Thesis - Events changing a kinetic Voronoi diagram within a simple non-convex polygon

Joes de Jonge (6228305)

June 2, 2024

Abstract

This thesis analyses the number of changes of a Voronoi diagram within a simple non-convex polygon with kinetic sites. Korman et al.[17] introduced an algorithm to keep track of such a Voronoi diagram. They showed this algorithm has $O(n^3 s^3 \beta_z(n))$ change events. This thesis tries to see what the results are with explicit instances. A simplified version of Korman et al.[17] has been created to get these results, which uses extended shortest path maps and a kinetic data structure. From the results, one can make the conjecture that vertex events that are part of a Voronoi diagram become less present when there are more vertices and sites in an instance. One can see that the vertex events that are not necessarily part of a Voronoi diagram are presumably linear in the number of sites when the number of vertices is fixed for specific types of polygons.

Contents

1 Introduction

1.1 Problem statement

When a shop gets robbed by a robber, you want to know which police agent is closest to the robber such that the robber can be arrested as soon as possible and the stolen goods can be returned to the shopkeeper. This problem can be solved by going through every police agent and see who is closer to the robber. The downside of this approach is that everyone moves in the time you are determining who is closest, we need a quick response to the robbery. To make this problem even more difficult, you cannot just take the Euclidean distance between each police agent and the robber. This all takes place in a city and a city is bounded by streets and buildings, making it impossible to use the straight line between a police agent and the robber as the shortest distance between these two. It is more beneficial to divide the space of the city into sections such that each section has a police agent associated with it that is the closest to all of the points that fall into that section. This can be seen in Figure 1. In this way, you know directly which police agent is the closest to the robber and the robber can be caught as soon as possible.

This structure is called a Voronoi diagram and the points of interest are called sites. In the robber problem, the police agents would have been called sites. A lot of research has been done on the topic of Voronoi diagrams. However, little research has been done on Voronoi diagrams in a bounded space with moving sites. In literature, sites that can move are called kinetic sites and sites that cannot move are called static sites. Korman et al.[17] are the only ones who have focused on this specific problem. They introduced an algorithm that can handle a Voronoi diagram with kinetic sites that are bounded by a simple polygon. With this algorithm you can have $O(n^3 s^3 \beta_z(n))$ events that change the combinatorial structure of the Voronoi diagram and each event can be handled in $O(k(\log\, n + \log^2 s))$ time. These bounds are extensive. It is of interest to see if these bounds are much lower in practice. There is the possibility that there is an algorithm that is faster in general or on specific types of instances. But first, it would be helpful to know if we need to

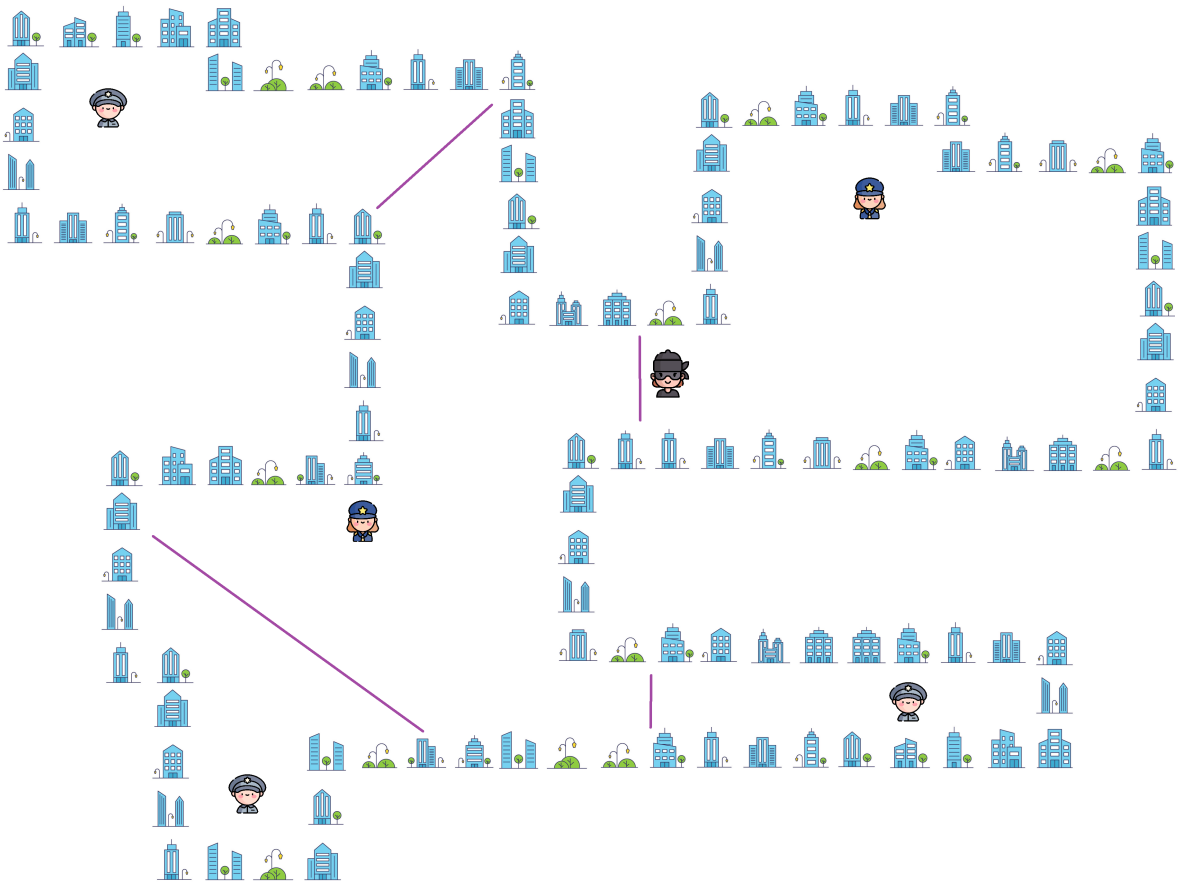


Figure 1: Rough example of the robber problem where the city has been divided into section with associated a police agent. In the middle of the figure, you can see the robber.

make the effort to find such an algorithm. Therefore, this thesis focuses on the number of changes in the Voronoi diagram of kinetic sites bounded by a simple polygon. We try to determine if on average this bound is much lower than Korman et al. stated and we will look at different types of polygons.
The research question for this thesis is:

- How many events that change the combinatorial structure of the Voronoi diagram will occur while maintaining the Voronoi diagram inside a simple polygon with kinetic sites, with explicit instances?

To answer this research question we are going to look at the next several sub-questions:

1. How many events will occur with explicit instances of certain polygon groups?
2. Does the star-shaped polygon group have $O(p * f(n, s))$ events, for p the number of legs on the star-shaped polygon and for some function f dependent on the number of polygon vertices, n, and the number of sites, s?
3. Do all the polygons have $O(ns^2)$ events, for n polygon vertices and s sites?
4. How many bisector events will occur?
5. How many Voronoi centre events will occur?
6. Do bisector events or Voronoi events occur more often?

For question 2, we assume it will have a bound of $O(p * f(n, s))$ events, because star-shaped polygons are constant in the change of internal and external angles of successive vertices. For question 3, I assume it will have $O(ns^2)$ events, because the average over all polygons can be much lower than the bound stated by Korman et al[17]. Finally, we assume the number of events depends more on the number of sites than on the number of vertices, because these create the changes. To answer questions 4, 5 and 6, we will focus on the same assumed bounds as stated with subquestions 2 and 3, but then focused on the bisector events and Voronoi centre events separately.
A group of polygons follows certain rules. There are many possible groups to create, but the ones we will look at are:

- Star-shaped polygons,
- Orthogonal polygons,
- Fully random polygons.

The fully random polygons are polygons that are simple but their points are chosen uniformly and the order of points on the border line is also chosen uniformly. While determining the order of points on the border it has to be taken into account that the polygon needs to be simple.

1.2 Prerequisites

1.2.1 Voronoi diagram

The sections of a Voronoi diagram are called Voronoi regions. To make things more precise, we shall introduce some notation. The Voronoi diagram lies within a space P with a distance function d. A Voronoi diagram has sites. These sites are the points of interest, in the robber problem these are the police agents. The sites are coming from a collection $S \subset P$. The Voronoi region can be depicted as $V(s, P)$, where $s \in S$ is a site and the Voronoi region is part of space P, thus $V(s, P) \subset P$. In some cases, you can drop the P specification when it is clear that you are working in space P. This cannot be dropped when you are talking about sub-Voronoi diagrams which use subspaces. For each point $x \in V(s, P)$ it holds that $\forall v \in S|\ \mathrm{d}(x, s) \leq \mathrm{d}(x, v)$. Where $\mathrm{d}(y, z)$ is the shortest distance between points y and z in space P. Besides sites, a Voronoi diagram also consists of Voronoi edges and Voronoi vertices. Voronoi edges are determined by sections of the bisectors of two points. Exactly at those places you have an equal distance between the two closest sites. Voronoi vertices are determined by more than two sites, from such a vertex you can draw a circle, by means of the distance function d, going through all the sites that are all closest to that vertex and which are all equidistant to that vertex.
An example of a Voronoi diagram in the open space can be seen in Figure 2 and an example of a Voronoi diagram in a space bounded by a polygon can be seen in Figure 3. Do mind that the latter is not completely accurate, due to the difficulty of drawing a Voronoi diagram in a space bounded by a polygon.

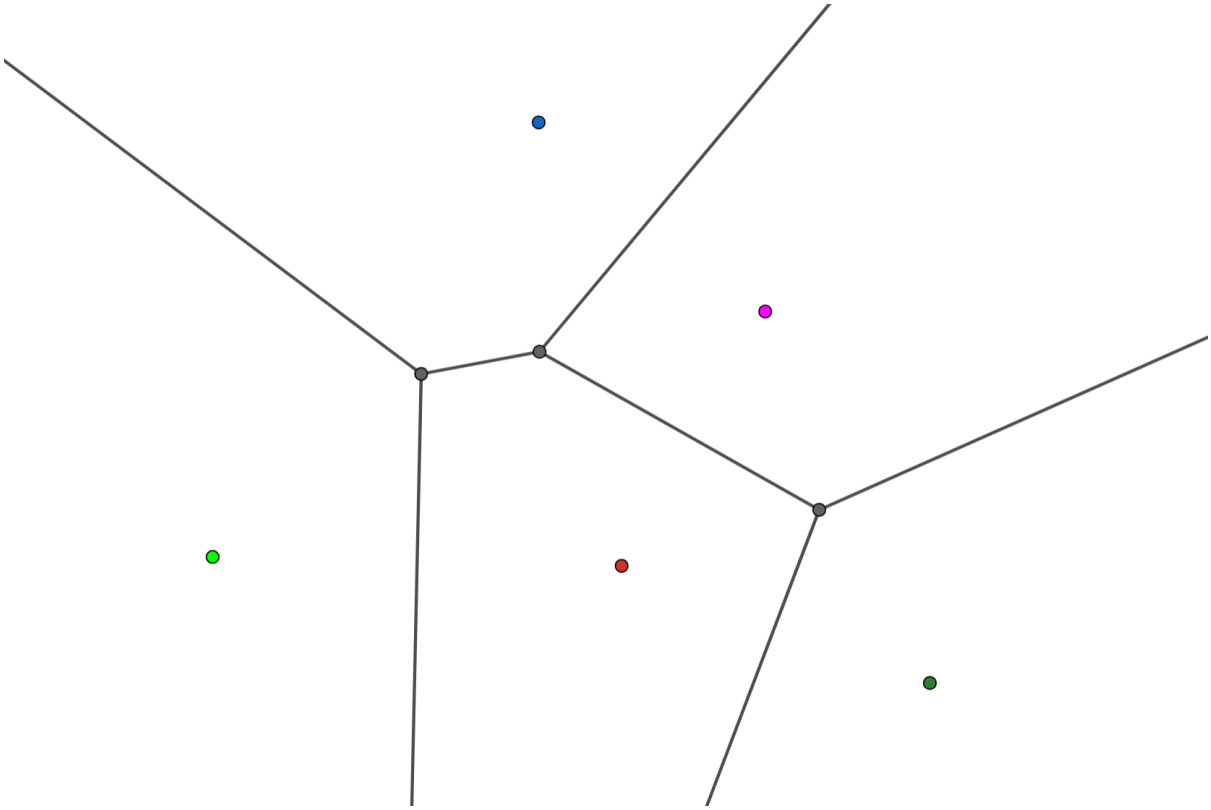


Figure 2: A part of a Voronoi diagram in the open space.

