Extensions on the Continuous Voronoi game on Graphs

Cuts & Ropes

Bas te Kolste
Supervisors: Marc van Kreveld, Maarten Löffler

A thesis presented for the master of
Computing Science

Computing Science
Utrecht University
The Netherlands
December 2016

Contents

1 Abstract

We study a puzzle game called *Lines* developed by Gamious. The game is a variant of the Continuous Voronoi game on graphs. The player is presented a graph containing sites and is tasked to alter the graph either by cutting an edge or adding an edge (Rope). The graph is then subdivided according to the closest sites, and the player who controls the most area on the graph wins.

We present an algorithm to find the best cut for arbitrary graphs in $O(|E| \cdot (|V| + |E|))$ time, where $|E|$ is the number of edges and $|V|$ is the number of vertices. For trees we present two algorithms; one runs in $O(|V||S| \log \Delta(G))$ time and one runs in $O(d \cdot |V| \log |S| \log \Delta(G))$ time, where d is the diameter of the graph, $|S|$ is the number of sites in the graph and $\Delta(G)$ is the maximum degree in the graph.

Finally, we made a start for ropes and laid out future research.

2 Introduction

Games and puzzles have consistently and perpetually interested mathematicians and computer scientists. Games do not only serve as models for certain problems but also serve an entertainment purpose. We will distinguish games from puzzles by saying that a puzzle has one solution, e.g. a Sudoku or a Rubik's cube. A game can have multiple solutions and the goal is to find a winning solution, e.g. chess. We will look at a commercial game called *Lines* developed by *Gamious*. This is a variant of the Voronoi game on graphs. For his thesis, Tom Rijnbeek [23] studied the game *Lines* for placing sites. This report will continue his study and extend it to other interactions.

2.1 Lines - Gamious

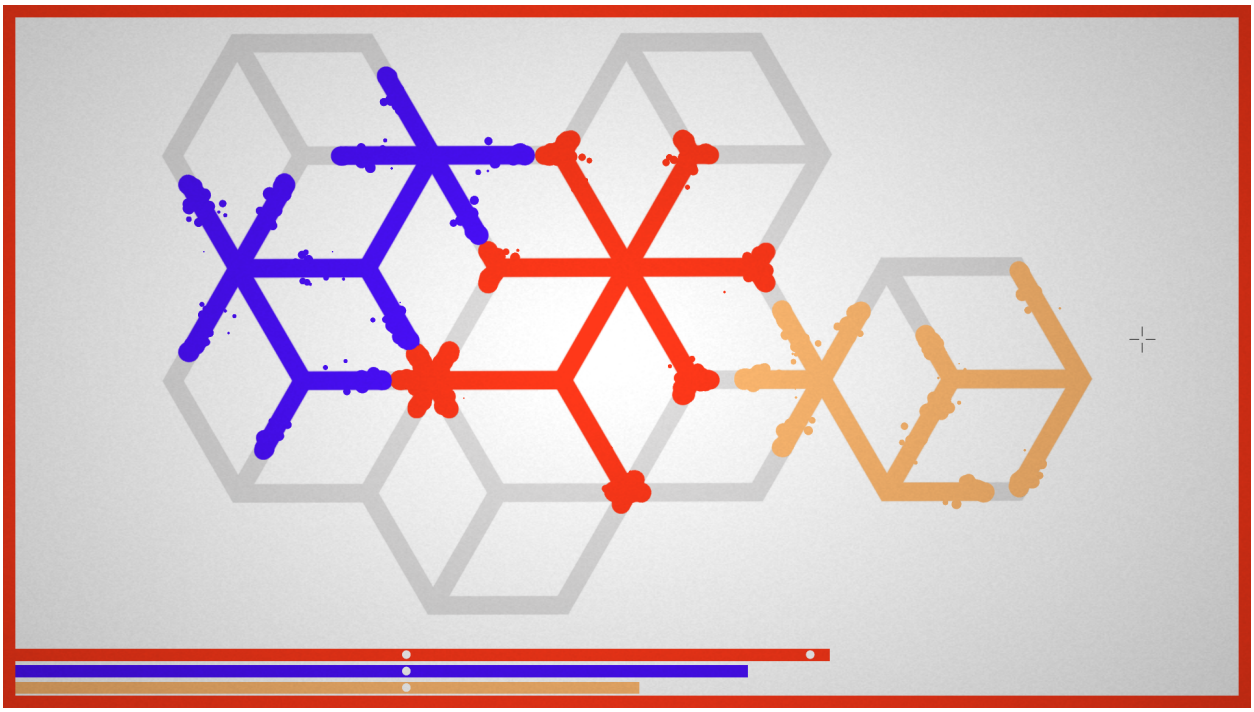

Figure 1: The game *Lines*.

A variant of the one-round Voronoi game on graphs can be played in the commercial game called *Lines* developed by Gamious. The player is tasked to have his facilities own more of the graph than any of his opponents. Each player is represented by a different colour in the graph. The game consists of two phases: the modification phase and the simulation phase. The interaction occurs only in the modification phase, after which the simulation phase happens in a deterministic manner. Upon starting the simulation phase, all sites start expanding at constant speed in all directions along the network, covering the edges in their colour. The sites only expand over the area of the graph that is not yet covered by any colour. The simulation phase ends when no expansion can occur anymore. Figure 1 shows a game state in which the simulation is still ongoing.

In the modification phase, given a graph, there are four operations that the player can do:

i Add a player-coloured site anywhere on the graph

ii Remove any existing site from the graph

iii Make a cut in one of the edges of the graph

iv Add a new line segment connecting two points in the graph

Rijnbeek [23] studied the mathematics behind the Voronoi game on graphs, and only focused on the placement of player-coloured sites anywhere on the graph. During his study he found a $O(|V||E||S| \cdot (|V| \log |V| + |E|))$ time algorithm for computing all winning placements of one site on the graph $G = (V, E)$, where S is the set of existing sites. Then he considered an extension of placing $k > 1$ sites and presented an $O(|V|^k |E|^{2k} k^{2k} \cdot (|C| + k^2)^2)$ time algorithm where $|C|$ is the number of different colours of the sites.

Note that since Gamious presents a puzzle game, it is important to find out whether a given graph G is solvable and whether it is easily solvable. For this reason Rijnbeek has not just looked at an optimal strategy for the player but at the solution space; the space where a site can be placed giving the player a winning configuration. Van Dieren [24] looked at the difficulty of a given game by statistical analysis; doing a random move a number of times to see how many times it is a winning move.

3 Problem definition

I will focus on two moves: making a cut in one of the edges and adding a new line segment connecting two points in the graph. First I will show how the graph and the Voronoi cells are defined.

3.1 The graph

Let $G = (V, E)$ be an undirected graph, where V is a set of vertices and E a set of edges. Each edge $e \in E$ has a weight $w(e) > 0$, usually the length of the edge connecting the corresponding two vertices. We define a point p on e by a parameter $0 \leq \lambda \leq 1$. Given some arbitrary order on V, a point e_λ (with $e = (v_1, v_2)$ such that v_1 is before v_2 in the arbitrary order) is defined as the point on e with distance $\lambda w(e)$ from vertex v_1 and automatically with distance $(1 - \lambda)w(e)$ from v_2. Note that e_0 coincides with v_1 and e_1 coincides with v_2.
The *Voronoi cell* of a site will be defined as in Tom Rijnbeek's paper [23]. To recapitulate we will denote the set of all points on the edges of the graph with $\mathcal{E}$. In other words,

$$\mathcal{E} = \bigcup_{e \in E} \{e_\lambda \mid \lambda \in [0, 1]\}$$

Further let $S = \{s_1, \ldots, s_l\} \subseteq \mathcal{E}$ be a non-empty finite set of disjoint points. The points in S represent the *sites*. We assume that the sites lie on the vertices of G. In other words, $S \subseteq V$. Note, if we have sites that lie on the edges we can trivially construct a new graph where the sites all lie on vertices.

Definition 3.1. *For any pair of points $p, q \in \mathcal{E}$ 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 $Vor_{G,S}(s)$ (or $Vor(s)$ if G and S are clear from the context) on G is defined as*

$$Vor_{G,S}(s) := \{p \in \mathcal{E} \mid \forall s' \in S : d_G(p,s) \leq d_G(p,s')\}.$$

The Voronoi region of a set of sites S' is the union of the Voronoi cells:

$$Vor_{G,S}(S') := \{p \in \mathcal{E} \mid s' \in S' : p \in Vor_{G,S}(s')\}.$$

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

$$\mathcal{V}_G(S) := \{Vor_{G,S}(s) \mid s \in S\}.$$

Definition 3.3. *Given are a set of l sites $S = \{s_1, \ldots, s_l\}$ and a group of k players $P_1, \ldots P_k$. Each site belongs to one player and we call $S_{P_i} \subseteq S$ the set of all sites belonging to player P_i; with $\cup_{1 \leq i \leq k} S_{P_i} = S$. Since each site belongs to only one player we have $S_{P_i} \cap S_{P_j} = \emptyset$ for $i \neq j$. In other words, the sets S_{P_i}, $1 \leq i \leq l$ are a partition of S. For any given site s, we say that $P(s)$ is a player such that $s \in S_{P(s)}$.*

A graph does not have to be connected. Therefore, it is possible to have a component that does not contain any sites. We have the following definition.

Definition 3.4. *Given are a graph G and a set of Sites $S = \{s_1, \ldots, s_l\}$. If there exists a point p for which there exists no path to any site in S, then p is not contained in any Voronoi Cell. We say that p belongs to* ***neutral space****.*

The payoff for a player is the volume of the captured area. In other words,

Definition 3.5. *Given are a set $S_P \subseteq S$ of sites belonging to a player P, the* **payoff** *for player P will be $vol(Vor(S_P))$, where $vol(\cdot)$ is the Lebesgue measure.*

From Definition 3.2 we can see that it is possible for a point to belong to multiple Voronoi Cells. We call such a point a **boundary point**. It is possible to have an infinite number of boundary points, as displayed in Figure 2. To avoid a situation with infinite boundary points, we assume $d(s,v) \neq d(s',v)$ for each vertex $v \in V$ and each pair of sites $s, s' \in S$, as this is sufficient to guarantee a finite number of boundary points on the graph. It is also convenient to assume that a site $s \in S$ coincides with a vertex on G. Therefore, we will assume $S \subseteq V$. If this is not the case, we can trivially construct a new graph $G' = (V', E')$ where $|V'| = |V| + |S|$. Note that in *Lines*, the sites are placed on edges. This means that a graph in which we consider sites to be vertices, these vertices have a degree of 2 and the two edges are at an angle of 180° degrees. In this thesis we do not limit vertices to a certain degree.

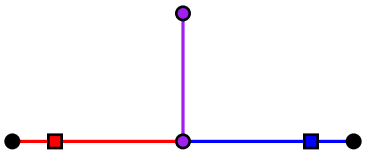

Figure 2: An example of a graph with an infinite number of boundary points. P_1 owns the red part of the graph, P_2 owns the blue part of the graph. The boundary points are colored purple.

Definition 3.6. *Given are a graph $G = (V, E)$, a set of sites $S = \{s_1, \ldots, s_l\}$ and a group of players $P_1, \ldots, P_k$; for any point p on the graph we define $d_G(S_{P_i}, p) = \min_{s \in S_{P_i}} d_G(s,p)$ the minimum shortest path distance from all sites in S_{P_i} to p.*

Definition 3.7. *Given are a graph $G = (V, E)$, a set of sites $S = \{s_1, \ldots, s_l\}$ and a group of players $P_1, \ldots, P_k$, we say a vertex $v \in V$ is **owned** (or **controlled**) **by** a player P_i if $d(S_{P_i}, v) < d(S_{P_j}, v)$ for each other player P_j, $j \neq i$.*

When we are working on trees, it is useful to be able to quickly define certain subtrees. Therefore we have the following definition.

Definition 3.8. *Given are a tree $G = (V, E)$, and a set of sites $S = \{s_1, \ldots, s_l\}$. We define a subtree $\mathcal{R}_{s_i,G}(y)$ as the set of all points p on G, such that the shortest path from s_i to p goes through y.*

$$\mathcal{R}_{s_i,G}(y) = \{p \mid p \in \mathcal{E} \text{ s.t. shortest path from } s_i \text{ to } p \text{ goes through } y\}$$

Definition 3.9. *Given are a tree $G = (V, E)$, and a set of sites $S = \{s_1, \ldots, s_l\}$. We define a subtree $\mathcal{S}_{s_i,G}(y)$ as the set of all points p on G in the Voronoi Cell of s_i such that the shortest path from s_i to p goes through y.*

$$\mathcal{S}_{s_i,G}(y) = \mathcal{R}_{s_i,G} \cap Vor_{G,S}(s_i)$$

3.2 Making a cut - Problem definition

In the game *Lines* by *Gamious* the player is allowed to make a cut on the graph. A Voronoi cell is unable to expand past such a cut. In the following paragraph I will describe how such a cut is made and how it affects the graph.

Given a graph $G = (V, E)$ and a set $S = \{s_1, \ldots, s_l\}$ of sites, a cut can be placed on any point $x \in \mathcal{E}$ on any edge $e = (v, w) \in E$ with $v, w \in V$. The point x can be described as $x = e_\lambda$. A cut can only be made on an edge and not on a vertex, meaning we can only cut at $0 < \lambda < 1$. But for practical purposes, when we desire to cut as close to the vertex as possible, we will also define a cut for either $\lambda = 0$ or $\lambda = 1$. Note that a cut on a vertex should not be possible and that graph G' in Figure 3 does not represent a correct cut.

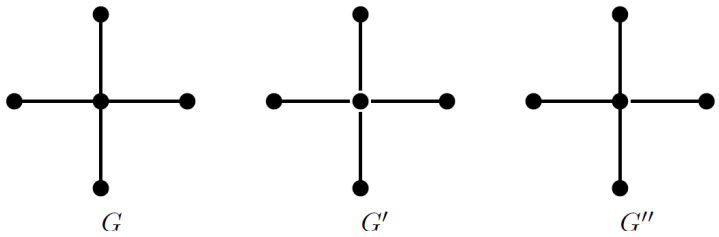

Figure 3: An example of how a cut with parameter $\lambda = 0$ should and should not behave. The resulting graph after the graph transformation should look like G''. It should not look like G'.

Definition 3.10. *Given are a graph $G = (V, E)$ and a set $S = \{s_1, \ldots, s_l\} \subseteq V$ of sites. In a graph G, a cut on an edge $e = (v, w)$ with parameter $0 < \lambda < 1$ is a graph transformation where the resulting graph G' is the same as G, except edge e is gone and there are two new vertices v' and w' and two new edges $\bar{e_1}' = (v, w')$ and $\bar{e_2}' = (v', w)$. The weights (or lengths) of the edges $\bar{e_1}'$ and $\bar{e_2}'$ are set to $\lambda w(e)$ and $(1 - \lambda)w(e)$ respectively. A cut with parameter $\lambda = 0$ creates a graph G', with edge e omitted, one new vertex v' and one new edge $\bar{e}' = (v', w)$ such that $w(\bar{e}') = w(e)$. A cut with parameter $\lambda = 1$ is defined analogously.*

Note that $w(\bar{e_1}') + w(\bar{e_2}') = w(e)$. The Voronoi regions are updated accordingly to the new graph structure.

Definition 3.11. *Given are a parameter λ either 0 or 1, an edge e and a vertex $v = e_\lambda$. A cut on edge e with parameter λ is called to **snap off** edge e from v.*

Given are a graph $G = (V, E)$ and a set of sites $S = \{s_1, \ldots, s_l\}$. Let G' be the graph transformation after cutting on edge e at point $q = e_\lambda$, with $0 \leq \lambda \leq 1$. In the rest of this paper, for any arbitrary point p on G,

we implicitly define p' as the same point on G'. If the point p lies on e with parameter μ then we get three cases. If $\mu < \lambda$, then $p' = \bar{e_1}_{\frac{\mu}{\lambda}}$. If $\mu > \lambda$, then $p' = \bar{e_2}_{\frac{\mu-\lambda}{1-\mu}}$. If $\mu = \lambda$ then we have $p = q$, we get that q' can be either of the new vertices, we therefore do not implicitly define q'.

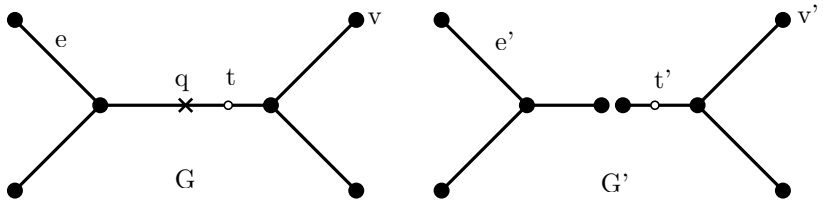

Figure 4: An example of a graph G and its transformed graph G' after cutting on point q. The cut in G' is only graphically separated as the new vertices in G' have the same position.

3.3 Adding a rope - Problem definition

Another move in the game *Lines* is that a player can be allowed to create a new edge on the graph. The edge does not have to be inserted between two vertices; any two points on the graph can be used. Everything until now works on abstract graphs where vertices need not have coordinates and the triangle inequality need not hold. From this point on, due to the nature of the rope operation, we need an embedding, so we assume that all vertices of the graph have coordinates, edges are straight-line connections between two vertices, and the weight of each edge is its Euclidean length. By inserting a line segment the player can drastically change how the Voronoi cells expand. In the following paragraphs I will describe how such a line segment is made and how it affects the graph.

Given are a graph $G = (V, E)$ and a set $S = \{s_1, \ldots, s_l\}$ of sites. Two points $x, y \in \mathcal{E}$ are chosen (restrictions on x and y will be discussed later). Let the edge containing x be e_i and let the edge containing y be e_j; now the points x and y can be described as $e_{i\lambda_x}$ and $e_{j\lambda_y}$ respectively.

Definition 3.12. *Given are a graph $G = (V, E)$ and a set $S = \{s_1, \ldots, s_l\}$ of sites. In a graph G,* **splitting an edge** *$e = (v, w)$ at parameter $0 \leq \lambda \leq 1$ is a graph transformation where the resulting graph G' is the same as G, except edge e is gone and there is one new vertex x' and two new edges $e' = (v, x')$ and $e'' = (x', w)$. The location of x' is that of point e_λ and the weights of e' and e'' are $\lambda_x w(e)$ and $(1 - \lambda_x)w(e)$ respectively.*

Definition 3.13. *Given is a graph $G = (V, E)$. In a graph G, a rope between two points x and y is a graph transformation where the resulting graph G' is obtained after splitting edges on both x and y and where there is a new edge $e' = (x', y')$. The weight of the new edge is the euclidean distance between x and y. The inserted line segment is called a* ***rope****. The Voronoi regions are updated according to the new graph structure.*

3.3.1 Types of ropes

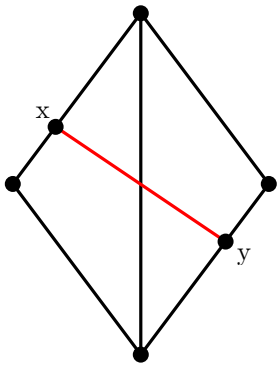

Figure 5: x and y are directly connected; when edges cross they do not connect.

