Paths Around Obstacles

An Honors Thesis for the Department of Computer Science

Marwan Al Jubeh

Tufts University, 2010.

Abstract

Connectivity augmentation is an important area of study in optimization. For a given graph, a connectivity augmentation problem asks to augment the graph (add edges) such that the augmented graph has the desired connectivity. Motivated by the problem of making a given non-crossing geometric graph 3-vertex connected, we consider the following augmentation problem: for a non-crossing geometric graph given in the form of convex obstacles inside a triangular container, augment the graph such that each obstacle has three disjoint paths to the container's boundary. We prove a lower bound on the number of edges needed for the augmentation, and also give an augmentation algorithm, which provides an upper bound.

Furthermore, motivated by the problem where the obstacles not only must have three disjoint paths to the boundary, but where those paths are also required to be locally-geodesic. We attempt to find an algorithm that would produce three disjoint locally-geodesic paths to the vertices of a triangular container from any point in the free space that is surrounded by line-segment obstacles. We document the various attempts we made to produce such an algorithm, and present counterexamples where these approaches fail. We conclude by presenting a promising line of attack for which we have not been able to find a counterexample.

Acknowledgments

I would like to express my deepest gratitude to my thesis committee of Professors Diane Souvaine and Benjamin Hescott who motivated me to partake in research and inspired in me great interest in the subject matter.

I also extend my most sincere thanks to Mashhood Ishaque, whose support and guidance were invaluable to me and without whom, this thesis would not have been possible.

I would also like to extend my deepest appreciation and thanks to Professors Csaba Tóth, Gill Barequet and Jack Snoeyink for their valuable expertise, time and help.

In addition, I would like to thank Andrew Winslow for his help and valuable suggestions.

This thesis presents joint work with Gill Barequet, Mashhood Ishaque, Diane L. Souvaine, Csaba D. Tóth, and Andrew Winslow.

Dedication

For my friends and family, who supported me in every possible way and whose encouragement and presence did not cease to inspire me throughout the course of this research.

Contents

1 Introduction

The study of graphs are important to both mathematicians and computer scientists, they have a wide range of applications and are relevant to networks and data structures as well as numerous other fields. A *simple graph* G is formally defined as a pair (V, E) of a set of vertices V and a set of edges E connecting those vertices. An *edge* is an unordered pair of two different vertices and represents a link between the vertices in question. A given graph is said to be k*-connected* if it remains connected after deleting any $k-1$ vertices in the graph, as well as their incident edges. According to Menger's theorem, a k-connected graph has k vertex-disjoint paths between any two vertices in the graph. The k*-connectivity augmentation problem* asks for the minimum number of edges that need to be added to augment a given graph to make it k-connected. Connectivity augmentation is a very important area of research in graph theory and optimization. Apart from its useful theoretical applications, it is extremely important for building and maintaining fault-tolerant networks while minimizing costs.

In abstract graphs, the connectivity augmentation problem can be solved in linear time for $k = 2$ [ET76, Ple76], and in polynomial time for any fixed k [JJ05]. For a given planar graph, an augmentation that preserves the graph's planarity is called a *planarity-preserving* augmentation. Unfortunately, planarity-preserving augmentations are NP-hard, even for $k = 2$ [BKB91]. NP-hard problems are an active area of research, however, they are believed to be infeasible. For a given planar graph that has been embedded in the plane (i.e: the vertices have been associated with coordinates in $\mathbb{R}^2$), an augmentation that respects the given embedding is said to be an *embedding-preserving* augmentation. For planar straight-line graphs, finding the minimum embedding-preserving augmentation using non-crossing straight-line edges is NP-Hard for any $2 \leq k < 5$ [RW08]. For $k \geq 5$, Kuratowski's theorem would imply that it is impossible for an augmentation to create a k-connected graph while maintaining the graph's planarity.

