Online Routing on Delaunay Triangulations in the Presence of Obstacles

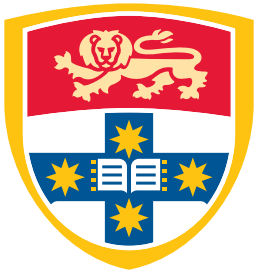
GUANGYAO GUO
SID: 490332933

Supervisor: Dr. André van Renssen

This thesis is submitted in partial fulfillment of
the requirements for the degree of
Bachelor of Advanced Computing (Honours)

School of Computer Science
The University of Sydney
Australia

30 December 2023



THE UNIVERSITY OF
SYDNEY

Student Plagiarism: Compliance Statement

I certify that:

I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure;

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to the University commencing proceedings against me for potential student misconduct under Chapter 8 of the University of Sydney By-Law 1999 (as amended);

This Work is substantially my own, and to the extent that any part of this Work is not my own I have indicated that it is not my own by Acknowledging the Source of that part or those parts of the Work.

Name: Guangyao GUO

Signature: **Date**:

Abstract

In Computational Geometry, Delaunay triangulations $\mathcal{DT}(V)$ on a point set V consist of non-overlapping triangles such that no point lies within the circumcircle of any triangle. It has been well studied with many nice properties found. Chew's routing algorithm proposed in [1, 2] is an online algorithm that can efficiently route on Delaunay triangulations using only local information stored in the current vertex and generate reasonable short paths.

Though it has been proved that constrained Delaunay triangulations that are equivalent to Delaunay triangulations in the presence of obstacles are plane spanners [3], which means there always exists a bounded path, the reachability and performance of online routing algorithms on these graphs are still unclear in the presence of obstacles.

In this thesis, we adapt Chew's online routing algorithm on Delaunay triangulations in the presence of obstacles and utilize recent advancement of best known bounds made by Bonichon et al. [4] and analyse the movement of the algorithm around obstacles, exploring the impact on the algorithm's performance.

We prove that Chew's online routing algorithm's reachability is not compromised in the presence of obstacles under the assumption that the obstacles are simple polygons not intersecting the line segment connecting the source and target. Furthermore, we show that the upper bound of the routing ratio remains at 5.90 in the presence of obstacles.

Acknowledgements

First, I want to show my deepest gratitude to my supervisor, Dr. André van Renssen. Your expert guidance, insightful suggestions and patient supervision constantly inspiring me to the final step of my undergraduate study. Your way of teaching, supervision and work made a great exemplar for me to look upon in the rest of my life.

I must also seize this opportunity to thank my parents, for their unwavering support and unconditional love. I am not a person of words, but I deeply appreciate all the sacrifices you have made for me.

Lastly, I wish to express my heartfelt appreciation to my Xiaoqing, despite the 5140 miles between us, you have always been the most steadfast pillar of my soul.

Thank you all for being an invaluable part of my life and journey.

CONTENTS

List of Figures

CHAPTER 1

Introduction

Given a set of points $V \in \mathbb{R}^2$, the Delaunay triangulation $\mathcal{DT}(V)$ is a computational geometry concept that generates a triangulated mesh using V such that no point in the set is inside the circumcircle of any triangle in the mesh. Coming with a number of desirable properties, such as maximizing the minimum angle of all the triangles in the mesh, and being unique for a given set of points, it firstly serves applications in the field of land surveying. Later, when computer science thrived in the late $20th$ century, many new fields were discovered, to which Delaunay triangulations are also applied, such as Geographic Information Systems, Computer Vision, Computational Geometry [5], and so on. Due to the potential ability of Delaunay triangulations to contribute in such areas, it has been studied intensively for efficient pathfinding algorithms to fulfil the need in both theory and application levels.

Consider such a Delaunay triangulation, how should we utilize those desirable properties? Path-finding? Feature-extraction? Distribution optimization? Yes, all of them have been explored in those fields mentioned above, and path-finding is one of the most classic problems among them that we are going to focus on.

When knowing the whole map of the graph, the shortest path can be easily found using search algorithms like BFS, DFS and Dijkstra's algorithm which are already fairly efficient. Thus it is trivial to investigate routing algorithms with the knowledge of the whole graph. Algorithms that know the whole graph are called offline algorithms. In other words, it is meaningful and realistic to consider a routing algorithm that does not know the whole graph but is still able to reach the destination. By definition, an online algorithm must serially process its input data piece by piece while an offline algorithm is aware of all input data before it starts running.

When discussing routing algorithms, first, we need to primarily prove the algorithm is able to solve all instances, and second we need to evaluate the performance of it. Online routing algorithms are usually efficient both in time and space complexity due to the piece-by-piece input of information, which means

the algorithm hardly needs to traverse the graph after it starts running. However, the output of an online algorithm is naturally not as 'good' as offline algorithm due to the limitation of information, so we also need to explore the bounds of the algorithm, in this case, to prove the length range the of path found by the algorithm, or, to find the best and worst case.

Routing without knowledge of the whole graph is a challenging problem, where the performance of online algorithms is usually hindered by worst-case scenarios. However, recent research has made significant progress in developing efficient online routing algorithms for the Delaunay triangulation. Specifically, Bonichon et al. [4] have adapted Chew's routing algorithm [1, 2] for $\mathcal{L}_1$-Delaunay triangulation, to achieve fairly tight upper 5.90 and lower bound 2.70 of path length for normal Delaunay triangulations. This means that there exists an online routing algorithm that guarantees reasonable path length for the Delaunay triangulation.

Additionally, research has also focused on the case of constrained Delaunay triangulations which could be treated as Delaunay triangulations in the presence of obstacles. Bose et al. [3] have demonstrated that the constrained Delaunay triangulations are plane spanners where the stretch factor depends on the shape used in the construction. This further implies that the existence of a path is guaranteed and the path length is also bounded by the value of the stretch factor in the Delaunay triangulation with obstacles.

Thanks to their contributions, we can now attempt to stand on the shoulders of giants. First, we know how to find a path in Delaunay triangulations without obstacles efficiently. Second, we know there exists a path with the length upper bounded by the stretch factor (the length of the path is t times their Euclidean distance, see 2.1 for detail) for Delaunay triangulations in the presence of obstacles. So, the question is, how should we find the path? Does the existing routing algorithm work well in the presence of obstacles? Will there be any compromise in the performance? In this thesis, we show that, given Delaunay triangulation $\mathcal{DT}(V)$ of a point set V, and arbitrary vertices $s,t \in \mathcal{P}$, when all obstacles are in the shape of simple polygons and never intersect $[st]$, Chew's routing algorithm works well on Delaunay triangulation in the presence of obstacles without any compromise of performance to find a path from s to t.

1.1 Main Results

Given a Delaunay triangulation $\mathcal{DT}(V)$ of point set V, a source vertex s and a target vertex t in V, with simple polygonal obstacles that do not cross $[st]$. We proved that the reachability of Chew's online

routing algorithm is not compromised in the presence of obstacles. We also show that the 5.90 upper bound of the routing ratio is not compromised as well in the presence of obstacles. As the lower bound of the routing ratio without obstacles proved by Bonichon et al. [4] is 5.72, and it can be deemed as the routing ratio with trivial obstacles that do not affect the triangles used to route, the upper bound is also almost tight in the presence of obstacles. In addition, we show some cases and properties of how Chew's routing algorithm interacts with obstacles to illustrate why the lower bound looks unassailable even in the presence of obstacles.

1.2 Outline

In Chapter 2, we briefly go through the preliminaries for this thesis. This includes the definition of t-spanner and stretch factor to show what it means for a graph, and metrics like competitive and routing ratio to evaluate the performance of an online routing algorithm. We also go into more detail about what the Delaunay triangulation is and the properties of it, and the detail of the previous research done related to our research question.

In Chapter 3, we analyse the difference between Delaunay triangulation and Delaunay triangulations in the presence of obstacles. Define the obstacles in our case and the motivation of such definition. We also go through some conjectures we had in mind and the counter-examples that show they do not hold in the presence of obstacles. This shows why we should do more work to evaluate the performance of Chew's routing algorithm in the presence of obstacles, especially with respect to bounds.

In Chapter 4, we prove the reachability of the algorithm is not compromised in the presence of obstacles (Theorem 2), under the assumption that the obstacles are simple polygons that do not intersect $[st]$, which is to make sure that s,t are visible to each other.

In Chapter 5, we prove that the upper bound of routing ratio of Chew's routing algorithm on Delaunay triangulations in the presence of obstacles is still 5.90 (Theorem 3), by showing how to adapt the proof by Bonichon et al. [4]. In detail, we identify the property that does not hold in our cases, target the affected lemmas in the original proof, then prove that the key lemmas still hold in our case, and finally show we can still get the same bound in the presence of obstacles.

In Chapter 6, we show some analysis about the routing specifically when around or on obstacles. As the upper bound proved in Chapter 5 and lower bound in [4] are almost tight. In the presence of obstacles,

we try to shed some light on the lower bound, though so far we were unable to improve on the existing bound.

In Chapter 7, we conclude this thesis and discuss the limitations of our research in order to suggest some future work.

CHAPTER 2

Preliminaries

In this chapter, we discuss some important concepts and definitions related to graph spanners, Delaunay triangulations and online algorithms. Furthermore, we will provide a more detailed discussion on the current understanding of online routing algorithms and constrained Delaunay triangulations that our further examination and evaluation are based on.

2.1 T-spanner and Stretch factor

The *t-spanner* in [6] is defined as follows: for a set of vertices S, build an undirected graph G for S, then for any two points p, q in S, there exists a path in G connecting p and q with length no longer than $t|[pq]||$ (We denote by $||[pq]||$ the Euclidian distance of line segment $[pq]$). Furthermore, in the context of geometric spanners, the *stretch factor* , also called *spanning ratio* of G is the smallest t for which G is a t-spanner. A small value of t and stretch factor means the longest detour we may find is guaranteed to not be too much longer than the straight line distance, therefore, a small t is the motivation for us to design routing algorithms for the corresponding graph.

2.2 Competitive and Routing ratio

When we have a routing algorithm for a spanner, we can define its competitive ratio and routing ratio. Given a set of points $P \in \mathbb{R}^2$, the routing algorithm has *competitive ratio c* if the length of the path found by the algorithm from an arbitrary vertex s to another arbitrary vertex t is at most c times the length of the shortest path between them. Similarly, the *routing ratio r* means that the length of the path found is at most $r|[st]||$.

2.3 Delaunay Triangulation

Though Delaunay triangulation have been defined by Boris Delaunay in 1934, this textbook [5] introduces it in a comprehensive way including its source, construction process and properties. As introduced in the textbook, a Delaunay triangulation is a set of triangles with no vertex lying inside the circumcircle of any triangle in the set (Fig. 2.1). With this property, it avoids most thin triangles as this usually corresponds to a giant circumcircle.

There are multiple ways to construct Delaunay triangulations, but all of them share the same main idea, which is that a triangle can exist in the Delaunay triangulation if and only if its circumcircle does not have any other vertex inside it. Different construction algorithms have achieved fairly fast with $O(nlogn)$ time complexity [7], so as long as we are aware of the main idea of construction, the efficiency of the algorithms is fast enough. Since the construction algorithms are not the main focus here, we do not elaborate on the construction of Delaunay triangulation in more depth.

In the context of Delaunay triangulations. In 1986, Chew [1] firstly introduced the lower bound of the stretch factor on the Delaunay triangulation as $\pi/2$, and an upper bound around 5.08 was introduced by Dobkin et al. [8]. Then, Keil and Gutwin [9] further successfully found the tighter upper bound between those which is around 2.42. Bose et al. [10] later improved the lower bound of the stretch factor to 1.581 which is strictly larger than $\pi/2$. Finally, in 2013, Xia managed to prove that Delaunay triangulations are plane spanners with a stretch factor less than 1.998 [11] which remains the best upper bound of the stretch factor for Delaunay triangulation till now. Besides, the best lower bound which is 1.593 was introduced in 2011 by Xia and Zhang [12].