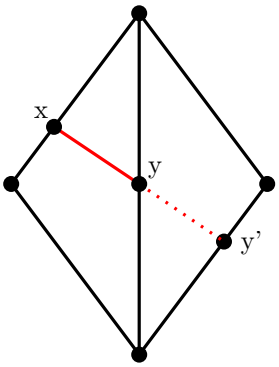

Figure 6: x and y are directly connected; x and y' may not be connected because the new edge would cross an existing edge. When the player tries to connect x with y' in *Lines* the line segment (x, y) will be connected instead.

There are multiple types of restrictions possible for x and y. The first option is to have no restrictions and when edges cross they do not connect as illustrated in Figure 5. A straight line segment is drawn from x to y and the graph is not guaranteed to be a planar graph. When two edges cross (in their $\mathbb{R}^2$-embedding) they might not actually cross in the network. The second option is that x and y may be connected only if the line segment between them in the planar representation does not cross any other line segment as in Figure 6. It is then guaranteed that the graph is always planar. Another option to guarantee graph planarity is to turn all crossing points into vertices as can be seen in Figure 7. So, when drawing a line segment from x to y, all points on (x, y) that also lie on another edge will be transformed into vertices and the two crossing edges are split at this point. It is now as if multiple edges have been drawn in the graph. The last restriction we can look at is that the new line segment must have a certain angle with the edges it is connected to. The angle must be bigger than some threshold α as happens in the game of Gamious.

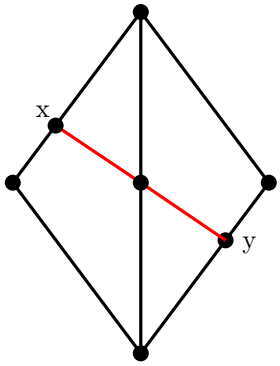

Figure 7: All points on the edges that cross the edge between x and y become a new vertex.

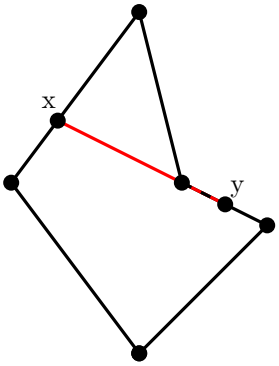

Figure 8: When connecting x with y we might get overlapping edges. To avoid this we impose a threshold α to give a minimum angle to the inserted line segment.

3.4 Win condition

There are two ways in which we describe a move to be the optimal move for a player P. The first way is to only look at the payoff for player P and try to maximize it. The second way is to maximize the rank of player P. The player with the highest payoff has the highest rank. We speak of a winning move when player P has the highest payoff in proportion to the other players.

In this paper, when there are only two players P_1 and P_2, we will always try to optimize the best relative payoff for P_1. In other words, we will try to maximize $vol(S_{P_1}) - vol(S_{P_2})$, where S_P is the set of sites belonging to P.

3.5 Extended Shortest Path Tree representation

Okabe et al. [22] described a method to calculate Voronoi Diagrams on arbitrary graphs. Let $G = (V, E)$ be a graph and let $S = \{s_1, \ldots, s_l\}$ be the set of sites. A new vertex dummy v_0 is added and connected to all sites in S by edges with weight 0. Now we calculate the Shortest Path Tree from the dummy vertex v_0. Note that each child of v_0 is the root of a subtree that represents the Voronoi Cell of that site. Figure 9 shows how the Voronoi Cells are calculated using this method. Figure 10 shows the same tree but then in the Shortest Path Tree representation. We call such a tree an Extended Shortest Path Tree (or *ESPT* for short).

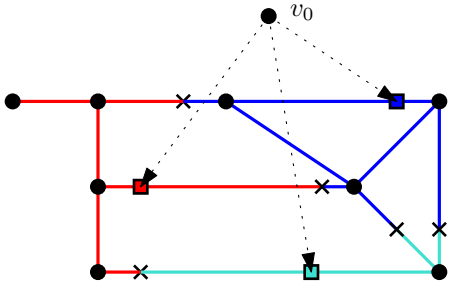

Figure 9: An example of a graph G where the Voronoi Cells are calculated by creating a Shortest Path Tree on v_0.

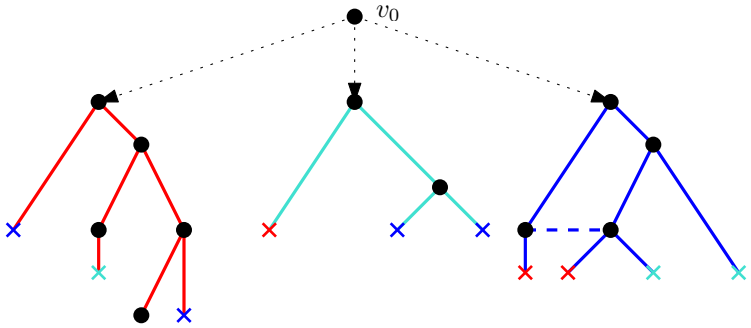

Figure 10: The graph given in Figure 9, but now in the ESPT representation. Note that a cross marks a boundary point, and that the color of the cross corresponds with the color of the opponent with which the boundary point is made. Note that the dashed line is not a tree edge.

The dashed line in Figure 10 represents that there is a connection between the two vertices, but that the edge itself is not in the *ESPT*. We call such an edge a cross edge. A cross edge is never between two vertices owned by different players, since then it would create a boundary point. A cross edge only exists between two subtrees of two different sites owned by the same player, or within one subtree of a site. Note that a

cross edge actually represents a boundary point between a player and himself, we call such a boundary point an inner boundary point.

The *ESPT* has two different kinds of leaf nodes. As illustrated in Figure 10, some leaf nodes are represented by a cross and others by a black disk. We first discuss the leaf nodes illustrated by a cross. A cross represents a boundary point p with the property that there exist two sites s_1 and s_2, such that $d(s_1, v) = d(s_2, v)$. Note that a boundary point p is represented twice in the *ESPT*; once in the subtree of s_1 and once in the subtree of s_2. Let p_1 and p_2 be these points in the *ESPT* respectively. We say that $P(p_1) = P(s_2)$ and $P(p_2) = P(s_1)$. We will now discuss the leaf node represented by a black disk. Such a leaf node is called an endpoint and represents a vertex in the graph with a degree of 1.

4 Making a cut

We will now consider the move of making a cut in one of the edges of the graph. First, we will give a simple solution that works on arbitrary graphs. Later, we will give more efficient solutions for specific problems on trees.

4.1 Simple solution

In this solution, we assume that we want to maximize the payoff of P_1.

Lemma 4.1. *Given a graph $G = (V, E)$. Only a cut on edge $e = (v, w) \in E$ will be considered. Snapping off v from e or snapping off w from e is optimal.*

Proof. We will consider the situations we can encounter when making a cut on the edge $e = (v, w)$; and optimizing the payoff for P_1. Note, that $S \subseteq V$, so the following cases are sufficient.

1. Both v and w are owned by P_1.
2. v is owned by P_1 and w is owned by P_2.
3. Both v and w are owned by P_2.

Case 1 and 3: Both v and w are owned by the same player

Let P_1 own both v and w. We first consider the case where the shortest path from S_{P_1} to v and the shortest path from S_{P_1} to w to not go through e. This implies that e is a cross edge in the *ESPT*. Let G' be the resulting graph after cutting at point e_λ for some $0 \leq \lambda \leq 1$. The vertices v' and w' are still owned by P_1 since the shortest path did not go through e. The values $d(S, v)$ and $d(S, w)$ remain unchanged and therefore we can conclude the cut to have no effect on the payoff for P_1. Therefore the cut e_λ with $\lambda \in \{0, 1\}$ is one of the optimal cuts on the edge. The same reasoning holds for when P_2 owns both v and w.

Now consider the case where the shortest path from S_{P_1} to w goes through e (Note that in this case it is impossible for the shortest path S_{P_1} to v to go through e). Again let G' be the resulting graph after cutting at point e_λ. There are again two cases; either P_1 still owns w' or P_2 now owns w'. Note that the location of the cut on e does not influence which player owns the vertex w'. This means that only the payoff on the edge e is influenced by the location of the cut. When w' is still owned by P_1, then the entire edge is still owned by P_1, thus the payoff remains unchanged for each value of λ. When w' is owned by P_2 then part of the edge e will count towards the payoff for P_1 and part of e counts towards the payoff for P_2. It is now optimal to make the cut as close as possible to w so that P_2 does not get the payoff of e. This implies that if we want to make the cut on e_λ it is optimal to have $\lambda = 1$. We prove it analogously if P_2 owns both v and w. Except that we want to place the cut nearest to v, which implies $\lambda = 0$.

Case 2: v is owned by P_1 and w is owned by P_2.

The shortest path from S_{P_1} to v does not run through e and the shortest path from S_{P_2} to w does not run through e. So after the cut is made the vertices are still owned by the same player. It is therefore optimal to place the cut as close to the vertex owned by P_2 so that the volume of the edge counts towards the payoff of P_1. This means either $\lambda = 0$ or $\lambda = 1$.

For a cut on an arbitrary edge e with parameter λ, we have shown in each case that there exists an optimal cut on e_λ such that $\lambda = 0$ or $\lambda = 1$. This is sufficient to prove the lemma. □

We will now use properties of the *ESPT* as described in Section 3.5 to get to the following lemmas.

Lemma 4.2. *The optimal cut in a graph G always lies on the tree edge of an opponent.*

Proof. Given an edge $e = (v, w)$. As seen in the proof for Lemma 4.1; when the same player owns both vertices v and w a cut on e only affects the payoff of the players when either $d(S, v)$ or $d(S, w)$ goes through e. When this happens we know, by definition, that e is an edge in the Extended Shortest Path Tree. When the shortest tree paths from both v to S and w to S do not go through e we know, by definition, that e is not a tree edge.

When the vertices v and w are owned by a different player then there exists a boundary point p on the edge e. By definition such a boundary point p forms a leaf child node for both the vertices v and w. Both the edges (v, p) and (w, p) form a tree edge. Since it is trivial that a cut must always be made on a point owned by the opponent, the lemma holds. □

Lemma 4.3. *The optimal cut in a graph G always lies on the tree edge of an opponent closest to the root v_0 in the Extended Shortest Path Tree representation.*

Proof. Given a tree edge $e = (v, w)$ with $d(S, v) < d(S, w)$. This means that v is the vertex closer to the root v_0 than w is to v_0. Lemma 4.2 already states that the optimal cut is on the tree edge of an opponent and Lemma 4.1 already states that there exists an optimal cut closest to the vertex. What is left to prove is that to snap off edge e from vertex v is a better (or equal) cut than to snap off e from vertex w. We already know that it does not matter where on the edge the cut is located to influence which player owns vertex v and w. We also know that vertex v will always be owned by the opponent after the cut, as the shortest path $d(S, v)$ does not go through e. If the cut is made at vertex w then we know the entire edge is owned by the opponent player so to cut at vertex v is never worse; therefore the lemma holds. □

We now know that an optimal cut exists on a tree edge of the opponent closest to the root v_0. So for each tree edge owned by P_2, we have exactly one candidate cut. Note that the *ESPT* does not have $O(|V|)$ tree edges as opposed to other shortest path trees, because an edge containing a boundary point is also considered a tree edge. Since there can be $O(|E|)$ boundary points, there are also $O(|E|)$ tree edges in the *ESPT*. This implies that there are $O(|E|)$ candidate cuts that we could test to find the optimal cut. For each such a candidate point we can do a cut and with Breadth-First Search we can find the payoff of each player. Since Breadth-First Search is an $O(|V| + |E|)$ algorithm, the entire algorithm takes $O(|E| \cdot (|V| + |E|))$ time.

4.2 A cut on a tree with two sites

Given are a tree $G = (V, E)$ and two players P_1 and P_2, both controlling one site (s_1 and s_2 respectively) on the tree G. For any two points on G we know that there exists exactly one path between these two points. Since there is exactly one path between s_1 and s_2 we know that there exists exactly one boundary point between P_1 and P_2. Due to the same property we know that there are no inner boundary points.

The candidate cuts for this problem are easy to spot. We try to minimize the Voronoi Cell of P_2, so we will have to cut as close to s_2 as possible. Let E_{s_2} be the set of edges connected to s_2. For each edge

$e \in E_{s_2}$, snapping off e from s_2 is a candidate cut. In the game *Lines*, a site only has a degree of 2 (since they are placed on edges), meaning we have 2 candidate cuts. In the general case we get $|E_{s_2}|$ candidate cuts. For each subtree spanned by s_2 and e for all $e \in E_{s_2}$, we can easily calculate the effect it has on the payoff. Let A be the subtree spanned by s_2 and e, and let G' be the transformed graph after snapping off e from s_2. If A does not contain the site s_1, then A will be neutral space in the transformed graph G'. This implies that the payoff of P_2 is decreased by $vol(A)$. When A does contain the site s_1, then the payoff for P_2 is decreased by $vol(Vor(s_2) \cap A)$. The payoff for P_1 is increased by this amount. These candidate cuts can be calculated with a Breadth-First Search on each subtree. Which adds up to take $O(|V| + |E|)$ time.

4.3 A cut on a tree with multiple opposing sites

Given are a graph $G = (V, E)$ a tree and two players P_1 and P_2. P_1 controls one site s_0 and P_2 controls k sites $S_{P_2} = \{s_1, s_2, \ldots, s_k\}$. Since G is a connected tree there exists exactly one path between any two points. We call p_i the point between s_0 and s_i, such that $d_G(s_0, p) = d_G(s_i, p)$. We say that the first r sites have a direct path to s_0, meaning that there is no other site on the path from s_0 to s_i, with $1 \leq i \leq r$. Let $\mathcal{P}$ be the set of all points p_i, $1 \leq i \leq k$ and let $\mathcal{P}_r = \{p_1, \ldots, p_r\}$ be the set containing all boundary points between P_1 and P_2.

4.3.1 Candidate cuts

We define $\sigma^*(p) = \{p_j \mid p_j \in \mathcal{R}_{s_0,G}(p) \cap \mathcal{P},\ \textit{path from } p_j \textit{ to } p \textit{ does not contain any point in } \mathcal{P}\}$, where p could be any point on the graph. In other words, $\sigma^*(p)$ is the set of all points in $\mathcal{P}$ that can be directly reached from p in the subtree $\mathcal{R}_{s_0,G}(p)$, Figure 11 shows an example. We say $\Delta vol(s_i)$ is the volume of all points solely owned by s_i. We get

$$\Delta vol(s_i) = vol(\mathcal{R}_{s_0,G}(p_i)) - \sum_{x \in \sigma^*(p_i)} vol(\mathcal{R}_{s_0,G}(x))$$

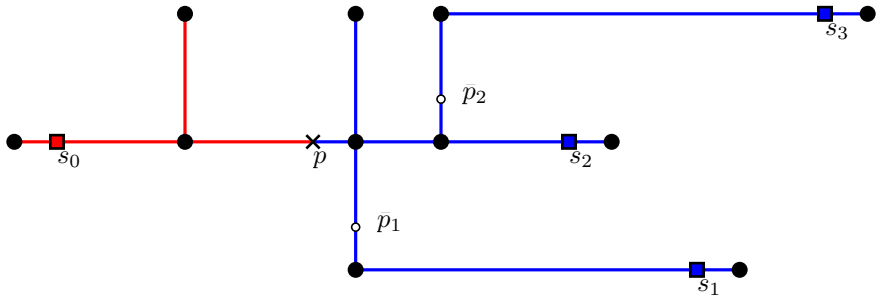

Figure 11: An example of a tree, where the red area is owned by P_1 and the blue area by P_2. We have $\sigma^*(p) = \{\bar{p}_1, \bar{p}_2\}$. This set basically consists of all new boundary points if site s_2 was removed from the graph.

Say we have a point $p_j \in \mathcal{P}_r$. If we were to snap off edge e from the corresponding site s_j such that the shortest path from s_j to p_j goes through e, the payoff for P_1 would increase with $\Delta vol(s_j)$. Since the area was previously owned by P_2 the payoff for P_2 would decrease by that same amount. Therefore the relative payoff increase for P_1 is $2\Delta vol(s_j)$. We call such a cut a **Type I candidate cut**. An example is shown in Figure 12, where the optimal cut is the candidate cut on the left.

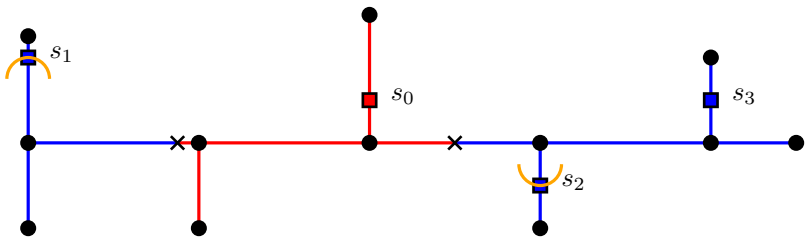

Figure 12: An example of a tree, where the red area is owned by P_1 and the blue area by P_2. The Type I candidate cuts are displayed by orange arcs. Note that there is no candidate cut at site s_3, because it does not have a direct boundary point with the area owned by P_1.

We define T' as a subset of vertices of V, such that for each element $t_i \in T'$ has the following properties.

- The closest site to t_i is a site s_j owned by P_2. Let e be the edge connected to t_i, such that the shortest path from t_i to s_j goes through e.
- The closest site to t_i, of which the path does not go through e, is s_0 (owned by P_1).

Let $T = T' \cup \{p\}$ be the union of T' and the boundary point p. Examples of T are given in Figure 13 and Figure 14; note that the boundary point indicated by a cross is also in T.

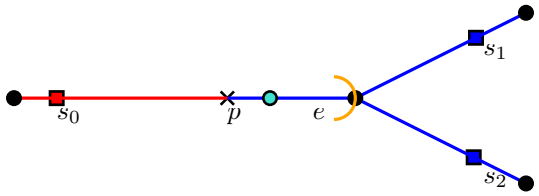

Figure 13: An example of a tree, where the red area is owned by P_1 and the blue area by P_2. The vertices in T are colored in light blue. The Type II candidate cut is given with an orange arc.