There are two possible approaches to get around the NP-Hardness of the augmentation problem: (*i*) approximation algorithms — for instance, there is a 2-approximation algorithm for planarity-

preserving connectivity augmentation for $k = 2$, which runs in $O(n \log n)$ time [BKB91]; and (*ii*) proving combinatorial bounds on the number of new edges in terms of the number of vertices. For instance, Al-Jubeh *et al.* [AJIR$^+$09] show that $2n - 2$ new edges are always sufficient and sometimes necessary for the embedding-preserving 3-edge-connectivity augmentation of a planar straight line graph with n vertices if augmentation is possible.

Tóth and Valtr [TV09] provided a characterization for planar straight-line graphs that can be augmented to 3-connectivity, calling such graphs *3-augmentable*. The minimum number of new edges sufficient for the 3-connectivity augmentation of any 3-augmentable planar straight-line graph with n vertices remains an open problem.

A *geometric graph* has been traditionally defined as a pair $G = (V, E)$, where V is a set of vertices in general position in the plane (i.e: no three of them are collinear) and where E is a set of distinct edges, whose endpoints lie in V. When no two edges of a geometric graph cross, that is, when any two edges can only intersect at a common endpoint, then the graph is called a *non-crossing geometric graph*.

Garcia *et al.* proved that for any point-set in the plane, surrounded by a 1-connected convex hull, if the point-set is augmented such that every point (except those on the convex hull) has three disjoint paths to distinct vertices on the convex hull, then the plane graph will be 3-connected. [GHH$^+$09]

Motivated by the 3-connectivity augmentation problem and the findings of Garcia *et al.* we consider the augmentation problem where a non-crossing geometric graph is given in the form of disjoint convex polygonal obstacles inside a triangular container, and this graph is to be augmented such that each obstacle has three disjoint paths to the container's boundary. In Chapter 2, we will prove lower bounds on the number of edges needed for the augmentation, and also give an augmentation algorithm (upper bound).

A *triangulation* of a non-crossing geometric graph G is a decomposition of G into triangles such that any two triangles either intersect in a common vertex, a common edge, or not at all.

An *obstacle* is defined as any region of space whose interior is forbidden to paths and new

edges. We will refer to the complement of the set of obstacles as the *free-space*.

We will use the term *locally-geodesic path* to refer to any path that cannot be improved by making a small change to it that preserves its *combinatorial structure* (the ordered sequence of triangles visited, for some triangulation of the free-space).

A locally-geodesic path can be thought of as an elastic rubber-band which has one endpoint fixed at the position of the path's first vertex and its other endpoint fixed at the position of the path's last vertex; initially, the rubber-band might be forced to take any shape, however, when it is allowed to snap by releasing it, the elastic rubber-band will start to contract and try to take the shape of the shortest path allowed by the surrounding obstacles that might stop it from taking a straight-line path. The path that the elastic rubber-band takes when it stabilizes would be the locally-geodesic path. We will refer to the process of replacing a path with the shortest path that has the same combinatorial structure (its corresponding locally-geodesic path) as *allowing the rubber-band (or path) to snap*.

We will use the term *geodesic shortest path* to refer to the Euclidean shortest path in the free-space. Keep in mind that this path does not necessarily have to be unique.

One might like to be able to obtain an augmentation for a given non-crossing geometric graph where the distance is minimized. One way to do this is to insure that each obstacle not only has 3 disjoint paths to the container, but also have these paths be locally-geodesic. However, the ability to obtain three disjoint locally-geodesic paths around obstacles from any point p in the free space to the vertices of the container is obviously a necessary precondition.

Arkin *et al.* considered the problem of finding monotone paths between two points in the presence of polygonal obstacles [ACM89]. For convex obstacles they proved there always exists a monotone path between any two points. This implies that we can always find two locally-geodesic paths from any point p in the free space to two vertices of the container, by simply finding monotone paths in the direction of the boundary edge connecting the two vertices, and then replacing each path with the shortest path in its homotopy type (i.e: letting the rubber-band snap). However, it remains an open problem whether this is also possible for three.