2.4 Online Routing Algorithms on Delaunay Triangulations

In 2004, Bose and Morin [13] proved there is no online routing algorithm works for all triangulations and further designed a competitive online algorithm for routing on Delaunay triangulations with only $O(1)$ memory use and competitive ratio around 45.749 [14], which was the best bound at that time. In 2014, Bose et al. [15] improved the competitive ratio of online routing on Delaunay triangulations down to 7.621. Later in 2015, Bonichon et al. [4] made an adaptation to Chew's algorithm on Delaunay triangulations with an upper bound of 5.90 and a lower bound of 2.70. They proved that the upper bound of Chew's algorithm is at least 5.72 thus proving the upper bound of 5.90 is very close to the

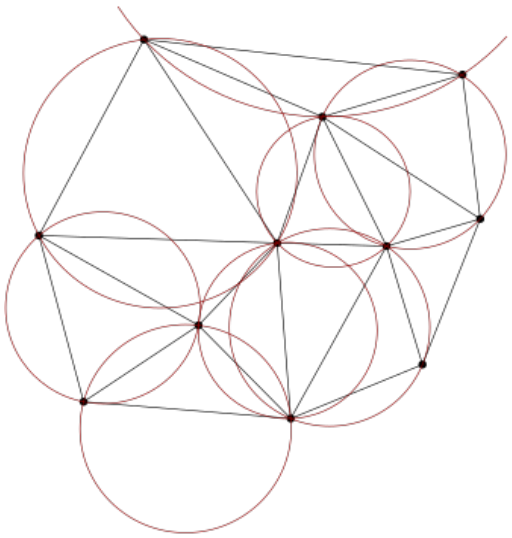


Figure 2.1. Each circumcircle has no other vertex inside it other than the vertices of its own triangle.

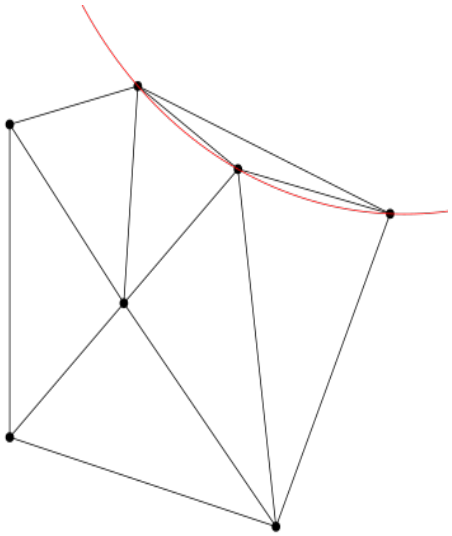


Figure 2.2. Thin obtuse triangle corresponds to giant circumcircle (red arc is part of the circumcircle).

best possible, and they also proved that the competitive ratio of any online algorithm on Delaunay triangulations is at least 1.70 and 2.70 for the specific L_1-Delaunay triangulation. In 2018, Bonichon et al. [16] designed a new algorithm called *MixedChordArc* with an upper bound of only 3.56 which is still the best known for online routing algorithms on Delaunay triangulations. As Chew's algorithm comes with fairly bounded path length and less specific assumptions compared to later Bonichon et al.'s *MixedChordArc* algorithm that only applies to L_2-Delaunay triangulations, in this thesis, we will focus on Chew's algorithm and the bounds proved by Bonichon et al.

Chew's algorithm [4] can be defined as follows (see Fig. 2.3): Given $\mathcal{DT}(V)$ as a Delaunay triangulation of a point set V, a source vertex s, and a target vertex t of V. The algorithm routing from s to t by, first, rotating $\mathcal{DT}(V)$ to set $[st]$ to be horizontal. Then start from s, let the vertex of the current routing position be p_i ($s = p_0$). When reach p_i, if there exists an edge connecting p_i and t, we set $p_{i+1} = t$. Otherwise, find the *rightmost* triangle T_i according to their *rightmost* intersection point with $[st]$ that contains p_i. Then, let w_i be the leftmost point of the circumcircle C_i of T_i, and r_i be the rightmost intersection of C_i with $[st]$. If p_i is on the upper arc $w_i r_i$, the algorithm goes next to the vertex met first in clockwise direction from p_i. If p_i is on the lower arc $w_i r_i$, the algorithm goes next to the vertex met first in counter-clockwise direction from p_i. The algorithm repeats this until it reaches t.

The following theorem is a main result of the upper bound from [4],

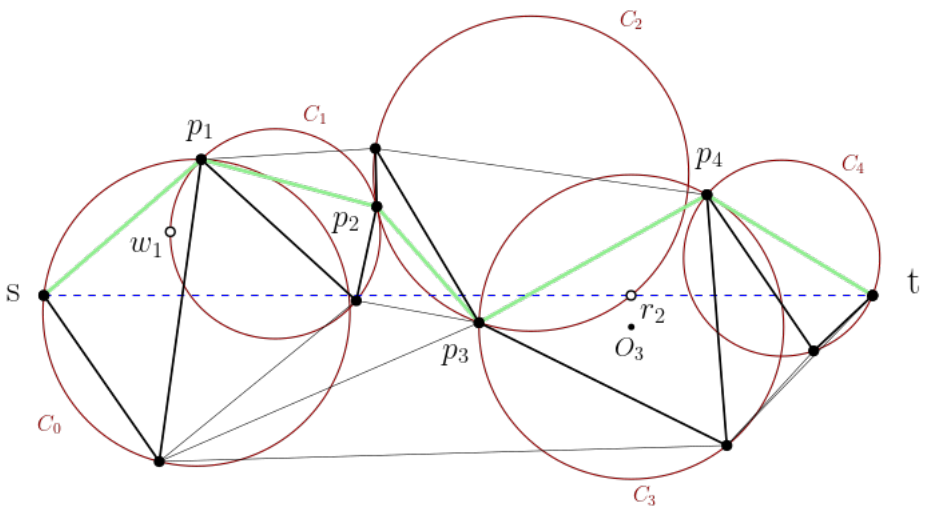
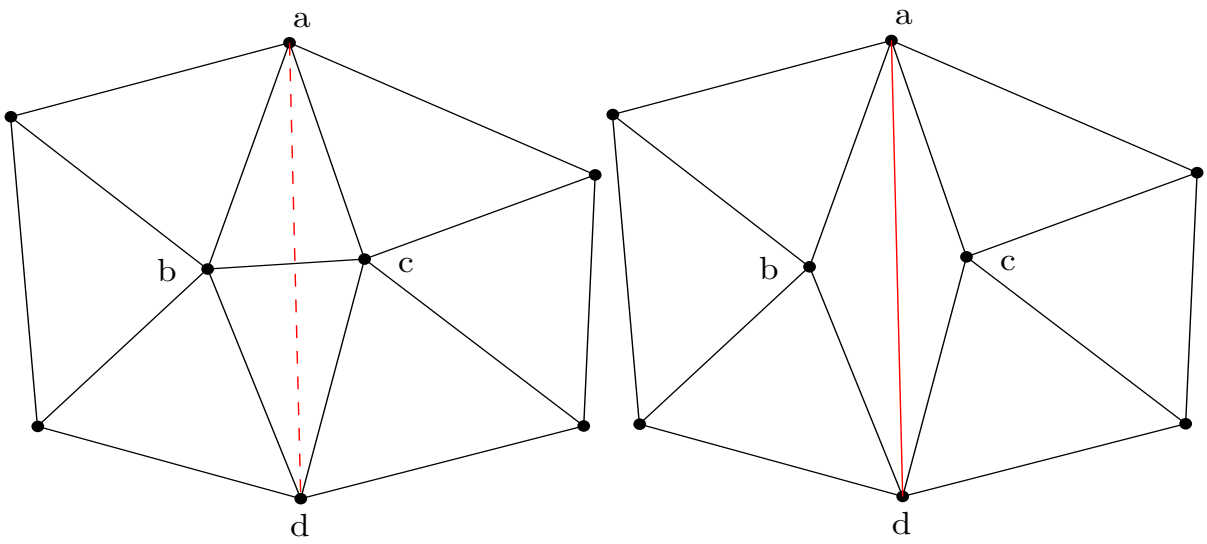


Figure 2.3. Chew's algorithm.

THEOREM 1. *The routing ratio of Chew's routing algorithm on the Delaunay triangulation is at most* $(1.185043874 + \frac{3\pi}{2}) \approx 5.90$.

2.5 Constrained Delaunay Triangulations

From the perspective of Delaunay triangulations in the presence of obstacles, the first thing is how we implement obstacles into Delaunay triangulations. After we put obstacles into the graph, the edges of obstacles are certainly not supposed to be modified. Therefore 'Constrained' could be considered as either forcing certain edges into the graph or forbidding the existence of edges between some vertices when constructing Delaunay triangulations (Fig. 2.4). By constructing constrained Delaunay triangulation, it is possible to simulate the presence of obstacles in Delaunay triangulations. In 2004, Bose and Keil [17] showed that constrained Delaunay triangulations are spanners with a spaning ratio of around 2.42. Later, Bose et al. [3] introduced how to generate constrained Delaunay triangulations for shapes other than circles, and showed that constrained Delaunay Triangulations are plane spanners where the stretch factor depends on the shape used in the construction. For example, Delaunay triangulations based on rectangles have stretch factor at least $\sqrt{2} \cdot \sqrt{(l/s)^2 + 1 + (l/s) \cdot \sqrt{(l/s)^2 + 1}}$ (l and s for the length of the long and short side of the rectangle). In a more special case, when the rectangle is a square, which means $l = s$, the stretch factor will be $\sqrt{4 + 2\sqrt{2}}$.



(a) Original Delaunay triangulation before the red dashed line segment $[ad]$ added as an obstacle.

(b) Constrained Delaunay triangulation after forcing the existence of $[ad]$ or forbidding the existence of $[bc]$.

Figure 2.4. Construction of constrained Delaunay triangulation with a line segment obstacle

2.6 Routing Algorithms in the Presence of Obstacles

Bose et al. [18] introduced the first local routing algorithm in the context of constraints for Θ-graph (a different type of spanner) with the length of path bounded. Later, Bose et al. [19] further introduced a local algorithm routing between non-visible vertices which is optimal in the worst case.

In recent studies, Bonichon et al. [4] proposed an online routing algorithm derived from Chew's algorithm on Delaunay triangulations with a fairly tight upper and lower bound, and Bose et al. [3] has proved that constrained Delaunay triangulations are plane spanners which bound the shortest path length on the constrained Delaunay graph.

However, though fairly tight bounds are already suggested on the spanning ratio of constrained Delaunay triangulations, we still do not have any idea about online routing or the upper and lower bounds of such algorithms. There exists a gap in recent research done in this area, and this is what we will explore.

CHAPTER 3

Obstacles

We have mentioned obstacles many times, but what kind of obstacles do we consider? Suppose s and t lie on different sides of the exterior boundary and interior boundary of a polygon with a hole, routing from s to t now becomes an impossible task. To avoid this, we need to set some assumptions for the obstacles.

In this chapter, we first show what kind of obstacles we consider and why. Then, we show more in detail how the Delaunay triangulations could be impacted by obstacles. Furthermore, we go through some conjectures of the algorithm's performance in the presence of obstacles and show the counterexamples we have found.

3.1 Definition of Obstacles

As we are adding obstacles to Delaunay triangulations, naturally we need to ensure that the new graph can still be triangulated, so we first want the obstacles to be made up of line segments rather than curves. Also, we can walk along a wall but cannot walk through it to the other side, and we can turn at the point the wall turns, so we set the edges of obstacles as also valid edges to route on in the graph, and also vertices of obstacles as vertices to be considered when triangulating. Considering the above, polygon obstacles become the perfect candidates.

3.1.1 Vertex Disjoint Simple Polygon Obstacles

In detail, we define the obstacles as vertex disjoint simple polygons. First, consider the scenario where obstacles pile up together, sharing edges, or forming a complex polygon with edges that extend into the constrained area. Even though the edges are passable by our definition, remember that Chew's

algorithm uses triangles for routing. Clearly, an edge flanked by obstacles on both sides cannot form any valid triangles. Therefore, such an obstacle can be treated as a simple polygon in our case.

Second, consider an obstacle with holes, as illustrated in Fig. 3.1. The vertices in the holes and the vertices on the boundary are mutually invisible. Whether we route around the obstacles or inside the holes, there is no need to store information on the other side, as the path would not be impacted by it.