1.2.2 Kinetic data structure

When you make it possible that the sites can move in space P, you have to use a kinetic data structure (KDS). A kinetic data structure has the structure of the data and a typical way of updating the structure. Although all objects move continuously, there are only certain moments in time when the combinatorial structure of the elements of the data structure changes. A KDS contains certificates that create a proof of the properties of interest, with a Voronoi diagram this could be the place of a Voronoi vertex or this could be that the straight line between two points within an SPM is within the polygon. Most of the time, certificates are simple inequalities that assert facts like "point c is on the left of the directed line through points a and b". These certificates are inserted in an event queue based on their time of expiration. For a Voronoi diagram, this could mean when a certain Voronoi vertex disappears. This happens when only two sites become the closest two points of a certain area of the polygon.
The event queue of the KDS creates a simulation. Per event, the KDS updates the structure such that all certificates are true once again. Afterward, it determines new events when some certificates are not true anymore[14]. Basch et al.[9] showed a way to create a KDS structure for the upper envelope of multiple lines. The algorithm does this by using a balanced binary tree. They also showed a way to create a KDS structure for the closest pair problem in two dimensions. This algorithm uses a plane sweep algorithm. This also uses a binary tree when making it applicable to a kinetic situation.

1.2.3 Shortest path map

When the space is bounded, by for example a polygon, the notion of shortest distance changes. In the open plane, the shortest distance is just a straight line. But within a polygon, this straight line can now be obstructed by the border of the polygon. Now, the shortest path is determined by the vertices and edges of the polygon. The path consists of a list of straight segments. There is an order of the vertices you have to pass before the final location is reachable with a straight line. For example, see Figure 4. To make a Voronoi diagram bounded by a polygon, it is convenient to use a shortest path map (SPM) per site to determine the Voronoi edges and vertices. An SPM is a tree structure that is defined per site s and can be denoted as SPM_s. SPM_s has site s as root and the vertices of the polygon are the nodes of the tree structure. An example of an SPM can be seen in Figure 5. SPM_s shows the shortest route from a vertex v towards s. To determine this route,

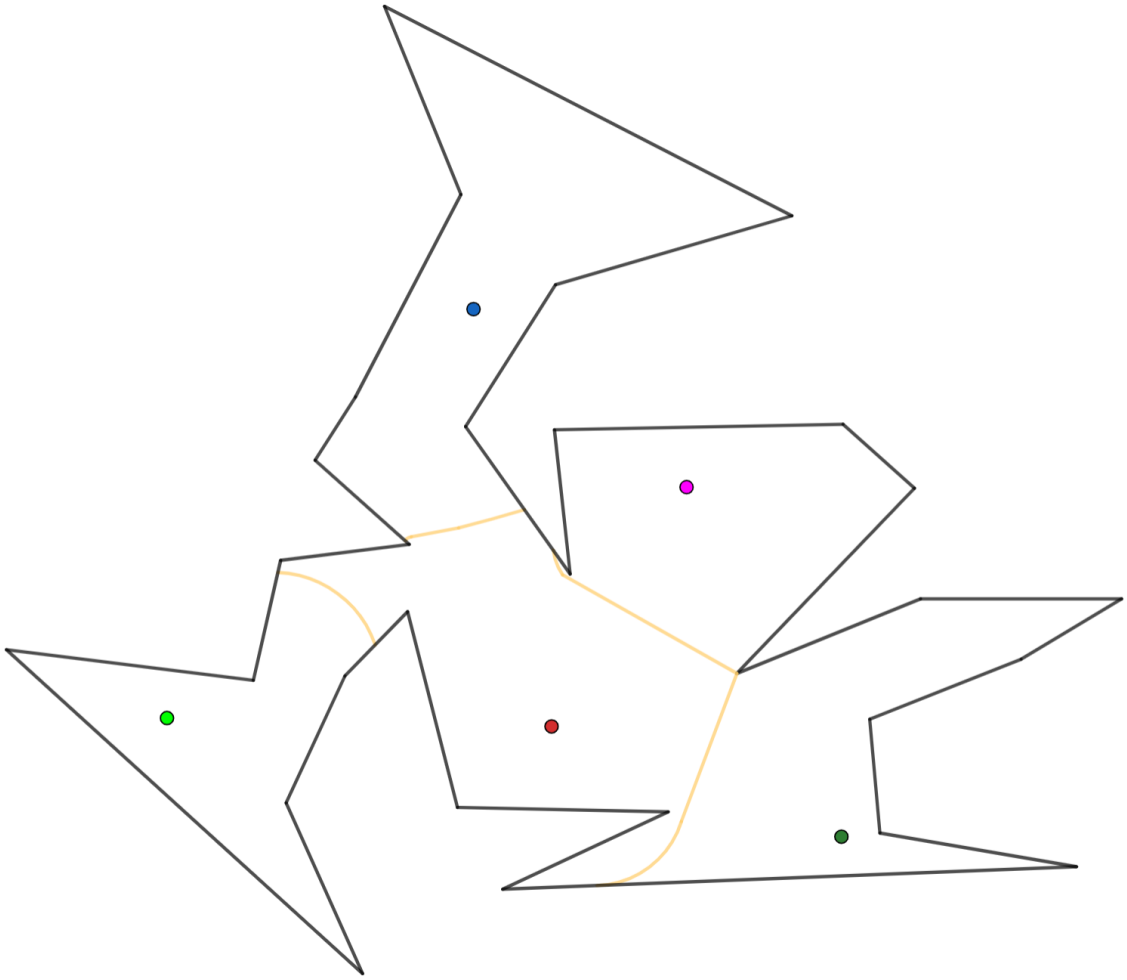


Figure 3: An approximation example of a Voronoi diagram in a space bounded by a polygon.

you start at the node of vertex v and continuously take the parent of the current node until you are at the root. SPM_s can also have more points besides the vertices of the polygon when you are also interested in the shortest path between s and other points than the vertices. You can extend SPM_s to divide the polygon into specific triangles such that it is easier to determine for a random point in the polygon what its shortest path towards s is. But also to determine when the shortest path of a kinetic point different than s combinatorially moves within the SPM_s. This can be done by introducing extension segments and extension segment points (ESP). An extension segment of vertex v is a segment drawn from parent u of v within SPM_s towards v that stops at the first intersection point with the border of the polygon that is not on the subsection from u towards v. This extension segment is denoted by $e_{vs}(t)$, the extension segment of v in SPM_s at time t. This intersection point that is the endpoint of the extension segment is the ESP of v in SPM_s. This can be donated as $\text{ESP}_{vs}(t)$, the extension segment point of v in SPM_s at time t. Only vertices that have children within SPM_s have an extension segment. The extension segment would otherwise directly enter the border of the polygon after v. The extension segments determine what area is visible from a certain vertex within a specific SPM. The extension segments determine the combinatorial order of the areas within the polygon.

Because of the aforementioned change in the definition of shortest distance, bisectors also change. A bisector in a polygon is not necessarily a straight line anymore. A bisector consists of multiple segments, where each segment is a hyperbolic arc. This also means that a Voronoi diagram bounded by a polygon has Voronoi edges that are not necessarily straight lines anymore, because Voronoi edges are parts of bisectors. The hyperbolic segments of a bisector of sites s_1 and s_2 begins and ends at the border of the polygon or the intersection of the hyperbolic arc with an extension segment from SPM_{s_1} or SPM_{s_2}. The intersection point between the bisector and an extension segment or the border is called a bisector point. These bisector points have a degree. The degree of a bisector point refers to the number of hyperbolic arcs that stop at that point. A hyperbolic

arc that ends at the border of the polygon has degree one. A bisector point of degree two is a point where the bisector will continue. A bisector point of a higher degree than two is a point where multiple bisectors come together. The latter would be a Voronoi centre. In the open plane, the Voronoi centre is also called a Voronoi vertex.

1.3 Outline of the thesis

To get statistical data to answer the research question and sub-questions an algorithm has been designed, implemented and run on multiple instances. This algorithm has certain aspects of the algorithm of Korman et al.[17], but it is not a complete implementation. This can be found in section 3. Related work can be found in section 2. There you can also get a better understanding of the specifications of the problem and how to apply certain ideas such as a KDS and an SPM. The findings on the results of the algorithm and the conclusions from this can be read in sections 5 up to and including 7.
For the remainder of this thesis, n is the number of polygon vertices and s is the number of sites.

2 Related work

2.1 Static sites and non-bounded space

One problem concerning Voronoi diagrams is the most standard one: creating a Voronoi diagram in the open plane $\mathbb{R}^2$ that is not bounded by anything and that has static sites. This means there is no border on the space and the sites are not moving. There is an algorithm that can compute the Voronoi diagram in this problem in $O(s \log s)$ time[13]. This algorithm uses a sweep line approach with a beach line. When the sweep line continues moving, the Voronoi diagram above the sweep line can still be affected by sites below the sweep line. The beach line separates the known and unknown parts of the Voronoi diagram. The beach line consists of parabolic bisectors between the sites above the sweep line and the sweep line itself. This algorithm is also optimal, because the Voronoi diagram also contains the convex hull and this can be done optimally in $\Omega(s \log s)$ time.
Even though we have this bound, researchers tried to create faster algorithms to create Voronoi diagrams. Chew[4] introduced an algorithm that can determine the Voronoi diagram in $O(n_1)$ expected time, where n_1 is the number of edges of the convex polygon that can be formed from the sites. The downside of this algorithm is that it does not apply to every set of sites. In that algorithm, the sites should be the vertices of

