement of the entire parameter space in $O((|E| \cdot k^2)^k)$ time using the algorithm as described by Edelsbrunner [9].

6.3. Comparing colours

The parameter space Λ is now separated in cells in which the intersection sizes of the k points are linear. Notice that if we look at the intersection of a subset of the k points, the intersection can still be represented as a single linear function for each cell, since the hyperplanes that we would get by comparing a subset of the r_i and ℓ_i functions are still present. This means we can also use this subdivision for the subsets of new Voronoi cells.

On each cell we can now represent the size of all Voronoi cells if the points $x_1, ..., x_k$ were placed according the given parametrisation $\vec{\lambda}$. This means we can calculate all winning points on each cell now in a similar was as before. Again we can express the subspace of winning k-sets as the intersection of half-spaces. Recall that for each colour we have the requirement

$$\sum_{0 \neq J \subseteq X} \left[(-1)^{|J-1|} A(J) \right] \geq \sum_{s_j \in C_q} \left[|\mathrm{Vor}_{G,S}(s_j)| - \sum_{0 \neq J \subseteq X} \left[(-1)^{|J-1|} \bar{A}_j(J) \right] \right],$$

which—since all the functions A, and $\bar{A}_j$ can be expressed in terms of $\vec{\lambda}$ given the elementary segment—gives us a set of half-spaces in the parameter space.

Calculating the intersection of these half-spaces can be done in $O(|\mathcal{C}|^k)$ time, since k is the dimension of the parameter space. We also have to add additional requirements to only retrieve the solutions in the current cell. Each of the hyperplanes bounding the cell originated in comparing two linear functions r_i and $r_{i'}$ (or ℓ_i and $\ell_{i'}$). All of these functions have the same slope, either positively or negatively, so for each pair of parameters $\lambda_i, \lambda_{i'}$ there are only two possible orientations for the hyperplanes, corresponding to r_i and $r_{i'}$ having the same slope or not. The total number of bounding hyperplanes for a single cell is thus $\Theta(k^2)$. Hence, calculating the set of winning k-sets within a single cell requires $O((|\mathcal{C}| + k^2)^k)$ time.

Given the elementary segments on which $x_1, ..., x_k$ are placed, we can calculate the solutions for all cells in $O((|E| \cdot k^2)^k \cdot (|\mathcal{C}| + k^2)^k)$ time. The number of possible combinations of k elementary segments is $O(|V|^k |E|^k)$, hence we can conclude that the overall running time of the algorithm is $O(|V|^k |E|^{2k} k^{2k} (|\mathcal{C}| + k^2)^k)$.

Theorem 6.1. *Let $G = (V, E)$ be a graph, S a set of sites, and $\mathcal{C}$ a set of colours as before. The set W^k of winning k-sets can be calculated in $O(|V|^k |E|^{2k} k^{2k} (|\mathcal{C}| + k^2)^k)$ time.*

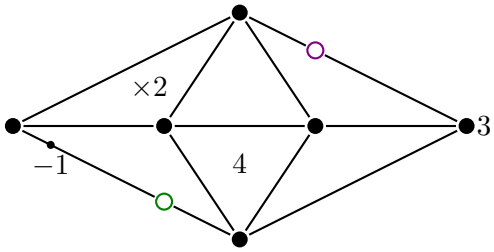


Figure 21: Proof of concept of a graph which has elements that provide additional score. Possibilities include points or vertices on the graph which give additional score or faces which award score or even multiply your score if fully or partly surrounded by a colour.

7. Extensions

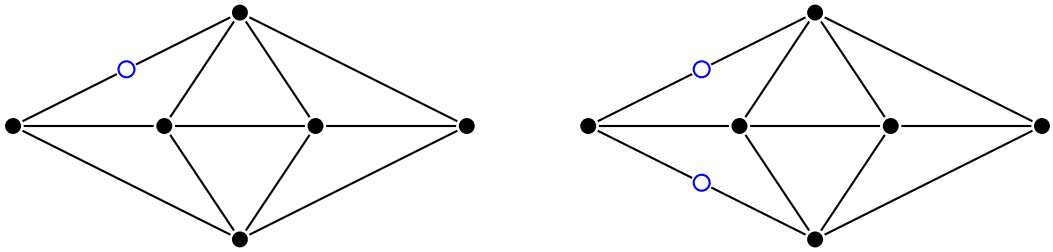
7.1. Different scoring functions

Until now we assumed that the conquered area of a player—or "score"—is equal to the length of the conquered edge segments. However, we can generalise the problem by detaching score from the edge lengths. We could trivially extend our algorithm to also consider vertex weights (i.e. the player gets awarded score for conquering a vertex) and additional points that provide score can also be added to the graph. Figure 21 shows a proof of concept of several score elements.

We can also detach the score function from the edge length. Some edges can be given a higher or lower value per distance unit—or a slower of faster propagation speed depending on the visualisation of the graph. Changing the *density* (the amount of awarded score per distance unit) on edges will be in conflict with some of the assumptions in our algorithm. Even if the density is constant on an edge, the parameter space spanned by $\vec{\lambda}$ will already be more complex since we can get arbitrary slopes for the planes dividing the parameter space. In particular, the cells in the parameter space can be bounded by an arbitrary number of the dividing cells, increasing their maximum complexity. Of course, if the number of different densities is bounded by some parameter or constant, we could parametrise the complexity of the algorithm by the number of different densities.