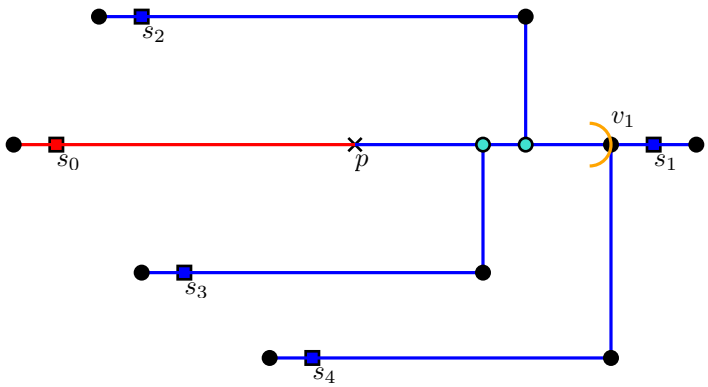

Figure 14: An example of a tree, where the red area is owned by P_1 and the blue area by P_2. The vertices in T are colored light blue. The Type II candidate cut is given with an orange arc. At v_1, both the site s_4 and the site s_1 are closer than the site s_0, therefore $v_1 \notin T$.

Let $\bar{T}$ be the set of points in T, such that for each t_i in $\bar{T}$, there does not exist another $t_j \in T$ such that the shortest path from t_j to the closest boundary point p goes through t_i. For each point $t \in \bar{T}$, we know the closest site s owned by P_2 and we know the edge $e = (v, w)$ such that the shortest path from t to s goes

through e. Note, there is only one such edge e, because we assumed there are no two different vertices $\bar{v}$ and $\bar{w}$ such that $d_G(t, \bar{v}) \neq d_G(t, \bar{w})$, and because there only exists one possible path between any two points on a tree. Let p be the closest boundary point (the one between s and s_0). If we were to snap off e from w, we know that the vertices between p and w are now owned by P_1. We call such a cut a **Type II candidate cut**. Note that a Type II candidate cut could also be a Type I candidate cut, but we make the distinction for practical purposes.

Lemma 4.4. *Given are a graph $G = (V, E)$ and a tree and two players P_1 and P_2. Let P_1 control one site s_0 and let P_2 control k sites $S_{P_2} = \{s_1, s_2, \ldots, s_k\}$. There is only one Type II candidate cut per boundary point p and all vertices between p and the location of the candidate cut are in T.*

Proof. Let v be a vertex in T'. Let s_i be its closest site and let $e = (v, w)$ be the edge connected to v such that the shortest path from v to s_i goes through e. Since v is in T' we have the property that s_0 is the closest site for which the path does not go through e. Let $e' = (v, w')$ be the edge on the path from v to s_0. We know that $d_G(v, s_i) < d_G(v, s_0)$. Let p be the boundary point, which exists along the path from v to s_0.

Let us consider the closest site to the vertex w'. Note that w' is a neighbour of v along the path e'. The path of w' to the closest site s_j goes either through e' or through another edge $e_j \in E_{w'}$. If the path to s_j goes through e', then $s_j = s_i$, because s_i is the closest site of v. If the shortest path does not go through e', then $s_j = s_0$, since s_0 is the closest site of v along the path e'. We therefore have two cases, w' is owned by P_1 or w' is owned by P_2, or more specifically, w' is in the Voronoi Cell of s_i. If w' is owned by P_1, then the boundary point exists somewhere along e'. In this case, we know that there exists a point in T along edge e', namely $p \in T$ itself. If w' is owned by P_2 then it must be in the Voronoi Cell of s_i. This means that its closest site is s_i. The closest site of w' for which the path does not go through e' must be s_0. So w' is also in T. We now know that for each $v \in T'$, there exists a path of vertices in T towards the boundary point p.

We will now prove that for each vertex v in T', there exists at most one vertex x that is in T' connected to an edge $e_x \in E_v \setminus \{e'\}$. Given any edge $e_x = (v, x) \in E_v \setminus \{e', e\}$, let the closest site of x be s_j. If s_j equals s_i, then the corresponding path will go through e_x. The closest site for which the path does not go through e_x can never be s_0, since the path from s_0 to x goes through e_x. Now, let $s_j \neq s_i$. Again, s_0 can never be the closest site to x along path e_x, since s_j will be the closest site along path e_x. So x is not in T. Therefore, there does not exist a vertex x connected to an edge in $E_v \setminus \{e', e\}$, for which x is in T. The vertex w connected to edge $e = (v, w)$ can still be in T. Since the boundary point p is connected to only one site owned by P_2, there only exists one possible vertex connected to p that can be in T. Therefore, there exists exactly one possible path from p to a Type II candidate cut, which means there exists one Type II candidate cut per boundary point p. □

Some edges in the graph are only reached by a site from one direction. Such an edge divides the graph into two subtrees A and B such that the subtree A contains no sites and subtree B contains all the sites. Let e be such an edge owned by P_2 and let v be the vertex connected to e in B. Let κ be the volume of the entire subtree A plus the volume of edge e. If we were to snap off edge e from v, the payoff for P_2 would decrease by κ. The payoff for P_1 would remain unchanged since this part of the graph cannot be reached by other sites. Therefore the relative payoff increase for P_1 is κ. We call such a cut with subtree A and B a **Type III candidate cut**, if there does not exist another such cut with subtree A' and B' such that $A \subset A'$. In Figure 15 all type III candidate cuts are displayed. Note that cutting off the subtree on the right is the optimal solution.

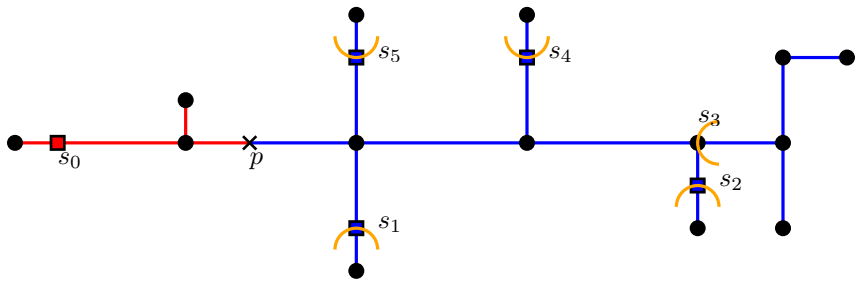

Figure 15: An example of a tree, where the red area is owned by P_1 and the blue area by P_2. The Type III candidate cuts are displayed by orange arcs.

Lemma 4.5. *Given a tree $G = (V, E)$ and two players P_1 and P_2. Player P_1 has one site s_0 and player P_2 has k sites $S_{P_2} = \{s_1, \ldots, s_k\}$. There exists an optimal cut which is either a Type I, Type II or Type III candidate cut.*

Proof. We will prove this lemma by showing that for an arbitrary cut, there exists a Type I, Type II or Type III cut which is at least as good. Consider any arbitrary cut on some edge e at point e_λ, such that e_λ is owned by P_2 before the cut, and such that the cut improve the relative payoff for P_1. Let s_t be the site owned by P_2 that is closest to e_λ. Let v' and w' be the vertices in the transformed graph G' at the location of the cut. Let w' be the vertex for which the shortest path from s_t to w' remains unchanged. This implies that $d_G(s_t, w) = d_{G'}(s_t, w')$. (Note that since we are on a tree, there does not exist a path from v' to s_t.) We have three cases for v':

i v' is owned by $P1$

ii v' is owned by P_2

iii v' is in neutral space

Case i

In this case vertex v' is owned by P_1. Let $E_{v'}$ be the set of edges connected to v'. On the original graph, if E_v contains an edge $\bar{e} = (v, \bar{w})$ such that v would be owned by P_1 after a cut on $\bar{e}$, then it would be a better to snap off $\bar{e}$ from $\bar{w}$. Note, this edge must exist because the original arbitrary cut also got v' to be owned by P_1. We could do this recursively on the edges $E_{\bar{w}} \setminus \{\bar{e}\}$, giving us a Type II candidate cut (and sometimes a Type I candidate cut). In conclusion, for an arbitrary cut such that v' is owned by P_1, there is a Type II candidate cut that is at least as good.

Case ii

In this case the vertex v' is still owned by P_2. The closest vertex to v' cannot be s_t and must be some other site s_j owned by P_2. Note that the relative payoff for P_1 has still increased, so either we gained some neutral space or P_1 must have taken some part of the graph. Since P_2 still owns v' and w', there is no added neutral space, therefore P_1 must have taken part of the graph. This implies that the Voronoi Cell of P_1 has expanded past a boundary point p in G. The shortest path from a site owned by P_2 to p must have been through e_λ. Which implies, s_t is the associated site of the boundary point p. An example of this situation is shown in Figure 16.

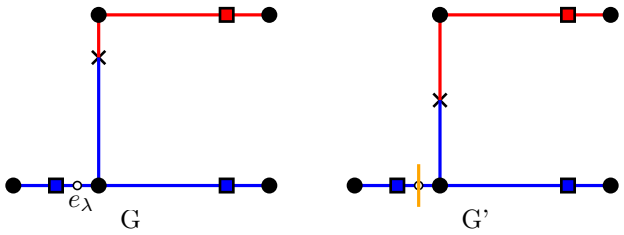

Figure 16: An example of a tree, where the red area is owned by P_1 and the blue area by P_2. A cut is made at e_λ and the Voronoi Cells before and after the cut are displayed.

Since the entire edge e is still owned by P_2, it does not matter if the cut is made elsewhere on the edge. If the edge e was connected to s_t, snapping off edge e from s_t would therefore have the same relative payoff increase for P_1 compared to the original arbitrary cut. Note that this would be a Type I candidate cut with the same relative payoff change for P_1.

We have two cases for what happens with the boundary point p. Either it got replaced by one different boundary point, or it got replaced by multiple different boundary points. An example of this is shown in Figure 17. Let $\bar{p}'$ be the boundary point closest to s_t' in the transformed graph G'. Since all area owned by P_1 is the Voronoi Cell of s_0, a cut at $\bar{p}$ in the original graph G would give the same relative payoff change as the arbitrary cut at e_λ. The Voronoi Cell of s_0 is the same after both cuts. Let $\bar{e} = (\bar{v}, \bar{w})$ be the edge containing $\bar{p}$ and let $d(s_0, \bar{v}) < d(s_0, \bar{w})$ (this means that the shortest path from s_0 to $\bar{p}$ goes through $\bar{v}$). If we were to snap off edge $\bar{e}$ from $\bar{w}$, P_1 would gain every point between p and $\bar{p}$ and every point between $\bar{p}$ and $\bar{w}$. In the original arbitrary cut, P_2 would still own all points between $\bar{p}$ and $\bar{w}$. Therefore, this new cut is better than the original arbitrary cut. Let $E_{\bar{w}}$ be the set of edges connected to $\bar{w}$. If there exists an edge $e_j = (\bar{w}, z)$ in $E_{\bar{w}} \setminus \{\bar{e}\}$ such that P_1 owns $\bar{w}$ after a cut on e_j on the original graph G, then it would always be a better to snap off e_j from z. We can do this recursively on E_z, giving us a Type II candidate cut.

This implies that for an arbitrary cut such that v' is owned by P_2, there is a Type I candidate cut or a Type II candidate cut that is at least as good.

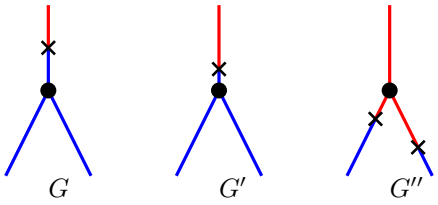

Figure 17: An example of a tree, where the red area is owned by P_1 and the blue area by P_2. Note that not the entire graph is drawn. In this example we can see how a boundary point can be split into multiple boundary points after a cut. Graph G shows the initial situation, the transformed graphs G' and G'' show how the graph might look after a cut.

Case iii

In this case vertex v' is neutral space. This implies that G' consists of two trees A' and B' such that A' contains all the sites and B' does not contain any site. We know that the subtree B in the original graph G was owned by P_2. If $E_{v'}$ contains an edge $\bar{e}' = (v', \bar{w}')$ such that v' would become neutral space after a cut on $\bar{e}$, then it would be better to snap off $\bar{e}$ from $\bar{w}'$ in the original graph G. We could do this recursively on the edges $E_{w'} \setminus \{\bar{e}\}$, giving us a Type III candidate cut. In conclusion, for an arbitrary cut such that v' is neutral space, there is a Type III candidate cut that is at least as good.

□

4.3.2 The algorithm

First we look at calculating Type I candidate cuts. We will need to know the boundary points in the graph and the corresponding sites. This is done by simply calculating the Extended Shortest Path Tree in $O(|V| + |E|)$ time. Let $\bar{S}$ be the set of sites that have a boundary point with s_0. We also need to know all points in $\sigma^*(p)$ where p is a boundary point. This can be done by calculating the Extended Shortest Path Tree after removing the sites in $\bar{S}$; note that Voronoi Cells must still be unable to expand past the sites in $\bar{S}$, therefore if we remove a site $s \in \bar{S}$ we snap off both of its edges. We now need to calculate $\Delta vol(p)$, for each boundary point p. Calculating all values of $vol(\mathcal{S}_{s_0,G}(p))$ for each p adds up to be linear in $|V|$ and $|E|$. Calculating all values of $vol(\mathcal{S}_{s_0,G}(x))$ for all $x \in \sigma^*(p)$ for all p also adds up to be linear in $|V|$ and $|E|$. This gives us the best relative payoff increase for P_1 for all Type I candidate cuts in $O(|V| + |E|)$ time. The algorithm can be seen in Algorithm 2.

We will now look at calculating Type II candidate cuts. First we need the boundary points in the graph, which can be found by calculating the Extended Shortest Path tree. Note that this was already calculated for Type I candidate cuts. Then the set of vertices T is calculated. A breadth-first search from all the sites can be done to calculate the closest site to each vertex; in each vertex v we store the closest site s_v and through which edge e_v the closest site is found. Let E_v be the set of edges connected to v. For each vertex v, we can determine the closest site $\bar{s_v}$ such that the shortest path from s_v to v does not go through e_v, by looking at the neighbours in E_v. If $\bar{s_v}$ is s_0, then the vertex is in T. The set T can be calculated in $O(|V| + |E|)$ time. Now we start at the boundary point p and navigate past all vertices in T, giving us the Type II candidate cut. This also takes $O(|V|+|E|)$ time. The payoff for this candidate cut can be calculated looking at the Extended Shortest Path Tree on the graph G', where G' is the transformed graph after the Type II candidate cut is done.

Type III candidate cuts can be calculated by doing a Depth-First Search in the Shortest Extended Path Tree. Note that the candidate points are points owned by P_2, so we can omit the part owned by P_1. For each vertex, we can easily find if there exists a leaf node that is either a boundary point or an inner boundary point. Let Z be the set of vertices whose subtree does not contain any kind of boundary point. The candidate cuts are at the vertices in Z for which the subtree does not have another vertex in Z. This can also be calculated with a Depth-First Search. Which gives us all Type III candidate cuts in $O(|V| + |E|)$ time. The algorithm can be seen in Algorithm 4.

Note that we treat all sites S to be a subset of V. If we would treat V and S separately, we would get a $O(|V| + |E| + |S|)$ algorithm. Given three sites on an edge e, the middle site would not contribute anything. These sites can easily be removed from the graph before the algorithm, which gives $|S| \leq 2|E|$. The algorithm for the best cut can be seen in Algorithm 1 in linear time. This algorithm calculates the best Type I, Type II and Type III candidate cut. Then returns the candidate cut that is the best. We formulate the following theorem.

Theorem 1. *Given a tree $G = (V, E)$ and two players P_1 and P_2. Player P_1 has one site s_0 and player P_2 has k sites $S_{P_2} = \{s_1, \ldots, s_k\}$. Finding the optimal cut takes $O(|V|)$ time.*

Algorithm 1 Best cut on tree with multiple opposing sites

function BestCut(Tree $T = (V, E)$, Sites $S = (s_0, s_1, \ldots, s_k)$)
  $ESPT \leftarrow$ Extended Shortest Path Tree of T
  $c_1 = (v_1, e_1, w_1) \leftarrow$ TypeICandidateCut$(T, ESPT, S)$
  $c_2 = (v_2, e_2, w_2) \leftarrow$ TypeIICandidateCut$(T, ESPT, S)$
  $c_3 = (v_3, e_3, w_3) \leftarrow$ TypeIIICandidateCut$(T, ESPT, S)$

  $x \leftarrow \text{argmax}\{w_1, w_2, w_3\}$
  Return c_x ▷ c_i is of the form (*Vertex* v, *Edge* e, *Relative Payoff increase* w)
end function

Algorithm 2 Best cut on tree with multiple opposing sites - Type I

function TypeICandidateCut(Tree $T = (V, E)$, Extended Shortest Path Tree $ESPT$, Sites $S = (s_0, s_1, \ldots, s_k)$)
  $\bar{P} \leftarrow$ the boundary points between P_1 and P_2 in $ESPT$
  $\bar{S} \leftarrow$ the corresponding sites of $\bar{P}$

  $T' \leftarrow T = (V \setminus \bar{S}, E)$ ▷ Remove the sites in $\bar{S}$ from the tree by snapping off their edges
  $ESPT' \leftarrow$ Extended Shortest Path Tree of T'
  $\sigma^* \leftarrow$ the boundary points between P_1 and P_2 in $ESPT'$

  for all $\bar{p} \in \bar{P} \cup \sigma^*$ **do**
    Calculate corresponding $vol(\mathcal{S}_{s_0,G}(\bar{p}))$
  end for
  for all $p_i \in \bar{P}$ **do** ▷ p_i is indexed to its corresponding site s_i
    $w_i \leftarrow \Delta vol(p_i) = vol(\mathcal{S}_{s_0,G}(p_i)) - \sum_{x \in \sigma^*(p)} vol(\mathcal{S}_{s_0,G}(x))$
  end for

  $x \leftarrow \text{argmax}\, w_i$
  $e_x \leftarrow$ corresponding edge between p_x and s_x
  Return candidate cut $(s_x, e_x, 2w_x)$.
end function

Algorithm 3 Best cut on tree with multiple opposing sites - Type II

function TypeIICandidateCut(Tree $T = (V, E)$, Extended Shortest Path Tree *ESPT* , Sites $S = (s_0, s_1, \ldots, s_k)$)
  $\bar{P} \leftarrow$ the boundary points between P_1 and P_2 in *ESPT*
  $\bar{S} \leftarrow$ the corresponding sites of $\bar{P}$

  Calculate T using a Breadth-First Search starting from all sites.
  for all $p_i \in \bar{P}$ **do**
    $t_i \leftarrow$ Furthest vertex in T encountered below p ▷ Using Breadth-First Search along vertices in T
    $w_i \leftarrow$ Volume of area between t_i and v_i.
  end for

  $x \leftarrow \operatorname{argmax} w_i$
  $e_x \leftarrow$ corresponding edge between p_x and t_x
  Return candidate cut $(t_x, e_x, 2w_x)$
end function

Algorithm 4 Best cut on tree with multiple opposing sites - Type III

function TypeIIICandidateCut(Tree $T = (V, E)$, Extended Shortest Path Tree *ESPT* , Sites $S = (s_0, s_1, \ldots, s_k)$)
  Do a DFS in *ESPT*, ignoring the area owned by P_1.
  $Z \leftarrow$ set of vertices whose subtree in DFS does not contain any boundary point
  $Z' \leftarrow$ set of vertices in Z directly reached by BFS in *ESPT*

  for all $z_i \in Z'$ **do**
    $w_i \leftarrow$ Volume of area in its subtree
  end for

  $x \leftarrow \operatorname{argmax} w_i$
  $e_x \leftarrow$ corresponding edge between z_i and root in *ESPT*
  Return candidate cut (z_x, e_x, w_x)
end function

4.4 A cut on a tree with multiple sites

Given are tree $G = (V, E)$ and two players P_1 and P_2. There are n sites controlled by P_1 and k sites by P_2. We have multiple boundary points on the graph. The Type I, Type II and Type III candidate cuts are not sufficient to solve this problem. Figure 18 shows a graph with the Type II candidate cuts and Figure 19 shows the resulting graph after a Type II candidate cut. An optimal cut is shown in Figure 20, there the optimal cut is not a Type I, Type II or Type III candidate cut.