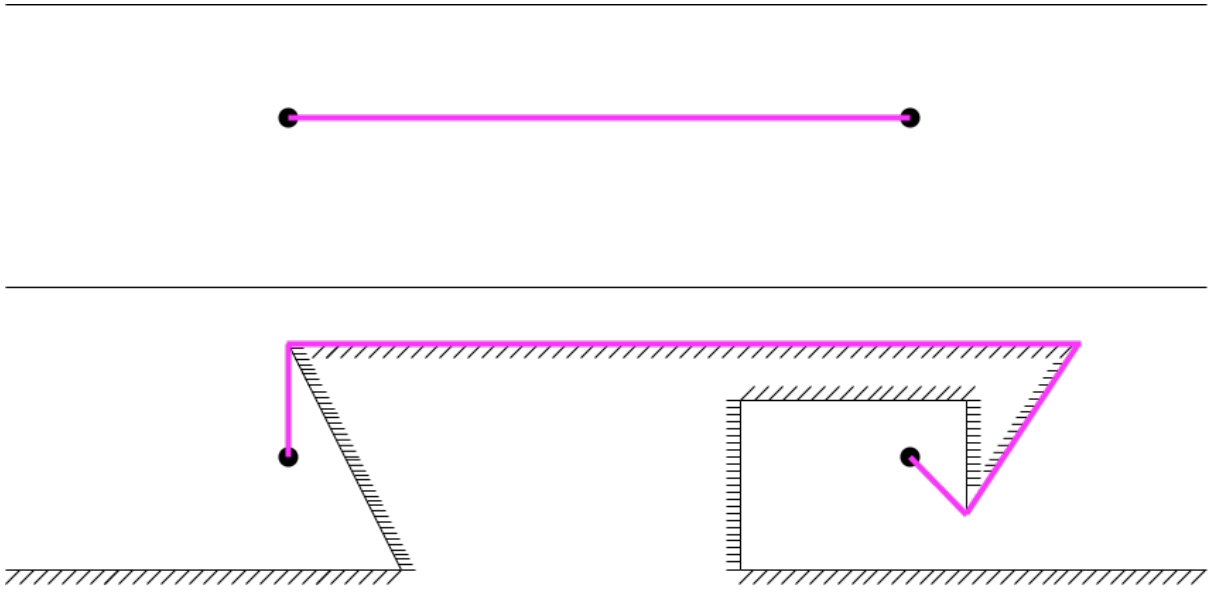
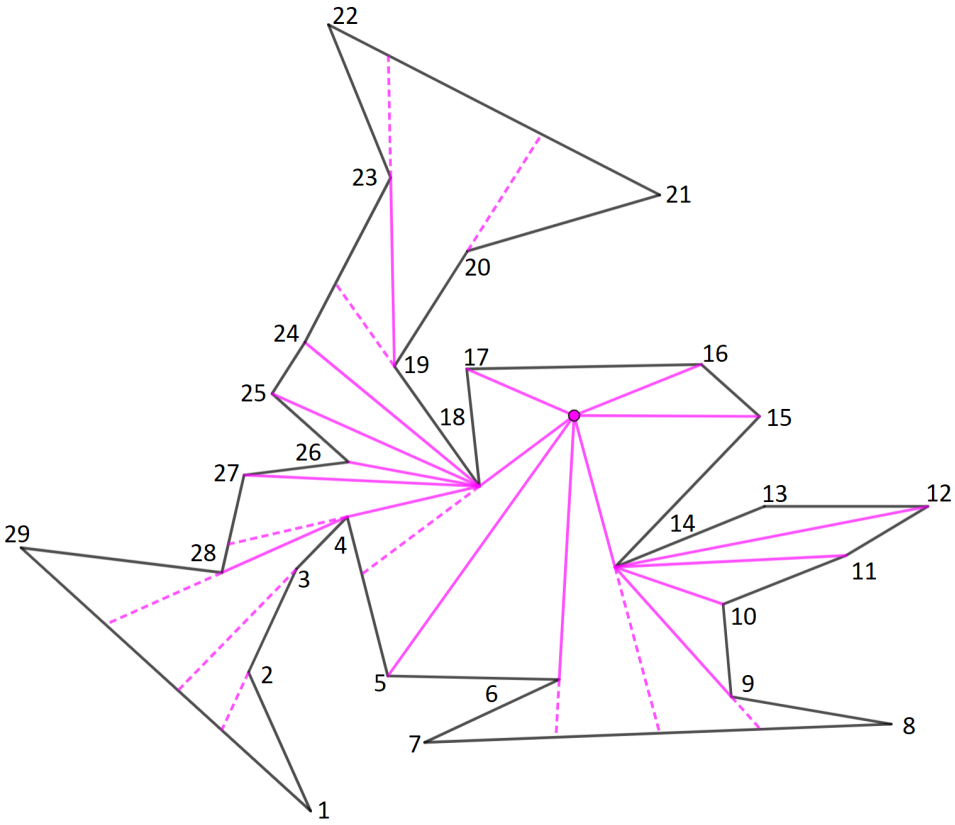
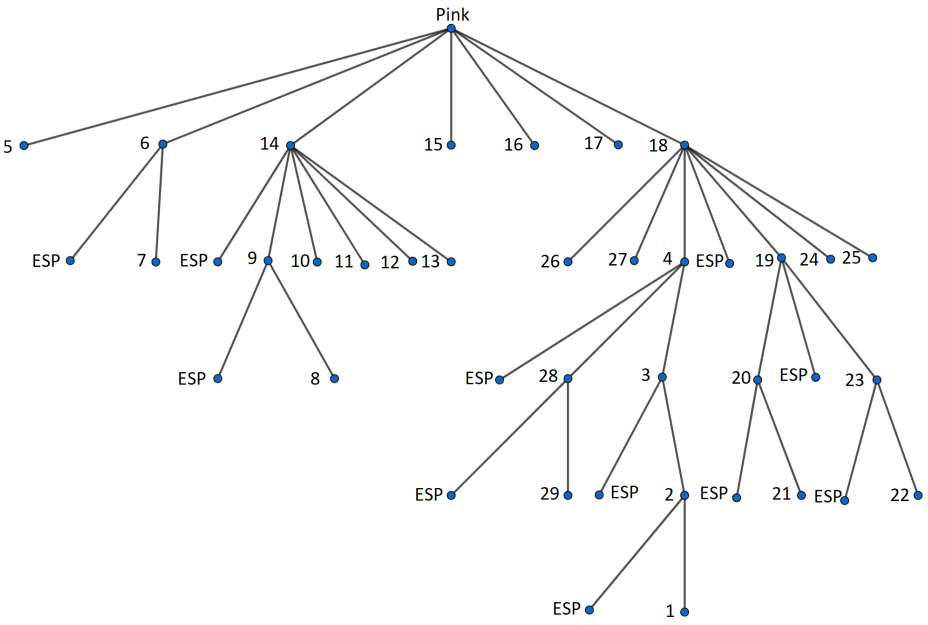


Figure 4: Visualisation of difference of definition shortest path between open plane and plane bounded by a polygon. Upper figure shows the shortest path in the open plane. Lower figure shows the shortest path in a plane bounded by a polygon, where the outside is shown by the dashed lines.



(a) The extended SPM of the pink coloured site displayed within the polygon. Most of the edges of the tree are coloured pink. Some edges are not coloured pink, because they are also edges of the polygon.



(b) The extended SPM of the pink coloured site displayed as a tree, where the numbers refer to the vertices as seen in (a) and ESP stands for extension segment point.

Figure 5: The extended SPM of the pink coloured site displayed within the polygon and displayed as a tree.

a convex polygon. Aggarwal et al.[2] introduced an algorithm that runs in $O(s)$ time. However once again, this algorithm has the constraint that the sites should be the vertices of a convex polygon. To construct the Voronoi diagram, the algorithm splits up the sites into blue and red points. It determines the convex hull of the blue points. Then it calculates a certain subset from the red points, called the crimson points. It does this by looking at the convex hull of the blue points. The crimson points are added to the convex hull of the blue points. Then it computes the convex hull of the remaining red points. Finally, it merges this remaining hull with the expanded blue hull. Other people have also looked at this problem by making it more general what sites can be. Until now, we only looked at sites as points, but these sites can also be line segments or complete polygons. In this situation, the shortest distance between an object and a point is the distance between a point on the border of the object and the point of interest. The border of the object can vary in meaning. For a point, it is the point itself. For a line segment, it is the whole line segment. But for a polygon, it consists of the polygon edges and vertices. Imai[6] focused on the problem of creating a Voronoi diagram with polygons and lines as sites and introduced an algorithm that runs in $O(n_2^2)$ time, where n_2 is the number of open segments in the collection of sites. This algorithm uses the topology of the objects to determine which areas are closer to which objects. To create the Voronoi diagram, it adds each object one by one. Per step, it determines the Voronoi diagram thus far. The downside of this algorithm is that it still can have errors.

2.2 Kinetic sites and non-bounded space

The previously mentioned problem of the Voronoi diagram in the open plane comprises static sites. Even though, there are also applications of the Voronoi diagram that uses kinetic sites. For example, you are playing a game and you want to know which moving character is closest to you. You do not only want to know it at a specific moment, but at any moment in time.
There are multiple ways to solve this. For example, you can determine a lot of information beforehand which can take up a lot of time, but because you did this you can now determine the Voronoi diagram per timestamp rapidly. You can also solve it the other way around, doing nothing beforehand and determining the Voronoi diagram per event. Fu et al.[5] introduced an algorithm that follows the first approach. This algorithm can work in $O(s)$ time to calculate the Voronoi diagram at a certain time while having a preprocessing time of $O(k^2 s^3 2^{O(\alpha(s)^{5k+1})})$, where k is the polynomial degree of the polynomial paths the sites take and $\alpha(s)$ is the functional inverse of Ackermann's function. This is done by determining which bisectors are partially part of the Voronoi diagram for which time ranges. Then the algorithm has per bisector time intervals when the bisector is part of the Voronoi diagram. The algorithm can then determine the Voronoi diagram on a certain timestamp by determining which bisectors are part of the Voronoi diagram and then it can construct the Delaunay triangulation. The best algorithm that works with preprocessing is the one introduced by Song et al.[18], that algorithm works in $O(k + s)$ time to create the Voronoi diagram at a certain timestamp with a preprocessing time of $O(s \log s)$ time, where k represents the number of events that will happen during the specific time the algorithm runs. That algorithm works on disk-shaped sites. Albers et al.[8] introduced an algorithm for the version of the problem where we have kinetic point sites and a non-bounded space, this algorithm does not have a preprocessing step. That algorithm works per event in $O(\log s)$ time with $O(s^d \lambda_p(s))$ events, where d is the dimension of the space and $\lambda_p(s)$ the maximum length of a (s, p)-Davenport-Schinzel sequence. The algorithm uses the structure of the Delaunay triangulation and its events to determine the Voronoi diagram. First, it determines the structure of the Delaunay triangulation at time 0. Then per tuple of $d + 1$ points, it calculates the potential events that could change the Delaunay triangulation structure. With all the potential events it creates a priority queue. Per event, it determines if the event is a transition of edges in the triangulation or a recalculation of the local topology, then it processes the event and updates the event queue. Other researchers focused on a similar problem, but instead of moving site points it has moving site polygons. Guibas et al.[11] introduced an algorithm that tackles this problem. They are also the only ones who tackled this version of the problem. The introduced algorithm of that research can handle an event in $O(\log s)$ time and it changes combinatorially in $O(kl^2\beta(k)\beta(s))$ time, where k is the number of polygons, l the number of polygon vertices and $\beta(x) = \lambda(x)/x$ and it is an extremely slowly growing function. The algorithm does not use the normal Voronoi diagram but a compact version of it. The algorithm uses a KDS with certificates represented by the junction triangles of the triangulation of the polygon and the non-polygonal convex hull edges. Junction triangles are triangles that are incident to three objects in the same space. This is used, because a Voronoi diagram remains a Voronoi diagram if the Delaunay triangulation remains a Delaunay triangulation. You can know that for certain when looking at the junction triangles and the non-polygonal convex hull edges.

2.3 Static sites and bounded space

The original problem focused on creating Voronoi diagrams in the open plane with static sites. Another take on this problem is also interesting. Namely, one can wonder how to create a Voronoi diagram bounded by a polygon P with static sites. For example, when you have multiple hospitals in a city and you want to know the hospital closest to you, you could use the original problem. But every city has streets and buildings that create boundaries. The streets and buildings create a polygon. This polygon should be simple, to make the problem easier.
Many researchers have also looked into this new problem. Aronov et al.[3] introduced an algorithm that runs in $O((n+s)\ \log(n+s)\ \log(n))$ time. That algorithm uses two phases. The first step triangulates the polygon and determines which sites are in which triangle. Finally, it will create a balanced decomposition of the triangulation. This is a tree structure of the triangulation, such that when you remove the root of the tree, or the sub-tree, you end up with two parts that have each a size that is a fixed fraction α of the size of the tree being split. The second step is a recursive step where it splits the polygon into two equally sized polygons, it determines the Voronoi diagrams of these polygons separately, and then combines the two Voronoi diagrams. The algorithm stops going further into the polygon when the polygon is a triangle or when there is only one site left in the polygon. Oh[15] introduced an algorithm that runs in $O(n+s\ \log\ s)$ time. This algorithm is the fastest at this moment. This algorithm works by first creating a triangulation of the polygon and determining which sites lie in which triangle. Then it creates a dual tree from the triangulation. After which it goes through the tree in post-traversal and pre-traversal. It combines the sub-Voronoi diagrams calculated from previous steps, to create new sub-Voronoi diagrams. After both traversals, it combines the two diagrams to create the final Voronoi diagram. In the previously mentioned solutions, the polygon could only be simple, but Mayya et al.[7] introduced an algorithm that works with non-simple polygons. That algorithm works in $O(s^2)$ time. Hershberger and Suri[10] introduced a way of determining the shortest path with polygons, such that you can determine the Voronoi diagram in a polygon with holes in $O((n+s)\ \log\ (n+s))$ time.