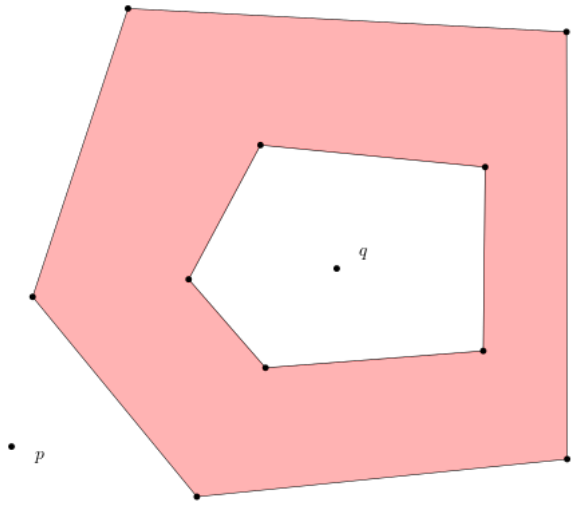


Figure 3.1. Vertex p around obstacle and vertex q inside hole are not visible to each other.

Finally, following the approach of van Renssen and Wong as shown in [20], we let the obstacles to be vertex disjoint. They demonstrated that a vertex shared by two polygonal obstacles can always be duplicated, depending on whether the path is allowed to pass through the vertex to another side. More important, they showed that this process does not impact the routing ratio (see Fig. 3.2).

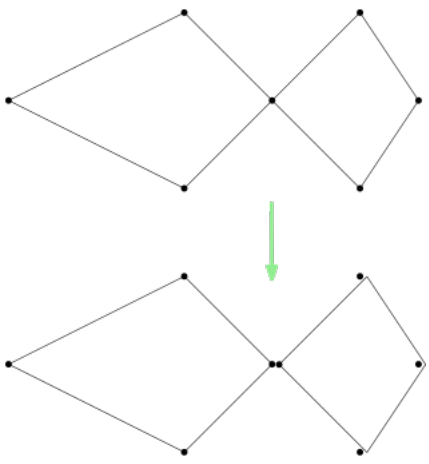


Figure 3.2. Sharing vertex is duplicated into two split vertices.

3.1.2 No Obstacles Cross $[st]$

Next, we consider where these obstacles can be located. As stated in [4], it is assumed that no other vertex lies on $[st]$, or we can split $[st]$ by that vertex, say t' and divide the original problem route from s to t into two sub-problems: from s to t' and from t' to t. Thus, for the same reason, we assume that no vertices of obstacles lie on $[st]$, though they are passable.

Moreover, recall that Chew's routing algorithm uses a set of triangles $T_0, T_1, ..., T_k$ that intersect $[st]$ to decide each move. The following lemma is proved by Chew in [1].

LEMMA 1. *The order in which Chew's algorithm uses triangles $T_0, T_1, ..., T_k$ is exactly the same as their order by intersecting $[st]$, even when not all triangles of the Delaunay triangulation intersecting $[st]$ are used.*

In Fig. 2.3, the triangles used by the algorithm consist of two edges drawn as black and one edge as part of the path drawn as green. The below corollary is also stated in [1, 4].

COROLLARY 1. *The algorithm will always terminate, and it will output a path from s to t.*

As in Fig. 3.3, putting a huge obstacle into the graph intersecting $[st]$ could instantly break this chain and give us two totally separated triangulations such that s,t are invisible to each other. Therefore, we assume that the obstacles never intersect the line segments $[st]$.

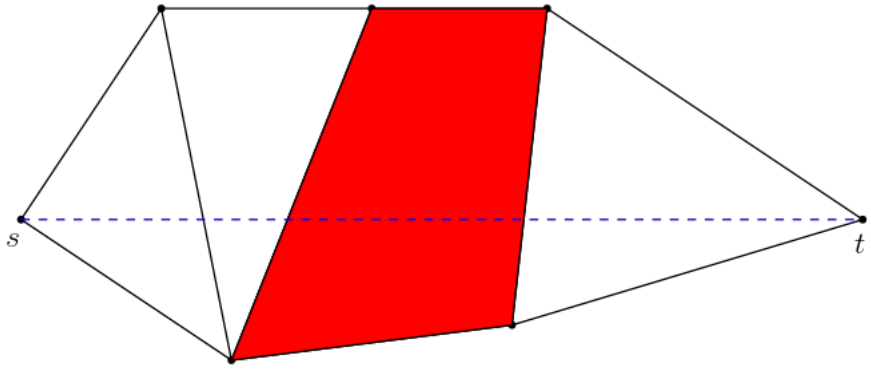


Figure 3.3. The red simple polygon obstacle breaks the chain of triangles intersecting $[st]$.

Now we have defined the shape of obstacles and the way they exist, but how will Delaunay triangulation be impacted? What does it mean if two vertices are invisible to each other? Let us move to the next section.

3.2 Delaunay Triangulations in the Presence of Obstacles

Among all properties, the most important one for Delaunay triangulations is the empty circle property, that is, given a Delaunay triangulation $\mathcal{DT}(V)$ based on a set of points V, no vertex in V lies inside the circumcircle of the circle defined by the points of any triangle in $\mathcal{DT}(V)$. This property ensures many advantages of Delaunay triangulations. However, now with obstacles, it does not hold anymore. As in Fig. 3.4, given a Delaunay triangulation $\mathcal{DT}(V)$ with obstacles, the circumcircles of $\triangle abc$ and $\triangle def$ contain each other's vertices, which is not allowed in Delaunay triangulations. However, now the obstacle $bcfd$ between them makes these two triangles invisible to each other, which means this is allowed to happen in the presence of obstacle $bcfd$, as constrained Delaunay triangulations forbid the existence of visible points in these circles.

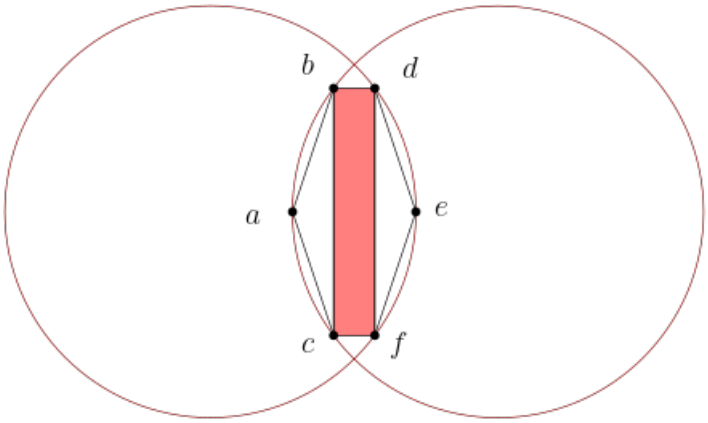


Figure 3.4. Circumcircles of triangles contain extra vertices in the presence of obstacles.

We notice that if we push a even closer to bc, we can get an extremely thin triangle with $\angle bac$ approximating 180°, thus there would be a huge circumcircle pointing to the right. This could happen in Delaunay triangulations without obstacles at some thin triangle on the boundary of the convex hull, but the circumcircle would point to nothing but an edge of the convex hull. Now with obstacles, it can happen in the middle of the graph.

3.3 Conjectures

We have defined the obstacles and observed that the crucial circumcircle property of Delaunay triangulations no longer holds in their presence. In this section, we explore some initial conjectures that we

anticipated would provide shortcuts. However, these conjectures were eventually disproved by corresponding examples.

3.3.1 Path May Go Above Obstacles

The most intuitive way to route that comes to mind is that, with the assumption that no obstacle can intersect $[st]$ which means s,t are visible to each other. If we could prove that the routing algorithm will never go around obstacles, does it mean the remaining graph enclosed by the path and $[st]$ would very likely be the same as a Delaunay triangulation without obstacles? Then, we could apply Chew's algorithm directly, and the bounds proved by Bonichon et al. in [4] still hold.

However, here we build a counterexample to show that the original routing algorithm can actually go around the obstacles. As in Fig. 3.5, consider such a Delaunay triangulation, we let the red small triangle be an obstacle, which does not intersect $[st]$, therefore the obstacle exists in the area enclosed by the path and $[st]$. The path starts from s, following the green path generated by Chew's routing algorithm and then goes around the obstacle by routing on a long triangle intersecting $[st]$ and above the obstacle.

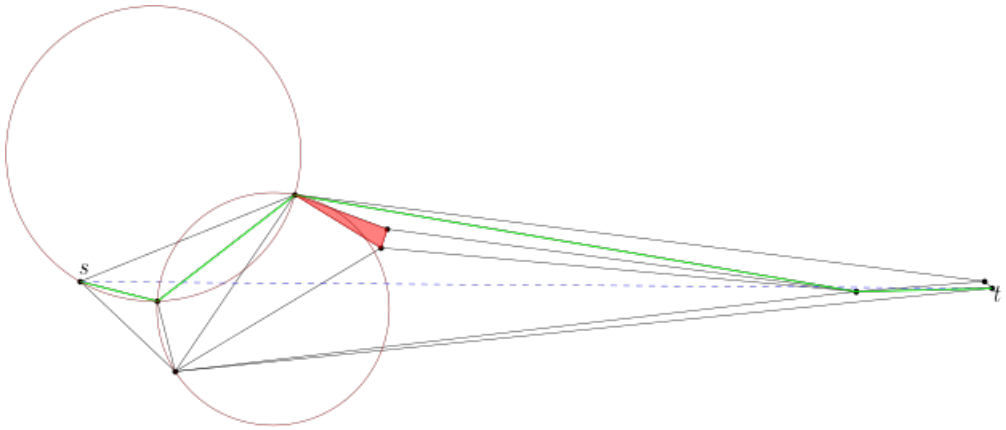


Figure 3.5. The path goes above the red obstacle.

Therefore, if we want to ensure that our algorithm does not go around obstacles, we need to apply some adaptation to Chew's algorithm to see if it is possible to let the algorithm make decisions that always go below obstacles.

However, when trying to adapt the algorithm to avoid going around obstacles, we noticed that the algorithm might not even be able to detect obstacles using only local information. Here we show an example that the algorithm is not able to know the obstacles using local information. Consider a similar Delaunay triangulation in Fig. 3.6, we let the tiny red triangle be the obstacle. Then by observation, the path

generated by Chew's algorithm here goes around the obstacle without even meeting it, which means the algorithm could go around the obstacle without detecting its existence.

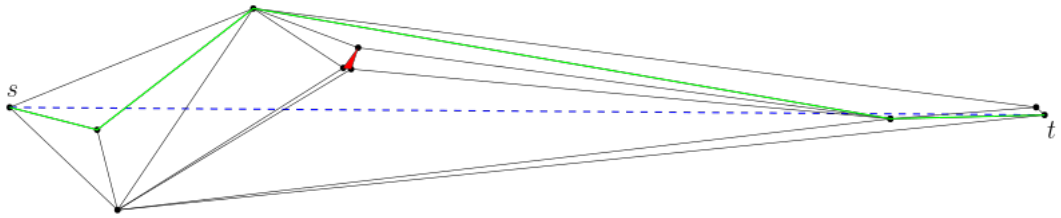


Figure 3.6. The path goes above the red obstacle without detecting its existence.

More specifically, the path generated will not include any vertex of the obstacle. This means that local information at the current vertex is not sufficient to detect the obstacle. So, the question is, can we use more information, so that the algorithm can detect the obstacles without meeting them?

3.3.2 Local Information Is Not Sufficient to Avoid Going Above Obstacles

If we let the algorithm scan the projecting area, which is the area enclosed by the possible edge of the next step, the vertical lines drawn from its vertices to $[st]$, and $[st]$, the algorithm can decide the next step of a path that does not go around obstacles by choosing another edge to go beforehand.

However, consider the Delaunay triangulation in Fig. 3.7, where we placed points on the upper arc of the circle, let $\triangle st't''$ be the lowest triangle so the path will not jump to $tfroms$ and let $\triangle p_{n-1}$, where n is the total number of triangles in this case, also connect each p_i to p_{i+1},t'. Then for each $[p_ip_{i+1}]$ on the arc, even though the projecting area $p_ip_{i+1}p'_{i+1}p'_i$ seems narrow enough, the adaptation requires the algorithm to keep checking all the triangles intersect it, which means it will need to check i triangles that it has passed. When it reaches p_n, the last vertex connects to t, if we let t' also approach t, then $n-1$ triangles in the whole graph need to be scanned only except $\triangle st't''$. As we have to constantly check i triangles that intersect $p_ip_{i+1}p'_{i+1}p'_i$ in each step, and the worst case of i here is n, which is not ideal for online algorithms that usually have $O(n)$ time complexity and $O(1)$ space complexity.