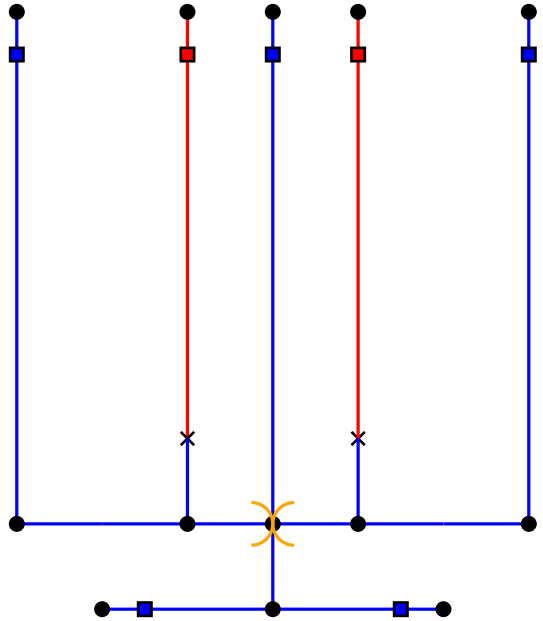

Figure 18: Example of a graph G, where the red area is owned by P_1 and the blue area by P_2. The orange arcs indicate a Type II candidate cut.

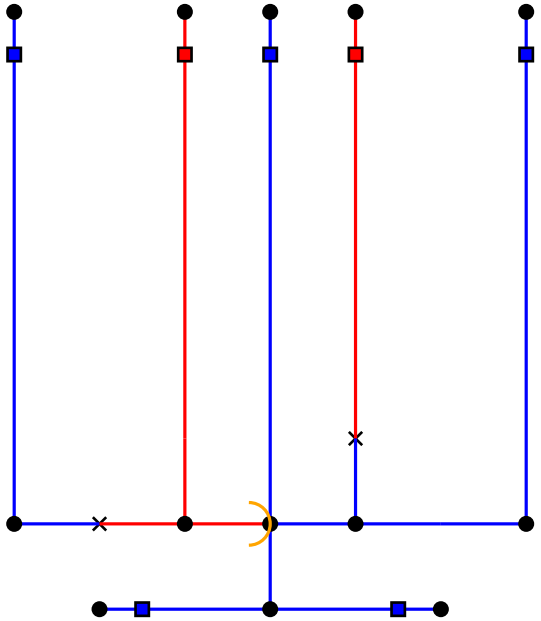

Figure 19: Example of the resulting graph G' after cutting at the orange arc in the graph in Figure 18.

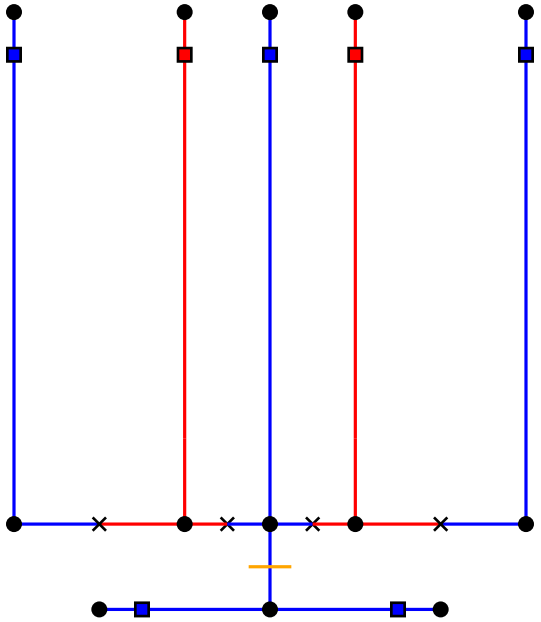

Figure 20: Example of an optimal cut (at the orange line) on the graph G in figure 18. Note, that this is the transformed graph of G after the cut. The optimal cut is not a Type I, Type II or Type III candidate cut. Note that it does not matter where on the edge the cut is made.

An example graph G in Figure 21 indicates that there can be $|E|$ candidate cuts (Note, a site always coincides with a vertex, therefore there can only be one boundary point per edge). For each candidate cut the subtrees can be formed in such a way, that the candidate cut is the optimal cut. Note that the combination of distances

from S_{P_2} to the subtrees is different for each candidate cut shown in Figure 21. For half of the candidate cuts, the distance from S_{P_2} to the right subtree is different. We therefore would like to define some function $f_{v,e}(x)$ for some $v \in V$ and some $e \in E$ connected to v, such that $f_{v,e,P_2}(x)$ indicates the payoff for P_1 on the subtree spanned by v and e, where a Voronoi Cell owned by P_2 reaches v with distance x and it will continue to expand down edge e.

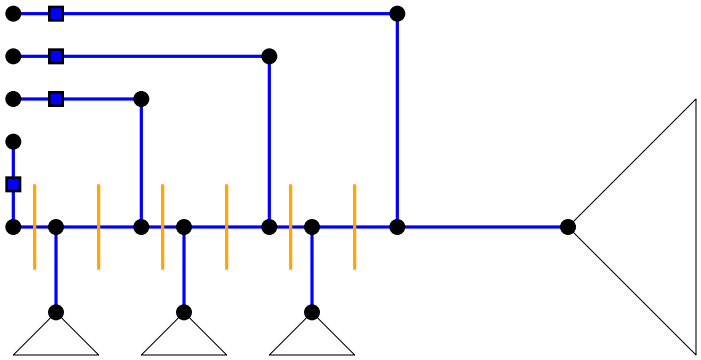

Figure 21: Example of a graph G that shows the possibility for $|E|$ candidate cuts. The triangles indicate a subtree, which may contain several sites by P_1 or P_2.

4.4.1 Payoff change in ESPT

In this subtree we will look at how the payoff for each player changes using the *ESPT*, we will use insights in this subsection to find a function f that calculates the payoff for a player on arbitrary trees. In the *ESPT*, inner boundary points, are displayed as normal boundary points. Given a tree $G = (V, E)$ and a set of sites S, we will look at a subtree of the *ESPT* of G such that the root v_1 of the subtree only has children which are boundary nodes or end nodes. We say v_1 has m children which are named b_1 to b_m; the first m' children are the boundary nodes and the remaining children are the endpoints. In other words, they are ordered in such a way that $b_1, \ldots, b_{m'}$ are boundary nodes and $b_{m'+1}, \ldots, b_m$ are endpoints. The set of boundary nodes of the root is called $B_{v_1} = \{s_i \mid \ s_i \textit{ boundary node and child of } v_1\}$. Let s_0, owned by P_2, be the site that owns v_1 and let s_i be the site for which $d_G(s_0, b_i) = d_G(s_i, b_i)$ for all $1 \leq i \leq m'$. In other words, sites s_i and s_0 share boundary point b_i. Let $S(B_{v_1}) = \{s_i \mid 1 \leq i \leq m'\}$. Let e_i be the edge connected to v and b_i for all $1 \leq i \leq m$. An example of such a subtree is given in Figure 22. We say $P(s_i) = P_1$ if the point s_i is a boundary point with a Voronoi Cell owned by P_1. If s_i is a boundary point with a Voronoi Cell owned by P_2, we say $P(s_i) = P_2$. Note that the subtree is owned by P_2. In this subsection we will only calculate the change in payoff for the edges in the subtree.

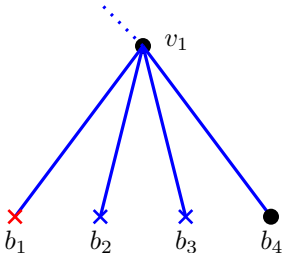

Figure 22: Subtree in the ESPT of which the root only has boundary nodes or endpoints as children. The blue area is owned by P_2 and the crosses indicate the boundary nodes.

If we were to snap off edge $e_i = (v_1, s_i)$ from v_1, where v_1 is the root of the subtree and where $1 \leq i \leq m$, we can easily calculate what will happen to the payoff for each player. If $0 \leq i \leq m'$ and s_i is a boundary

node with $P(s_i) = P_2$ then the payoff of each player will not change, since the ownership of edge e_i remains unchanged. However, if $P(s_i) = P_1$, then e_i will now be owned by P_1. This means that the payoff of P_2 decreases by $w(e_i)$ and the payoff of P_1 increases by $w(e_i)$. Lastly, if $m' + 1 \leq i \leq m$ then s_i is an endpoint and the edge becomes neutral space. Meaning that only the payoff of P_2 is decreased by $w(e)$.

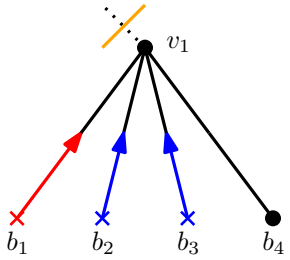

Figure 23: Example of how the Voronoi Cells expand after cutting above the root.

If we cut on the parent edge of the root v_1, then v_1 will be owned by a different site on the graph. As illustrated in Figure 23, the Voronoi cells now start expanding beyond the boundary points. The first Voronoi cell to reach v_1 will continue to expand upwards along the parent edge of v_1 and downward along the edges e_i. We have the following lemma to see which player will own v_1 after the cut. Note that since G is a tree, the cut will never be on the path between any site in $S(B_{v_1})$ and its corresponding boundary point in B_{v_1}.

Lemma 4.6. *Given a subtree consisting of a root v_1 (owned by P_x) only consisting of children that are boundary nodes or endpoints. Let the children be ordered in such a way that $s_1, \ldots, s_{m'}$ are boundary nodes and $s_{m'+1}, \ldots, s_m$ are endpoints. If a cut is made on the parent edge of v_1 then v_1 will be controlled by $P(s_\lambda)$ where $\lambda = \arg\min_{1 \leq i \leq m'} w(e_i)$. This implies that the player that controls the boundary node closest to v_1 will gain control of v_1. The shortest distance $d_{G'}(P(s_\lambda), v_1')$ in its transformed graph G' will be $d_G(P_2, v_1) + 2w(e_i)$.*

Proof. Let G' be the resulting tree after a cut on the parent edge of v_1 in the *ESPT*. The site that will own v_1' must be one of the sites in $S(B_{v_1})$. For $1 \leq i \leq m'$, we know that $d_G(P_x, s_i) = d_G(P_x, v_1) + w(e_i)$, which is the sum of the length of the path from P_x to v_1 and the length of the path from v_1 to s_i. Since s_i is a boundary node we also know that $d_G(P(s_i), s_i) = d_G(P_x, s)$. We also have $d_G(P(s_i), s_i) = d_{G'}(P(s_i'), s_i')$, meaning that the shortest path remains unchanged in G'. After the cut above v_1, which cuts the shortest path from P_x to v_1, the new shortest path to v_1' will be $\min_{1 \leq i \leq m'} d_{G'}(P(s_i'), s_i') + w(e_i')$. After the substitution $d_{G'}(P(s_i'), s_i') = d_G(P(s_i), s_i) = d_G(P_x, v_1) + w(e_i)$ we get that the new shortest path will be $\min_{1 \leq i \leq m'} d_G(P_x, v_1) + 2w(e_i)$. Since $d_G(P_x, v_1)$ is constant for all i the newest shortest path is only dependent on $\min_{1 \leq i \leq m'} w(e_i)$, which implies that the boundary node with the lowest weighted edge will take ownership of v_1. □

We say $\lambda = \arg\min_{1 \leq i \leq m'} w(e_i)$, meaning that $P(s_\lambda)$ will be the new owner of v_1. We will now show how the payoff for each player is affected when cutting in a parent edge of v_1. We know that player $P(s_\lambda)$ will own v_1'. Note that all the boundary points s_i' have now moved along the edge towards v_1' by $w(e_\lambda)$. We have two cases; either $P(s_\lambda) = P_1$ or $P(s_\lambda) = P_2$. We will first look at the case where $P(s_\lambda) = P_2$. Let e_i, $i \neq \lambda$, be an arbitrary edge. If $P(s_i) = P_2$, then both endpoints of e_i are controlled by P_2 before and after the cut. In this case the payoff attributed by edge e_i is the same. The same holds for the edges with an endpoint $(m' + 1 \leq i \leq m)$. They will be owned by P_2 before and after the cut so they do not affect the payoff. The other edges however, where $P(s_i) = P_1$, will be affected. We can then use Lemma 4.7 to see how the payoff changes.

Lemma 4.7. *Given an edge $e = (v, s)$ between some vertex v and a boundary node s. Let there be a cut on G resulting in a transformed graph G' such that $d_{G'}(P(v), v') > d_G(P(v), v)$ and such that $P(v') \neq P(s')$, where $P(v')$ is the owner of v' in the transformed graph G'. The payoff on e for $P(s')$ is increased by $f = \frac{1}{2}(d_{G'}(P(v'), v') - d(P(v), v)) = \frac{1}{2}\Delta d$ and decreased by f for player $P(v')$.*

Proof. We know that $P(v') \neq P(s')$, meaning that the newest shortest path $d_{G'}(P(v'), v)$ does not go through s; this means that $d_G(P(v), v) + w(e) > d_{G'}(P(v), v') > d_G(P(v), v)$. The new shortest path from $P(s')$ to s' stays unaffected, in other words $d_{G'}(P(s'), s') = d(P(v), v) + w(e)$. We assume that the newest shortest distance from a site of $P(v')$ to v' is Δd bigger than the old distance $d_G(P(v), v)$ in G. We have $d_{G'}(P(v'), v') = d(P(v), v) + \Delta d$. We now calculate the distance to the new boundary point p' on e, which is:

$$\begin{aligned} d_{G'}(P(s'), p') &= \frac{d_{G'}(P(v'), v') + d_{G'}(P(s'), s') + w(e)}{2} \\ &= \frac{1}{2}(d_G(P(v), v) + \Delta d + d_G(P(s), s) + w(e)) \\ &= \frac{1}{2}\Delta d + d_G(P(v), v) + w(e) \end{aligned}$$

Note that $d_{G'}(P(v'), p') = d_{G'}(P(s'), p')$. The payoff gained by $P(s)$ along edge e then equals $d_{G'}(P(v'), p') - d_{G'}(P(s'), s')$. We get:

$$\begin{aligned} d_{G'}(P(v'), p') - d_{G'}(P(s'), s') &= \frac{1}{2}\Delta d + d_G(P(v), v) + w(e) - d_G(P(v), v) - w(e) \\ &= \frac{1}{2}\Delta d \end{aligned}$$

This means that the payoff for $P(s')$ increases by $\frac{1}{2}\Delta d$ and the payoff of $P(v')$ decreases by $\frac{1}{2}\Delta d$. □

Due to Lemma 4.6 we know that $\Delta d = 2w(e_\lambda)$. So overall, the payoff for player P_1 will be increased by the function $\bar{F}_{v_1:=P_2}$, where $v_1 := P_2$ means that v_1 will be owned by P_2 after cutting above v_1. The function $\bar{F}_{v_1:=P_2}$ is defined as:

$$\bar{F}_{v_1:=P_2} = \sum_{0 \leq i \leq m} \bar{f}_{v_1:=P_2}(s_i)$$

where

$$\bar{f}_{v_1:=P_2}(s_i) = \begin{cases} w(e_\lambda), & \text{if } P(s_i) = P_1 \\ 0, & \text{otherwise.} \end{cases}$$

Note that in this case if the payoff of player P_1 increases by $\bar{F}$ that the payoff of player P_2 gets decreased by $\bar{F}$.

We will now look at the case where $P(s_\lambda) = P_1$. Once v_1 is owned by P_1 we get that all edges e_i with endpoints are now owned by P_1. This implies that the payoff for P_1 is increased by $w(e_i)$ and the payoff for P_2 is decreased by $w(e_i)$. The edges e_i, of which $P(s_i) = P_1$ will now be entirely owned by P_1. Meaning that the payoff for P_1 is increased by $w(e_i)$ and the payoff for P_2 is decreased by $w(e_i)$. If the edge e_i has a boundary point s_i for which $P(s_i) = P_2$, the payoff for P_1 is increased by $w(e_\lambda)$ and for P_2 decreased by $w(e_\lambda)$. Overall the payoff for player P_1 will be increased by:

$$\bar{F}_{v_1:=P_1} = \sum_{0 \leq i \leq m} \bar{f}_{v_1:=P_1}(s_i)$$

where

$$\bar{f}_{v_1:=P_1}(s_i) = \begin{cases} w(e_i), & \text{if } s_i \text{ is an endpoint.} \\ w(e_i), & \text{if } P(s_i) = P_1. \\ w(e_\lambda), & \text{if } P(s_i) = P_2. \end{cases}$$

4.4.2 Payoff change on entire tree

Given a graph G and two players P_1 and P_2. Let P_1 have n sites and P_2 have k sites. We define a function $f_{v,e,P_i}(x)$ such that $v \in V$ and $e \in E$ connected to v. The function $f_{v,e,P_i}(x)$ gives the payoff for P_1 on the subtree spanned by v and e when v is owned by P_i and the shortest path from $S(P_i)$ to v does not go through e, with $d_G(S(P_i), v) = x$. In other words, a Voronoi Cell owned by P_i reaches v with distance x and continues to expand along e, we say the Voronoi Cell **pushes** from v onto e with a **force** of x. The payoff for P_2 will then be $vol(\mathcal{T}_{v,e}) - f_{v,e,P_i}$, where $\mathcal{T}_{v,e}$ is the subtree spanned by v and e. We say $g_{v,e,P_i}(x) = vol(\mathcal{T}_{v,e}) - f_{v,e,P_i}(x)$.