2.4 Kinetic sites and bounded space

The previously stated problem of creating a Voronoi diagram bounded by a polygon focuses on static sites, but this problem can also be changed such that it uses kinetic sites. This new problem is a problem that comes very close to the real world and is the problem stated in the introduction of this thesis.
Korman et al.[17] are the only ones who focused themselves on this problem. They introduced an algorithm that can handle a Voronoi diagram within a polygon with moving sites. With this algorithm, there are at most $O(n^3s^3\beta_z(n))$ events and each event can be handled in $O(k(\log\ n+\log^2 s))$ time, where $\beta_z(n)=\lambda_z(n)/n$ and $\lambda_z(n)$ is the maximum length of a Davenport-Schinzel sequence of n symbols of order z.The algorithm uses a KDS for bisectors and a second KDS for Voronoi centres. These two types of KDSs are necessary, because a change in the bisector or the Voronoi centre determines a change in the structure of the Voronoi diagram. The bisector KDS handles an event in $O(\log\ s)$ time and the centre KDS handles an event $O(\log^2 s)$ time. The algorithm of Korman et al.[17] uses extended SPMs to determine these changes quickly.
Korman et al.[17] stated that the Voronoi diagram changes during the next following events:

- Vertex events, at which a degree 1 bisector point or a Voronoi centre coincides with a vertex,
- 1.2-collapse/expand events, at which a degree 2 bisector point disappears by colliding with a degree 1 bisector point, or vice-versa,
- 2,2-collapse/expand events at which a degree 2 bisector point disappears by colliding with another degree 2 bisector point, or vice-versa,
- 1,3-collapse/expand events at which a Voronoi centre disappears by colliding with a degree 1 bisector point, or vice-versa,
- 2,3-collapse/expand events at which a Voronoi centre disappears by colliding with a degree 2 bisector point, or vice-versa, and
- 3,3-collapse/expand events at which a Voronoi centre disappears by colliding with another Voronoi centre, or vice versa.

All these events happen when specific distances for two or three sites towards specific points are the same. In the following situations, we talk about sites p, q and r and without loss of generality focus on the site p.

- Vertex events happen when two or three sites are equidistant to one vertex,

- 1,2-events happen when two sites are equidistant to one specific ESP_{vp},
- 2,2 events happen when two sites are equidistant to one specific degree 2 bisector point coming from SPM_p,
- 1,3-events happen when three sites are equidistant to one specific ESP_{vp}, and
- 2,3-events and 3,3-events happen when three sites are equidistant to one specific degree 3 bisector point coming from SPM_p.

This thesis focuses on the aforementioned bound on the number of change events of the Voronoi diagram. We try to determine if on average this bound is much lower than Korman et al. stated and we will look at different types of polygons.

3 Method

3.1 Algorithmic approach

3.1.1 Overview

To answer the research question, we can implement the whole algorithm stated by Korman et al.[17], but that gives us too much information. Because we want to know when the Voronoi diagram changes and not necessarily what it will look like afterward. We will look at a simplified version of that algorithm. Our simplified version of the algorithm only determines when the aforementioned change events of Korman et al.[17] happen. That will be done by determining when the aforementioned distances of Korman et al.[17] are the same and then checks if these events are also part of the Voronoi Diagram. In this way, less machinery is needed than in the original algorithm.
The steps in the algorithm are:

1. Determine the SPMs of all sites. Each SPM contains the site, the vertices, the ESPs of its own site, the ESPs of all the other sites, the degree 2 bisector points of all the bisectors and the Voronoi centres of all the bisectors.
2. Create a queue of events that can potentially change the structure of each SPM.
3. Per potential change event:
   (a) Check which equidistant events occurred during the last potential change event and the current potential change event and check if these equidistant events are part of the Voronoi diagram.
   (b) Update the SPMs such that all the information is correct once again.
   (c) Determine new potential change events with the updated SPMs.

3.1.2 Determining the SPMs

To determine the SPMs of all sites, we used the algorithm described by Guibas et al.[1]. This algorithm uses the triangulation of the polygon. First, the triangle of the triangulation where the site is, has to be found. This found triangle needs to be split up into multiple sub-triangles such that there is a tree structure of triangles where the site is the root of the tree. These sub-triangles can be created by making extra edges, starting from the site to the vertices of the found triangle. When the site is on one of the vertices of the found triangle, there is no need to split up the found triangle. When the site is on one of the edges of the found triangle, the found triangle has to be split up into only two sub-triangles. When the site is not on the border of the triangle, the found triangle needs to be split up into three sub-triangles. In this way, a tree structure of triangles will be created. The only way the triangles will not create a tree structure, is when there is a hole in the polygon. But the polygons we use are simple. This is true, because a triangulation of a polygon only uses the vertices of the polygon itself. Suppose there is a simple polygon, and the algorithm uses this way to create a connecting structure of the triangles, and it ends up with a structure that is not a tree, it means it has created an extra vertex in the triangulation that was not part of the polygon.
With the created tree structure of the triangles of the polygon, the algorithm of Guibas et al.[1] can be continued. This algorithm must be applied on every sub-triangle the site is part of. The algorithm will maintain a funnel. This is a set of points that represents a part of the shortest paths of two vertices that both lie on the

same diagonal of a triangle of the triangulation of the polygon. When comparing the two shortest paths with each other, there is a part on both paths that is not the same and there is a part on both paths that is the same. The first vertex on the shortest paths, seen from the vertices toward the site, from which the shortest paths are the same is called the apex. A funnel consists of the two vertices that are part of the same diagonal that is part of the current lowest triangle in the tree structure, the vertices that are not shared between the two shortest paths and both form two connected paths going from one of the two diagonal vertices towards the apex, and the apex itself. For all vertices in the funnel, that are not the apex, it holds that the next vertex of their shortest path towards the site is within the funnel. The funnel also has to keep track of the order how these other vertices appear on the legs of the funnel, such that you have the complete shortest paths of the two diagonal vertices, for the current location within the tree. An example of a funnel can be found in Figure 6.
In depth-first order, the algorithm introduces a new triangle of the triangulation that is attached to the current triangle. With this new triangle the funnel has to be changed, because the introduction of a new triangle also means the introduction of a new point x. x has to be put somewhere into the funnel to determine the shortest path from x toward the site. In this way, two new funnels can be created to go deeper into the tree. Because the algorithm keeps track of the funnel of the current triangle, it knows for certain that the beginning part of the shortest path for x toward the site is within the collected vertices in the current funnel. The remaining part of the shortest path of x is the shortest path from the apex toward the site. This is also true for all points of the funnel. To determine the first vertex on the shortest path from x toward the site, the algorithm will go through all the vertices of the funnel, which are stored in a list. This is different from what Guibas et al.[1] suggested, they suggested the use of a finger tree. However, going through a list in order of the list is easier to implement. When the algorithm has determined which of the vertices in the funnel is the first vertex in the shortest path from x toward the site, this information can be stored. Then the algorithm goes further into the tree. There are two edges of the triangle the algorithm has not further discovered yet, the third edge is part of the previous triangle. The algorithm goes into both new triangles separately with a different funnel in both cases. The funnels need to be different, because x becomes one of the diagonal points and thus the other vertices in the funnel have to keep track of the shortest paths of both diagonal vertices. When there is no

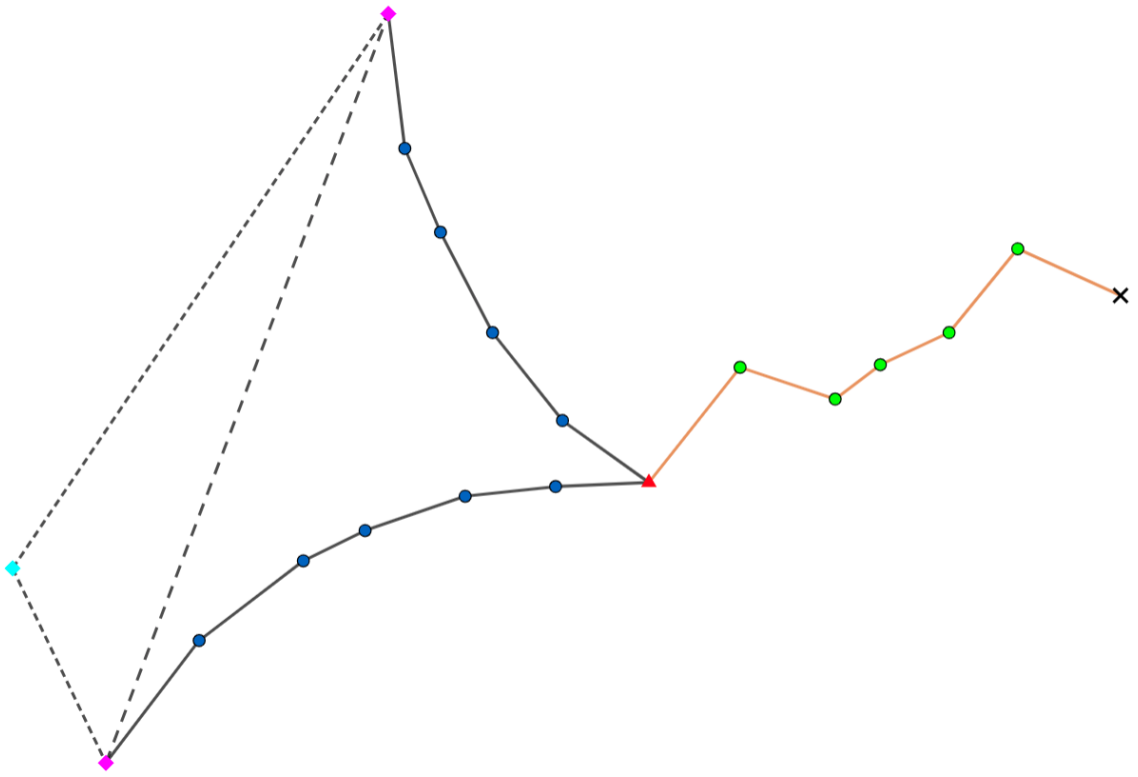


Figure 6: Funnel consisting of two shortest paths, coming from both pink diamond vertices towards the black cross site. The parts between the two shortest paths that is shared has orange sections and green vertices. The apex is coloured red. In a new step of the SPM algorithm explained in section 3.1.2, a new vertex has been introduced. This vertex is light blue. The algorithm has to find the shortest path towards the apex.

other triangle on a new diagonal between x and one of the vertices of the current diagonal, the algorithm has reached one of the leaves of the triangle tree.
For our approach of the algorithm of Korman et al.[17], we need more information than only the shortest path from the sites toward the vertices. We also need information on the ESPs, the degree two bisector points and Voronoi centres. This can be achieved during the process of creating the SPMs inserting these points. The ESPs can be added when the algorithm gets to the end of a leaf in the triangle tree with a funnel. This is because an ESP should be at an edge and should come from one of the vertices of a funnel, by definition of the funnel and ESP. If a vertex in the funnel does not have its ESP on this edge of the leaf in the triangle tree, then this vertex should not be part of this funnel. This vertex is then obstructed by another vertex and that cannot be true within a funnel. The obstructing vertex should have been an apex at some point in the program. Therefore when the algorithm gets to the end of a leg of the tree, it knows that every vertex of that funnel has its ESP on that current diagonal.
Determining the SPM of a site takes O(n^2) time. The program goes through the whole tree once. Per newly introduced vertex, it takes O(n) time. This linear time comes from the fact that it goes through the whole funnel every time to find the point that is part of the shortest path from the newly introduced point toward the site. When the program gets to a leaf in the tree, it takes O(n) time to find all the ESPs of the whole funnel. This means that creating all of the SPMs takes O(sn^2) time.