Introducing the concept of density does allow for interesting puzzle variations. Instead of arbitrarily assigning densities to edges, one can consider the variation for planar graphs where each face is assigned a value and score is awarded based on the portion of the adjacent edges conquered. This concept can be made more interesting by awarding the full face value to the player who conquers the largest portion of the adjacent edges of all players, or each player who conquers at least a certain percentage of it (e.g. 50% or 100%). Since this is an explicit threshold, additional mechanics and/or critical points may have to be introduced to keep the algorithm useful in these situations.

These concepts can be used in combination to find more interesting—and difficult—instances of what is in fact the same game, which can likely be solved using minor alterations to the developed algorithms.



(a) Input (b) Output

Figure 22: The expected in- and output for the instance generation problem.

7.2. Instance generation

The ability to completely generate new puzzles from nothing would be a very useful application of the developed algorithms. Puzzle generation, or instance/level generation in general, comes with its own set of challenges.

The first element of a puzzle instance is the graph on which the puzzle is laid out. To make appealing puzzles, a lot of aesthetic measures have to be kept in mind when building this graph. It is therefore more likely that the graphs are designed manually by a game designer.

The algorithms can still be helpful by taking the graph as designed by the game designer and adding points to it to turn it into an interesting puzzle. Since the problem of placing all existing points is still quite complex, we will simplify the problem further by assuming that positions for all but one of the opponent sites are given and that the puzzle is designed to be solved by placing one point in the player's colour. The problem can be generalised to the placement of multiple points using a similar process as before.

The problem is thus as follows: given the graph $G = (V, E)$ and positions for all but one opponent sites, generate a position for the last opponent site that creates an instance fulfilling predefined requirements where the player's move is placing one point. See Figure 22 for an example of the expected in- and output of this problem.

The requirements for our solution are based on design choices and can be used to define how difficult to solve the puzzle should be. Parameters could for example involve the fraction of the graph that should provide a winning solution for the player, the number of connected components the solution space for the player should consist of, and the (minimum) distance the second point should have from existing features of the graph. We will assume a general objective function is given.

We can consider this problem as a sequence of two problems: first we have to find a position for the opponent point and then we have to solve the finished instance to find the solution for the player. Since the optimal result for the first problem depends on the outcome of the second, this complicates the overall problem.

We can also look at it as a single problem. The order in which we add points to the graph does not influence the result, so we can solve both problems at the same time. Consider the problem of finding a placement of two points—an enemy point and a player point—such that the player wins.

This problem turns out to be very similar to the problem of placing two player points. The largest difference lies in the final evaluation of the solution, since our objective functions are different. However, there is one complication: since the points we are adding have different colours, one way or another we are introducing new critical points. In particular, we have to revisit how we deal with overlapping Voronoi cells if we add the points individually. In our original problem we subtracted the area that we counted doubly (remember the size of this area is given by $A(x_1, x_2)$, where x_1, x_2 are the newly placed points). This was warranted since both Voronoi cells were assigned the same colour and it does not matter to which Voronoi cell the overlapping area was eventually assigned. However, if the cells are assigned to different colours, we have to be more careful about redistributing the overlapping area. Therefore we can introduce a new function $\tilde{A}_i$ that tells us how much a point x_i conquers from the intersection of a set of Voronoi cells:

Definition 7.1.

$$\tilde{A}_i(X) := |\mathrm{Vor}_{G,S\cup X}(x_i) \cap \bigcap_{y\in X} \mathrm{Vor}_{G,S\cup\{y\}}(y)|.$$

Observe that just like the family of functions $\bar{A}_j$, the functions $\tilde{A}_i$ come from a subdivision of the overlapping area and thus

$$A(X) = \sum_{x_i\in X} \tilde{A}_i(X).$$

See also Figures 23 and 24 for an example of how the function $\tilde{A}_i$ relates to the functions A and $\bar{A}_j$. Recall that the functions A and $\bar{A}_j$ in the generalised definition are equal to the functions B and $\bar{B}_j$ from Section 5.

Using this function we can define the size of each newly added Voronoi cell as a composition of the functions A:

$$|\mathrm{Vor}_{G,S\cup X}(x_i)| = \sum_{\emptyset\neq J\subseteq X : x_i\in J} \Big[(-1)^{|J-1|}[A(J) - \tilde{A}_i(J)]\Big]$$

or, in the simpler case of two points x_1, x_2:

$$|\mathrm{Vor}_{G,S\cup X}(x_i)| = A(x_i) - A(x_1, x_2) + \tilde{A}_i(x_1, x_2).$$

Recall that we fixed the elementary segments—maximum edge segments between critical points—on which the points were placed so that we could parametrise their position with scalars λ, μ, and considered their behaviour on a single edge. We will now revisit the interesting cases:

(a) If both Voronoi cells only partly intersect the edge, there are two possibilities as shown in Figure 25. If the conquered segments are at different sides of the edge (Figure 25a), the overlapping area will always be equally divided between the colours. If the conquered segments are at the same side of the edge (Figure 25b),