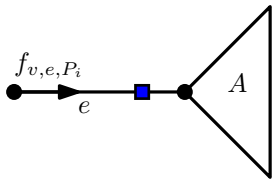

Figure 24: An example of a graph G. An arrow is drawn indicating the function f to push onto the edge.

Let v be a vertex and let e be an edge connected to v and a site s, such as in the graph in Figure 24. The function f_{v,e,P_i} can easily be calculated. Note that the Voronoi Cell can never expand past a site, so the subtree A behind the site s (as shown in Figure 24) does not influence the value for f. If $P_i = P(s)$, then edge e is entirely owned by $P(s)$. Therefore, we have $f_{v,e,P_i}(x) = w(e)$ for each value of x. If $P_i \neq P(s)$, then part of the edge can be owned by P_i. We can use Lemma 4.7 to find $f_{v,e,P_i}(x) = \frac{1}{2}\Delta d_x = \frac{1}{2}(w(e) - x)$. Note that if $x \geq w(e)$, the Voronoi Cell pushes onto e with a force x such that the Voronoi Cell of s already reached v. Since this cannot happen we get $f_{v,e,P_i}(x) = \max\{\frac{1}{2}(w(e) - x), 0\}$. Finally, for an edge e connected to a site s we get:

$$\begin{aligned} f_{v,e,P_1}(x) &= \begin{cases} w(e), & \text{if } P(s) = P_1 \\ \max\{\frac{1}{2}(w(e) - x), 0\}, & \text{if } P(s) = P_2 \end{cases} \\ g_{v,e,P_2}(x) &= \begin{cases} w(e), & \text{if } P(s) = P_2 \\ \max\{\frac{1}{2}(w(e) - x), 0\}, & \text{if } P(s) = P_2 \end{cases} \end{aligned} \tag{1}$$

The property $g_{v,e,P_i} = vol(\mathcal{T}_{v,e}) - f_{v,e,P_i}(x)$ implies:

$$\begin{aligned} f_{v,e,P_2}(x) &= \begin{cases} 0, & \text{if } P(s) = P_1 \\ \min\{\frac{1}{2}(w(e) + x), w(e)\}, & \text{if } P(s) = P_2 \end{cases} \\ g_{v,e,P_1}(x) &= \begin{cases} 0, & \text{if } P(s) = P_2 \\ \min\{\frac{1}{2}(w(e) + x), w(e)\}, & \text{if } P(s) = P_2 \end{cases} \end{aligned} \tag{1}$$

Now consider the case that the edge e is connected to vertices v and w, for which the degree of w is 1. In other words, w is an endpoint in the graph. It is easy to see that the Voronoi Cell will completely expand upon edge e. We therefore get:

$$f_{v,e,P_i}(x) = \begin{cases} w(e), & \text{if } P_i = P_1 \\ 0, & \text{if } P_i = P_2 \end{cases} \tag{2}$$

And transversely:

$$g_{v,e,P_i}(x) = \begin{cases} 0, & \text{if } P_i = P_1 \\ w(e), & \text{if } P_i = P_2 \end{cases} \tag{2}$$

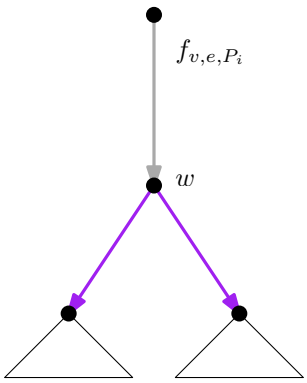

Figure 25: An example of a graph G. A gray arrow is drawn indicating the function f to push onto the edge. The purple edges indicate the functions that are already known.

Now consider the case that the edge e is connected to some arbitrary vertex w. Let E_w be all edges connected to w. Say for all edges $\bar{e} \in E_w \setminus \{e\}$, we know the function $f_{w,\bar{e},P}$ for both players $P \in \{P_1, P_2\}$. An example can be seen in Figure 25. The Voronoi Cell at v pushes down e with a force of x, so this same Cell pushes down the edge $\bar{e}$ with a force of $x + w(e)$ for all $\bar{e} \in E_w \setminus \{e\}$. Let μ be the shortest distance from vertex w to any site in the subtree spanned by v and e, note that μ is the shortest distance to any site in graph G, for which the path does not go through e. If we have $x + w(e) < \mu$, then the Voronoi Cell at v is the first to reach vertex w. We obviously get $f_{v,e,P_i}(x) = w(e) + \sum_{\bar{e} \in E_v \setminus \{\bar{e}\}} f_{w,\bar{e},P_i}(x + w(e))$. But as soon as $x + w(e) \geq \mu$, it means that a different Voronoi Cell has already reached vertex w. If the other Voronoi Cell that reached v first is also owned by P_i, then the area of player P_i is still expanding down the other edges, but the force does not depend on x anymore. The edge e will also be entirely owned by P_i. If the new Voronoi Cell is owned by the opponent P_j however, then vertex w is not owned by P_i anymore. Let $\bar{x}$ be the value of x such that $x + w(e) = \mu$ and let $\bar{e} \in E_w$ be the edge containing the shortest path from w to the closest site in the subtree. Now we assume that the vertex w is owned by the opposing player. We get $f_{v,e,P_i}(\bar{x}) = w(e) + \sum_{\bar{e} \in E_w \setminus \{e,\bar{e'}\}} f_{v,e,P_j}(\bar{x} + w(e))$. Note, that in the summation the function f is parametrized with P_j. Now that w is owned by P_j, we can consider this just as a site, so like before we get $f_{v,e,P_i}(\bar{x} + x)$ to equal the function in (1) plus the constant $\sum_{\bar{e} \in E_w \setminus \{e,\bar{e'}\}} f_{v,e,P_j}(\bar{x} + w(e))$. We get:

$$f_{v,e,P_1}(x) = \begin{cases} w(e) + \sum_{\bar{e} \in E_v \setminus \{\bar{e}\}} f_{w,\bar{e},P_1}(x + w(e)), & \text{if } x < \bar{x} \\ (w(e) - \frac{1}{2}x) + \sum_{\bar{e} \in E_w \setminus \{e,\bar{e'}\}} f_{v,e,P_2}(\bar{x} + w(e)), & \text{if } x \geq \bar{x} \text{ and } P_2 \text{ owns } w \\ w(e) + \sum_{\bar{e} \in E_w \setminus \{e,\bar{e'}\}} f_{v,e,P_1}(\bar{x} + w(e)), & \text{if } x \geq \bar{x} \text{ and } P_1 \text{ owns } w \end{cases} \tag{3}$$

$$g_{v,e,P_2}(x) = \begin{cases} w(e) + \sum_{\bar{e} \in E_v \setminus \{\bar{e}\}} f_{w,\bar{e},P_2}(x + w(e)), & \text{if } x < \bar{x} \\ (w(e) - \frac{1}{2}x) + \sum_{\bar{e} \in E_w \setminus \{e,\bar{e'}\}} f_{v,e,P_1}(\bar{x} + w(e)), & \text{if } x \geq \bar{x} \text{ and } P_1 \text{ owns } w \\ w(e) + \sum_{\bar{e} \in E_w \setminus \{e,\bar{e'}\}} f_{v,e,P_2}(\bar{x} + w(e)), & \text{if } x \geq \bar{x} \text{ and } P_2 \text{ owns } w \end{cases} \tag{3}$$

The values for $f_{v,e,P_2}(x)$ and for g_{v,e,P_1} can easily be found due to the property $g_{v,e,P_i}(x) = vol(\mathcal{T}_{v,e}) - f_{v,e,P_i}(x)$. Note that if the subtree does not have any sites, the value for $\bar{x}$ is not defined. In this case we have $\bar{x} = \infty$.

Lemma 4.8. *The function f_{v,e,P_i} is a non-continuous piecewise linear function.*

Proof. As we can see in (3), the function is clearly not continuous in point $\bar{x}$. We also know that the sum of piecewise linear functions is a piecewise linear function. Since the functions (1) and (2) are piecewise linear, we know that the function (3) is also piecewise linear. □

4.4.3 Data structure

A piecewise linear function is composed of linear segments. Each segment of f can therefore be defined by an interval $[a, b)$, a slope $\frac{\Delta y}{\Delta x}$ and a value $f(a)$. Note that if the segment is $[-\infty, b)$, the value $f(a)$ is ill-defined. But since a negative input for f_{v,e,P_i} does not mean anything, we only need to store segments that have non-negative boundaries. These segments are then stored in order of a, i.e. the breakpoints are stored as a sorted array $\{a_1, a_2, \ldots, a_k\}$. Note that each segment can be described as $[a_i, a_{i+1})$, $1 \leq i \leq k$, with $a_{k+1} = \infty$. Each segment, indicated by its breakpoint a_i, has a corresponding slope m_i and its value $f(a_i) = v_i$. The value of $f(x)$, with $x \geq 0$, can then be calculated by first finding the segment $[a_j, a_{j+1})$ which contains x, which can be done with a binary search over the values a_i. Then we get $f(x) = v_i + m_j \cdot (x - a_j)$. So calculating a value of f takes $O(\log n)$ time.

Let f_{w,e,P_i} be known functions, stored as a data structure, for all $e \in E_w$. If we want to calculate $\sum_{e \in E} f_{w,e,P_i}$, we will need to merge all the segments in f_{w,e,P_i}. A k-way merge sort is done on all breakpoints, which takes $O(n \log |E_w| + |E_w|)$ time, where n is the total number of breakpoints. For each new segment a_i the corresponding slope m_i is calculated by taking the sum of all slopes of all overlapping segments, of which there are at most $|E|$ many. The value v_i is calculated by taking the sum of all values $f(a_i)$ of all overlapping segments, of which there are $|E|$ many. Calculating $\sum_{e \in E} f_{w,e,P_i}$ therefore takes $O(n + |E|)$, where n is the number of breakpoints, which is bounded by the number of leaves in the subtree spanned by w and e. Note that in the function (3), the sum is taken over the function f_{w,e,P_i} with parameter $x + w(e)$. Since $x \geq 0$, the definition of the function $f_{w,e,P_i}(x')$ with $x' < 0$ does not need to be known. A segment $[a_i, a_{i+1})$ such that $a_{i+1} < w(e)$ is therefore ignored. A segment $[a_i, a_{i+1})$ with $a_i < w(e)$ and $a_{i+1} > w(e)$, is trivially converted to a segment $[w_e, a_{i+1})$ with value $v_i = f(w(e))$.
If we have some function f_{v,e,P_i}, the value of f_{v,e,P_i} can be calculated in $O(\log n)$ time, where n is the number of segments of the function, i.e. the number of sites in the subtree.

4.4.4 Graph simplification

In a tree $G = (V, E)$, the Type III candidate cuts can still be optimal cuts. If a Type III candidate cut is made next to a site, the graph transformation contains a subtree T' which is neutral space. A different cut on T would then generate a smaller subtree, hence the optimal cut is never on any part of T. This implies that if a Type III candidate cut next to a site of P_i is not optimal, the subtree T' will always be owned by P_i afterwards, i.e. it stays unaffected. So the corresponding subtree T can be removed from G, without changing the location of the optimal cut.

The algorithm first searches for the best Type III candidate cut and the corresponding subtrees connected to a site can be removed from the graph. At this point, there is no need to check for candidate cuts that generate neutral space.

4.4.5 The algorithm

As discussed before, calculating a function f_{v,e,P_i} takes $O(n \log(\Delta(G) + \Delta(G))$ where $\Delta(G)$ is the maximum degree in G and where n is the number of breakpoints, i.e. the number of sites in its subtree. But, as indicated in function (3), we need to know which player will own w. We therefore need to know the closest site s to each vertex v and its corresponding edge e which is contained in the shortest path from s to v. The closest site of each vertex is easily calculated in the *ESPT*, which takes $O(|V| + |E|)$ time. The closest site to v for which the shortest path does not go through e also needs to be known. Which can also be found in $O(|V| + |E|)$ time. Lastly, a function f_{v,e,P_i} gives the payoff for P_1, and transversely the payoff for P_2. But what we need is the difference in payoff for each player. We therefore have to calculate the total volume of each subtree and the payoff for each player for each subtree spanned by v and e. This can be calculated in $O(|V| + |E|)$ time.

We will discuss two methods, which both give a different running time. The first method will calculate

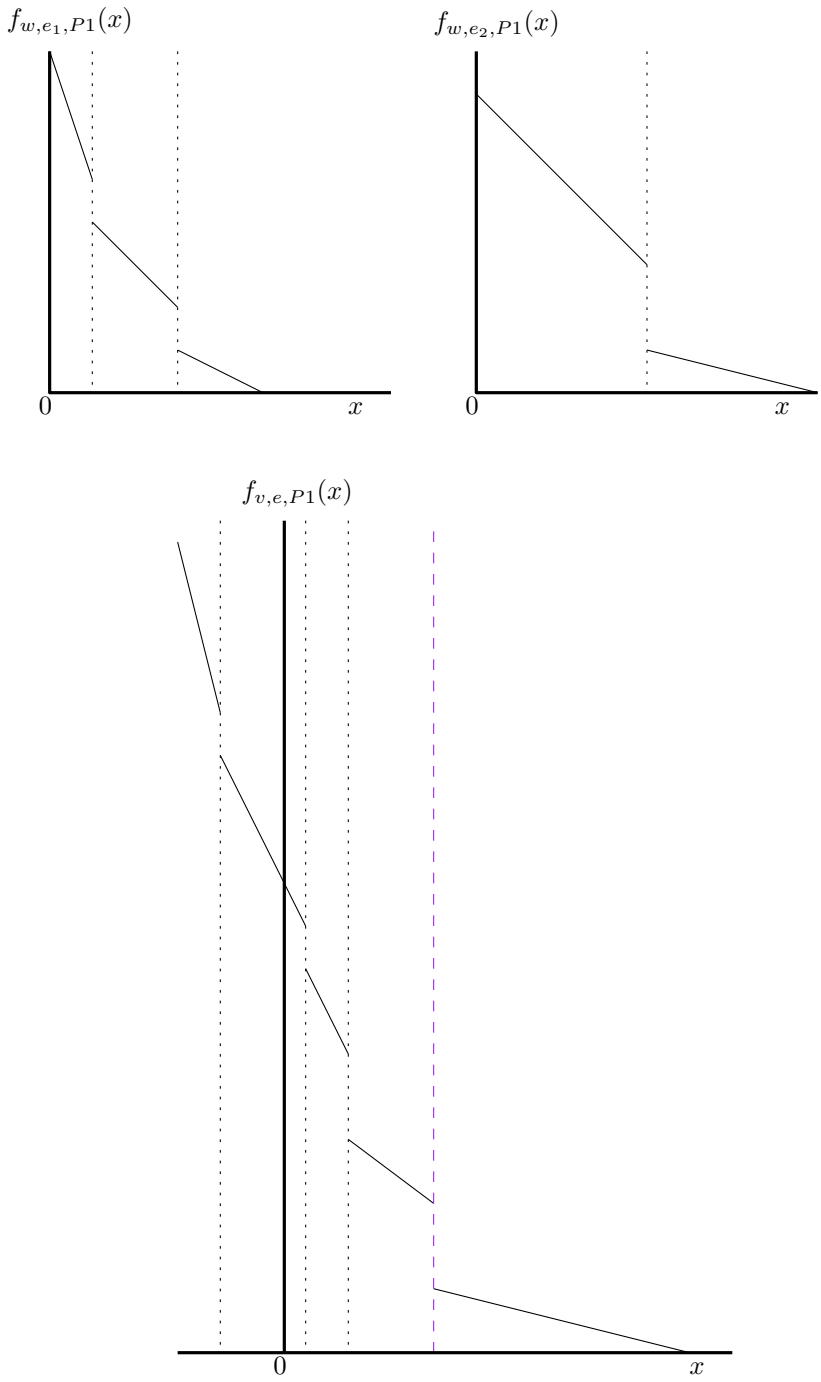

Figure 26: Example of a function that consists of a sum of other functions according to the graph in Figure 25. The purple line indicates $\bar{x}$, i.e. the first point at which one of the functions in the sum reaches zero. Everything right of the purple line is $(w(e) + x) + c$, with c some constant. Note that the functions are also shifted $w(e)$ to the left.

all the functions of f_{v,e,P_i} and g_{v,e,P_i}. The second method will artificially transform the tree G into a rooted (directed) tree.

Method 1

First, we will calculate all functions. For each edge $e = (v', w') \in E$, a function f_{v,e,P_i} has to be calculated four times. Since P_i can have the values P_1 and P_2 and v can be either v' or w'. Therefore we have to calculate $4|E|$ functions. Note that the functions g_{v,e,P_i} can be trivially found in $O(1)$ by the property: $g_{v,e,P_i}(x) = vol(\mathcal{T}_{v,e}) - f_{v,e,P_i}(x)$. Calculating all functions f_{v,e,P_i} for all $v \in V$ and all $e \in E_v$ for both P_1 and P_2 takes $O(|E| \cdot |S| \log \Delta(G) + |V|)$. After the initialization of the function, any query would take $O(\log S)$ time.

Method 2

We will now discuss the second method, where we have an artificially rooted tree G. Let d be the diameter of the graph G. The rooted tree will have depth $O(d)$. Calculating all the functions going downwards, i.e. the functions f_{v,e,P_i} such that e is connected to a child of v, takes $O(d \cdot |S| \log \Delta(G) + |E|)$. Note that if G is a balanced binary tree, $d = \log |V|$ and $\Delta(G) = 2$, we would get a running time of $O(|S| \log |V| + |E|)$. A query on a function f_{v,e,P_i} has two cases. If e is connected to a child of v, the query takes $O(\log |S|)$ time, since the function is already precomputed. But if the edge e is connected to the parent of v, extra calculation is needed. An example graph is drawn in Figure 27, where the query $f_{v,e,P_i}(x)$ is indicated. Let v be a vertex and let $e = (v, w)$ be an edge. Say we know the functions f_{w,e',P_i} for all $e' \in E_w \setminus \{e\}$. The value $f_{v,e,P_i}(x)$ can be calculated by calculating the correct sum $\sum_{e' \in E_w \setminus \{e\}} f_{w,e',P_t}(x + w(e))$ as described in function (3). This takes $O(\log |E_w|)$ time. This has to be done $O(d)$ times, as the length from the path from v to the root is $O(d)$. Therefore a query takes $O(d \cdot \log |S| \cdot \log \Delta(G))$ time. If G was a balanced binary tree, a query takes $O(\log |V| \log |S|)$ time.