3.1.3 Running the simulation

In the second part of the simplified version of the algorithm of Korman et al.[17], a simulation will be run. During the process, the algorithm keeps track of the SPMs off all sites, such that at every moment we know exactly what the SPMs are combinatorial. Furthermore, the program keeps track of an event queue. This event queue says when the SPMs need to be updated. Finally, the program keeps track of the results. These results are the information about every equidistant moment of the sites towards points of interest as explained earlier. But these results also contain when every change event of the SPMs happened.
Aronov et al.[12] have stated when the basic structure of an SPM with only a site and vertices can change with a kinetic site. One change event is when a site is colinear with two children of its SPM. But more specifically, these two children have to be direct neighbours of each other in the ordered circle around the site. This circle is an ordering of the children of the site using their angle towards the site as an ordering measurement. Another change event is when a site is colinear with a child of its SPM and with the principal child of that child. A principal child w is the child of a vertex v such that it has the smallest absolute angles over all children of v between the vectors $\overrightarrow{vw}$ and $\overrightarrow{uv}$, where u is the parent of v in the SPM.
However, this only updates the vertices in the SPM. In the extended SPM, the ESPs, degree 2 bisectors points and Voronoi centres have to be updated too. Because these points can also move. This can be done by determining when these points collide with other points or when they are colinear with other points. For the ESPs, the place of an ESP within an SPM only changes when it collides with vertices or with ESPs of its own SPM or other SPMs. These collisions can only happen with vertices or other ESPs, because an ESP is always on the border of the polygon. An example of the collision of two ESPs of different SPMs can be seen in Figure 7. During a change event when the site is colinear with two children or with a child and its principal child, the algorithm also has to update the status of the ESPs that are part of the two vertices. Therefore, the program does not have to determine the collision of ESPs with vertices nor with ESPs of the same SPM. It should only determine the collision of ESPs with other ESPs of other SPMs. When ESP_{vs} collides with vertex w, site s is colinear with vertices v and w. That event has already been determined with the aforementioned ideas of Aronov et al.[12]. This is also true when looking at an ESP colliding with another ESP of the same SPM. This last situation can be seen in Figure 8.
To start this process, the first potential change events will be determined. This will be done for each first segment of each site in their movement path. When a site gets to the end of the segment of its current movement path, the potential change events during the time on the next segment will be determined. In this way, the queue will not be overfilled with potential change events. Some potential change events cannot be calculated directly even though they happen during the time a site is on the current segment of its movement path. This can happen when the current combinatorial situation of the SPMs makes the event undetectable. But when the SPMs have been updated this event can be detected. For example, this happens when a site is colinear with a child and its principal child, and afterward the child has a new principal child and the algorithm can now determine when the site is colinear with the child and this new principal child. This future colinear event could not be detected before this current colinear event had happened, because the program only tries to find these events between the current situation of a child and its principal child and not of all of its children. The same holds for colinear event of two children of a site. This is only determined between two children that

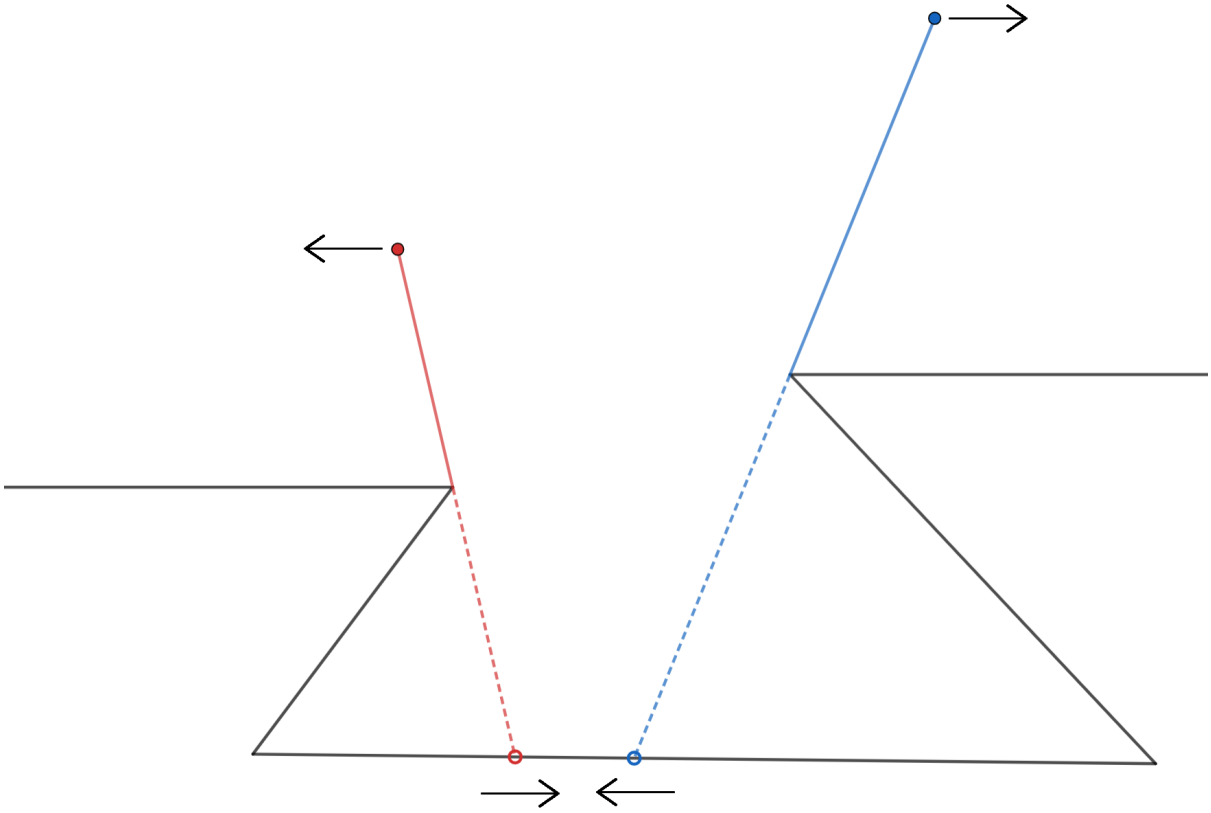


Figure 7: Two sites within a polygon are moving in opposite directions and create a collision of two ESPs.

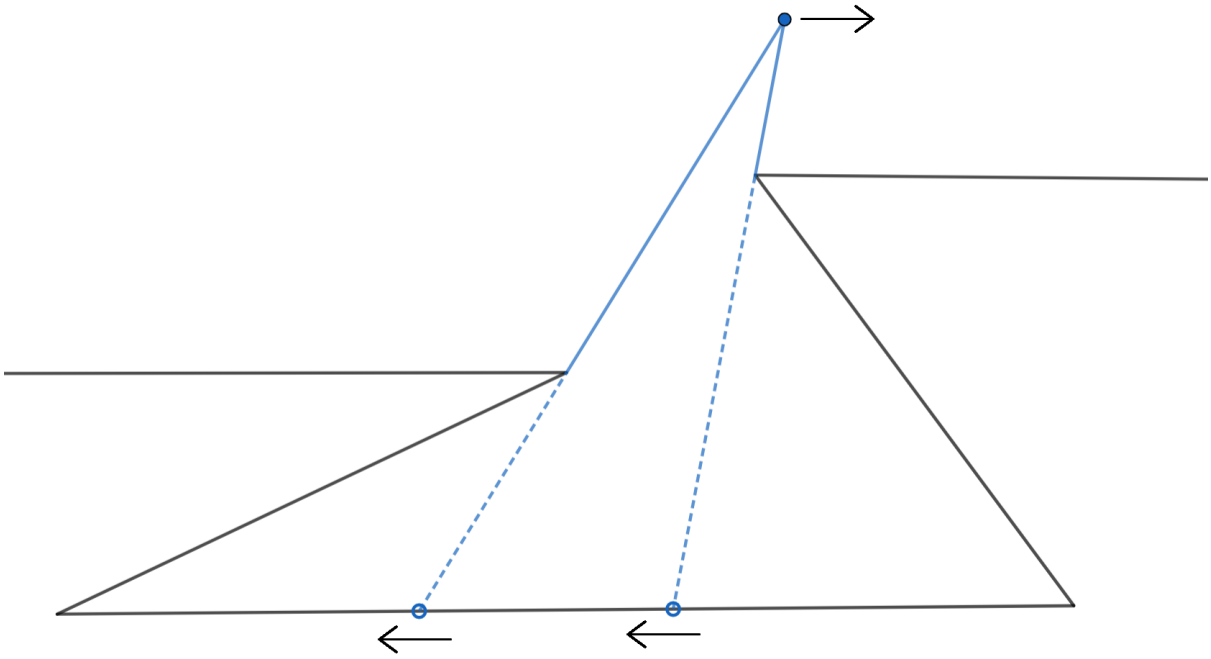


Figure 8: A site within a polygon is moving and creates a collision of two ESPs of its own SPM. When this collision happens the site is also colinear with two children.

are direct neighbours on the angular circle of the site. For all the other types of change events this is also true. These events can also only be determined by the current combinatorial situation of all of the SPMs.
Running the simulation takes O(es^3nr) time, where e is the number of change events during the simulation and r is the maximum length of the range of time that had been used during the root findings to determine the equidistant moments and the collision moments. Per change event, the equidistant moments for all sites and all vertices had to be determined. This takes O(s^3nr) time. Because the implemented way goes through all combinations of sites and takes all of the vertices and their associated ESPs, degree two bisector points and Voronoi centres and for all of the combinations it determines their root if it exists. Finding a root takes O(r) and there are O(s^3n) combinations of sites and vertices, which means determining all of the roots takes O(s^3nr) time. When handling a change event, the other steps only take up O(n) time or O(r) time. Updating the structure of the SPM can take up O(n) time. Finding the roots for new collision change events take O(r) time. Thus for handling just one event, it takes O(s^3nr) time. For handling all of the events, it takes O(es^3nr) time.

3.1.4 Additional information

The program has been created in C# with the use of NetTopologySuite library. This library has mostly been used for determining the triangulation of the polygon, the distance function between two points and finding the intersection of two lines. The program also uses the library MathNet.Numerics.RootFinding for several root-finding algorithms.

3.2 Correction on the algorithmic approach

When implementing the algorithm, there was one problem that kept on coming, namely rounding errors. This occurred mostly when determining the time of collision for ESPs. Two points collide when their distance becomes zero. The moment when the distance function becomes zero can be found with a root-finding algorithm. The earlier mentioned root-finding library has been used to find these events. This library has multiple different ways of determining the root of a general function that is of type Double $\mapsto$ Double. The distance function between two points is always of type Double $\mapsto$ Double and is always an approximation. This is because the distance function of NetTopologySuite only returns the rounded value to a certain decimal and not the mathematical expression of the calculation. After using every usable algorithm in the MathNet library, a combination of four different algorithms was used. Because each algorithm on its own did not always give the root when the root was actually present. The combination algorithm will split up the domain such that it can find the root in these sections instead of the whole domain. This tries to surpass the problem that when the root-finding algorithm gets to a local minimum that is not the root, it will not try to find another local minimum and returns that there is no root, even though if it had tried to find another local minimum it could have found it. The combination algorithm tries to find the root first with the Newton-Raphson algorithm, then with the Secant algorithm, then with the Brent algorithm and finally with the Bisection method. If it finds the root it will stop and will not continue to the other root-finding algorithms. If the program went through all of these algorithms and through all the sections of the domain, and still could not find it, it would try it again with sections of different sizes. The used section sizes were 100, 50, 20, 5 and 1.
Even though, the algorithm uses multiple root-finding algorithms with multiple sizes of sections, it still could not always find the roots. This is due to rounding errors. Many other options have also been looked at. For example, the distance function could be stored in a different way. The standard way was storing the function itself. But the coefficients can also be stored. This idea was also not continued, because the distance function uses multiple divisions of linear functions and uses the square root. This makes storing the function as a list of coefficients inefficient and unreadable.
After multiple attempts to try making the program work with the distance functions such that the program can determine when the collisions of moving ESPs happen, it has been decided to only focus on the vertex events. Keeping track of changes in the vertices does not involve finding a root. That only involves the intersection of the path segment and the line going through two vertices.
Even though, the root-finding algorithm was not accurate enough to determine the collision events, this algorithm was still used to determine when the equidistant events happen.