In Chapter 3, we demonstrate that it's not always possible to obtain 3 disjoint geodesic shortest paths around obstacles from any point p to the vertices of a triangular container. We also document the various attempts we made to provide an algorithm that produces three disjoint locally-geodesic paths for the special case where the obstacles are line-segments, and we present examples where these approaches fail. We conclude the chapter by presenting a promising line of attack for which we have not found a counterexample.

In Chapter 4, we briefly summarize our results and plans for future work.

2 Connectivity Augmentation

Problems concerning connectivity-augmentation are very important in network design and graph theory. In this chapter, we investigate the problem of augmenting a planar straight-line geometric graph given in the form of a triangular container and a set of disjoint polygonal obstacles in its interior, such that each obstacle must have 3 vertex-disjoint paths to the vertices of the container. We show that this is not always possible for non-convex obstacles. Then, we present lower and upper bounds for the case where the obstacles are convex.

2.1 Problem Definition

Given a set, O, of k disjoint convex polygonal obstacles inside a triangular container C, add straight-line non-crossing edges such that each obstacle has 3 disjoint paths to the three vertices of the container. The three paths must start at distinct vertices of the obstacle, end at distinct vertices of the container, and not contain steiner vertices. The paths can use the edges along the boundary of the obstacles as long as the path remains disjoint. The augmentation is not allowed to add edges in the interior of the obstacles (see Figure 2.1).

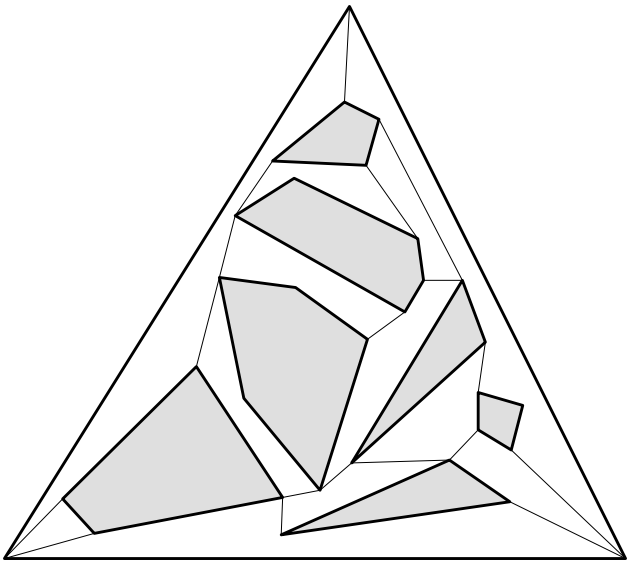


Figure 2.1: A triangular container with a set of disjoint convex obstacles in its interior. The graph is augmented by adding non-crossing straight-line edges to make each obstacle connected by three vertex-disjoint paths to the triangular container.

2.2 When is Augmentation Possible?

If the obstacles are not convex, it is not always possible to add edges such that each obstacle has three vertex-disjoint paths to the container. In Figure 2.2, only three vertices are visible to the inner-most obstacle, and all of those vertices belong to the same obstacle. Since it is not possible for paths to go through the interior of obstacles, this means that any path leaving the inner-most obstacle will have to go through one of the two lower endpoints of the outer obstacle. However, because we require three paths, then by the pigeon-hole principle, one endpoint must have two paths going through it. Hence, this example is not augmentable with vertex-disjoint paths. Therefore, non-convex obstacles are not always augmentable, which is why we will only be concerned with convex obstacles.