Besides, checking triangles that intersect an area is not possible without knowing the whole graph. Therefore, such adaptations are not acceptable because online algorithms should not get more than local information during routing. As these modifications seem to not have the desired effect or properties, we now focus our attention on Chew's original algorithm and evaluate its performance in the presence of obstacles.

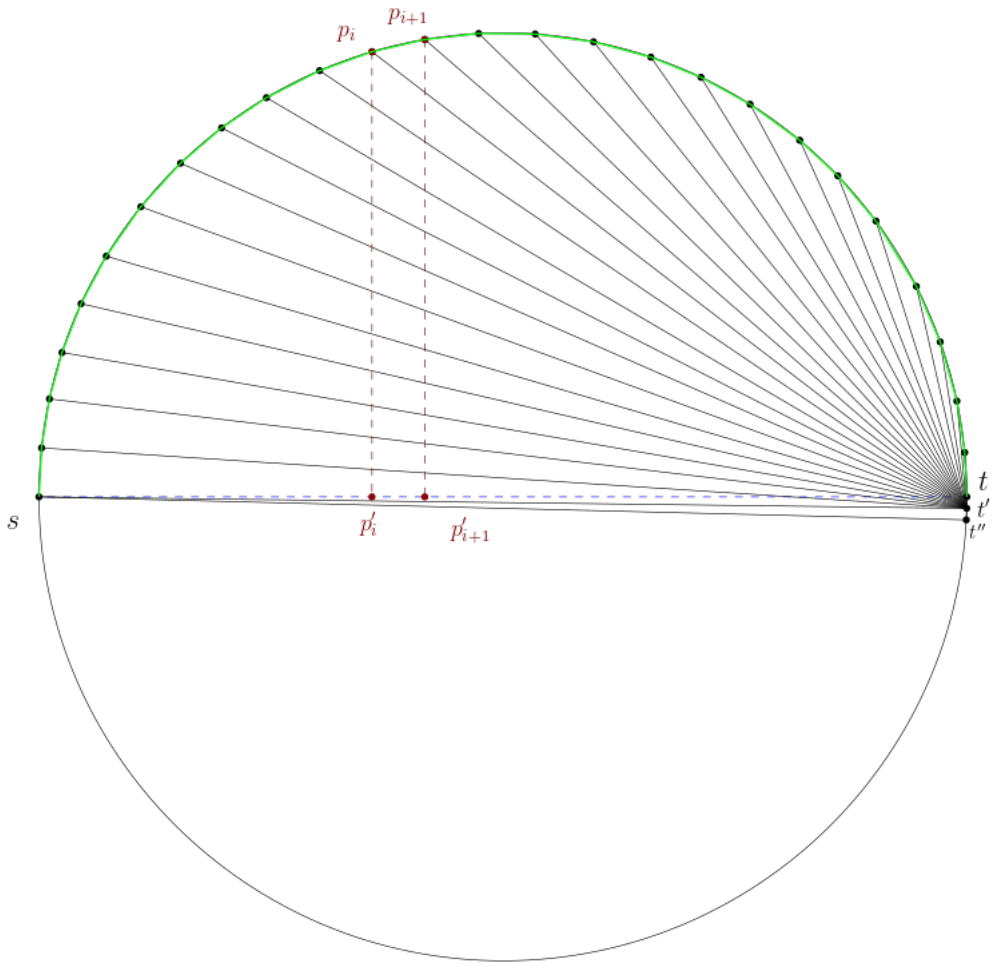


Figure 3.7. The projecting area of $[p_i p_{i+1}]$ to $[st]$ could include i triangles as in each step, and include $n-1$ triangles as p_{i+1} approaches t.

CHAPTER 4

Reachability

Although Corollary 1 is already stated in [1, 4] to show Chew's algorithm terminates and produces a path from s to t, here we still need a proof to show that the reachability of Chew's algorithm is not compromised by the presence of obstacles.

In this Chapter, we focus on analysing how Chew's algorithm move from vertex to vertex, and the consistent existence of a rightmost triangle T_i, thus proving the reachability of Chew's algorithm in the presence of obstacles.

4.1 The Existence of Consecutive Set of Triangles

As Chew's algorithm uses a consecutive set of triangles, we first show that such a set always exists. Recall that in Chapter 3, to make sure s and t are visible to each other, we set that no obstacles could intersect with $[st]$.

COROLLARY 2. *By the assumption that no obstacles intersect with $[st]$, any edge in the triangulation that intersects with $[st]$ must be an edge of a triangle that intersects $[st]$.*

LEMMA 2. *Given a Delaunay triangulation $\mathcal{DT}(V)$ with obstacles of point set V, source and target vertex s,t, with the assumption that no obstacles intersect $[st]$. There always exists a consecutive set of triangles $\mathcal{T}$ intersect $[st]$.*

PROOF. By Corollary 2, starting from the first edge contains s, we can always find the triangle contains itself and another edge intersects $[st]$ to the right of it, therefore we can repeatedly go to the right edge and eventually reach the last triangle intersect $[st]$ at t. Thus such a consecutive set of triangles that intersect $[st]$ always exists. □

4.2 Chew's Algorithm Always in the Right Direction

LEMMA 3. *Ordering the triangles by their rightmost intersecting with* $[st]$*, Chew's algorithm will always visit these triangles from left to right it may slip triangles and at least be able to use an edge of the triangle right next to the current vertex.*

PROOF. This lemma is illustrated in Fig. 4.1. As only triangles intersecting with $[st]$ are considered when routing, therefore, for the current vertex c, the current rightmost triangle $\triangle T$ has 3 vertices c, p_1, p_2. c can only be on the side of $[st]$ which has 2 vertices of T and on the edge that has intersection with $[st]$ on the left. We prove this by contradiction.

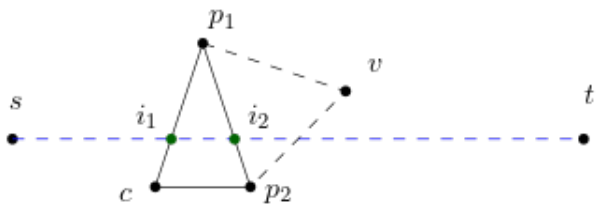


Figure 4.1. The current vertex of current rightmost triangle intersects witch $[st]$ could only be c.

For one situation, assume that the current vertex is p_1, $\triangle cp_1p_2$ is the rightmost triangle, and p_1 lies on the side of $[st]$ which has only has 1 vertex of $\triangle cp_1p_2$. Then there is definitely a triangle $\triangle p_1p_2v$ to the right of $\triangle cp_1p_2$ that includes $[p_1p_2]$ and also intersects $[st]$ at i_2. Hence, $\triangle cp_1p_2$ cannot be the current rightmost triangle, which contradicts the assumption.

For another situation, assume that the current vertex is p_2, it is also guaranteed that there exists a triangle $\triangle p_1p_2v$ that contains $[p_1p_2]$, leading to the same contradiction as above. Therefore the current vertex must be c which is on the 2 vertices side and on the left edge that intersects $[st]$ at i_1 which is to the left of another intersection of edge.

Furthermore, by observation, no matter whether vertex p_1 or p_2 it is the next vertex on the routing path, it is guaranteed that the algorithm always finds a new rightmost triangle. Thus the algorithm will always be able to visit these triangles from left to right and at least go to the triangle right to the current one.

□

4.3 For Triangles Including s, t

As s,t are both on $[st]$ we need to take special care of them. In the first step of the routing algorithm, the algorithm starts at s, thus the edges including s intersect $[st]$ on the leftmost position. Therefore this case can be treated as if it lies on both sides of $[st]$. Then Lemma 3 also holds for this case.

For triangles intersect $[st]$ including t, this triangle intersects $[st]$ on t which is the rightmost position of $[st]$. Thus the triangle is the rightmost triangle of the graph. Lemma 3 implies that the algorithm eventually gets to this triangle and route to t since the algorithm will go to t if it can.

THEOREM 2. *The reachability of Chew's algorithm is not compromised by the presence of obstacles.*

PROOF. By Lemma 2 and 3, along with special cases solved above about triangles including s,t, we show that Chew's routing algorithm in the presence of obstacles terminates, produces a path from s to t. □

CHAPTER 5

Upper Bound

Now we see that though online routing on Delaunay triangulations in the presence of obstacles seems difficult to analyse, at least we can prove the reachability of Chew's algorithm is not compromised first. With this, we now can start thinking about the performance, but we bring obstacles into consideration, we need a bit of context in more depth about Chew's algorithm.

In this chapter, we first discuss a crucial property of Delaunay triangulations that does not hold in the presence of obstacles, and its impact on Chew's algorithm, and Bonichon et al.'s proof of upper bound in [4]. Furthermore, we identify the lemmas that are broken or need to be adapted and show how to prove them with the existence of obstacles, thus conclude the upper bound is not compromised.

5.1 Routing Back in the Circumcircle

Recall that the following lemma is a crucial property of Delaunay triangulations,

LEMMA 4. *Given a Delaunay triangulation $\mathcal{DT}(V)$ of points set V, for any triangle $\triangle T$ in $\mathcal{DT}(V)$, there is no point p in V that lies inside the circumcircle of $\triangle T$.*

PROOF. This lemma follows the definition of Delaunay triangulations. □

However, in Chapter 3, we show that in the presence of obstacles, this property does not hold anymore (see Fig. 3.4), and the path inevitably goes above obstacles into the area that this property does not hold.

Furthermore, as shown in Fig. 5.1, given such a Delaunay triangulation with the red polygon as an obstacle, the path from s to t generated by Chew's algorithm has p_{i+2} inside the circumcircle C_i.

Why does this look so problematic for the bounds? If we look partially into the part from p_i to p_{i+2}, the routing ratio of this part could be further worsened if we squeeze p_i, p_{i+2} horizontally and elevate

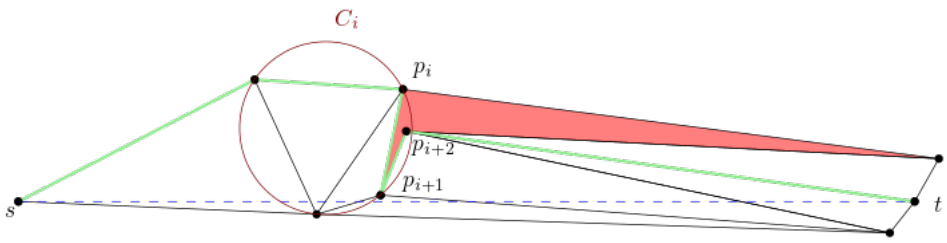


Figure 5.1. Green path from s to t goes along p_i, p_{i+1}, p_{i+2} with p_{i+2} lies inside circumcicle C_i.

them vertically. Though, by observation, we see that the circumcircle on both sides will be bigger if we do this, and further push the possible position of s,t away. However, it is not convincing nor feasible to consider all possible shapes of obstacles.

More specifically, the best-known upper bound of the routing ratio without obstacles, 5.90, is proved by Bonichon et al. [4] in 2017. In their proof, Lemma 4 is used to prove the key lemmas, which are later incorporated to prove Theorem 1, that is the upper bound of routing ratio. As this lemma breaks in the presence of obstacles, does it mean the upper bound breaks as well? We can finally look at the upper bound of the routing ratio now.

5.2 Bonichon et al.'s Upper bound

Because our proof is heavily based on Bonichon et al.'s proof in [4], we start by introducing additional definitions and notions to look into the original proofs.

Recall that as shown in Fig. 2.3, we denote by $s = p_0, p_1, ..., p_i, p_k = t$, as the vertices visited by Chew's algorithm to generate the path. Also we denote by $\mathscr{P}(p_i, p_{i+1})$ the path from p_i to p_{i+1}. We let C_i be the circumcircle of the rightmost triangle T_i used by the algorithm at p_i, and we denote w_i the leftmost point of C_i, r_i the intersection point of C_i with $[st]$ and O_i the center of C_i.

We further denote by $\mathscr{A}_i\langle p_i, p_{i+1}\rangle$ the corresponding oriented arc on C_i used by the algorithm to route on edge (p_i, p_{i+1}) (see Fig. 5.2,5.3).

The main idea in [4] to upper bound the routing ratio of Chew's algorithm, is to analyze three types of possible positions of p_i, p_{i+1} along with the worst case and combination to show the routing ratio is always bounded by 5.90 between any two points.