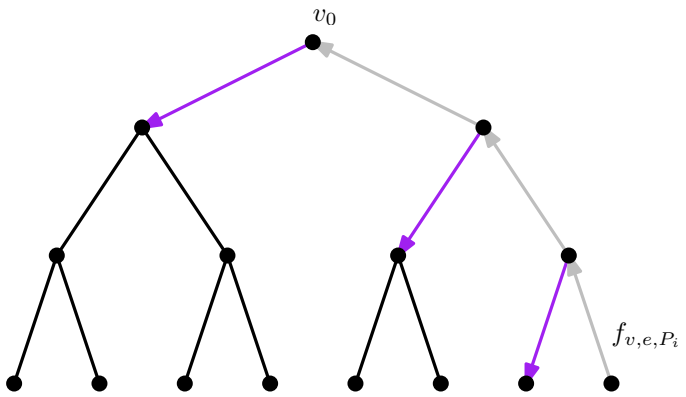

Figure 27: Example of a graph G, rooted at some vertex v_0. The functions needed for the query f_{v,e,P_i} are indicated by an arrow. The purple arrows initialized and takes $O(1)$, the gray ones need to be calculated for the query.

After initializing the tree we can calculate any value $f_{v,e,P_i}(x)$ in $O(Q)$ time, where Q is the query time. We will now show how the best cut is found. Given any vertex v in V, let s be the closest site to v and let $e = (v, w) \in E_v$ be the edge contained in the shortest path from v to s. We know that e is a tree edge in the *ESPT*. Let $\bar{s}$ be the closest site of v for which the shortest path does not go through e. Let $\bar{e} \neq e$ be the edge contained in the shortest path from $\bar{s}$ to v. If we cut on edge e, we know that $\bar{s}$ will be the newest closest site to v. We can now calculate the payoff for P_1 easily by calculating f_{v,e',P_i} for each $e' \in E_v \setminus \{e, \bar{e}\}$ with P_i the owner of $\bar{s}$. Note that if $P(\bar{s})$ is P_1, we would snap off e from w, since then e will also be owned by P_1, otherwise we would snap off e from v. We already know the payoff for P_1 for each subtree, and since we know that there is no neutral space, we can easily calculate the relative payoff change for P_1 which takes $O(|E_v| \cdot Q)$ time, where Q is the time it takes to query a function. According to Lemma 4.2 and Lemma 4.1,

it would be sufficient to check such a cut for all $v \in |V|$ that are owned by P_2. So calculating all these cuts would take $O(|E| \cdot Q)$. The algorithm can be seen in Algorithm 5.

Analysis of the two methods

We have described two methods, the first method where we initialize all the functions gave an initialization time of $O(|E| \cdot |S| \log \Delta(G) + |V|)$ and a query time of $O(\log |S|)$. Therefore, the entire algorithm takes $O(|E| \cdot |S| \log \Delta(G) + |E| \log |S| + |V|)$ which simplifies to $O(|E| \cdot |S| \log \Delta(G) + |V|)$. The second method, that initializes functions on an artificial rooted tree, has an initialization time of $O(d \cdot |S| \log \Delta(G))$ and a query time of $O(d \log |S| \log \Delta(G))$. The entire algorithm therefore takes $O(d \cdot |S| \log \Delta(G) + |E| d \log |S| \log \Delta(G))$. We formalize the following theorem.

Theorem 2. *Given a graph G and two players P_1 and P_2, where P_1 has n sites and P_2 has k sites. There exists an algorithm to find the best cut with a running time as presented in Table 1.*

	Best case	Worst case
Method 1	$O(n \cdot \|S\|)$	$O(n \cdot \|S\| \log \Delta(G))$
Method 2	$O(n \log n \log \|S\|)$	$O(nd \log \|S\| \log \Delta(G))$

Table 1: The best case and worst case running times of the algorithm, we say that $|V| = n$ and because G is a connected tree we get $|E| = n - 1$. We have d the diameter of the graph, S is the set of sites and $\Delta(G)$ is the degree of the graph.

Note that S is a subset of V and that in the worst case we have $|S| = O(|V|)$. But $|S| = O(|V|)$ is not a realistic scenario, therefore we can assume only the practical cases and say that $|S|$ is a constant. Table 2 shows the running times when we consider $|S|$ constant and Table 3 shows the running times when we consider $|S| = |V|$.

	Best case	Worst case
Method 1	$O(n)$	$O(n \log \Delta(G))$
Method 2	$O(n \log n)$	$O(nd \log n \log \Delta(G))$

Table 2: The best case and worst case running times of the algorithm, we say that $|V| = n$ and because G is a connected tree we get $|E| = n - 1$. In this table the running times are displayed assuming $|S|$ is constant.

	Best case	Worst case
Method 1	$O(n^2)$	$O(n^2 \log \Delta(G))$
Method 2	$O(n \log^2 n)$	$O(nd \log n \log \Delta(G))$

Table 3: The best case and worst case running times of the algorithm, we say that $|V| = n$ and because G is a connected tree we get $|E| = n - 1$. In this table the running times are displayed assuming $|S| = O(|V|)$.

Algorithm 5 Best cut on a tree with 2 players

function TypeIICandidateCut(Tree $T = (V, E)$, Sites $S = (s_1, s_1, \ldots, s_r)$)
  $ESPT \leftarrow$ calculate Extended Shortest Path Tree
  $c_0 = (v_0, e_0, t_0) \leftarrow$ TypeIIICandidateCut$(T, ESPT, S)$
  for each $v_i \in V$ **do**
    $s_i \leftarrow$ closest site of v_i
    $e_i \leftarrow$ edge connected to v_i contained in shortest path from s_i to v_i
    $\bar{s_i} \leftarrow$ edge closest to v_i of which the path does not go through e_i
    $\bar{e_i} \leftarrow$ edge connected to v_i contained in shortest path from $\bar{s_i}$ to v_i
    for each $e \in E_{v_i}$ **do**
      $\mathcal{T}_{v_i,e} \leftarrow$ subtree spanned by v_i and e
      $\mathcal{V}_{v_i,e} \leftarrow vol\mathcal{T}_{v_i,e}$
      $F_{v_i,e} \leftarrow$ Payoff for P_1 on $\mathcal{T}_{v_i,e}$
      $G_{v_i,e} \leftarrow \mathcal{V}_{v_i,e} - F_{v_i,e}$ ▷ Payoff for P_2 on the subtree.
    end for
  end for
  InitializeFunctions(T,S)
  for each $v_i \in V$ owned by P_2 **do** ▷ This loop tries a cut for each vertex owned by P_2.
    $w_i \leftarrow$ vertex connected to e_i s.t. $v_i \neq w_i$
    $f \leftarrow 0$
    for each $e \in E_{v_i}$ s.t. $e \neq e_i$ **do**
      $f' \leftarrow$ Query$(v, e, P(\bar{s_i}), d(\bar{s_i}, v_i))$ ▷ Queries payoff for P_1 $f_{v,e,P(\bar{s_i})}$ with force $d(\bar{s_i}, v_i)$.
      $f' \leftarrow f' - F_{v_i,e}$ ▷ Calculates payoff gained for P_1 on subtree $\mathcal{T}_{v_i,e}$.
      $f \leftarrow f + f'$
    end for
    if $S(\bar{s_i} = P_2$ **then**
      $c_i \leftarrow (v_i, e_i, 2 \cdot f)$
    else
      $c_i \leftarrow (w_i, e_i, 2 \cdot f + 2w(e_i))$
    end if
  end for
  $C \leftarrow \{c_0, c_1, \ldots, c_n\}$
  Return best cut in C
end function

Algorithm 6 Initializes all the functions f_{v,e,P_i} for a Tree

function InitializeFunctionsAll(Tree $T = (V, E)$, Sites $S = (s_1, s_2, \ldots, s_r)$)
  for each $v \in V$ **do**
    for each $e \in E_v$ **do**
      for $P \in \{P_1, P_2\}$ **do**
        $v.function_{e,P} = f_{v,e,P}$
      end for
    end for
  end for
end function

Algorithm 7 Initializes the functions f_{v,e,P_i} for a Tree T as if T is a directed rooted tree

function InitializeFunctionsRootedTree(Tree $T = (V, E)$, Sites $S = (s_1, s_2, \ldots, s_r)$)
  $v_0 \leftarrow$ vertex in V ▷ Will be the root
  for each $v \in V$ **do**
    $\bar{e} \leftarrow$ parent edge of v ▷ Is empty for v_0
    for each $e \in E_v \setminus \{\bar{e}\}$ **do**
      $v.function_{e,P} = f_{v,e,P}$
    end for
  end for
end function

Algorithm 8 Queries a function f_{v,e,P_i} after the initialization *InitializeFunctionsAll*, Algorithm 6

function QueryFunctionsAll(Vertex v, Edge e, Player P, Force x)
  Segment $R = (a, \frac{\Delta y}{\Delta x}, f(a)) \leftarrow$ segment in $v.function_{e,P}$ containing x ▷ Using binary search
  Return $f(a) + \frac{\Delta y}{\Delta x} \cdot (x - a)$
end function

Algorithm 9 Queries a function f_{v,e,P_i} after the initialization *InitializeFunctionsRootedTree*, Algorithm 7

function QueryFunctionsRootedTree(Vertex v, Edge e, Player P, Force x)
  $v_0 \leftarrow$ root node
  if e is child edge of v **then**
    Segment $R = (a, \frac{\Delta y}{\Delta x}, f(a)) \leftarrow$ segment in $v.function_{e,P}$ containing x ▷ Using binary search
    Return $f(a) + \frac{\Delta y}{\Delta x} \cdot (x - a)$
  else
    $\bar{v} \leftarrow v$
    $\bar{e} \leftarrow e$
    $\bar{e}' \leftarrow$ *empty*
    $f \leftarrow 0$
    while *true* **do**
      for each $e \in E_{\bar{v}} \setminus \{\bar{e}, \bar{e}'\}$ **do**
        Segment $R = (a, \frac{\Delta y}{\Delta x}, f(a)) \leftarrow$ segment in $v.function_{e,P}$ containing x ▷ Using binary search
        $f \leftarrow f + f(a) + \frac{\Delta y}{\Delta x} \cdot (x - a)$
      end for
      if $\bar{v} = v_0$ **then**
        Return f ▷ Breaks the while loop
      end if
      $\bar{v} \leftarrow$ parent of $\bar{v}$
      $\bar{e}' \leftarrow \bar{e}$
      $\bar{e} \leftarrow$ parent edge of $\bar{v}$
    end while
  end if
end function

5 Placing a rope

In this section we assume that each graph has an associated planar embedding. When adding a rope between the points p_1 and p_2, the weight of the rope is the Euclidean length of these two points in its planar

embedding. In the game *Lines* the player is given a planar graph and is not allowed to place a rope between points p_1 and p_2, such that the new rope crosses other edges. In other words, the transformed graph G' must stay planar after inserting a straight line segment. However, in this section we will look at a rope that can be placed between any two points; if the rope crosses a different line segment they do not connect, as discussed in Section 3.3.1 and as illustrated in Figure 5. Both the initial graph G and the transformed graph G' need not be planar.

5.1 A rope on a tree with two sites

Given are a connected tree $G = (V, E)$ and two sites s_1 and s_2 belonging to player P_1 and P_2 respectively. When placing a rope between points p_1 and p_2 there are three cases. Either both points in G are owned by P_1, both points in G are owned by P_2, or each point is owned by a different player. It is obvious that a rope is never placed between points p_1 and p_2, if these points are owned by P_2. When p_1 is owned by P_1 and p_2 by P_2, then the rope will be connected to s_1. When both points p_1 and p_2 are owned by P_1, the length of the rope can yield more influence. In this case an optimal rope can be the two points owned by P_1 such that the euclidean distance between these points are the largest.

Lemma 5.1. *Given a tree $G = (V, E)$ and two sites s_1 and s_2 belonging to player P_1 and P_2 respectively. If there exists an optimal rope placement, such that one endpoint gets connected to a point on G owned by P_1 and the other endpoint gets connected to a point on G owned by P_2. Then there exists an optimal rope placement where one endpoint is connected to s_1.*

Proof. Let x be any point owned by P_1 on the tree $G = (V, E)$ and let y be any point owned by P_2 on the tree G. Let G' be the resulting graph after the graph transformation when adding a rope between the points x and y. Let x' and y' be the new vertices on the resulting graph G' and let $e' = (x', y')$ be the new edge. There are three cases to consider:

i x' and y' are both owned by P_1.

ii x' is owned by P_1 and y' is owned by P_2.

iii x' and y' are both owned by P_2.

For each case, it is sufficient to prove that there is an equal or better solution where x coincides with s_1 for any y. In other words, for every arbitrary rope with endpoints x and y, we can find a rope at least as good where one endpoint is connected to s_1.

Case i

We now consider another rope from s_1 to y. Let G'' be the resulting tree after the graph transformation. Figure 28 shows an example. Let $x'' = s_1''$ and y'' be the new vertices and let $e'' = (x'', y'')$ be the new edge. By using the triangle inequality: $d_{G''}(s_1, y'') = d_{\text{eucl}}(s_1, y'') \leq d_{G'}(s_1, x') + d_{G'}(x', y')$ we can show $d_{G''}(s_1, y'') \leq d_{G'}(s_1, y')$. We know y' is owned by P_1, therefore y'' must also be owned by P_1. Let $e^y = (v_1, v_2)$ be the edge containing point y. We assume that v_1 is closer to s_2 than v_2 is to s_2. This implies that the shortest path from s_2 to v_2 goes through v_1 and e^y. We will now look at the set $\mathcal{S}_{s_2,G}(y)$. Since point y' and y'' are now owned by P_1 in the transformed graphs G' and G'' respectively, we have that P_2 is cut off from all points in $\mathcal{S}_{s_2,G}(y)$ in the corresponding graph transformation. We call $vol(\mathcal{S}_{s_2,G}(y)) = \kappa$, in both graph transformations the payoff for P_1 is increased by κ and for P_2 decreased by κ.

Now let $\Delta d' = d_{G'}(s_2', y') - d_{G'}(s_1', y')$ and $\Delta d'' = d_{G''}(s_2'', y'') - d_{G''}(s_1'', y'')$. Note that the shortest path from s_2 to y remains unchanged in the graph transformations, so we have $d_G(s_2, y) = d_{G'}(s_2', y') = d_{G''}(s_2'', y'')$. The increase in payoff for P_1 in G' among the path s_2' to y' will be $\frac{1}{2}\Delta d'$. Similarly, the increase in payoff for P_1 in G'' among the path s_2'' to y'' will be $\frac{1}{2}\Delta d''$. Note that the decrease in payoff for P_2 is also equal to $\frac{1}{2}\Delta d'$ and $\frac{1}{2}\Delta d''$ on G' and G'' respectively. The relative payoff between P_1 and P_2 is therefore changed

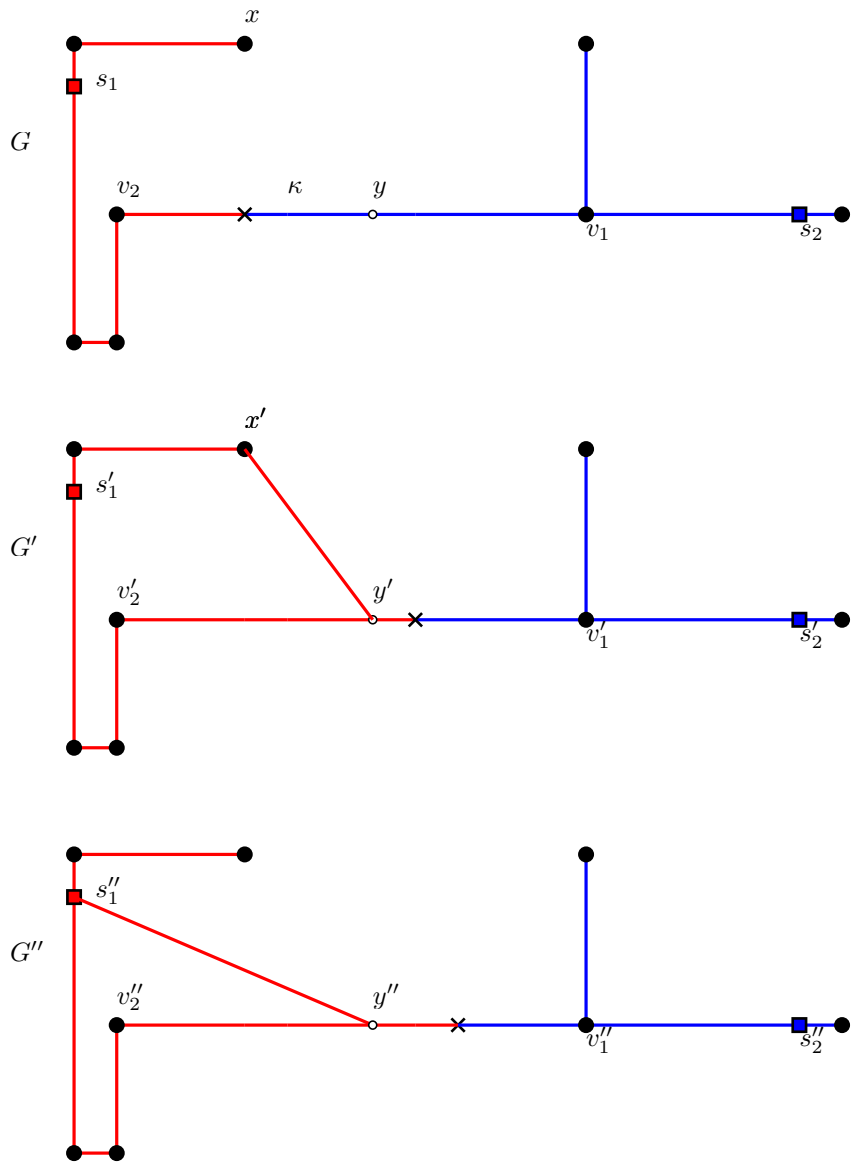

Figure 28: Example of a graph transformations G' and G'', where P_1 is the red color and P_2 is the blue color. Note that κ is given in G and that this area is owned by P_1 in both G' and G''.

in favor of P_1 by $\Delta d'$ and $\Delta d''$ on G' and G'' respectively. The relative payoff increase for P_1 in G' among the new shortest path s'_1 to s'_2 equals $w(e') + \Delta d'$. For G'' the relative payoff for P_1 among the path s''_1 to s''_2 is increased by $w(e'') + \Delta d''$. Now we calculate the difference in payoff among the path s_1 to s_2 in both transformed graphs G' and G''. We get $(w(e') + \Delta d') - (w(e'') + \Delta d'') = w(e') - w(e'') + \Delta d' - \Delta d''$. First

we calculate the value of $\Delta d' - \Delta d''$:

$$\begin{aligned} \Delta d' - \Delta d'' &= d_{G'}(s'_2, y') - d_{G'}(s'_1, y') - \\ &\qquad (d_{G''}(s''_2, y'') - d_{G''}(s''_1, y'')) \\ &= d_{G''}(s''_1, y'') - d_{G'}(s'_1, y') \\ &= w(e'') - d_{G'}(s'_1, y') \\ \rightarrow \qquad \Delta d' - \Delta d'' &= w(e'') - w(e') - d_{G'}(s_1, x) \end{aligned}$$

We can now substitute this value:

$$\begin{aligned} w(e') - w(e'') + \Delta d' - \Delta d'' &= w(e') - w(e'') + w(e'') - w(e') - d_{G'}(s_1, x) \\ &= -d_{G'}(s_1, x) \qquad \leq 0 \end{aligned}$$

From this we can conclude that the relative payoff for P_1 in G'' is equal or better than in G' among path s_1 to s_2. Note that the equation only looks at the relative payoff among the edges on the path s_1 to s_2 on G and on the newly placed rope. When P_1 owns a vertex along the path y' to s'_2 while it was owned by P_2 in G, then P_1 will cut off P_2 from a potential subgraph. Also since $d_{G''}(s_1, y'') \leq d_{G'}(s_1, y')$, we have that the vertices owned by P_1 among path y' to s'_2 are also owned by P_1 in G''. Therefore, the rope from s_1 to y, in this case, is at least as good as any arbitrary rope from x to y, where x is owned by P_1 and y is owned by P_2.