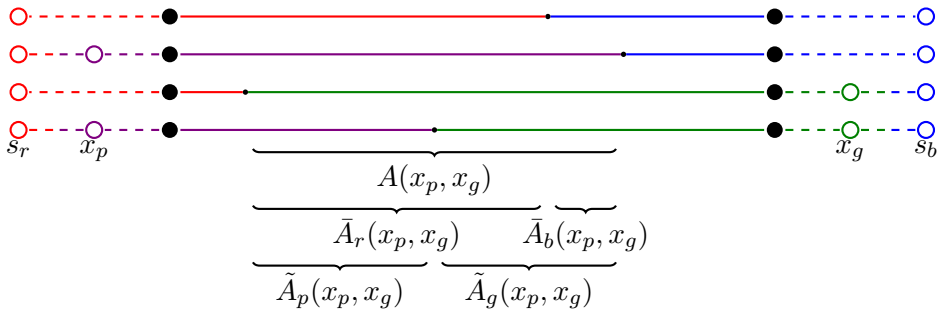


Figure 23: An illustration of the functions A, $\bar{A}_j$, and $\tilde{A}_i$. Lengths of dashed edges are not drawn proportionally.

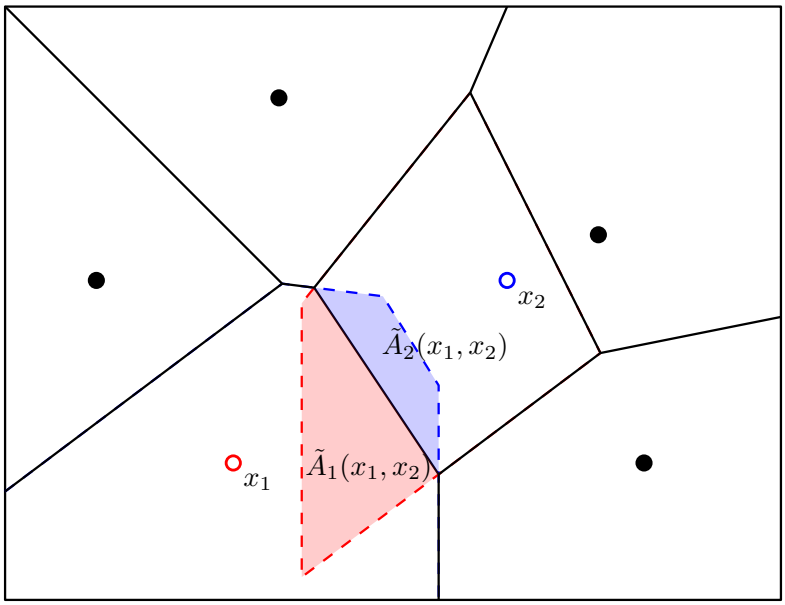


Figure 24: An illustration of the functions $\tilde{A}_i$ in a Euclidean setting.

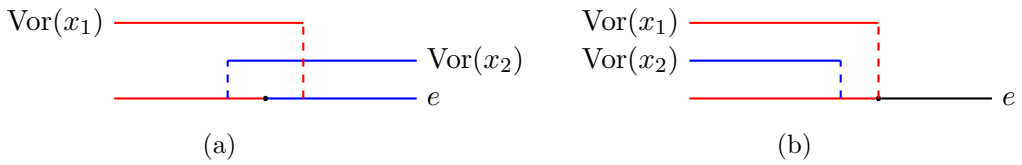


Figure 25: The possibilities of two differently coloured Voronoi cells overlapping, showing how the overlapping area is divided amongst the two colours.

the colour that conquers the largest segment will be given the entire segment. This means that if the boundaries cross while changing λ and μ, the colour conquering the edge segment will change, causing a break in the function $\tilde{A}_i$. This break coincides with the two boundaries coinciding and thus the discontinuity in the function $\tilde{A}_i$ coincides with the break we already identified in the function A.

(b) If one of the Voronoi cells contains the entire edge, there are many different distributions of that edge possible depending on the graph structure. The entire edge may be conquered by the colour containing the entire edge or the edge is only conquered partly. It is possible that for some λ and μ we have the first situation, and for some other λ' and μ' we have the latter. This means that there is a break in the contribution of this edge to the function $\tilde{A}_i$.

The point at which we change between the two situations is the point in which the boundary of the two Voronoi cells in the final Voronoi diagram coincides with an endpoint of the edge (remark that we did not identify this as a critical point, since both involved points are newly added to the graph). If this vertex has degree 2 there will be no break in $\tilde{A}_i$ itself. If the vertex has a larger degree, in addition to the situation changing there will be a larger section of the graph flipping colours.[2] This means that another edge must be in the situation described above (see Figure 25b) where both Voronoi cells conquer an edge segment on the same side of the edge.

We find that $\tilde{A}_i$ is linear under most circumstances. In the situations in which $\tilde{A}_i$ has a break or discontinuity, the function A has a break or discontinuity as well. This means that the structure of $\tilde{A}_i$ is entirely equal to the structure of A and the subdivision of the parameter space we built in Section 5 is still valid even when our points are coloured differently. Therefore, we can use the exact same approach as before to solve this problem.

We can further generalise this problem for the placement of k additional points in different colours. The parameter space is already divided in cells in which only at most two of the newly placed points are relevant for the edge, and within these cells the same cases as above can be distinguished.

[2]There is one exception involving two equally long shortest paths between one of the sites and the vertex, but this only occurs in critical points.