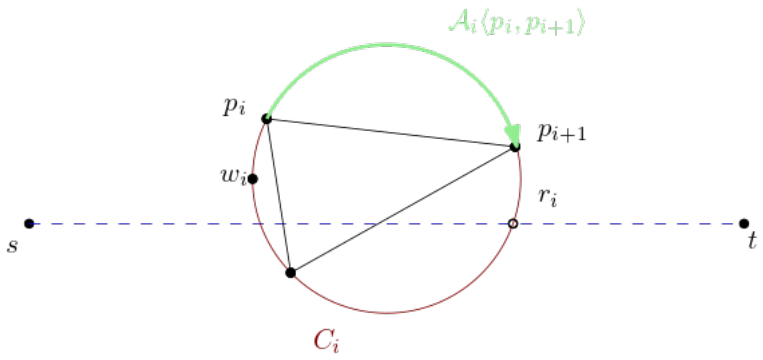


Figure 5.2. The algorithm routes from p_i to p_{i+1} clockwisely using the upper arc, $\mathscr{A}_i\langle p_i, p_{i+1}\rangle$ is also in clockwise direction on the upper arc of C_i.

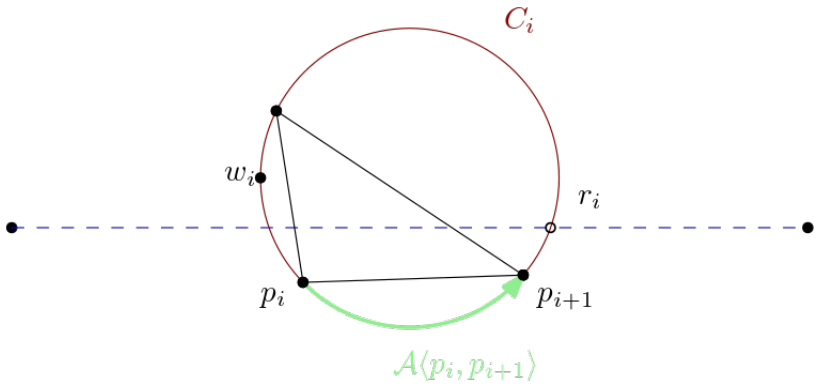


Figure 5.3. The algorithm routes from p_i to p_{i+1} counter-clockwisely using the lower arc, $\mathscr{A}_i\langle p_i, p_{i+1}\rangle$ is also in count-clockwise direction on the lower arc of C_i.

5.2.1 The Worst Cases of C_i'

In order to bound the length of $\mathscr{A}$, *Worst Case Circles* C_i' are introduced in [4] that also go through p_i, p_{i+1} with center O_i' obtained by starting at O_i, moving along the perpendicular bisector of $[p_i, p_{i+1}]$ to the direction of $\mathscr{A}_i\langle p_i, p_{i+1}\rangle$, until C_i' is tangent to line st or p_i is the leftmost point of C_i' (i.e. $p_i = w_i'$).

By definition, the possible situations of C_i' can be generalized to three cases:

Case A_1: $p_i \neq w_i'$, $[p_i, p_{i+1}]$ does not cross $[st]$, and C_i' is tangent to $[st]$. (See Fig. 5.4)
Case A_2: $p_i = w_i'$ and $[p_i, p_{i+1}]$ does not cross $[st]$. (See Fig. 5.5)
Case B: $p_i = w_i'$ and $[p_i, p_{i+1}]$ crosses $[st]$. (See Fig. 5.6)

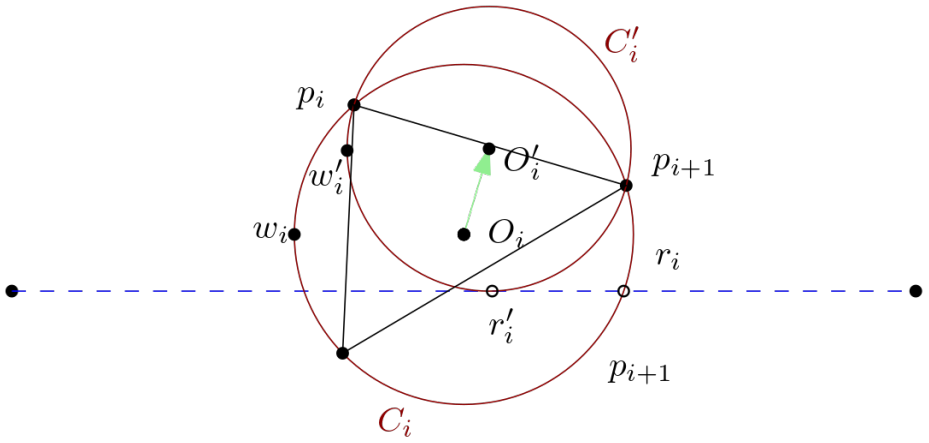


Figure 5.4. Illustration of Type A_1 of C_i'.

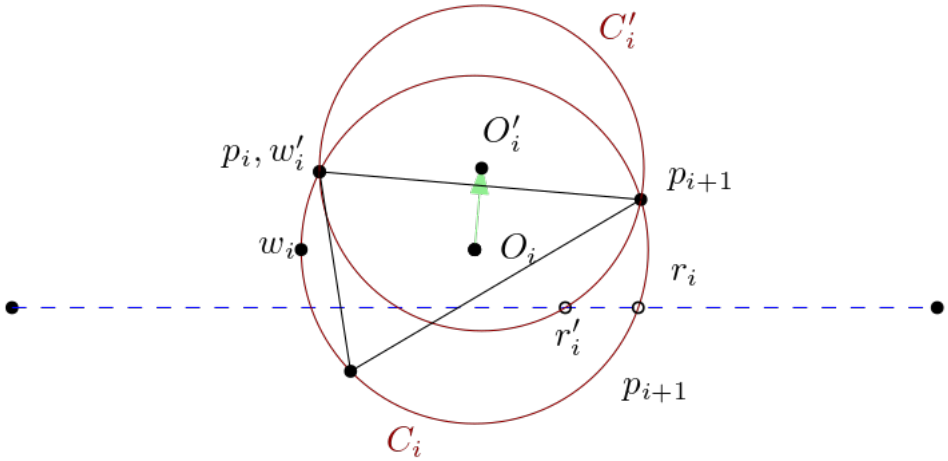


Figure 5.5. Illustration of Type A_2 of C_i'.

Now we also define $\mathscr{A}_i'\langle p_i, p_{i+1}\rangle$ as the oriented arc on C_i' according to p_i, p_{i+1} similar to $\mathscr{A}_i\langle p_i, p_{i+1}\rangle$.

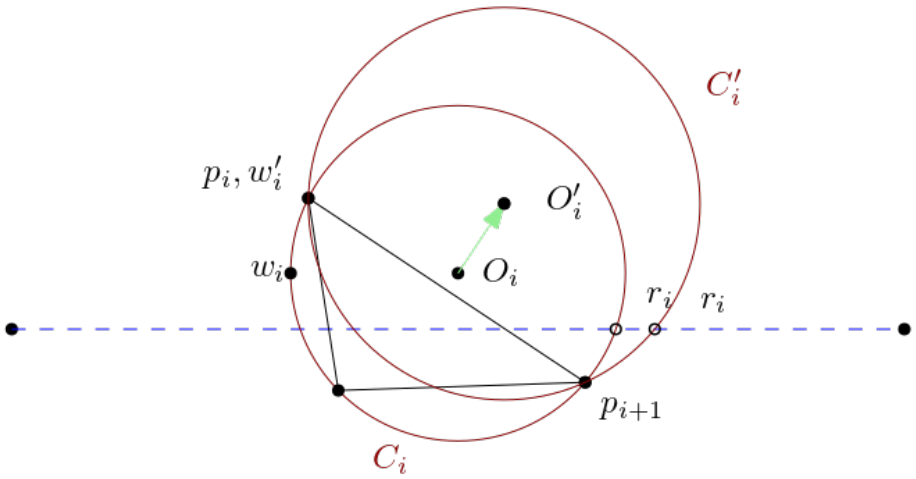


Figure 5.6. Illustration of Type B of C_i'.

Therefore, by construction of C_i' and definition of $\mathscr{A}_i\langle p_i, p_{i+1}$ we get,

COROLLARY 3. $|\mathscr{A}_i'\langle p_i, p_{i+1}\rangle| \geq |\mathscr{A}_i\langle p_i, p_{i+1}|\rangle \geq |[p_i p_{i+1}]|$

The following three lemmas are proved as Lemma 2, 3, and 5 in [4],

LEMMA 5. *Let angles $\angle w_i O_i p_i$ and $\angle w_i O_i p_{i+1}$ be defined using the orientation of $A_i\langle p_i, p_{i+1}\rangle$ for any $i = 0, \ldots, k-1$. Then, the following inequalities hold:*

$$0 \leq \angle w_i O_i p_i \leq \angle w_i O_i p_{i+1} \leq \frac{3\pi}{2}.$$

When $[p_i p_{i+1}]$ crosses $[st]$, this can be strengthened to:

$$0 \leq \angle w_i O_i p_i \leq \angle w_i O_i p_{i+1} \leq \pi.$$

LEMMA 6. *Given C_{i-1} and C_i as circles such that the orientations of $A_{i-1}\langle p_{i-1}, p_i\rangle$ and $A_i\langle p_i, p_{i+1}\rangle$ are the same, and defining angles $\angle w_{i-1} O_{i-1} p_i$ and $\angle w_i O_i p_i$ using that orientation, we have:*

$$\angle w_i O_i p_i \leq \angle w_{i-1} O_{i-1} p_i.$$

LEMMA 7. *Let angles $\angle w_{i-1}' O_{i-1}' p_i$ and $\angle w_i' O_i' p_i$ be defined using the orientations of arcs $A_{i-1}'\langle p_{i-1}, p_i\rangle$ and $A_i'\langle p_i, p_{i+1}\rangle$, respectively. Then, for every $i = 1, \ldots, k-1$, the following inequalities hold:*

$$0 \leq \angle w_i' O_i' p_i \leq \angle w_{i-1}' O_{i-1}' p_i \leq \frac{3\pi}{2}.$$

When C'_{i-1} is of type B, this can be strengthened to:

$$0 \leq \angle w'_i O'_i p_i \leq \angle w'_{i-1} O'_{i-1} p_i \leq \pi.$$

As Lemma 5 and 6 are all proved simply based on the definition of Chew's algorithm in the original proof, they are not affected by the presence of obstacles. Furthermore, Lemma 7 is proved simply by using Lemma 5 and 6, therefore it holds in the presence of obstacle as well.

5.2.2 $\mathscr{P}_i$, and *Snail Curve* $\mathscr{S}$

We still need a couple of definitions before we look into to original proof. We denote by $\bar{p}$ the perpendicular projection of p on $[st]$, and we define $\mathscr{P}_i$ to be $[\bar{w'_i} w'_i] + \mathscr{A}'_i \langle w'_i, p_i \rangle$ for $0 \leq i \leq k-1$ (see Fig. 5.7). Besides, given p, q such that $x(p) < x(q)$ and $y(p) = y(q)$, $[pp']$ perpendicular to $[st]$, and a circle C with radius $R = |[pq]|$, tangent to $[st]$ at q and tangent to $[pp']$ at p', we denote by $\mathscr{S}(p,q)$ the *Snail Curve* from p to q, that is $[pp'] + \mathscr{A} \langle p', q \rangle$ as shown in Fig. 5.8.

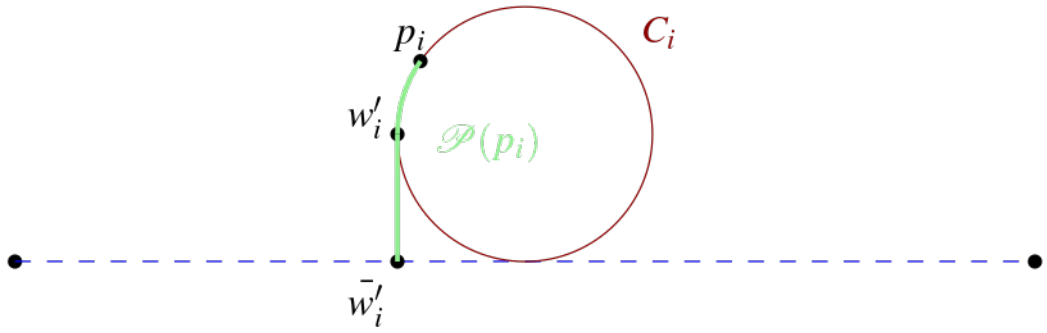


Figure 5.7. Illustration of $\mathscr{P}_i$.