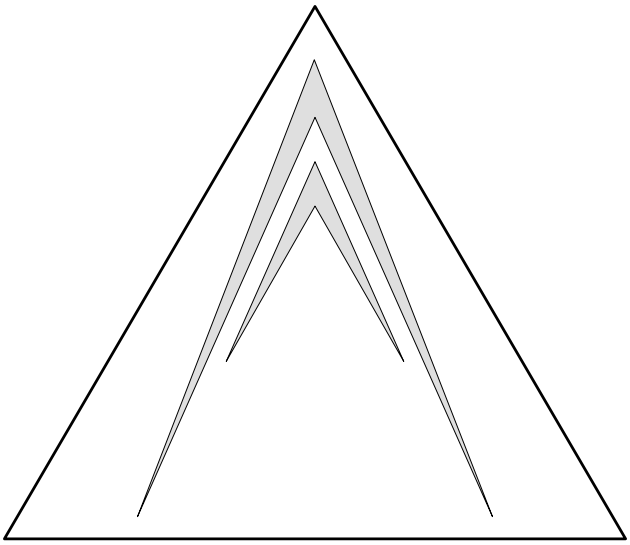


Figure 2.2: A triangular container with disjoint non-convex obstacles in its interior. Inner-most obstacle does not have three vertex-disjoint paths to the container.

On the other hand, for a set of disjoint convex obstacles inside the triangular container, every triangulation of the free space around the obstacles is a 3-connected graph [TV09]. It is easy to see that there are three vertex-disjoint paths from every obstacle to the container along the edges of a triangulation. To show this for any particular obstacle, add a new internal node p and connect it to any three vertices on the boundary of the obstacle. Similarly, add a node q outside the triangular container, such that q is visible to the container's three vertices (which can always be done by placing it on the angle bisector of any vertex). Then, add the three edges connecting q to the vertices of the container. It can be easily verified that the new graph is 3-connected, which means that there are three vertex-disjoint paths from p to q. Removing the points p and q from those paths would result in three vertex-disjoint paths that start at distinct vertices of the chosen obstacle and end at distinct vertices of the container. These three paths can be determined using any max-flow algorithm [AMO93]. This can be done for any obstacle, and it therefore proves that convex obstacles are always augmentable and that the triangulation of the free space around those obstacles constitutes a possible augmentation (See Figure 2.3).

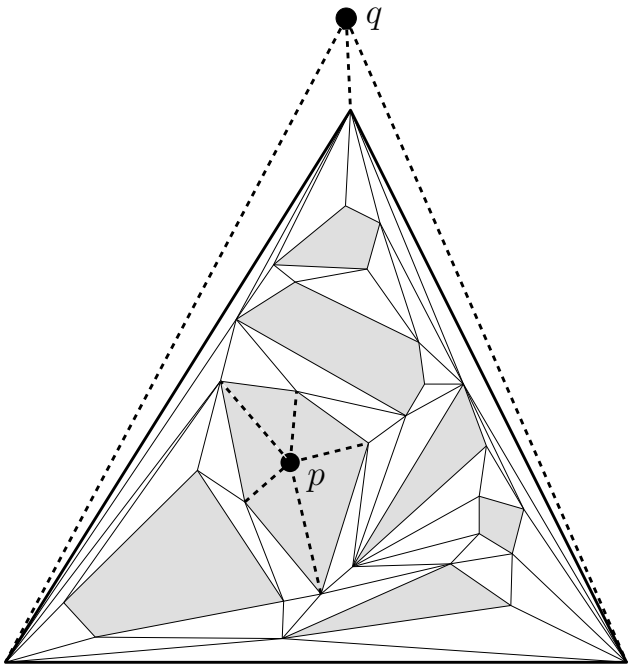


Figure 2.3: A triangulation of the free space around convex obstacles in a triangular container is a 3-connected graph. This implies that triangulation results in every obstacle having three vertex-disjoint paths to the vertices of the container.

Although triangulation is a possible augmentation, it adds too many edges and some subgraphs of the triangulation would constitute more desirable augmentations. We show how to obtain an augmentation that uses much fewer edges.

2.3 Lower Bound Constructions

When there is only one convex obstacle, three edges are obviously necessary (and sufficient) for connecting it to the container. However, for k convex obstacles (where $k > 1$ and can be arbitrarily-large), $3k - 1$ edges are necessary in the worst case. Our lower bound construction is depicted in Figure 2.4. It includes one large convex obstacle which hides one small obstacle behind each side (except the base), such that only three different vertices are visible to each small obstacle (Namely, the top vertex of the container and two adjacent vertices of the large obstacle). Therefore, we need three edges for each small obstacle and only two edges for the larger obstacle, each connecting a

bottom vertex to one of the container's base vertices.