3.3 Data set

As stated before, the algorithm had to be run on different types of polygons, namely star shaped polygons, orthogonal polygons and fully random polygons. Star-shaped polygons and fully random polygons are being generated and the orthogonal polygons are being read from an already existing dataset. This dataset is the

Salzburg Database of Geometric Inputs[16].
To create the star-shaped polygons, the program generates l points uniformly randomly in a circle of radius r_1. Between each two consecutive points, a point will be generated uniformly randomly on the arc defined by the angles of the two consecutive points and radius r_2 that is between r_1 and $2r_1$. All generated points will be connected in counterclockwise order as seen from the centre of the circle.
To create the fully random polygons, the program generates n points uniformly random. These points get ordered by their x-coordinate. Two disjoint sets are created of all the generated points. A point gets placed in one of the two sets if it lies above or below the line that can be formed by the first and last points in the ordered list. The above-line list will be sorted ascending and the below-line list will be sorted descending. The first point of the original ordered list will be connected to the first point of the above-line list. All the points of the above-line list will be connected in order of the list. The last point of the above-line list will be connected to the last point of the original ordered list. The last point of the original ordered list will also be connected to the first one of the below-line list. The below-line list will be connected in order of the list. The last point in the below-line list will be connected to the first point of the original ordered list.
After the triangulation has been made of the polygon, the sites will be generated. The m sites will be generated uniformly. To generate the site, first each triangle of the triangulation has to get a certain probability. This probability is the area of the triangle divided by the total area of the polygon. A random double will be generated to determine in which triangle the new site will lie. Then to determine the real coordinates of the new site, two new doubles p and q will be generated. The coordinate will be determined by $p * e_1 + q * e_2$, where e_1 and e_2 are two edges of the triangle. If $p+q \geq 1$, then $p = 1-p$ and $q = 1-q$. This is to make sure the site is definitely in the triangle. The direction of the site's path is uniformly randomly chosen. The whole path of the site is a set of straight lines. At places where the straight line would intersect with the border of the polygon, the straight line bounces off to create a new segment. The new segment is also a straight line and the direction is determined by the angle of incidence and angle of reflection rule. This rule uses the normal of the edge a path segment bounces onto, but when a segment bounces into a vertex there is no clear definition of a normal. To solve this problem, the normal is the angle bisector of the angle formed by the two edges joining at this vertex. Within one unit of time, a site moves one unit of space. In this simulation, the amount of time passed is equal to the length of the path a site has passed. All of the sites travel at the same speed. There is a maximum of x bounces of a site on the border of the polygon. The time an instance runs, is equal to the lowest length of paths. This is to make sure every site still moves after the final bounce of some site.

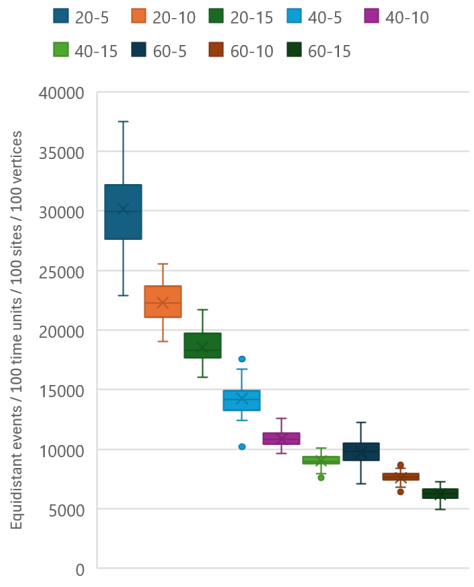


Figure 9: Star-shaped polygons: Number of equidistant events between two sites and one vertex, where the events are part of the Voronoi diagram. Sorted by number of vertices.
A category of x-y means x vertices and y sites.

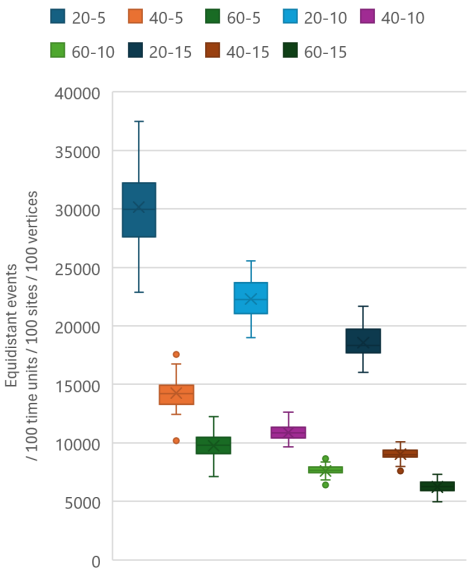


Figure 10: Star-shaped polygons: Number of equidistant events between two sites and one vertex, where the events are part of the Voronoi diagram. Sorted by number of sites.
A category of x-y means x vertices and y sites.

4 Results

The program has been run on star-shaped, fully random and orthogonal polygons. For the star-shaped polygons and the fully random polygons, it holds that for each polygon type polygons have been created of three different values of number of vertices, namely: 20, 40 and 60. For each specific number of vertices, five different polygon instances have been created. For each polygon instance, different site instances have been created of three different values of number of sites, namely: 5, 10 and 15. For each specific number of sites, ten different site instances have been created. With all these different combinations of number of vertices and number of sites, 450 instances of star-shaped polygons and 450 instances of fully random polygons have been created.
During testing the orthogonal polygons, the program needed a much longer time with the orthogonal polygons than was needed with the star-shaped and fully random polygons. Thus it was decided that for the orthogonal polygons only ten different polygons from the dataset would be used. Per polygon, there have been created different site instances of three different values of number of sites, namely: 5, 10 and 15. For each specific number of sites, there have been created ten different site instances. With all these combinations there have been 210 instances.
Each site of each instance has a maximum of 20 bounces onto the border of the polygon. Even though the number of sites, vertices and bounces have been kept very low, each polygon group still ran for at least six hours.
A part of the diagrams of the results has been placed in the apendix of this thesis. In Figures 9, 10, 11 and 12, we can see that for star-shaped and fully random polygons it holds when there are more vertices and more sites there will happen less equidistant events between two sites and a vertex that are part of the Voronoi diagram. The differences are not significant enough to determine if there is a linear or an exponential decrease. In Figure 13, the number of the same type of events does not following a certain rule. The order of magnitude between the different types of polygons is significant. Within the fully random polygons the events happen the least. The events with star-shaped polygons happen 10 times more often than with fully random polygons. The events with orthogonal polygons happen 10 times more often than with star-shaped polygons.

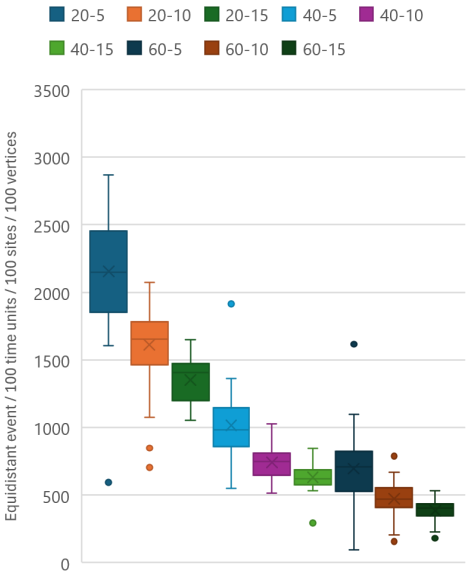


Figure 11: Fully random polygons: Number of equidistant events between two sites and one vertex, where the events are part of the Voronoi diagram. Sorted by number of vertices.
A category of x-y means x vertices and y sites.

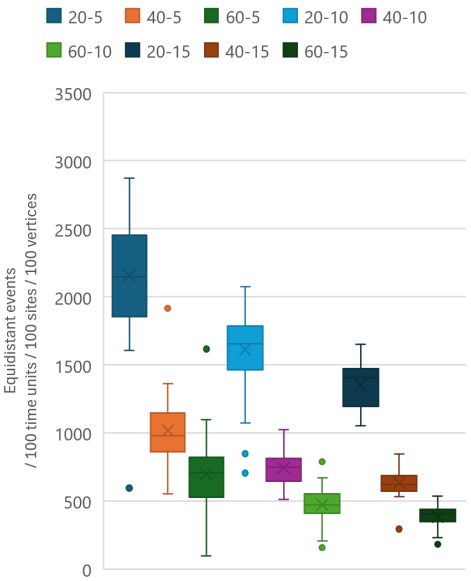


Figure 12: Fully random polygons: Number of equidistant events between two sites and one vertex, where the events are part of the Voronoi diagram. Sorted by number of sites.
A category of x-y means x vertices and y sites.

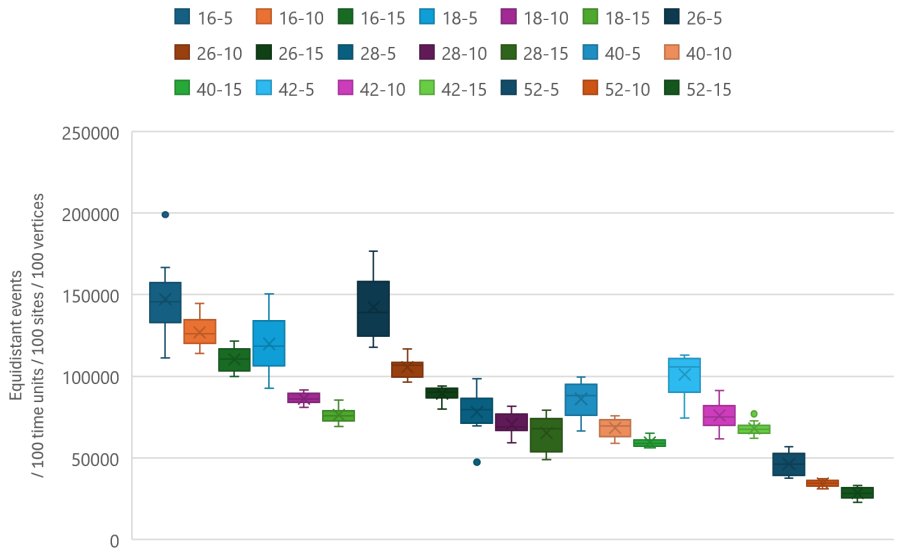


Figure 13: Orthogonal polygons: Number of equidistant events between two sites and one vertex, where the events are part of the Voronoi diagram. A category of x-y means x vertices and y sites.