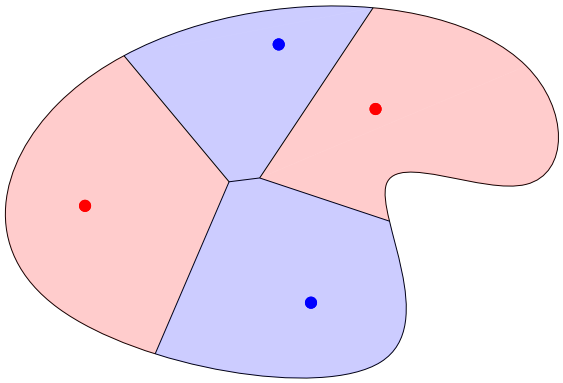


Figure 26: A finished instance of a two-player Voronoi game in a Euclidean domain.

Now that we have established the structure of the entire solution space, we can return to our original problem of finding the placement of an enemy site which makes the puzzle instance as interesting as possible. To find the best solution according to our objectives we can construct all possible solutions and maximise our objective function. If our objective function is simple enough (e.g. linear in A) a form of linear programming can be used instead of calculating the full intersection of half-planes to slightly improve the efficiency of the algorithm.

Alternatively, the algorithm can be designed to return a list of promising candidates or even all possibilities, so that the game designer can make the final decision based on the results of the algorithm. This shows how game design can benefit greatly from the assistance of computers.

We remark that using the more general description of the size of Voronoi cells using $\tilde{A}_i$ is still valid in the simpler case where all sites to be placed have the same colour. The applications of this generalisation of the algorithm are not limited to puzzle generation. Any problem that involves the placement of sites in different colours can utilise the extended techniques from this section. Section 7.4 discusses how these techniques can be used in a more interactive version of the game.

7.3. Euclidean variants

So far, we have only considered the game on graphs, but the concept of *Lines* extends to two- or higher-dimensional Euclidean space as well. Consider a bounded, connected subspace of $\mathbb{R}^2$, which can have holes in it (see Figure 26). We can again add coloured sites to the space and allow the player to add one or more sites. Instead of the sites expanding along the graph, the expansion will be omni-directional, generating the traditional (generalised) Voronoi diagram in two dimensions.

Some concepts from our graph problem have a complement in the Euclidean problem. For example, consider an instance where part of the space is only connected to the rest through a single point. The site closest to that point will conquer the entire subspace behind it if there is no site in that subspace, so placing a site closer than the existing closest site will change the colour of the subspace. This corresponds to a vertex flipping

colours in the graph problem: moving a point closer to the vertex (singularity) than any other site will cause a portion of the graph (subspace) to switch owners. In the graph problem the points for which the vertex switched owners were type II critical points. In the continuous case we will find a *critical curve* (a circular arc in the simplest case).

The Euclidean variant is a very challenging problem and only some heavily simplified cases on two-dimensional Euclidean space have been solved [1, 7, 12]. However, it is possible to create a discrete version of the problem by dividing the Euclidean space into small simplices or general polytopes. The two-dimensional space could for example be divided into triangles. We could then generate a graph by replacing each triangle by a vertex and adding an edge for each pair of triangles that are adjacent with the distance between the centres of mass as length. By reducing the size of the triangles, the continuous solution can be approached, but due to the growth of the underlying graph structure, the algorithms would be rendered infeasible.

While this thesis provides a solid starting point, it appears that extended additional analysis is necessary to find out how the concepts on the graph problem extend to the geometric variant.

7.4. Interactivity and uncertainty

The problem we solved in this thesis assumes that all sites in the puzzle not placed by the player are given in advance. In the general case the Voronoi game does not assume that all this information is available when deciding on a strategy. A large part of the difficulty of finding optimal strategies for a Voronoi game is the fact that the opponent's moves have to be anticipated for.

Another form of incomplete information we might be dealing with in a generalised version of the problem is uncertainty about the existing sites. Uncertainty is different from interactivity since we often have some information about the likely location of the site in the form of a finite set of possibilities or a more general probabilistic distribution. Another key difference is that in interactive settings the opponents can be assumed to play the game according to an optimal strategy in response to the player's choices, where in problems involving uncertainty the strategy of the opponent does not change according to the player's choices.

It is possible to use very similar concepts for both cases by adapting some of the principles we saw in Section 7.2. We can invert the roles and consider our own move the generation of the instance, and the opponent's move or the reveal of the actual site positions the solving of the puzzle. Our goal then becomes to find a site placement—or "generate an instance"—such that there is no winning solution left for the opponent or, in the case of uncertainty, to minimise the probability of the opponent winning.

The solution does not extend to multiple rounds. The winning positions might change depending on the sites the opponent places, because it might not be possible to plan all site placements ahead of time in such a way that the player wins for every possible opponent site placement. As the game progresses, more information becomes available and the strategy might have to be altered.

While this algorithm was not designed for interactive applications, with some small alterations it can be made responsive to relatively simple instances involving interactivity and uncertainty.

(a) Before (b) After

Figure 27: The redistribution of an edge after a snip.

8. Extended Lines game

Placing points is only one of the four operations the player is given in the game *Lines*. In this section we will briefly discuss remarks and observations about the remaining three operations.