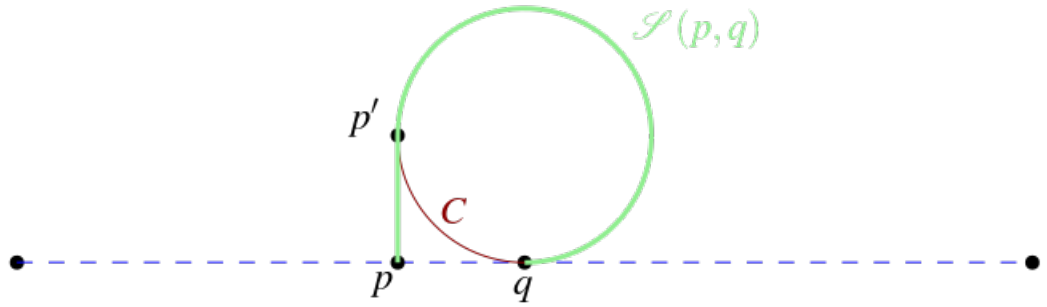


Figure 5.8. Illustration of $\mathscr{S}(p,q)$.

5.2.3 The Broken Lemmas

We also define f_i, to be the first vertex on the path visited by the algorithm after p_i, such that $[p_i, f_i]$ intersects $[st]$.

Now we can finally say something about the broken lemmas in the original proof, not that the following lemmas are implied in [4] as Lemma 4,6,7,8.

LEMMA 8. *Given P as vertices visited by Chew's algorithm to generate a path, no point of P lies inside the region bounded by* $[p_i p_{i+1}] + \mathscr{A}'_i\langle p_{i+1}, p_i\rangle$ *for every* $i = 0, \ldots, k-1$.

In the original proof, this lemma is simply implied by the region bounded by $[p_i p_{i+1}] + \mathscr{A}'_i\langle p_{i+1}, p_i\rangle$ is always inside C_i, and by Lemma 4, there is no other point in P lies inside C_i. However, recall that in Fig. 5.1, we show that Lemma 4 does not hold any more in the presence of obstacles, as the path can route back into the previous C_i. Therefore, Lemma 8 does not hold as well.

The following lemmas are implied by Lemma 6, 7, 8 in Bonichon et al.'s proof [4], and further used to prove Theorem 1,

LEMMA 9. *For all* $0 < i \leq k$*:*

$$x(w'_{i-1}) \leq x(w'_i) \leq x(f_{i-1}) \leq x(f_i).$$

LEMMA 10. $|\mathscr{A}'_{k-1}\langle p_{k-1}, t\rangle| \leq |\mathscr{S}_{w'_{k-1},t}| - |\mathscr{P}_{k-1}|$.

PROOF. This follows from the fact that path $\mathscr{P}_{k-1} + \mathscr{A}'_{k-1}\langle p_{k-1}, t\rangle$ from w'_{k-1} to t is convex and inside $\mathscr{S}(w'_{k-1}, t)$. □

LEMMA 11. *For all* $0 < i < k$ *and* $\delta = 0.185043874$*,*

$$|\mathscr{A}'_{i-1}\langle p_{i-1}, p_i\rangle| \leq |\mathscr{P}_i| - |\mathscr{P}_{i-1}| + |\mathscr{S}_{w'_{i-1},w'_i}| + |y(f_i)| - |y(f_{i-1})| + \delta|[f_{i-1}f_i]|.$$

Though Lemma 10 go through as well by observation, we notice that Lemma 9,11 are both proved using Lemma 8 that is broken in the presence of obstacles, and do not hold in this case. Does this mean Theorem 1 is also broken in the presence of obstacles? Not really, we later show how we prove these two lemma without using 8.

5.3 Upper Bound in the Presence of Obstacles

In this section, because the proof by Bonichon et al. [4] uses Lemma 9, 10, and 11 to calculate the final upper bound 5.90, we focus on the lemmas affected by the presence of obstacles. Specifically Lemma 9 and 11, that need to be adapted in our case.

5.3.1 Adapting the First Use of Lemma 8

The original proof of Lemma 9 in [4] here holds still for the first two inequalities, and Lemma 8 is used to prove only the third equality, $x(f_{i-1}) \leq x(f_i)$. We now argue this inequality instead as follows:

Recall that f_i is the first vertex on path visited by the algorithm after p_i, such that $[p_i f_i]$ intersects $[st]$. In order to prove $x(f_{i-1}) \leq x(f_i)$, we first assume p_{i-1} and p_i are on different sides of $[st]$ (otherwise this trivially hold as $f_{i-1} = f_i$). Then, without loss of generality, we assume p_{i-1} lies above $[st]$ and p_i lies below $[st]$, thus $p_i = f_{i-1}$.

LEMMA 12. *The path cannot move leftward through* $[st]$.

PROOF. Illustrated in Fig. 5.9, we prove this by contracdiction. Given $[p_{i-1}p_i]$ crosses $[st]$ and $x(p_i) > x(p_{i+1})$, to route from p_{i-1} to p_i, we need C_{i-1} be on the left side of $[p_{i-1}p_i]$, otherwise we get $w_{i-1}O_{i-1}p_i > 3/2\pi$. The latter contradicts Lemma 5. Now with C_{i-1} on the left side, p_{i-1} lies on the $\mathscr{A}(w_{i-1}r_{i-1})$, thus routes from p_{i-1} to p_i in clockwise direction. The third vertex of the rightmost triangle T_{i-1} containing p_{i-1} is on the left of $[p_{i-1}p_i]$, we see there exists a triangle also containing $[p_{i-1}p_i]$ and intersect $[st]$ that lies to the right of $[p_{i-1}p_i]$. This contradicts T_{i-1} is the rightmost triangle on p_{i-1}, and thus the algorithm would not have used T_{i-1}. □

LEMMA 13. *The path cannot go up and leftward without crossing* $[st]$.

PROOF. Given p_{i-1}, p_i are on the same side of $[st]$, $x(p_i) \leq x(p_{i-1})$ and $|y(p_i)| > |y(p_{i-1})|$, therefore no matter $\mathscr{A}\langle p_i - 1,\ p_i\rangle$ is in clockwise or counter-clockwise direction, we get $\angle w_{i-1}O_{i-1}p_i > 3\pi/2$, which contradicts Lemma 5. □

Now we try to build an example where $x(f_{i-1}) > x(f_i)$. First, given p_{i-1} above $[st]$ and $p_i = f_{i-1}$ below $[st]$, we need the third vertex of T_{i-1} to lie above $[p_{i-1}p_i]$ so that T_{i-1} is the right most triangle at p_{i-1}. By Lemma 12 and 13, we need an intermediate vertex p_{i+1} also below $[st]$ between p_i and f_i, as otherwise

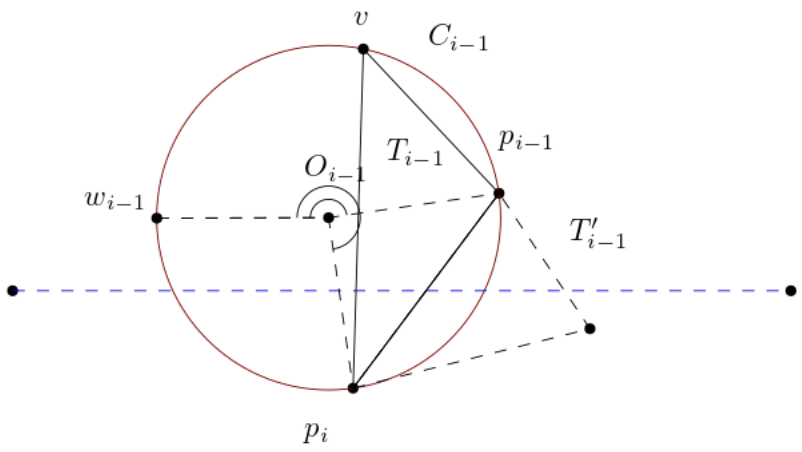


Figure 5.9. Path cannot go backward through $[st]$.

$x(f_{i-1}) \leq x(f_i)$. Without loss of generality, we assume $x(p_i) > x(p_{i+2}) \geq x(p_{i+1})$ and $[p_{i+1}p_{i+2}]$ cross $[st]$.

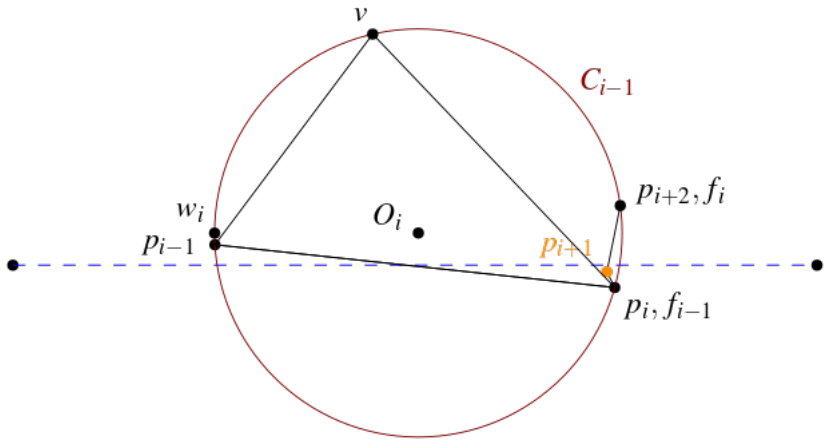


Figure 5.10. Intermediate vertex p_{i+1} lies inside C_{i-1} and is visible to v, p_i.

As in Fig. 5.10, v is above $[p_{i-1}p_i]$, so $\measuredangle\langle p_{i-1}, p_i\rangle$ is in counter clockwise order, thus p_{i-1} is below w_{i-1}. Therefore, given $x(p_{i+1}) < x(p_i)$ and p_{i+1} lies between $p_i, [st]$, p_{i+1} must lies inside C_{i-1} because O_i lies above $[st]$.

The following lemma is implied by the more general Lemma 3 by Bose et al. [3]:

LEMMA 14. *Given two vertices p and q that can see each other, and a circle C with p and q on its boundary, if there exists any another vertex x in C such that p sees x, then there exists another vertex y such that p and q both see y.*

By Lemma 14, to make this graph a valid constrained Delaunay triangulation, p_{i+1} should not be visible to v and p_i, otherwise there exists some visible vertex inside C_{i-1}. However, it is impossible to add polygon obstacles that make p_{i+1} invisible without crossing $[st]$ or adding new visible vertices, (the latter follows from Lemma 14).

Therefore, we show that such an intermediate vertex p_{i+1} cannot exist, so the third inequality holds without Lemma 8, thus completing the proof of Lemma 9.

5.3.2 Adapting the Second Use of Lemma 8

In the proof of Lemma 11 by Bonichon et al. [4], Lemma 8 is used to show that the lemma holds for *Case 3* when C'_{i-1} is of type B and C'_i is of type A or B, with the assumption $|y(p_{i-1}) - y(p_i)| \geq |y(p_{i-1}) - y(f_i)|$.

First, we show an example where f_i lies inside C'_{i-1} in the presence of obstacles in Fig. 5.11. The path generated by Chew's algorithm is shown in light green, and a red polygon obstacle with all vertices on the boundary of C'_1. Therefore, the area to the right of the obstacle is not visible to p_1, p_2 that is inside the area to the left of the obstacle in C'_1. Then along with an intermediate vertex p_3, the path goes back into C'_1 to $p_4 = f_2$. Thus showing that f_i always lies outside C'_{i-1}, as was done in the original proof, does not hold in the presence of obstacles.

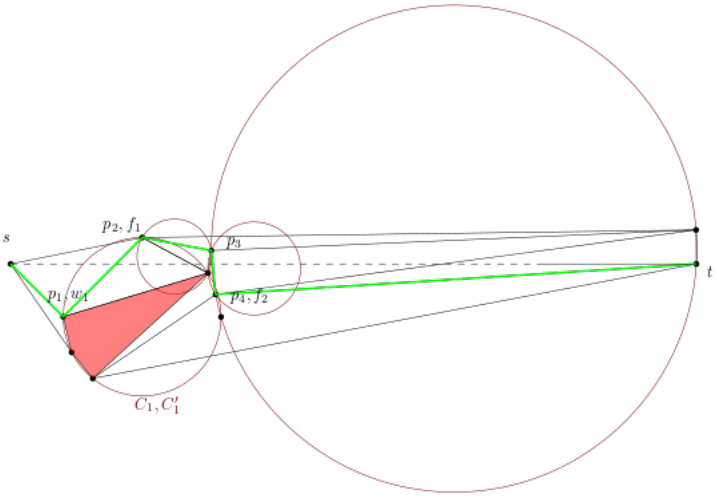


Figure 5.11. Example that f_i exists in C_{i_1}, f_2 lies inside C_1.