In Figure 14, 15, 16, 17 and 18, we can see that if the number of vertices is fixed, the number of equidistant events between two sites and one vertex where the events are not necessarily part of the Voronoi diagram, follow an increasing linear function. It is difficult to see if there is a specific relation between the number of events and the number of vertices when the number of sites is fixed. With some fixed numbers of sites, it looks like the number of events follows a decreasing linear line in the number of vertices, but in other numbers of sites it looks like a decreasing exponential function. The only thing that can be concluded from the data, is that the number of events decreases when the number of sites is fixed. The order of magnitude between the different types of polygons is significant. Within the fully random polygons the events happen the least. Between star-shaped and orthogonal polygons the magnitude is the same. With star-shaped and orthogonal polygons, the equidistant events happen 10 times more often than with fully random polygons.
In Figure 19, 21 and 23, we can see that there is almost no change in colinear change events with two direct neighbours when the number of vertices is fixed. It is interesting to see that the number of events decreases when there are more vertices when looking at the star-shaped polygons and the fully random polygons, but this is not true for the orthogonal polygons. The order of magnitude between the different types of polygons is significant. Within the fully random polygons the events happen the least. Between star-shaped and orthogonal polygons the magnitude is the same. With star-shaped and orthogonal polygons, the colinear change events happen 10 times more often than with fully random polygons.
In Figure 20, 22 and 24, we can see that there is almost no change in colinear change events with a child and its principal child when the number of vertices is fixed. It is interesting to see that the number of events decreases when there are more vertices when looking at the star-shaped polygons and the fully random polygons, but this is not true for the orthogonal polygons. The order of magnitude between the different types of polygons is significant. Within the fully random polygons the events happen the least. Between star-shaped and orthogonal polygons the magnitude is the same. With star-shaped and orthogonal polygons, the colinear change events happen 10 times more often than with fully random polygons.
It is also interesting to see that the number of events of colinear change event of two children and a child and the principal child are of the same magnitude and have the same behaviour when focusing on the same type of polygon.

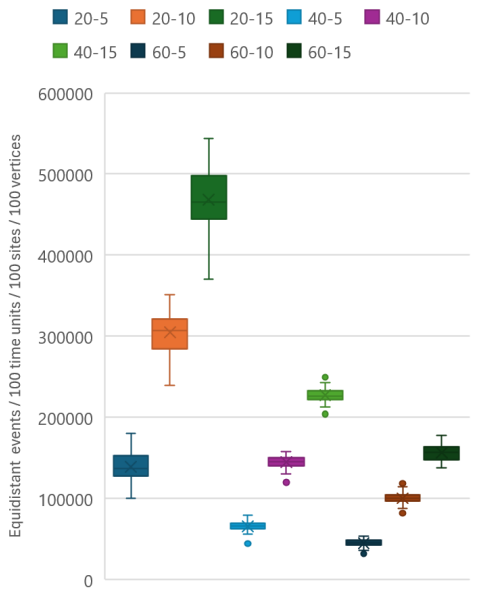


Figure 14: Star-shaped polygons: Number of equidistant events between two sites and one vertex, where the events are **not** necessarily part of the Voronoi diagram. Sorted by number of vertices.
A category of x-y means x vertices and y sites.

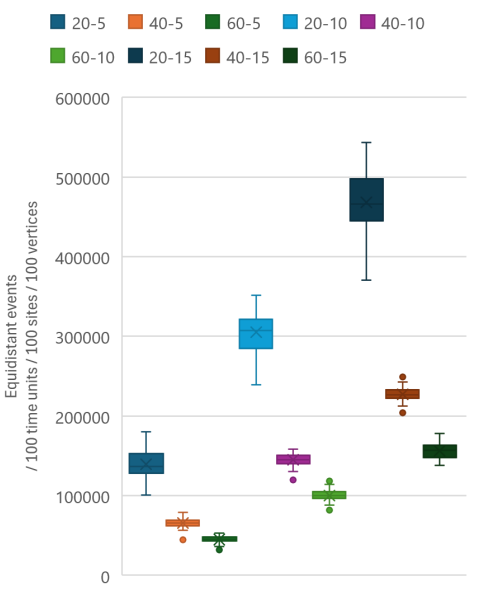


Figure 15: Star-shaped polygons: Number of equidistant events between two sites and one vertex, where the events are **not** necessarily part of the Voronoi diagram. Sorted by number of sites.
A category of x-y means x vertices and y sites.

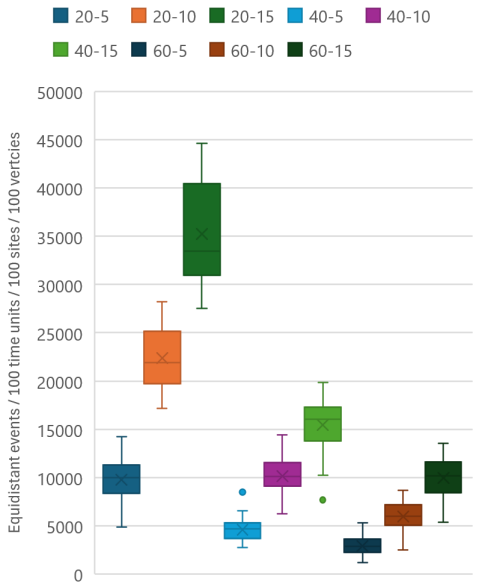


Figure 16: Fully random polygons: Number of equidistant events between two sites and one vertex, where the events are **not** necessarily part of the Voronoi diagram. Sorted by number of vertices.
A category of x-y means x vertices and y sites.

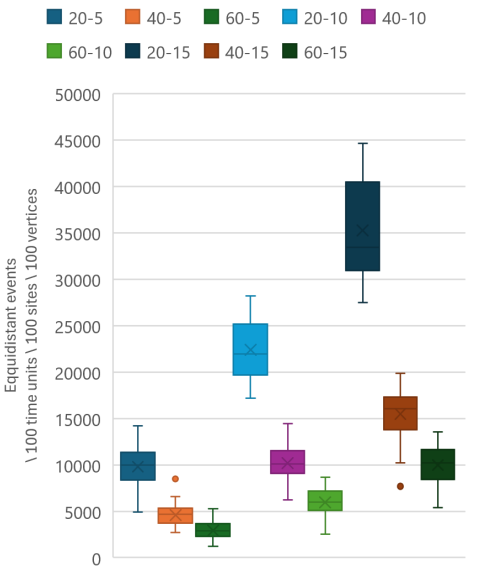


Figure 17: Fully random polygons: Number of equidistant events between two sites and one vertex, where the events are **not** necessarily part of the Voronoi diagram. Sorted by number of sites.
A category of x-y means x vertices and y sites.

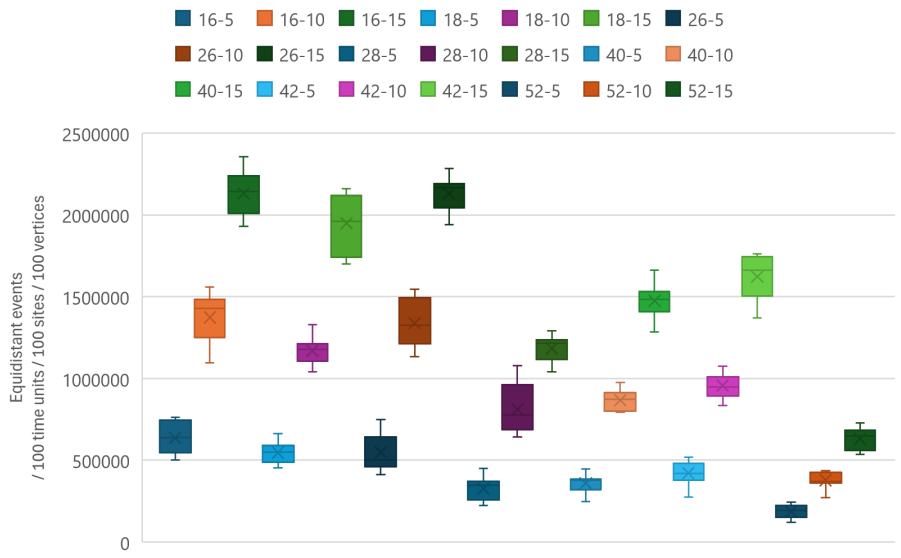


Figure 18: Orthogonal polygons: Number of equidistant events between two sites and one vertex, where the events are **not** necessarily part of the Voronoi diagram. A category of x-y means x vertices and y sites.

5 Conclusion

With the current data, only conclusions can be made on the vertex events. This can still be translated to the other types of events, because we know from Korman et al.[17] the bounds and we can relate the conclusion from this thesis to the other bounds. For example, they stated that vertex events happen O(m^2) times and 1,2-events also happen O(m^2) times. When looking at the data coming from the simulations, the conclusions that we can make for the vertex events can potentially also be made for the 1,2-events.
When focusing only on vertex events, the number of changes to the Voronoi diagram bounded by a polygon with kinetic sites is difficult to specify. Per type of polygon, there are better conclusions to make. The magnitude and behaviour of the data differ between the different types of polygons. Orthogonal and star-shaped polygons have more vertex events than fully random polygons, when the events are part of the Voronoi Diagram and when they are not.
If you fix the number of vertices, it seems that for all the polygon groups it holds that the number of vertex events that are not necessarily part of the Voronoi diagram is linear in the number of sites. But this linearity is completely different per polygon group and per number of vertices. When looking at it the other way, when you fix the number of sites, there is no clear decreasing function visible in any of the polygon groups.
The number of vertex events that are part of the Voronoi diagram decreases drastically when the instance becomes more difficult in means of its number of vertices and number of sites, when looking at star-shaped and fully random polygons. These do not follow any specific function when you fix the number of vertices or the number of sites. Looking at the orthogonal polygons, the data partially shows that vertex events that are part of the Voronoi diagram do happen less when the instance becomes more difficult just like with the star-shaped and fully random polygons. But there are clear spikes in the data. Thus, concluding vertex events that are part of the Voronoi diagram happen less when the instance becomes more difficult is not a conclusion that can be made for orthogonal polygons. More information on the polygons is needed for such conclusions.
It is interesting to see that in general it holds that if the instance becomes more difficult in means of the number of vertices and the number of sites, vertex events that are part of the Voronoi diagram happen less. You would assume that this happens more often, because you have more vertices and more sites. But this observed result could come from the fact that if there are more sites, less pairs of vertices can have an equidistant event and being part of the Voronoi diagram. The values in the diagrams are per 100 time units, per 100 sites and per 100 vertices, this means that it happens less in general.
When comparing the results with the bound of Korman et al.[17] of $O(n^3 s^3 \beta_z(n))$, we can make the conjecture that with specific types of polygons this bound is indeed significantly lower. It would be of interest to have further research into specific polygon groups such as the star-shaped polygons. The star-shaped polygon group had the most interesting results. Creating specific algorithms with specific data structures for these polygon groups can have interesting results, because they have to most remarkable results from this thesis.

6 Discussion

The algorithm itself can still be improved. For making an SPM, we chose to go through the whole funnel to determine the first vertex of the shortest path from a newly introduced vertex toward the site. However, it is also possible to use specific data structures such that the first vertex can be found quickly, for example using a finger tree as Guibas et al.[1] suggested. Determining when two sites are equidistant can be optimized. The algorithm goes per change event through all the sites and all the vertices. Even though, you know that only one leg of the tree will be changed. Thus determining equidistant events is only necessary in the part that has to be changed. Furthermore, there are multiple places in the code that can be refactored.
With an improved program, more data can be created. Because each polygon group took at least six hours to run, it was decided to decrease the amount of data that would be created. This also meant that the conclusions that could be made were not as truthful as desired. More data is needed to make the conjectures of the conclusion section more concrete.
Initially, the choice was made to program in C#, because of the present knowledge of C#. But within C++ there is the library CGal. To a certain degree, this library stores the mathematical expression as the result of a calculation instead of the rounded value. That helps with rounding errors. If you can delay the moment when you are specifying the result and thus the need for rounding, you can have more precise answers. This could have helped with the problem of not being able to determine the collision events of ESPs. Another way of solving this problem, could have been spending more time on saving the mathematical expression in the current structure. There can be more C# libraries that can do this, but this has not been researched further.
To determine the equidistant events, the same root-finding algorithm had been used as the one that was used to try to find the collision event. This algorithm does find most of the roots, but not all of them. This means that more events could have happened than we could see in the data, since the algorithm could almost always find all of the roots. It is good enough to make conclusions from it. Especially, when you run the algorithm multiple times. But to have a more truthful picture of the different instances, this algorithm has to be changed too.
Comparing the data with each other data set is difficult, due to the multi-dimensional nature of the problem. There are the number of vertices, the number of sites, the number of legs of a polygon, the number of vertices with angles above 180° and many other variables. This thesis only focused on specific polygon groups and the number of sites and vertices. The polygon groups could have been more specified through other variables. More variables have to be taken into account.
Finally, there are more polygon groups to discover with the approach of this thesis. Polygons can also be created with a specific set of vectors, such that there will only be a discrete number of angles in the polygon. The orthogonal polygon group is an example of this. But there are more polygon groups to be used.