Case ii

Since x' is owned by P_1 and y' is owned by P_2, we know that $d_{G'}(s'_2, y') \leq d_{G'}(s'_1, x') + w(e')$. Analogously, we know $d_{G'}(s'_1, x') \leq d_{G'}(s'_2, y') + w(e')$. The increase in payoff for P_1 (between G and G') is equal to $\frac{1}{2}(d_{G'}(s'_2, y') - d_{G'}(s'_1, x') + w(e'))$, the increase in payoff for P_2 is equal to $\frac{1}{2}(d_{G'}(s'_1, x') - d_{G'}(s'_2, y') + w(e'))$. So the relative increase in payoff for P_1 becomes: $\frac{1}{2}(d_{G'}(s'_2, y') - d_{G'}(s'_1, x') + w(e')) - \frac{1}{2}(d_{G'}(s'_1, x') - d_{G'}(s'_2, y') + w(e')) = d_{G'}(s'_2, y') - d_{G'}(s'_1, x')$.

We now consider another rope from s_1 to y. Let G'' be the resulting tree after the graph transformation. Let $x'' = s''_1$ and y'' be the new vertices and let $e'' = (x'', y'')$ be the new edge. If y'' is owned by P_2 (and we know $s''_1 = x''$ is owned by P_1), then the relative increase in payoff (between G and G'') for P_1 equals $d_{G''}(s''_2, y'') - d_{G''}(s''_1, x'')$. We know that $d_{G''}(s''_1, x'') = 0$. So the relative increase in payoff for P_1 equals $d_{G''}(s''_2, y'') = d_{G'}(s'_2, y')$. Clearly $d_{G'}(s'_2, y') - d_{G'}(s'_1, x') \leq d_{G''}(s''_2, y'')$ so the rope in G'' is at least as good as any other arbitrary rope.

If y'' is owned by P_1 then the relative increase in payoff for P_1 among the path s''_1 to s''_2 equals $w(e'') + \Delta d''$, where $\Delta d'' = d_{G''}(s''_2, y'') - d_{G''}(s''_1, y'')$. Note, that $d_{G''}(s''_1, y'') = w(e'')$ because the shortest path from s_1 to y is obviously via the new edge e''. The relative increase in payoff for P_1 will be at least $w(e'') + d_{G''}(s''_2, y'') - w(e'') = d_{G''}(s''_2, y'')$. We obviously have $d_{G'}(s'_2, y') - d_{G'}(s'_1, x') \leq d_{G''}(s''_2, y'')$ so the rope in G'' is, in this case, at least as good as any other arbitrary rope from x to y, where x is owned by P_1 and y is owned by P_2.

Case iii

If x' is owned by P_2 in the transformed graph G', then the payoff for P_1 has obviously not increased. Furthermore, the payoff for P_2 has increased at least by $w(e') = d_{eucl}(x, y) \geq 0$. We now consider another rope from s_1 to s_1. Let G'' be the resulting tree after the graph transformation. We now obviously have that the payoff for both P_1 and P_2 remains unchanged in G''. Therefore, the rope from s_1 to s_1 is equal or better than the original rope.

□

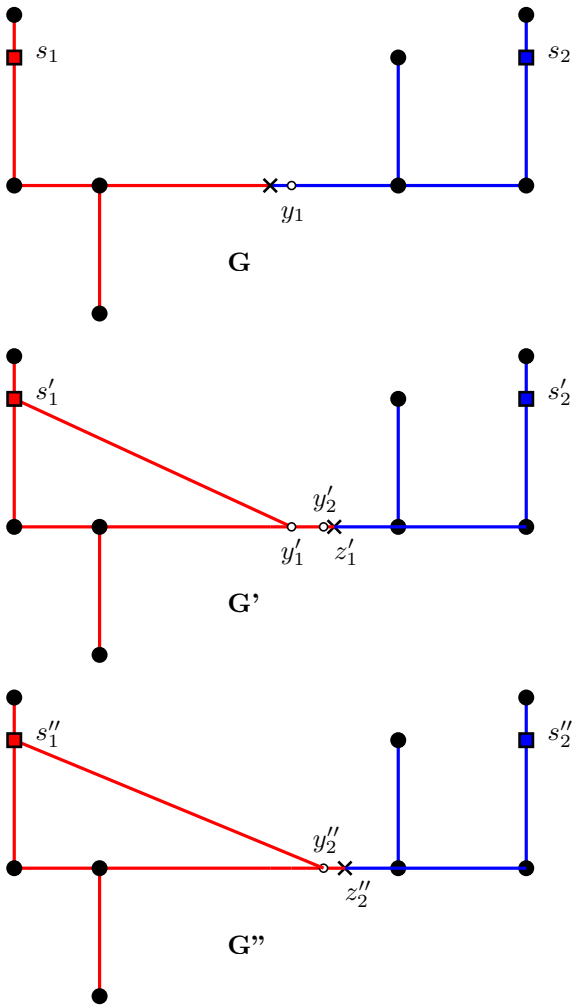

Figure 29: An example of a tree, where the rope is iteratively placed closer to a balance point.

Lemma 5.2. *Given a tree $G = (V, E)$ and two sites s_1 and s_2 belonging to player P_1 and P_2 respectively. If the best rope placement has one endpoint x connected to a point on G owned by P_1 and the other endpoint y is connected to a point on G owned by P_2, and such that y' is owned by P_1 in the graph transformation, then there exists a rope placement such that there does not exist another rope placement that is ϵ better. This rope has endpoint y such that $d_{G'}(s'_1, y') + \epsilon = d_{G'}(s'_2, y')$ with $\epsilon > 0$ arbitrarily small. Where $d_{G'}(s', p')$ is the new shortest distance from s to p after the rope placement on the transformed graph G'.*

Proof. Let y_1 be any point owned by P_2 on the tree $G = (V, E)$. We consider the rope from s_1 to y_1 and let G' be the resulting graph after the graph transformation, where s'_1 and y'_1 are the new vertices and $e' = (s'_1, y'_1)$ the new edge. We choose y_1 such that y'_1 is owned by P_1 after the graph transformation. Since y'_1 is owned by P_1 we have that $d_{G'}(s'_1, y'_1) = w(e')$. We call z'_1 the new boundary point between s'_1 and s'_2. We have that $d_{G'}(s'_1, z'_1) = w(e') + \frac{1}{2}\Delta d'$, where $\Delta d' = d_{G'}(s'_1, y'_1) - d_G(s_2, y_1)$. We define y_2 as a point on the path from y_1 to z_1 (with y'_2 the same point but on the graph G'), such that $d_{G'}(y'_2, z'_1) = \epsilon_1$ arbitrary small. We know that the point y_2 is owned by P_2 and the point y'_2 is owned by P_1. An example can be seen in Figure 29.

Now consider another rope from s_1 to y_2 and let G'' be the resulting graph after the graph transformation, where s''_1 and y''_2 are the new vertices and $e'' = (s''_1, y''_2)$ the new edge. By triangle inequality

$d_{G''}(s_1'', y_2'') = w(e'') \leq d_{G'}(s_1', y_1') + d_{G'}(y_1', y_2') = d_{G'}(s_2', y_2') + \epsilon_1$, we can see that the point y_2'' is owned by P_1. We call z_2'' the boundary point between s_1'' and s_2'' on G''. Note that the path from y_2'' to z_2'' will pass the point z_1'', unless $d_{G''}(s_1'', y_2'') = w(e'') = d_{G'}(s_1', y_1') + d_{G'}(y_1', y_2')$ then we have $z_2'' = z_1''$ are the same points. Now we will calculate the difference in payoff for both these graph transformations. The relative increase in payoff for P_1 in G' equals $w(e')+2w(\mathcal{S}_{s_2,G}(z_1))$. The relative increase in payoff for P_1 in G'' equals $w(e'')+2w(\mathcal{S}_{s_2,G}(z_2))$. We know that all points in $\mathcal{S}_{s_2,G}(z_1)$ are owned by P_1 in both graph transformations G' and G'', which implies, $\mathcal{S}_{s_2,G}(z_1) \subseteq \mathcal{S}_{s_2,G}(z_2)$. We also know that all points κ in the path from z_1 to z_2 are in the set $\mathcal{S}_{s_2,G}(z_2)$. Since $w(\kappa) = \frac{1}{2}(w(e'')-w(e'))$ we know that $w(\mathcal{S}_{s_2,G}(z_2) \geq \frac{1}{2}(w(e'')-w(e'))+w(\mathcal{S}_{s_2,G}(z_1))$. Therefore $w(e') + 2w(\mathcal{S}_{s_2,G}(z_1)) \leq w(e'') + 2w(\mathcal{S}_{s_2,G}(z_2))$. The rope from s_1 to y_2 is at least as good as the rope from s_1 to y_1.

We can now use induction to prove that for any rope from s_1 to y_1, we can find a better rope s_1 to y_2, where y_2 lies closer to the boundary point between s_1 and s_2. Note, we need the iterative method because we assume that each vertex v must have a different distance to each site. If the rope was directly connected to a point p_1, such that p_1' becomes a boundary points in the transformed graph, then we can get an infinite number of boundary points. If we assume that the endpoint of the rope, in case of a tie, will be owned by P_1, then the optimal rope placement will be from s_1 to the boundary point. Otherwise we must avoid vertices being boundary points. In this case an optimal rope does not exist and we can approximate the optimal rope with a rope from s_1 to the point y_i where y_i is arbitrarily close to the boundary point. □

Lemma 5.2 uses points with the property $d_{G'}(s_1', y') = d_{G'}(s_2', y')$. Since the endpoints of the rope are s_1 and y' we have the property $d_{eucl}(s_1, y) = d_G(s_2, y)$, for which we have the following definition.

Definition 5.1. *Given a tree $G = (V, E)$ and two sites s_1 and s_2 belonging to player P_1 and P_2 respectively. A **balance point** y is a point on the graph G such that the euclidean distance from s_1 to y equals the shortest distance from s_2 to y on the graph. In other words, $d_{eucl}(s_1, y) = d_G(s_2, y)$.*

Lemma 5.3. *Given a tree $G = (V, E)$ and two sites s_1 and s_2 belonging to player P_1 and P_2 respectively. There always exists at least one balance point p on the graph G such that P_1 does not own p.*

Proof. The tree is connected so we know there is exactly one path from s_1 to s_2. Along this path there is exactly one boundary point p. We have that $d_{eucl}(s_1, s_2) > d_G(s_2, s_2) = 0$ (if $d_{eucl}(s_1, s_2) = 0$ then s_1 and s_2 have the same location, but we omit this case). We know, by triangle inequality that $d_{eucl}(s_1, p) \leq d_G(s_1, p) = d_G(s_2, p)$. If $d_{eucl}(s_1, p) = d_G(s_2, p)$ then we found a balance point. Now assume $d_{eucl}(s_1, p) < d_G(s_2, p)$. Since the distance function on G and in the euclidean metric is continuous along the path from p to s_2, we can use the intermediate value theorem. We have $d_{eucl}(s_1, s_2) > d_G(s_2, s_2)$ and $d_{eucl}(s_1, p) < d_G(s_2, p)$ so there must be a point p' along this path such that $d_{eucl}(s_1, p') = d_G(s_2, p')$. □

In Lemma 5.2 the endpoints x and y of the rope are chosen in such a way that x equals the site s_1 owned by P_1 and that y is a point on G such that y is owned by P_2 and y' is owned by P_1. We now consider the case where y' is still owned by P_2.

Lemma 5.4. *Given a tree $G = (V, E)$ and two sites s_1 and s_2 belonging to player P_1 and P_2 respectively. If there exists an optimal rope placement to maximize for P_1, such that one endpoint is $x = s_1$ and the other endpoint y is owned by P_2 in G and after the rope in the transformed graph G'. Then there exists an optimal rope placement such that y is the point owned by P_2 for which the distance $d_G(s_2, y)$ is maximal.*

Proof. As shown in case ii of the proof for Lemma 5.1, the relative increase in payoff for P_1 equals $d_{G'}(s_2', y') - d_{G'}(s_1', x')$. Because one endpoint of the rope x equals the site s_1, we get $d_{G'}(s_1', x') = 0$. Therefore, the relative increase in payoff becomes $d_{G'}(s_2', y') = d_G(s_2, y)$. So, the rope from s_1 to a point y for which $d_G(s_2, y)$ is maximal is obviously an optimal rope. □

Both Lemmas 5.2 and 5.4 look at ropes with endpoints x and y where x is owned by P_1 and y is owned by P_2. We will now look at ropes where both endpoints are owned by P_1.

Lemma 5.5. *Given a tree $G = (V, E)$ and two sites s_1 and s_2 belonging to player P_1 and P_2 respectively. If there exists an optimal rope placement to maximize for P_1, such that both endpoints get connected to points on G owned by P_1, then there exists an optimal rope placement between two points x and y on the graph, such that x and y are the two furthest away points, using the Euclidean metric, owned by P_1.*

Proof. Say we have two points x and y on the tree $G = (V, E)$. Both points x and y are owned by P_1. After the rope placement we obtain the transformed graph G', such that the points x' and y' have the same location as x and y but are now vertices that share the added edge e'. Both x' and y' are owned by P_1. We have two cases:

i $d_{G'}(s_1, x') = d_G(s_1, x)$ and $d_{G'}(s_1, y') = d_G(s_1, y)$

ii $d_{G'}(s_1, x') < d_G(s_1, x)$

Case i

In this case, the shortest path from s_1 to either endpoint remains unchanged. This means that there is an inner boundary point on the edge e'. This implies that there exists a point e'_λ such that the path from s_1 to e'_λ via x' has the same distance as the path from s_1 to e'_λ via y'. Since the distances to x' and y' remain unchanged, the payoff of P_1 is only increased by $w(e') = d_{eucl}(x, y)$. Let $\bar{x}$ and $\bar{y}$ be any two points owned by P_1. A rope from $\bar{x}$ to $\bar{y}$ increases the payoff of P_1 by at least $d_{eucl}(\bar{x}, \bar{y})$. Therefore, if $d_{eucl}(x, y) \leq d_{eucl}(\bar{x}, \bar{y})$ then we have that the rope from $\bar{x}$ to $\bar{y}$ is at least as good as the rope from x to y.

Case ii

In this case we have $d_{G'}(s_1, x') < d_G(s_1, x)$, which implies that the new shortest path from s_1 to x' runs through the rope e'. Now let the boundary point between P_1 and P_2 be the point p (with p' the same point on G', note that p' is not necessarily a boundary point). Again we have two cases, either the shortest path on G from s_1 to p goes through x, or it does not go through x.

We first discuss the case in which the shortest path on G from s_1 to p goes through x. We now know that $d_G(s_1, p) = d_G(s_1, x) + d_G(x, p)$. We also know that $d_{G'}(s_1, p') = d_{G'}(s_1, x') + d_G(x', p')$. Since $d_G(x, p) = d_G(x', p')$ we get that $d_{G'}(s_1, p') < d_G(s_1, p)$. The shortest distance from s_2 to p' remains unchanged so p' is not a boundary point. This implies that the point p' is now owned by P_1. We can now use the same method used to proof Lemma 5.1, that the rope from s_1 to p' is a better rope than the rope from x to y. This implies that x and y is not an optimal rope if the shortest path on G from s_1 to p goes through x.

We will now discuss the case in which the shortest path on G from s_1 to p does not go through x. Since G is a tree, there is no path from x to p. Therefore the payoff is only increased by the weight of e' which is equal to the euclidean distance $d_{\text{eucl}}(x, y)$ of the two points x and y. For any two points $\bar{x}$ and $\bar{y}$ owned by P_1 such that $d_{eucl}(x, y) \leq d_{eucl}(\bar{x}, \bar{y})$ we get that the rope from $\bar{x}$ to $\bar{y}$ is at least as good as the rope from x to y.

In both the cases i and ii, we have that there exists a rope at least as good if $d_{eucl}(x, y)$ is maximized. Thus the lemma holds. □

5.1.1 Candidate ropes

In Lemma 5.1 and 5.2 we can conclude that a rope between two points p_1 and p_2, such that both points in G are owned by a different player but will both be owned by P_1 in the transformed graph G', is optimal when one point is s_1 and the other point is as close to a balance point as possible. A rope can sometimes not be placed on a balance point p itself because if p coincides with a vertex then both P_1 and P_2 has the same distance to the vertex in the transformed graph, which can result in an infinite number of boundary points. For practical purposes we say that a rope to a balance point can be placed and that P_1 will win the 'tie'.

This means that to avoid an infinite number of boundary points, that the Voronoi Cell of P_1 will expand beyond the vertex coinciding with p' while the Voronoi Cell of P_2 will stop expanding beyond this vertex. Note that the payoff will be $\lim_{\epsilon \downarrow 0} f(p-\epsilon)$, where $f(x)$ is the payoff when the rope is placed between s_1 and x, and where $\epsilon \downarrow 0$ means that ϵ approaches 0 such that $d_{G'}(s_1', p-\epsilon) < d_{G'}(s_2', p-\epsilon)$. We call such a rope a **Type I candidate rope**. An example can be seen in Figure 30.

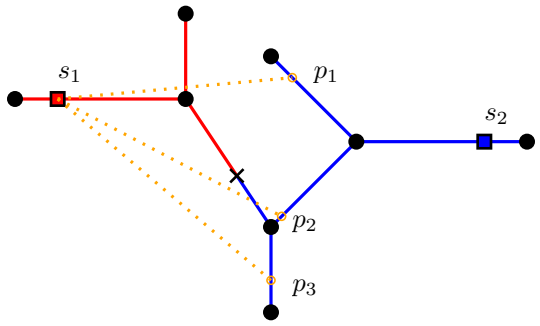

Figure 30: An example of a tree, where the red area is owned by P_1 and the blue area by P_2. The orange circles indicate the balance points. The optimal rope in this example is from s_0 to p_2.