8.1. Cutting edges

One of the other operations is the ability to choose a point on an edge in the graph where the edge is cut. To be precise, we allow snips to be made in any point of $\mathcal{E} \setminus V$, since we exclude vertices. A snip means that an edge is separated in two disconnected edge segments through which shortest paths cannot pass, technically creating two new vertices with degree one. This can be used to reduce the size of a Voronoi cell and thus make the difference that allows the player to cover the largest portion of the graph.

In this section we will discuss several observations about this problem and provide a simple procedure to find all *winning snips*.

For this problem we will assume that the original Voronoi diagram is given. Further we will assume that the Voronoi cells are given as proper extended shortest path trees with the site as root. Thus, if a cycle is contained within a Voronoi cell, the cycle will be split at the point that has two equally long shortest paths to the site which we will consider as a boundary between the Voronoi cell and itself.

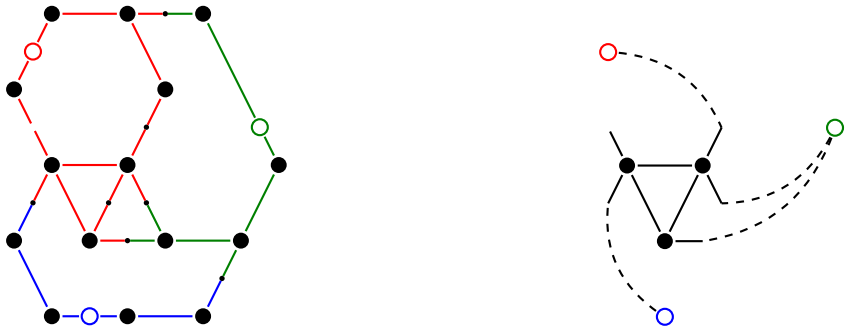
If we now look at all possible locations where we can cut an edge in two we have to consider two types of edges: edges with a boundary on them and edges without a boundary. Remark that since we can never have more than one boundary on an edge, these cases cover all edges in the graph.

It is easy to see that if the edge already contains a boundary, making a snip on that edge will merely move the boundary to that point, as seen in Figure 27. The area of the Voronoi cell that is separated by the snip can only be conquered by the Voronoi cell already adjacent to it.

A very similar situation occurs when making a snip on an edge that does not have a boundary. Again we separate a part of the Voronoi cell, but this time the part we separate is more complex than an edge segment and might even touch multiple other Voronoi cells, making the calculation of the redistribution of this part more complicated.

We extract the subgraph that is the separated part of the Voronoi cell. This subgraph will be a subtree of the extended shortest path tree that represents the Voronoi cell. This does not exclude cycles from the subgraph, since we might have two leaves that correspond to a boundary of the Voronoi cell with itself.

To calculate how this subgraph will be distributed we construct a new graph from it. We also include all sites of the Voronoi cells adjacent to the subgraph as artificial sites



(a) (b)

Figure 28: An example of a graph with a snip and the instance that can be used to calculate the Voronoi diagram on the separated part. Additional boundary markers have been added to show boundaries within one Voronoi cell.

and connect the boundaries with artificial edges that puts them at the proper distance from the boundaries (see Figure 28). Finally we calculate a new Voronoi diagram on this subgraph. We thus require $O(|E| + |V| \log |V|)$ time to calculate the redistribution of the separated area.

The separated area will be exactly the same, independent of where along a given edge e we make the snip, except that the length of the segment of e in the separated area will change. Therefore we only have to calculate the redistribution once per edge to deduce the new Voronoi cell sizes for all snips along every edge.

Doing this for every edge would give a total running time of $O(|E|^2 + |E||V| \log |V|)$ time.

The two operations the player can perform that remain to be discussed are the ability to remove existing sites and the ability to connect two points on the graph with a new edge. We will only touch upon the remaining two operations very briefly.

8.2. Removing sites

To find all combination of existing sites for which the player wins if they would be removed, we can simply try all possible combinations. If the player can remove $k < |S|$ sites, there are $\binom{|S|}{k}$ or $O(|S|^k)$ combinations to be checked. We can recalculate the entire Voronoi diagram in $O(|E| + |V| \log |V|)$ time, hence this algorithm would require $O(|S|^k(|E| + |V| \log |V|))$ time.

8.3. Adding connections

The final operation available to the player is the ability to create a new edge between two points on the graph. The framework that is used so far is not sufficient to deal with this problem. *Lines* uses a graph that is embedded in two-dimensional Euclidean space, so the Euclidean metric can be used to find the length of the newly added edge. Furthermore, *Lines* does not allow the addition of a new edge that intersects with an existing edge. We could expand our framework with a boolean function that returns 1 if a pair of points can be connected by a new edge and 0 otherwise, and a distance function for each pair of points in the graph to determine the length of the newly added edge.

Since the restriction that the new edge can not cross an existing edge and the distance measure are inherent to the Euclidean embedding of the graph, it is likely that the underlying geometric structure of the graph should be used to develop an efficient algorithm. The abstract version of the problem, in which the distance and boolean functions can be arbitrary, cannot use the properties of Euclidean space and is probably very hard to solve due to the potential complexity of the abstract functions.

8.4. Combining operations

The final extension *Lines* provides is the possibility to have instances where the player is provided operations from multiple categories. From what we have seen the problems for each of the operations are solved using completely different methods, so combining their algorithms to solve the problem given any combination of operations will most likely be a challenging process.