7 References

[1] Leonidas Guibas et al. "Linear time algorithms for visibility and shortest path problems inside simple polygons". In: *Proceedings of the second annual symposium on computational geometry*. 1986, pp. 1–13.

[2] Alok Aggarwal et al. "A Linear-Time Algorithm for Computing the Voronoi Diagram of a Convex Polygon". In: *Discret. Comput. Geom.* 4 (1989), pp. 591–604. DOI: 10.1007/BF02187749. URL: https://doi.org/10.1007/BF02187749.

[3] Boris Aronov. "On the Geodesic Voronoi Diagram of Point Sites in a Simple Polygon". In: *Algorithmica* 4.1 (1989), pp. 109–140. DOI: 10.1007/BF01553882. URL: https://doi.org/10.1007/BF01553882.

[4] L Paul Chew. "Building Voronoi diagrams for convex polygons in linear expected time". In: *Computer Science Technical Report* PCS-TR90-147 (1990).

[5] Jyh-Jong Fu and Richard C. T. Lee. "Voronoi diagrams of moving points in the plane". In: *Int. J. Comput. Geom. Appl.* 1.1 (1991), pp. 23–32. DOI: 10.1142/S0218195991000037. URL: https://doi.org/10.1142/S0218195991000037.

[6] Toshiyuki Imai. "A Topology Oriented Algorithm for the Voronoi Diagram of Polygons". In: *Proceedings of the 8th Canadian Conference on Computational Geometry, Carleton University, Ottawa, Canada, August 12-15, 1996*. Ed. by Frank Fiala, Evangelos Kranakis, and Jörg-Rüdiger Sack. Carleton University Press, 1996, pp. 107–112. URL: http://www.cccg.ca/proceedings/1996/cccg1996%5C_0019.pdf.

[7] Niranjan Mayya and V. T. Rajan. "Voronoi diagrams of polygons: A framework for shape representation". In: *J. Math. Imaging Vis.* 6.4 (1996), pp. 355–378. DOI: 10.1007/BF00123352. URL: https://doi.org/10.1007/BF00123352.

[8] Gerhard Albers et al. "Voronoi Diagrams of Moving Points". In: *Int. J. Comput. Geom. Appl.* 8.3 (1998), pp. 365–380. DOI: 10.1142/S0218195998000187. URL: https://doi.org/10.1142/S0218195998000187.

[9] Julien Basch, Leonidas J. Guibas, and John Hershberger. "Data Structures for Mobile Data". In: *J. Algorithms* 31.1 (1999), pp. 1–28. DOI: 10.1006/jagm.1998.0988. URL: https://doi.org/10.1006/jagm.1998.0988.

[10] John Hershberger and Subhash Suri. "An Optimal Algorithm for Euclidean Shortest Paths in the Plane". In: *SIAM J. Comput.* 28.6 (1999), pp. 2215–2256. DOI: 10.1137/S0097539795289604. URL: https://doi.org/10.1137/S0097539795289604.

[11] Leonidas J Guibas, Jack Snoeyink, and Li Zhang. "Compact Voronoi diagrams for moving convex polygons". In: *Scandinavian Workshop on Algorithm Theory*. Springer. 2000, pp. 339–352.

[12] Aronov et al. "Visibility queries and maintenance in simple polygons". In: *Discrete & Computational Geometry* 27 (2002), pp. 461–483.

[13] de Berg Mark et al. *Computational geometry algorithms and applications*. Spinger, 2008, pp. 147–171.

[14] Bettina Speckmann. "Kinetic Data Structures". In: *Encyclopedia of Algorithms*. Ed. by Ming-Yang Kao. Boston, MA: Springer US, 2008, pp. 417–419. ISBN: 978-0-387-30162-4. DOI: 10.1007/978-0-387-30162-4_191. URL: https://doi.org/10.1007/978-0-387-30162-4_191.

[15] Eunjin Oh. "Optimal algorithm for geodesic nearest-point Voronoi diagrams in simple polygons". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2019, pp. 391–409.

[16] Günther Eder et al. *Salzburg Database of Polygonal Data*. Version 20200507. Work supported by Austrian Science Fund (FWF): Grants ORD 53-VO and P31013-N31. Zenodo, May 2020. DOI: 10.5281/zenodo.3784789.

[17] Matias Korman et al. "Kinetic Geodesic Voronoi Diagrams in a Simple Polygon". In: *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*. Ed. by Artur Czumaj, Anuj Dawar, and Emanuela Merelli. Vol. 168. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 75:1–75:17. DOI: 10.4230/LIPIcs.ICALP.2020.75. URL: https://doi.org/10.4230/LIPIcs.ICALP.2020.75.

[18] Chanyoung Song et al. "Dynamic Voronoi Diagram for Moving Disks". In: *IEEE Trans. Vis. Comput. Graph.* 27.6 (2021), pp. 2923–2940. DOI: 10.1109/TVCG.2019.2959321. URL: https://doi.org/10.1109/TVCG.2019.2959321.

8 Apendix: Diagrams

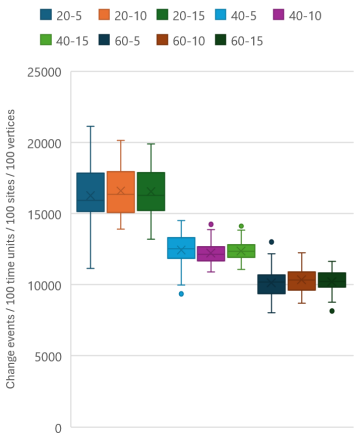


Figure 19: Star-shaped polygons: Number of colinear change events between a site and two children that are direct neighbours in angular circle. A category of x-y means x vertices and y sites.

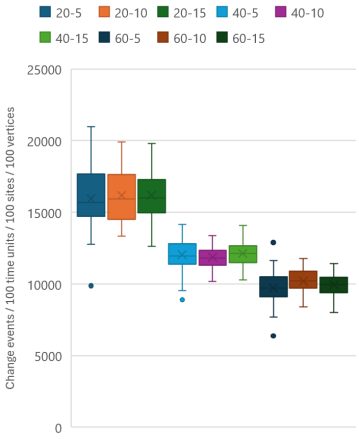


Figure 20: Star-shaped polygons: Number of colinear change events between a site and a child and its principal child. A category of x-y means x vertices and y sites.

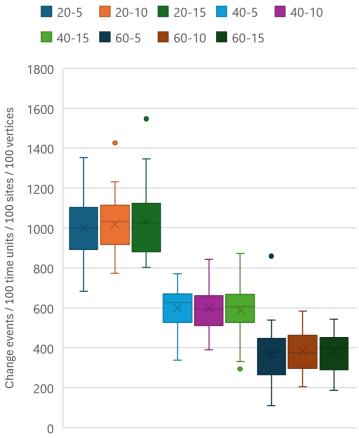


Figure 21: Fully random polygons: Number of colinear change events between a site and two children that are direct neighbours in angular circle. A category of x-y means x vertices and y sites.

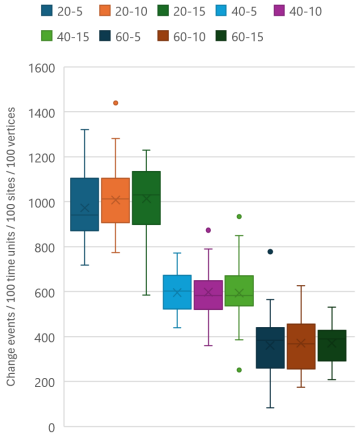


Figure 22: Fully random polygons: Number of colinear change events between a site and a child and its principal child. A category of x-y means x vertices and y sites.

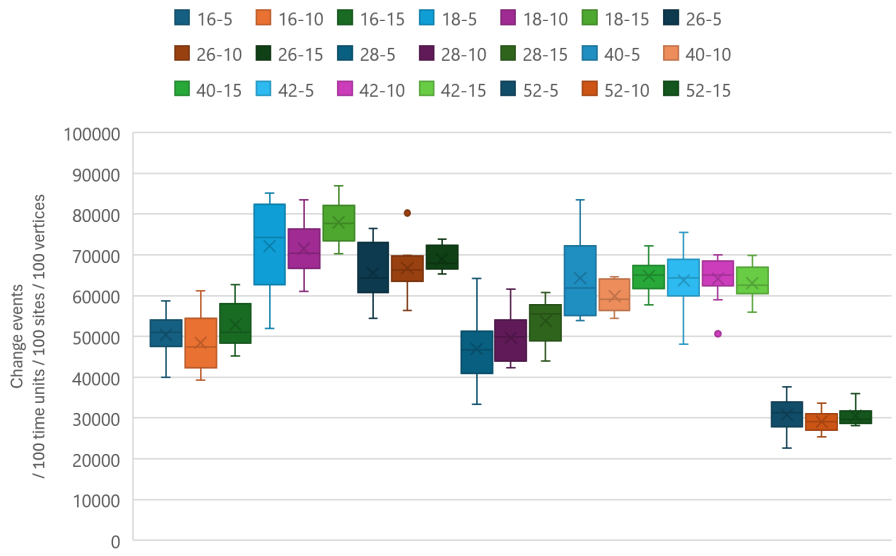


Figure 23: Orthogonal polygons: Number of colinear change events between a site and two children that are direct neighbours in angular circle. A category of x-y means x vertices and y sites.

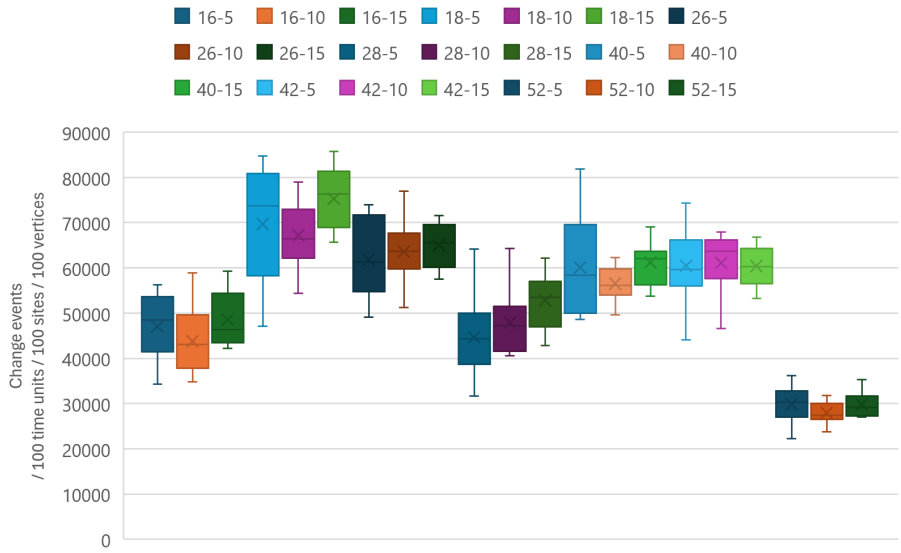


Figure 24: Orthogonal polygons: Number of colinear change events between a site and a child and its principal child. A category of x-y means x vertices and y sites.