From Lemma 5.5 we can conclude that a rope between two points p_1 and p_2, such that both points are owned by P_1, is optimal when p_1 and p_2 are the two furthest points owned by P_1. We call such a rope a **Type II candidate rope**. An example of an optimal Type II candidate rope can be seen in Figure 31.

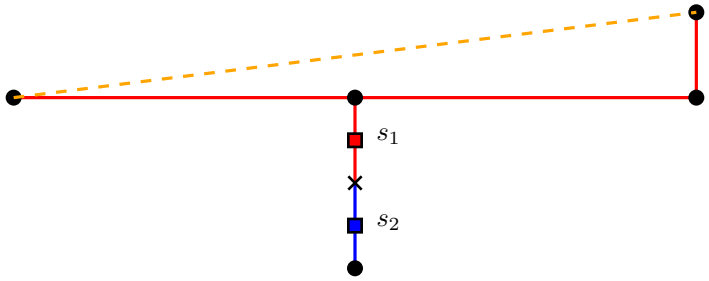

Figure 31: An example of a tree, where the red area is owned by P_1 and the blue area by P_2. The dashed orange line indicates a Type II candidate rope, which is also an optimal rope.

From Lemma 5.4, we see that a rope between two points p_1 and p_2, such that both points in G and the transformed graph G' are owned by a different player, is optimal when one endpoint is p_1 and the other endpoint is the point owned by P_2 furthest from s_2. We call such a rope a **Type III candidate rope**. An example of an optimal type III candidate rope can be seen in Figure 32. Obviously, the optimal rope on a tree G is either a Type I, Type II or a Type III candidate rope.

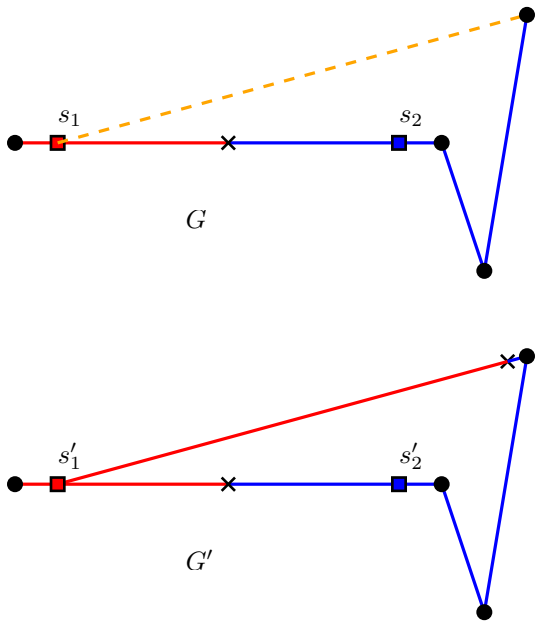

Figure 32: An example of a tree, where the red area is owned by P_1 and the blue area by P_2. The dashed orange line indicates a Type III candidate rope, which is also an optimal rope. The transformed graph G' is also shown after inserting the Type III candidate rope.

5.1.2 The algorithm

Given a tree $G = (V, E)$ and two sites s_1 and s_2 belonging to player P_1 and P_2 respectively. We say s_1 has coordinates (x_0, y_0).

First we will try to find the type I candidate ropes. Given any edge segment e owned by P_2 spanned between two points $p_1 = (x_1, y_2)$ and $p_2 = (x_2, y_2)$, where x_i and y_i are the coordinates in the planar embedding and such that the edge segment e does not contain any boundary point. Note, we use edge segments here because some edges contain a boundary point and we only want to look at the part of the edge owned by P_2. We say that an edge segment is spanned between two vertices or a boundary point and a vertex. We will denote a distance function from s_1 to e_λ in the euclidean metric and from s_2 to e_λ along the graph G. If $\lambda = 0$ we get $e_\lambda = p_1$ and if $\lambda = 1$ we get $e_\lambda = p_2$. The function $f(\lambda)$ denotes the distance from s_2 to e_λ via the graph, in other words $f(\lambda) = \lambda d_G(s_2, p_2) + (1 - \lambda) d_G(s_2, p_1)$. We denote the function $g(\lambda_x)$ as the euclidean distance from s_1 to e_λ, which is $g(\lambda) = \sqrt{(x_0 - (\lambda x_1 + (1 - \lambda)x_2))^2 + (y_0 - (\lambda y_1 + (1 - \lambda)y^2))^2}$. A Type I candidate cut will be at e_λ for which $f(\lambda) = g(\lambda)$. This point can be calculated in $O(1)$ time. For each edge-segment, of which we have $O(|E|)$ many, we have to calculate the balance point. Finding all balance points therefore takes $O(|E|)$ time. For each balance point, we have to calculate how the payoff is affected. Recalculating the Voronoi Cells takes $O(|V| + |E|)$, so finding the best Type I candidate rope takes $O(|E| \cdot (|V| + |E|))$ time. Since the graph is a tree the algorithm takes $O(|V|^2)$ time.

We will now try to find the type II candidate rope. There is only one type II candidate rope needed to be found. Since the Voronoi Cell is one connected component, all we need to do is find the diameter of the points owned by P_1. When we found the two points owned by P_1 which are furthest apart using the Euclidean metric, then the payoff for P_1 is increased by the length of this rope. Finding the (best) Type II candidate rope therefore takes $O(|V| \log |V|)$ time.

Lastly, we will find the Type III candidate rope. We can simply do a Breadth-First Search from s_2 to find the furthest point p from s_2 owned by $P2$. If the Euclidean distance $d_{eucl}(s_1, p)$ is smaller than distance

along the graph $d_G(s_2, p)$, then point p will be owned by P_1 after inserting the rope. In this case, there does not exist a Type III candidate rope. Otherwise the relative payoff increase for P_1 will be $d_G(s_2, p)$. Calculating this candidate rope takes $O(|V| + |E|)$ time.

6 Extensions

6.1 Cuts on arbitrary graphs

We presented an algorithm that works on balanced trees in $O(n \log^2 n)$ and a simple algorithm that works on arbitrary graphs in $O(n^2)$. I expect there exists an algorithm that works on arbitrary graphs in sub-quadratic time. Let G be an arbitrary graph and let the graph in Figure 33 be its *ESPT*. If we know how the payoff changed after applying cut c_1 or cut c_2 on the graph G, then we can calculate what will happen if we apply cut c_3 on the graph G.

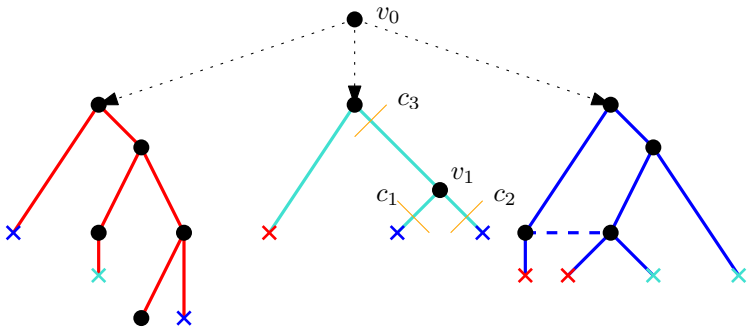

Figure 33: An example of a tree, where the red area is owned by P_1 and the blue area by P_2. Two example cuts are presented in the *ESPT* representation.

We consider the cut c_3. Let e_1 and e_2 be the child edges of v_1. The Voronoi Cell corresponding to the closest boundary point in the subtree of v_1 will take control of v_1. The Voronoi Cell then proceeds to flow downwards. We need some function f'_{v,e,P_i} initialized on the *ESPT*. The function f'_{v,e,P_i} gives the payoff of P_1 solely on the subtree in the *ESPT* spanned by v and e. Let v be a vertex with a child edge e connected to a boundary node p in the *ESPT*. If we cut somewhere in a subtree containing v, a new Voronoi Cell never reaches v with a smaller distance then before the cut. Let $d(p)$ be the distance of the boundary point p to the closest site. If we want to evaluate a function $f'_{v,e,P_i}(x)$, we get that x can never be smaller then $d(p) - w(e)$. Given that the boundary point is with an opposing player, we get $f'_{v,e,P_1}(x) = w(e) - \frac{1}{2}(x - (d(p) - w(e)))$. So in the example given in Figure 33, cut c_3 can be calculated by finding out the closest boundary point to v_1. Then we can use a function query to find out how the payoff of each player is affected.

However, this does not work with cross edges. An example *ESPT* is drawn in Figure 34. Note, that in this Figure, the boundary nodes b_1 and b_2 actually represent an inner boundary point and that the parent edges of b_1 and b_2 actually represents a cross edge. Let v_1 be the lowest common ancestor of b_1 and b_2. If we cut in e_2, the Voronoi Cell at b_1 starts to expand upwards. In other words, the boundary node b_1 is active. The same holds for a cut in e_3; the boundary node b_2 would be active. But once we cut in the edge e_1, the Voronoi Cell to both b_1 and b_2 is cut off. In other words, b_1 and b_2 become inactive boundary nodes. This implies that the set of functions $f'_{v,e,P_i}(x)$ are different with respect to cutting below or above the node v_1. Let $\mathcal{B}$ be the set of the lowest common ancestors of the endpoints of all cross edges. In other words, $\mathcal{B} = \{v \mid v = LCA(a, b) : (a, b) \in E \text{ a cross-edge }\}$. We have $|B| \leq |V|$. I suspect that there is an algorithm to find the optimal cut on arbitrary graphs in $O(|B| \cdot |S| \cdot (|V| + |E|))$ time.

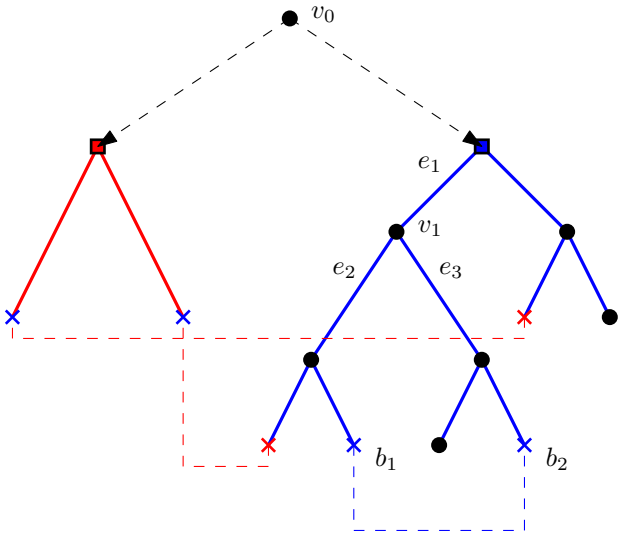

Figure 34: An example of an *ESPT*, where the red area is owned by P_1 and the blue area by P_2. The boundary nodes are connected with a dashed line to illustrate the connections on the actual graph.

6.2 Percentage of winning cuts

A winning cut is a cut, such that the payoff of P_1 is greater then the payoff of P_2 in the transformed graph. More generally, the payoff of P_1 is greater than the payoff of any other player. This paper presents an algorithm that finds the optimal cut in $O(|E| \cdot (|V| + |E|))$ time. It does so by simulating a cut on each Tree Edge following Lemma 4.3. The algorithm can be slightly adapted to find the percentage of winning cuts. Let $e = (v, w)$ be an edge in the graph G. Let G' be the transformed graph after cutting at point e_λ. Note that the location of the cut does not influence which player owns v' and w' after the cut. The location of the cut only affects the payoff gained on edge e'. Whereas $P(v')$ would gain $\lambda w(e)$ and $P(w')$ would gain $(1 - \lambda)w(e)$. We can design an algorithm that simulates a cut at each edge $e \in E$ and calculates the payoff for each player. The percentage of winning moves on e can then be calculated in $O(1)$, so the entire algorithm would take $O(|E| \cdot (|V| + |E|))$.

This implies that if the algorithm works in such a way that it calculates the payoff for a cut on each edge in the graph, it does not take any extra time complexity to check for the percentage of winning moves. The algorithms presented to solve arbitrary trees with 2 players therefore have the same time complexity as the algorithm for getting the percentage of winning moves.

6.3 Ropes

In this thesis, a small start was made on Ropes in the Continuous Voronoi game on graphs. The next step would be to look at multiple opposing sites. But when the opponent has multiple sites, it is not always guaranteed that the rope is a Type I, Type II or Type III candidate rope. An example is shown in Figure 35, where a tree is drawn with two opposing sites. The balance points are indicated by the orange circles. The optimal rope here is from the site s_0 to the vertex v_1.

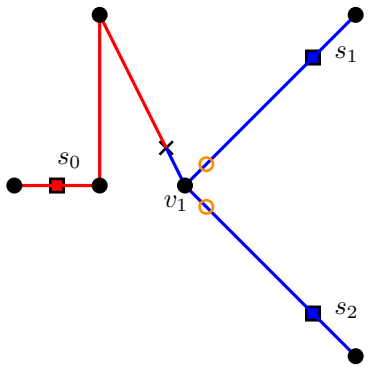

Figure 35: An example of a tree, where the red area is owned by P_1 and the blue area by P_2. The vertices in T are colored in light blue. The balance points are indicated by orange circles.

It is still unclear that adding the candidate cut from s_0 to any inner boundary point of P_2 would always be sufficient to encapsulate the optimal solution. However, it does seem that when there is no Type II or Type III candidate rope that is optimal, a rope should always be drawn from s_0. For general graphs, a rope should probably always be connected to a site owned by P_1. Such a claim greatly reduces the search space and the candidate ropes needed to be checked, but it still needs proof. Due to this property, this kind of rope, where any two arbitrary points x and y can be connected and where crossing edges are not connected (Figure 5), seems to be the least interesting. A type of rope, which can only connect x and y such that the graph remains planar (Figure 6), therefore seems to be a harder and more interesting problem. Since now, one of the end points of the rope is not necessarily fixated to a site owned by P_1.

6.4 Percentage of winning ropes

Finding the percentage of winning ropes is still an open problem. The problem of finding a rope is a two-dimensional problem, where the placement of both endpoints must be found. The algorithm presented in this paper for ropes indicates that the problem can be reduced to a one-dimensional problem since one endpoint is always connected to a site owned by P_1. In other words, the search space is reduced significantly for finding the optimal rope. Finding the percentage of winning ropes is therefore a different, more computationally expensive task.

7 Discussion

In this paper we extended Tom Rijnbeek's study on the game *Lines*. We showed that the problem of finding the best cut on a graph can be done in polynomial time.

For ropes we mainly focused on developing a framework for the set of problem. A start was made on the problem where there are no restrictions on the endpoints of the rope, and showed that this variant is probably the least interesting. However, the problem was not studied extensively and needs further research.

Further research is required to find an efficient algorithm for solving ropes and to find a sub-quadratic algorithm for cuts on arbitrary graphs. Section 6 outlines the set of problems for future research.

References

On graph games

[1] H.L. Bodlaender and D. Kratsch, "Kayles and Nimbers", *Journal of Algorithms 43*, pages 106-119, 2002.

[2] H.L. Bodlaender, "On the complexity of some colouring games", *R. Möhring, (Ed.), Graph Theoretical Concepts in Computer Science, vol. 484, Lecture Notes in Computer Science*, Springer, Berlin, pp. 30–40, 1991.

[3] A. Lee and I. Streinu, "Pebble game algorithms and sparse graphs", *Discrete Mathematics 308*, pages 1425-1437, 2008.

[4] J.-W. Hong and H.T. Kung, "I/O Complexity: The red-blue Pebble game", *Proc. ACM Symposium on Theory of Computing*, pages 326-333, 1981.

[5] J.F. Hopcroft, P. Wolfgang and L.G. Valiant, "On Time versus Space", *Journal of the ACM 24*, pages 332-337, 1977.

[6] H.A. kierstead and W.T. Trotter, "Planar Graph colouring with an Uncooperative Partner", *Journal of Graph Theory 18*, no. 6, pages 569 - 584, 1994.

[7] E. Friedgut, Y. Kohayakawa, V. Rödl, A. Rucński and P. Tetali, "Ramsey games against a one-armed bandit", *Combinatorics, Probability and Computing 12*, pages 515-545, 2003.

[8] J. Beck, "Ramsey games", *Discrete Mathmatics 249*, pages 3-30, 2002.

[9] G.D. Prichett, "The Game of Sprouts", *The Two-Year College Mathematics Journal 7* no 4, pages 21-25, 1976.

[10] G. Cairns and K. Chartarrayawadee, "Brussels Sprouts and Cloves", *Mathematics Magazine 80*, no 1, pages 46-58, 2007.

[11] R. Nowakowski and P. Winkler, "Vertex to Vertex pursuit in a graph", *Discrete Mathematics 43*, pages 235-239, 1983.

[12] R.M. Wilsox, "Graph Puzzles, Homotopy, and the Alternating Group", *Journal of Combinatorial Theory 16*, pages 86-96, 1974.

On Voronoi games

[13] H.-K. Ahn, S.-W. Cheng, O. Cheong, M. Golin, R. van Oostrum, "Competitive facility location: the Voronoi game", *Theoretical Computer Science 310* pages 457-467, 2004.

[14] R. Hosseini, M. Khosravian, M. Davoodi and B.S. Bigham, "One Round Voronoi Game on Grid", *EuroCG*, 2016.

[15] O. Cheong, S. Har-Peled, N. Linial and J. Matousek, "The One-Round Voronoi Game", *Discrete and Computational Geometry 31*, no 1, pages 125-138, 2014.

[16] S. Bandyapadhyay, A. Banik, S. Das and H. Sarkar, "Voronoi Game on Graphs", *Theoretical Computer Science 562*, pages 270-282, 2015.

[17] 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*, no 4, pages 655-669, 2013.

On graphs / polygons

[18] J. Luo, and C. Wulff-Nilsen, "Computing Best and Worst Shortcuts of Graphs Embedded in Metric Spaces", *Algorithms and Computation*, pages 764-775, 2008.

[19] O. Cheong, A. Efrat and S. Har-Peled, "Finding a Guard that Sees Most and a Shop that Sells Most", *Discrete and Computational Geometry 37*, pages 545-563, 2007.

[20] M. Farshi, P. Giannopoulos and J. Gudmundsson, "Finding the Best Shortcut in a Geometric Network", *Proc. 21th Symposium on Computational Geometry*, pages 327-335, 2005.

[21] C. Gutwenger, P. Mutzel, and R. Weiskircher, "Inserting an Edge into a Planar Graph", *Algorithmica 41*, pages 289-308, 2005.

[22] 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), pages 965-994, 2008.

On Lines - gamious

[23] T. Rijnbeek, "Continuous Voronoi Games on Graphs with Multiple Opponents", *Department of Information and Computing Science, Utrecht University*, 2015.

[24] J. van Dieren, "Small project", *Department of Information and Computing Science, Utrecht University* 2015.