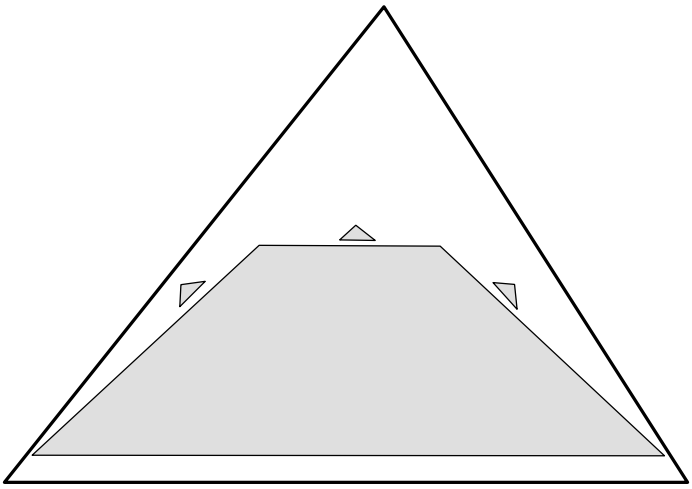


Figure 2.4: Lower bound construction: a triangular container with k disjoint convex obstacles in its interior, where $k = 4$. This non-crossing geometric graph requires at least $3k - 1$ new edges in order for every obstacle to have three-vertex disjoint paths to the container's vertices.

It's necessary for this construction that the large obstacle be a convex k-gon. If every obstacle has at most s sides, for some fixed $3 \leq s < k$, then the construction will not work. In that case we use a similar construction: hide $s - 1$ smaller obstacles behind each side (except one) of a big s-sided obstacle, and repeat the construction recursively for each smaller obstacle. This construction corresponds (in the worst case) to a complete tree with a branching factor of $s - 1$, in which the smaller obstacles are the children of a larger obstacle. For a fixed value of s, we set h as the height of the complete $(s - 1)$-ary tree of obstacles. Thus, the number of obstacles,

$$k = \frac{(s-1)^h - 1}{s-2}, \tag{2.1}$$

can be made arbitrarily high by changing h. The number of leaves in the tree is $(s - 1)^{h-1}$. A simple manipulation of Equation 2.1 shows that this number equals $k - \frac{k-1}{s-1}$. Hence, the number of internal nodes in the tree is $\frac{k-1}{s-1}$. For the 3-vertex-disjoint path augmentation, each leaf obstacle needs three edges and each non-leaf obstacle needs two edges. The total number of edges required

is, thus,

$$3\left(k - \frac{k-1}{s-1}\right) + 2\left(\frac{k-1}{s-1}\right) = 3k - \frac{k-1}{s-1},$$

which ranges from $\frac{5}{2}k + \frac{1}{2}$ to $3k - 1$ for $3 \leq s \leq k$.

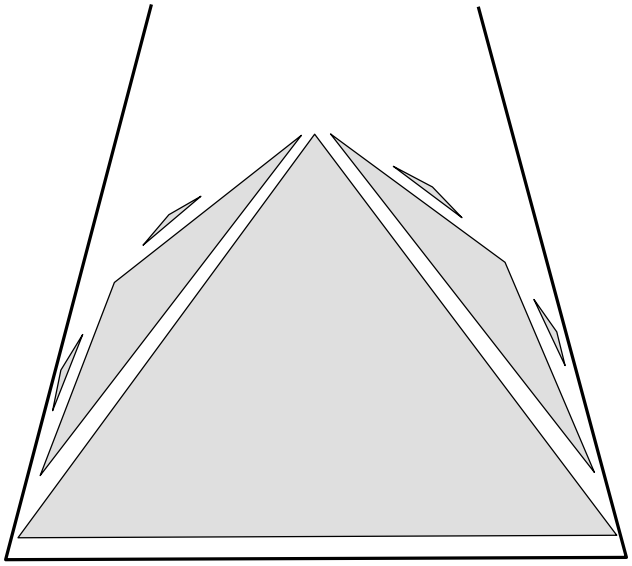


Figure 2.5: Lower bound construction: a triangular container with k disjoint convex obstacles in its interior such that every obstacle has at most s edges, where $k = 7$ and $s = 3$. This non-crossing geometric graph requires at least $3k - \frac{k-1}{s-1}$ new edges in order for every obstacle to have three-vertex disjoint paths to the container's vertices.