As in Fig. 5.12, let p'_i be the intersection of the horizontal line through p_i with C'_{i-1}, and let p''_i be the intersection of the vertical line through p'_i with C'_{i-1}. It is implied by Lemma 8 that no point p on the

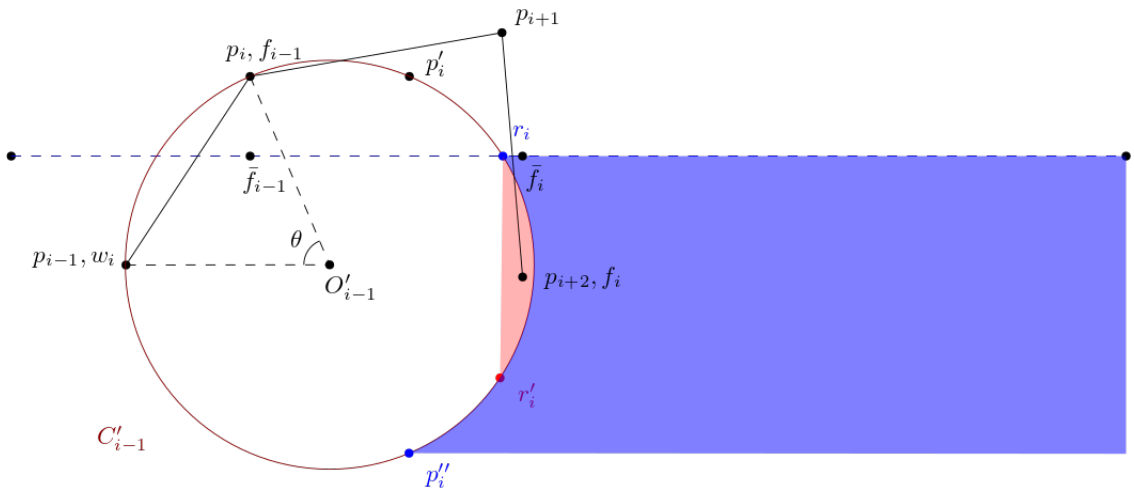


Figure 5.12. The original proof bounds f_i inside the blue area with assumption, while we bound f_i inside the blue area plus the area enclosed by a chord r_i, r'_i and $\mathscr{A}\langle r_i, r'_i\rangle$

path lies inside C'_{i-1}, along with the assumption $|y(p_{i-1}) - y(p_i)| \geq |y(p_{i-1}) - y(f_i)|$, the original proof bounds the possible position of f_i to be inside the blue area to the right of $\mathscr{A}\langle r_i, p''_i\rangle$, thus showing that $|[f_{i-1}f_i]| \geq |[f_{i-1}p'_i]| = 2\cos\theta$ holds here. We now argue this instead as follows.

First, as in Fig. 5.11, the existence of f_i requires an obstacle also inside C'_{i-1} to block the visibility, thus requires at least one intermediate vertex to route to f_i. By Lemma 14, the vertices of the obstacle should also be on the boundary of C'_{i-1} to avoid any new visible vertex. Therefore, such an obstacle can only exist on the same side of f_i as p_{i-1} and p_i are type B ($[p_{i-1}p_i]$ crosses $[st]$).

Without loss of generality, we assume there exists one intermediate point p_{i+1} and an obstacle that is thin enough to be treated as a chord (see Fig. 5.12). As the obstacle cannot cross $[st]$, it can at most approach r_i to achieve maximum flexibility for the position of f_i. Then, by Lemma 12, the intermediate point p_i can be placed along the line $x = x(r_i)$ to achieve maximum flexibility for the position of f_i. Therefore, let r'_i be the intersection of $x = x(r_i)$ with C'_{i-1}. Since by Lemma 12, the path cannot go left while crossing $[st]$, we have shown that we can bound f_i inside the light red area enclosed by chord $r_i r'_i$ and $\mathscr{A}\langle r_i, r'_i\rangle$ plus the original blue area in this case.

As any f_i inside the area has $x(f_i) > x(p'_i)$, therefore we proved that $|[f_{i-1}f_i]| \geq |[f_{i-1}p'_i]| = 2\cos\theta$ still hold without using Lemma 8. As the rest of the original proof and calculation is not affected, here we proved Lemma 11 in the presence of obstacles.

THEOREM 3. *The routing ratio of Chew's routing algorithm on the Delaunay triangulation in the presence of obstacles is at most* $(1.185043874 + \frac{3\pi}{2}) \approx 5.90$.

PROOF. We proved Lemmas 9 and 11 in the presence of obstacles, and Lemma 10 is not affected by the presence of obstacles, therefore the calculation and proof of the routing ratio go through in the presence of obstacles. □

Thus, we show that Chew's routing algorithm on the Delaunay triangulation in the presence of obstacles has a routing ratio of at most $(1.185043874 + \frac{3\pi}{2}) \approx 5.90$. Which also translates to a competitive ratio of 5.90.

CHAPTER 6

Lower Bound

As a main result in [4], the lower bound of routing ratio of Chew's algorithm on Delaunay triangulations is at 5.72, which is close to the upper bound of 5.90 . In the presence of obstacles, we try to shed some light on the lower bound in our case, though so far we were not able to improve on the existing bound.

In this section, we focus more in detail on the movement of the algorithm closely related to the obstacles, giving some cases that show why such a lower bound is even more difficult to beat in the presence of obstacles.

6.1 Bonichon et al.'s Lower Bound of Routing Ratio

Here we show how Bonichon et al. [4] find the lower bound of 5.72, the following Delaunay triangulation is re-drawn using the same idea:

As in Fig. 6.1, the lower bound 5.72 is slightly bigger than $(1+3\pi/2)|[st]|$, where the path shown in light green from s to p_j approaches $|[st]|$ and from p_j to t approaches $3\pi/2|[st]|$, this is a main lower bound in [4]. Now, we can show that in many cases, paths that go around obstacles behave similarly to this bound as well.

6.2 Cases When Path Goes Above Obstacles.

Recall that in Chapter 3, we show some cases that the paths go above obstacles, now we look into detail about this case.

If the path goes above the obstacles, whether it eventually goes over it or not, it is not easy to claim a case is the worst case due to the arbitrary shape of simple polygon obstacles. However, we can still analyse

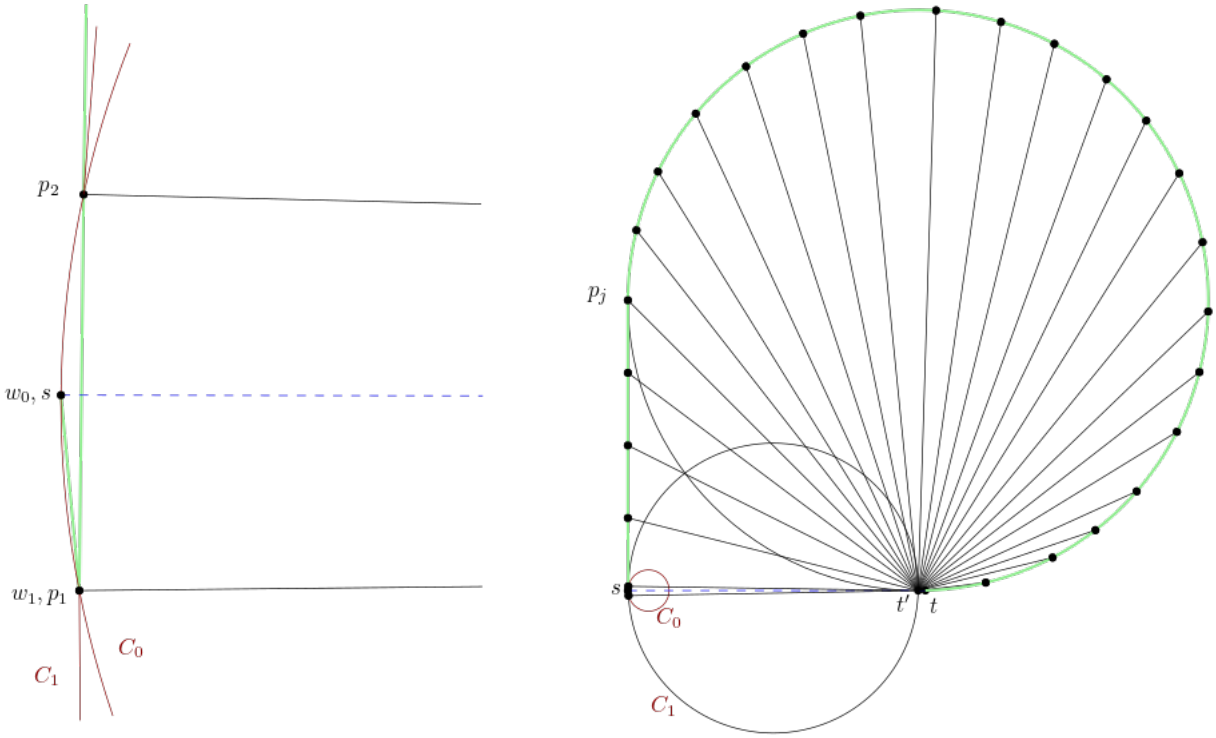


Figure 6.1. Bonichon et al.'s worst case of lower bound

how it enters an area above obstacles, what kind of movement it can perform when above obstacles and how it leaves the area.

6.2.1 Routing Along Edges of Obstacles

If in some way the path can route along edges of obstacles, we may somehow design an obstacle with edges designed to maximize the detour in the path. Therefore we first focus on edges above obstacles that are visible to another side of $[st]$ so that routing triangles containing such edges can exist.

LEMMA 15. *Given an edge $[pq]$ of an obstacle, the path cannot route on p to q if $x(p) < x(q), y(p) < y(q)$ and $[pq]$ lies is above part of the obstacle.*

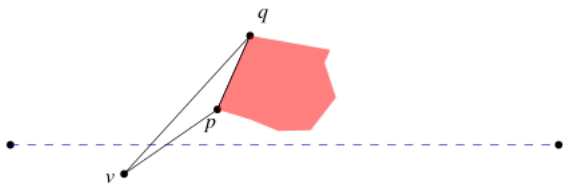


Figure 6.2. Triangles contain $[pq]$ cannot be the rightmost triangle contains p

PROOF. As in Fig. 6.2, given such an edge $[pq]$, it requires a vertex v on the other side of $[st]$ to form a triangle intersecting $[st]$. There always exists a triangle containing $[vp]$ to the right of $\triangle vpq$, therefore the path cannot go from p to q in this case. □

LEMMA 16. *Given an edge $[pq]$ of an obstacle, the path cannot route from p to q if $x(p) > x(q), y(p) > y(q)$ and $[pq]$ lies is above part of the obstacle.*

PROOF. As in Fig. 6.3, similarly, such a required triangle containing $[pq]$ cannot be the rightmost triangle as there always exists a triangle containing $[pv]$ to the right of it. □

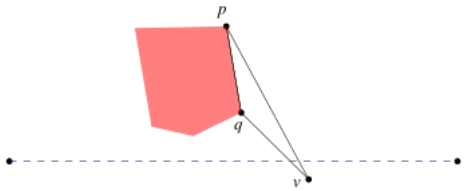


Figure 6.3. Triangles contain $[pq]$ cannot be the rightmost triangle contains p

On another side, if we flip positions of p, q in these two situations, though we can now route on edges above obstacles, we are still not able to go over it and stay in the area that is not impacted by the obstacle. Therefore, we fall into the same lower bound in the original [4].

For edges below obstacles, in Fig. 6.4, assume the path goes along p_{i-1}, p_i, p_{i+1}. If we narrow $[p_{i-1}p_i]$ and p_ip_{i+1}, it seems we get large routing ratio. However, we look at the circumcircle and make it approach tangent to $[st]$ as it must cross $[st]$ to provide a routing triangle. Therefore, the approximate tangent point s' means that source vertex s is also limited to the left of it. Same on $[p_ip_{i+1}]$ for target vertex t. Therefore, we again fall into the original bound.