9. Discussion

In this thesis we studied the game *Lines* and in particular its Voronoi game aspect. We have shown that if the number of sites k placed by the player is a fixed constant, the space containing all winning points can be determined in polynomial time. We also discussed several generalisations and extensions.

Previous research has shown that the Voronoi game is generally hard to solve, even in some seemingly trivial cases. The NP-hardness proof by Bandyapadhyay et al. [2] can be adjusted to show NP-hardness of the problem discussed in this paper. On the other hand, the algorithm outlined in Section 6 shows that the problem is at most XP-hard. Hence, despite the NP-hardness of the problem, we have shown that solving this problem for small k is still feasible.

We have mainly focussed on developing a framework to solve the problem, without spending much time on optimising the procedures within the algorithm. It is likely that the running time of the algorithm can be improved by using some of the additional structure the problem gives us. For example, *Lines* only uses planar graphs embedded in Euclidean space. The properties of planar graphs and the Euclidean metric could be used to make additional observations on the solution structure and in turn speed up its evaluation.

During the analysis of the problem we also observed that the parameter space Λ has a very specific structure. For example, one can show that all the hyperplanes that divide Λ will intersect each other along a common line parallel to one of the main diagonals of the hypercube. Since these lines may be different for every edge—and are very likely to be different—we can not use this property directly to simplify the parameter space. However, the property does give us an opportunity to exploit the structure of the parameter space to speed up the calculation of the solution, which may be significant when the dimension of the problem grows.

Furthermore, the functions that are compared in each cell are very similar for neighbouring cells. While attempts to use this to speed up the algorithm during its development were not fruitful, it is not unthinkable that these concepts can be used to reduce the amount of time needed to calculate the solution.

Further research is required to explore these potential improvements to the algorithm we developed. It would be interesting to see if these improvements or observations from further analysis can be used to show that the problem is fixed parameter tractable with k as parameter.

We also listed several possible directions for future extensions in Section 7, including the extensions to geometric and interactive variants. Another interesting topic for future research is to investigate if the findings of this thesis can also be applied to previously studied variants of the Voronoi game. In general, since previous research to Voronoi games was limited to finding the optimal solution, considering the entire set of solutions in different variants of the Voronoi game would already be an interesting topic on its own for future research.

Despite the potential possibilities to improve upon the algorithm we developed in this thesis, the introduced techniques are novel. We have opened up a new class of problems by considering the entire set of winning moves by the player, as opposed to calculating only the optimal move. This opens up interesting applications in game and level design, as well as allowing informed decision making.

In conclusion, this thesis has made significant steps towards the understanding of the game *Lines*, and with it, the Voronoi game on graphs.

References

[1] H. Ahn, S. Cheng, O. Cheong, M. Golin, and R. van Oostrum. Competitive facility location: the Voronoi game. *Theoretical Computer Science*, 310(1):457–467, 2004.

[2] S. Bandyapadhyay, A. Banik, S. Das, and H. Sarkar. Voronoi game on graphs. *Theoretical Computer Science*, 562:270–282, 2015.

[3] A. Banik, B.B. Bhattacharya, and S. Das. Optimal strategies for the one-round discrete Voronoi game on a line. *Journal of Combinatorial Optimization*, 26(4):655–669, 2013.

[4] A. Banik, B.B. Bhattacharya, S. Das, and S. Mukherjee. One-round discrete Voronoi game in $\mathbb{R}^2$ in presence of existing facilities. Canadian Conference on Computational Geometry, 2013.

[5] A. Banik, S. Das, A. Maheshwari, and M. Smid. The discrete Voronoi game in a simple polygon. In *Computing and Combinatorics*, pages 197–207. Springer, 2013.

[6] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry*. Springer-Verlag Berlin Heidelberg, 2008.

[7] O. Cheong, S. Har-Peled, N. Linial, and J. Matousek. The one-round Voronoi game. *Discrete & Computational Geometry*, 31(1):125–138, 2004.

[8] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[9] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag New York, 1987.

[10] H.A. Eiselt and G. Laporte. Competitive spatial models. *European Journal of Operational Research*, 39(3):231–242, 1989.

[11] H.A. Eiselt, G. Laporte, and J.F. Thisse. Competitive location models: A framework and bibliography. *Transportation Science*, 27(1):44–54, 1993.

[12] S.P. Fekete and H. Meijer. The one-round Voronoi game replayed. *Computational Geometry*, 30(2):81–94, 2005.

[13] M. Kiyomi, T. Saitoh, and R. Uehara. Voronoi game on a path. *IEICE Transactions on Information and Systems*, 94(6):1185–1189, 2011.

[14] A. Okabe, B. Boots, K. Sugihara, and S.N. Chiu. *Spatial tessellations: concepts and applications of Voronoi diagrams*. Wiley series in probability and statistics: Applied probability and statistics. John Wiley & Sons, 2000.

[15] A. Okabe, T. Satoh, T. Furuta, A. Suzuki, and K. Okano. Generalized network Voronoi diagrams: Concepts, computational methods, and applications. *International Journal of Geographical Information Science*, 22(9):965–994, 2008.