2.4 The Upper Bound

We present in this section an augmentation algorithm that augments the given graph in order to make every obstacle have 3 vertex-disjoint paths to the container. We will also show that this algorithm uses at most $3k$ edges, which proves that $3k$ edges are always sufficient to to augment any set, O, of k convex obstacles such that each obstacle has 3 vertex-disjoint paths to C — the triangular container.

The augmentation algorithm works by picking an arbitrary obstacle, finding three vertex-disjoint paths to the designated vertices on the container boundary, and then recursively augment-

ing the generated faces by considering them as subproblems. The container for each subproblem is a simple polygon (not necessarily convex) that is bounded by the three vertex-disjoint paths, the obstacle boundary, and the enclosing container. The algorithm maintains the following two invariants while generating subproblems:

1. The container P for each subproblem has three designated vertices on the boundary such that each designated vertex has a vertex disjoint path to one of the vertices of the triangular container C.

2. The triangulation of the free space in P along with the boundary of P forms a 3-connected graph.

Notice that the two invariants are initially true: the vertices of C serve as the designated vertices, and the triangulation of the free space is 3-connected as demonstrated in Section 2.2. In the following subsection, we explain how the algorithm maintains the two invariants, and why the number of edges added during augmentation is only $3k$.

2.4.1 The Augmentation Algorithm

We start by coloring the three vertices v_R (red), v_G (green) and v_B (blue) of the triangular container C to mark them as the designated vertices. To proceed, we choose an arbitrary obstacle o, and find three vertex-disjoint paths to v_R, v_G, and v_B, such that each path starts at a distinct vertex of o. We can find these paths π_R, π_G and π_B using a max-flow algorithm on the triangulation (includes obstacles and container's boundaries) of the free space , since the triangulation is 3-connected (see Section 2.2). For a path π_I ($I \in \{R, G, B\}$) we mark the obstacle o as being connected to v_I. We add the edges in the path π_I to our augmentation until we reach an obstacle that is already connected to v_I. We mark every obstacle along the path π_I for being connected to the designated vertex v_I. These markings are important for our charging scheme.

The augmentation algorithm in invoked by calling the procedure AUGMENT(C, v_R, v_G, v_B). Every two of the three produced vertex-disjoint paths create a simple polygon P, along with por-

tions of o and the container's boundaries. These polygons are considered as subproblems where each polygon acts as a container.

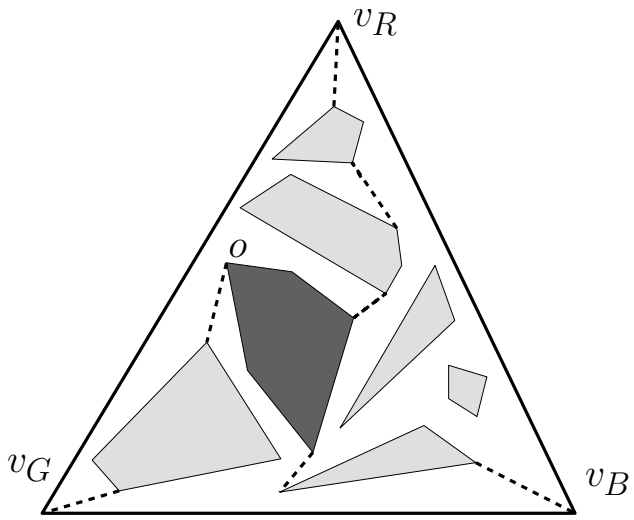


Figure 2.6: A triangular container with a set of disjoint convex obstacles in its interior. In this figure, edges have been added to make the obstacle o connected with the container's vertices via three vertex-disjoint paths.