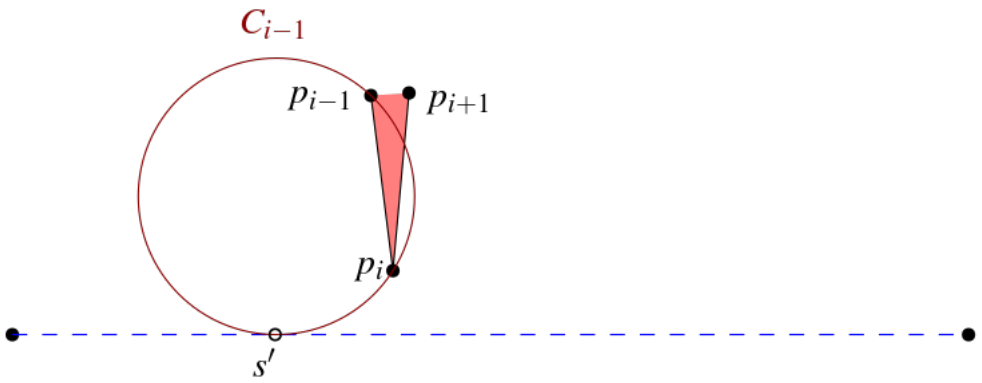


Figure 6.4. C_{i-1} further pushes possible position of s to at least s'

For another situation in Fig. 6.5, given edges of obstacle $[pq], [qr]$, the area below it will be triangulated exactly the same as for Delaunay triangulation. To guarantee the existence of $[qv]$, the position of v is heavily decided by $\angle\theta$ as in normal Delaunay triangulation, and further decides the possible position of s,t.

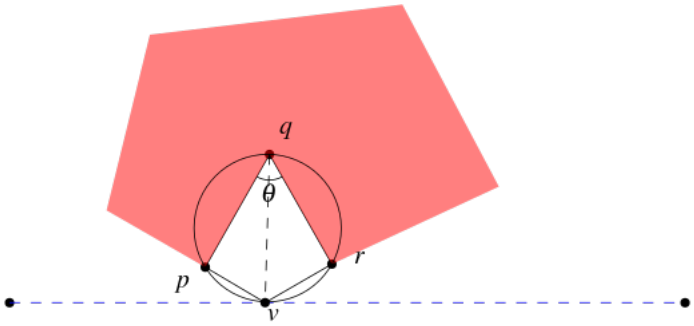


Figure 6.5. To guarantee the existence of $[qv]$, position of v is position of v is heavily decided by $\angle\theta$ as same as in normal Delaunay triangulation

Thus, we show that cases routing along edges are very similar to routing without obstacles. We therefore try to explore situations when the path goes above obstacles.

6.2.2 Path Goes Above Obstacles

Recall that in Chapter 3, we show that the path may go around an obstacle through a vertex of it or without meeting any vertex of it. However, by Lemma 15,16, when a path enters or leaves the area above obstacles, it requires long triangles that further push s,t away as seen in Fig. 6.4 and instantly fall into the original bound. Therefore, we now focus on the case that routing inside the area above obstacles.

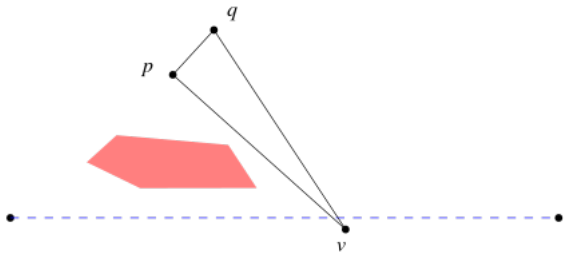


Figure 6.6. q can only be right of p when both vertices are above obstacle

As in Fig. 6.6, assume p is the current vertex. To keep routing above the obstacle, we need v to form a triangle that intersects $[st]$, and q can only exist on the right of pq or $\triangle pqv$ is not the rightmost triangle. Therefore, as p and v are visible to each other, they can at most approach the vertical line in order to

keep p above the obstacle and visible to v. Now we fix p and v, in order to break the routing ratio, we let q also approach the vertical line and v approaches $[st]$. Therefore, in Fig. 6.7, when $y(q) < y(p)$, C_p pushes the obstacles to the far left and when $y(q) > y(p)$, C_p pushes t to far right, which falls into the original bound as well.

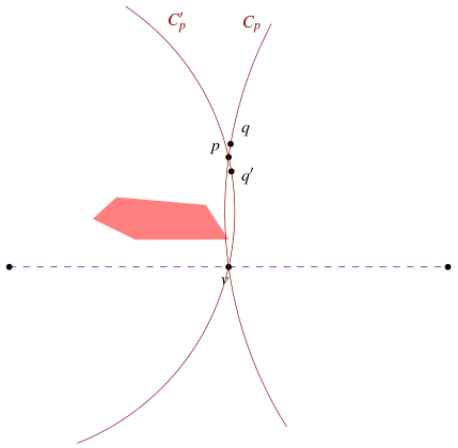


Figure 6.7. Circumcircle C_p, C'_p pushes either obstacle or t far away

There seems no case of routing around obstacles that can beat the current best lower bound found by Bonichon et al. [4]. We therefore believe that obstacles may not suffice to construct worse lower bounds. However, as obstacles can be arbitrary, it is problematic to iterate all cases and convince ourselves that we considered all cases, so we leave this as a conjecture here.

CHAPTER 7

Conclusion

Delaunay triangulations and Chew's routing algorithm have been well studied, with the spanning ratio and routing ratio both bounded nicely and tightly. However, in the presence of obstacles, the performance of them was unknown. Now we solve this puzzle partially, the reachability of Chew's routing algorithm is not compromised, and the upper bound of the routing and competitive ratio is 5.90 in the presence of obstacles, as good as when there is no obstacles. The lower bound of the routing ratio is shown in a more general case without obstacle by Bonichon et al. [4] is 5.72, therefore the upper bound we gave is almost tight. Due to the tightness, we were unable to give cases that improve the 5.72 but leave it as a conjecture that it is unlikely to beat 5.72 in the presence of obstacles. However, our work specifically focuses on Chew's routing algorithm and the Delaunay triangulations in the presence of obstacles, and there are still many other properties, algorithms and graphs waiting to be explored, which may provide better routing bounds.

7.1 Future Work

We look at the whole map of Computational Geometry about routing algorithms and constrained graphs, there are still many interesting topics that inspire us about what we can improve and enlighten us on what we can further contribute.

7.1.1 Tighter Bound

Though the upper bound of the routing ratio of Chew's algorithm on Delaunay triangulations in the presence of obstacles is 5.90 and the lower bound 5.72 is known. However, the question remains whether obstacles can give a worse lower bound.

7.1.2 Algorithms

An algorithm named *MixedChordArc* was proposed by Bonichon et al. in [16] with the upper bound of routing ratio at 3.56, which uses the length of arcs instead of the position of current vertices in Chew's algorithm to decide each move. As it has a better upper bound than Chew's algorithm, we want to ask, does that algorithm still works in the presence of obstacles? What about the performance?

7.1.3 Improved Graphs

In [21], an improved version of the Delaunay triangulation named $\mathcal{LMBDG}(V)$ was introduced with a constant degree at 25 and bounded weight. It is also shown that an adapted version of Chew's algorithm works well on $\mathcal{LMBDG}(V)$ with the same routing ratio as the original version. Therefore, we want to know, what if we put obstacles in it? What if we further apply another algorithm such as *MixedChordArc* on it?

7.1.4 Local Graph

Though as an online algorithm, Chew's algorithm does not need the information of the whole map. However, it still requires the Delaunay triangulations generated with local information stored in each vertex. We now consider a more realistic question, can we do more locally? Li et al. [22] applied a technique to their autonomous harvesting and transportation problems, which generates local Delaunay triangulations using a $3D$ laser scanner. Inspired by this, assuming we are given a robot with only a camera and wheels, can we design a Delaunay triangulation-based graph locally generated based on the current vertex and a set of possible adjacent neighbours, and an algorithm routes to the destination efficiently?

Bibliography

[1] P. Chew, "There is a Planar Graph Almost as Good as the Complete Graph," in *Proceedings of the Second Annual Symposium on Computational Geometry*, 1986, p. 169–177.

[2] P. Chew, "There are planar graphs almost as good as the complete graph," *Journal of Computer and System Sciences*, vol. 39, no. 2, pp. 205–219, 1989.

[3] P. Bose, J.-L. De Carufel, and A. van Renssen, "Constrained generalized Delaunay graphs are plane spanners," *Computational Geometry*, vol. 74, pp. 50–65, 2018.

[4] N. Bonichon, P. Bose, J.-L. D. Carufel, L. Perkovic, and A. van Renssen, "Upper and Lower Bounds for Online Routing on Delaunay Triangulations," *Discrete & Computational Geometry*, vol. 58, p. 482–504, 2017.

[5] M. de Berg, O. Cheong, M. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer Berlin, Heidelberg, 2008.

[6] G. Narasimhan and M. Smid, *Geometric spanner networks*. Cambridge University Press, 2007.

[7] D.-T. Lee and B. J. Schachter, "Two algorithms for constructing a Delaunay triangulation." *International Journal of Computer & Information Sciences*, vol. 9(3), p. 219–242, 1980.

[8] D. P. Dobkin, S. J. Friedman, and K. J. Supowit, "Delaunay graphs are almost as good as complete graphs," in *28th Annual Symposium on Foundations of Computer Science*, 1987, pp. 20–26.

[9] J. M. Keil and C. A. Gutwin, "The Delaunay triangulation closely approximates the complete Euclidean graph," in *Algorithms and Data Structures*, 1989, pp. 47–56.

[10] P. Bose, L. Devroye, M. Löffler, J. Snoeyink, and V. Verma, "Almost all delaunay triangulations have stretch factor greater than $\pi/2$," *Computational Geometry*, vol. 44, no. 2, pp. 121–127, 2011.

[11] G. Xia, "The Stretch Factor of the Delaunay Triangulation Is Less than 1.998," *SIAM Journal on Computing*, vol. 42, no. 4, pp. 1620–1659, 2013.

[12] G. Xia and L. Zhang, "Toward the Tight Bound of the Stretch Factor of Delaunay Triangulations." in *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry*, 2011, p. 175–180.

[13] P. Bose and P. Morin, "Online Routing in Triangulations," *SIAM Journal on Computing*, vol. 33, no. 4, pp. 937–951, 2004.

[14] P. Bose and P. Morin, "Competitive online routing in geometric graphs," *Theoretical Computer Science*, vol. 324, no. 2, pp. 273–288, 2004.

[15] P. Bose, J.-L. Carufel, S. Durocher, and P. Taslakian, "Competitive Online Routing on Delaunay Triangulations," *International Journal of Computational Geometry & Applications*, vol. 27, pp. 241–253, 12 2017.

[16] N. Bonichon, P. Bose, J.-L. de Carufel, V. Despré, D. Hill, and M. Smid, "Improved Routing on the Delaunay Triangulation," in *ESA 2018 - 26th Annual European Symposium on Algorithms*, Aug. 2018.

[17] P. Bose and J. M. Keil, "On the Stretch Factor of the Constrained Delaunay Triangulation," in *International Symposium on Voronoi Diagrams in Science and Engineering*, 2006, pp. 25–31.

[18] P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot, "Competitive Local Routing with Constraints," *Journal of Computational Geometry*, vol. 8(1), p. 125–152, 2017.

[19] P. Bose, M. Korman, A. van Renssen, and S. Verdonschot, "Constrained routing between non-visible vertices," *Theoretical Computer Science*, vol. 861, pp. 144–154, 2021.

[20] A. van Renssen and G. Wong, "Bounded-degree spanners in the presence of polygonal obstacle," *Theoretical Computer Science*, vol. 854, pp. 159–173, 2021.

[21] V. Ashvinkumar, J. Gudmundsson, C. Levcopoulos, B. Nilsson, and A. van Renssen, "Local routing in sparse and lightweight geometric graphs," *Algorithmica*, vol. 84, no. 5, pp. 1316–1340, 2022.

[22] Q. Li, P. Nevalainen, J. Peña Queralta, J. Heikkonen, and T. Westerlund, "Localization in unstructured environments: Towards autonomous robots in forests with delaunay triangulation," *Remote Sensing*, vol. 12, no. 11, 2020.