[16] S. Teramoto, E.D. Demaine, and R. Uehara. Voronoi game on graphs and its complexity. In *Computational Intelligence and Games, 2006 IEEE Symposium on*, pages 265–271. IEEE, 2006.

A. Software tool

To gain additional insights in the problem we wrote a software tool.[3] The tool provides the user with a graphical user interface (see Figure 29) in which graphs can be constructed. Coloured sites can be placed on the graph and the Voronoi diagram of these sites is calculated in real-time. The user can modify the graph and the positions of the sites to see how these changes influence the Voronoi diagram.

The tool also includes an implementation of the algorithm that determines all winning points. It uses the coloured points on the graph as opponents and shows the solution for a player currently not represented on the graph. The critical points and solution space can be calculated in real-time, thus making it possible to see how these are influenced by (small) changes in the instance.

The following figures show screenshots of the tool in use.

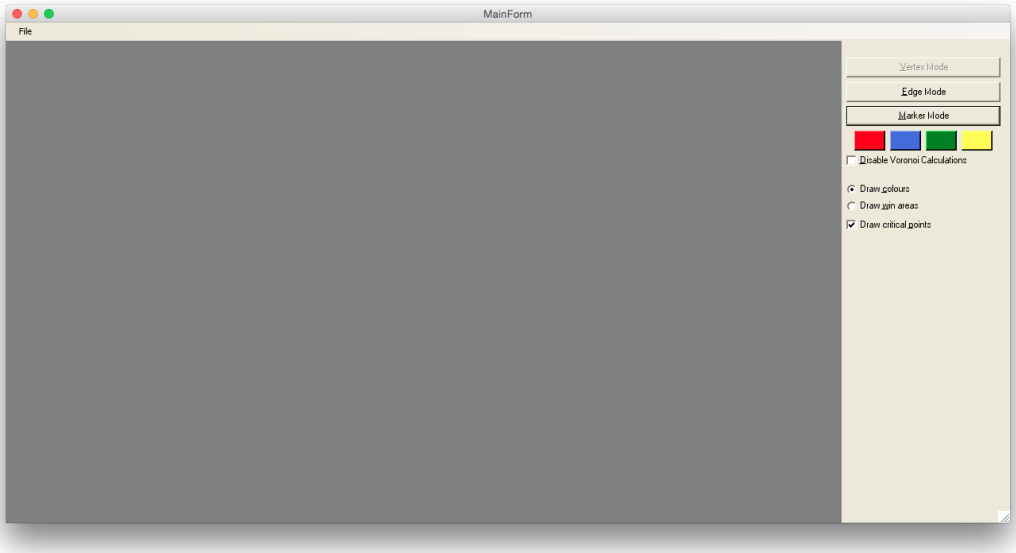


Figure 29: The graphical user interface of the software tool.

[3] https://github.com/tomrijnbeek/graph-voronoi

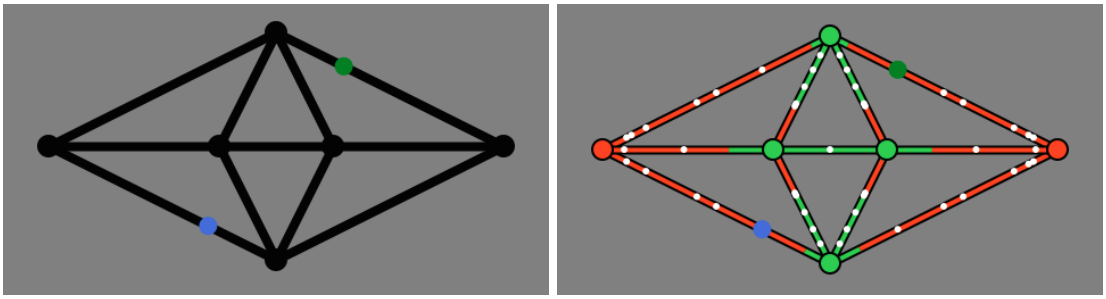


Figure 30: One of the example graphs from the introduction modelled in the software tool and its solution. The black dots represent vertices and the coloured dots represent sites. Critical points are represented by smaller white dots. The green areas represent areas in which a player would win if they place a site there and dots with a black outline and coloured interior are vertices with the colour showing if the player would win if a point was placed in the vertex.