Algorithm 1 AUGMENT(P, v_R, v_G, v_B)

Pick an arbitrary obstacle o inside P.

Find three vertex-disjoint paths π_R, π_G, and π_B to the vertices v_R, v_G, and v_B, respectively.

for all $I \in \{R, G, B\}$ **do**

$\pi_I =$ SHORTENPATH(π_I)

for all edges e along the path π_I from o to v_I **do**

Mark the obstacle incident to e for v_I

if e is a part of some obstacle boundary **then**

Do nothing; the edge is already part of the graph.

else if e in incident to the boundary of P **then**

Add the edge e and exit the loop.

else if e in incident to the vertex v_I **then**

Add the edge e and exit the loop.

else if e is incident to an obstacle marked for v_I **then**

Add the edge e and exit the loop.

else

Add the edge e.

end if

end for

end for

HANDLESUBPROBLEM(P, o, π_R, π_G)

HANDLESUBPROBLEM(P, o, π_R, π_B)

HANDLESUBPROBLEM(P, o, π_B, π_G)

However, it is possible that the triangulation of free space in the generated subproblem is not 3-connected (the triangulation contains a chord [TV09]). Therefore, in order to maintain the algorithm's invariants, we use the two subroutines: SHORTENPATH(π) and HANDLESUBPROBLEM(P, o, π_I, π_J). The subroutine SHORTENPATH(π) shortens the path π while keeping it disjoint from the other two

paths. The purpose of path-shortening is to ensure that if a subproblem contains a chord, the boundary of the container on the either side of the chord does not comprise of a single path. Hence, there is always a designated vertex on either side of the chord. The subroutine HANDLESUBPROBLEM(P, o, π_I, π_J) divides the subproblem into smaller subproblems until there is no chord.

Algorithm 2 SHORTENPATH(π)

Let $\{v_1, v_2, \ldots, v_m\}$ be the vertices in path π.

while for some $i < j - 1$, v_i and v_j see each other or are incident on the same obstacle **do**

Let P' be closed polygon formed by π and the line segment $v_i v_j$.

Let $\pi_{i,j}$ be shortest geodesic path between v_i and v_j inside P'.

Replace the portion of π between v_i and v_j by $\pi_{i,j}$.

Exit loop when π stops changing.