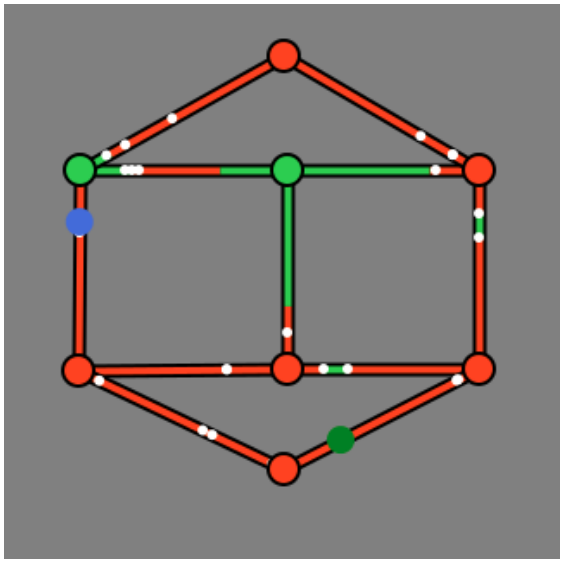
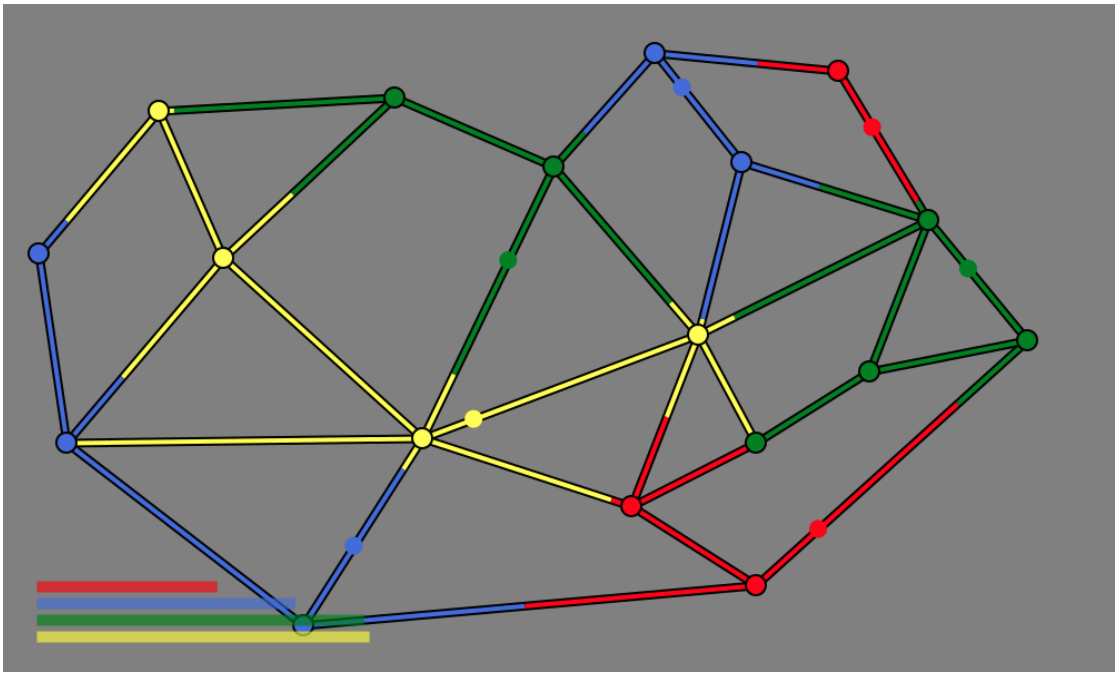
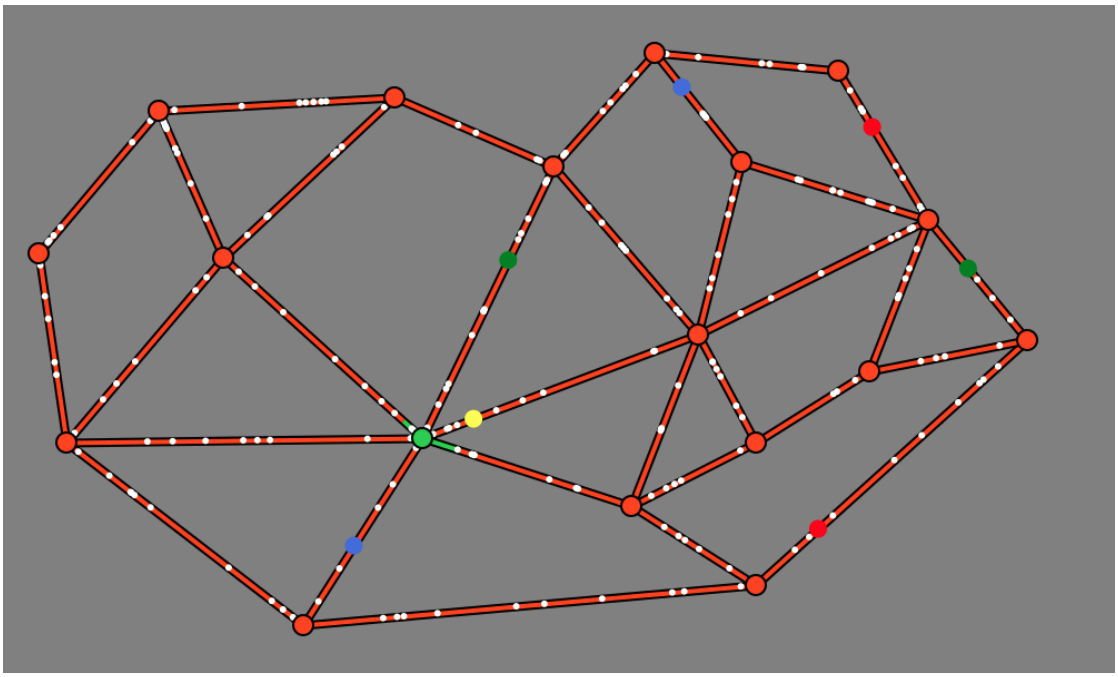


Figure 31: The critical points solution space of a different instance of *Lines*.



(a) The Voronoi diagram of a more complex instance. The coloured bars in the bottom represent the portion each of the colours covers. In this instance, yellow wins.



(b) The critical points and solution space of the instance from the image above.

Figure 32