end while

return π

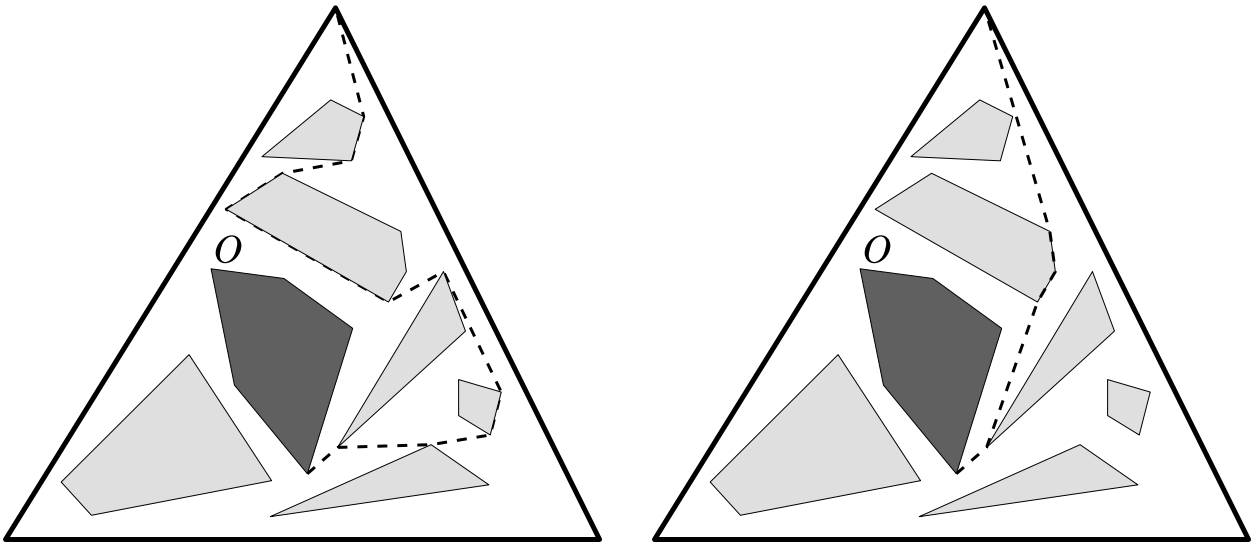


Figure 2.7: A triangular container with a set of disjoint convex obstacles in its interior, and a path from an obstacle, o to one of the vertices of the container. This figure demonstrates the process of path-shortening.

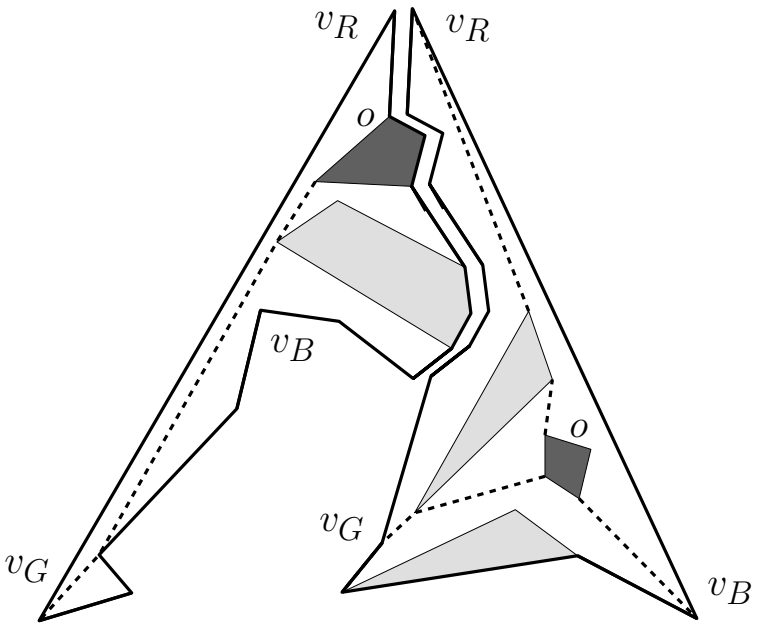


Figure 2.8: Two of the three subproblems created by the three vertex-disjoint paths from an obstacle to the container's vertices. This figure demonstrates the algorithm's recursion on the two subproblems.

Algorithm 3 HANDLESUBPROBLEM(P, o, π_I, π_J)

Obstacle o together with π_I and π_J creates a closed polygon P' inside P

Let v_I, v_J be the designated vertices of the paths π_I and π_J.

Let $l \in \{R, G, B\} \setminus \{I, J\}$.

Designate a vertex on the obstacle o as v_L.

if There is a 3-connected triangulation of P' **then**

AUGMENT(P', v_I, v_J, v_L)

else

Let C_1 be an extremal 2-cut.

Let P_1 be the polygon created by C_1.

Let v_R be one of the designated vertices the right of the C_1 (w.l.o.g).

Designate the two vertices of the 2-cut as V_G and V_B

AUGMENT(P_1, v_R, v_G, v_B)

HANDLESUBPROBLEM$(P \setminus P_1, C_1, \pi_I, \pi_J)$

end if

Lemma 1 *The procedure* AUGMENT(P, v_R, v_G, v_B) *is always invoked such that a triangulation of P is 3-connected, and the vertices v_R, v_G, v_B have three vertex-disjoint paths to the triangular container C.*

Proof. The first time the procedure AUGMENT is invoked is for the triangular container C, which has a 3-connected triangulation (See Section 2.2) and the designated vertices v_R, v_G, v_B are part of the container C. The procedure finds three vertex-disjoint paths from an arbitrary obstacle to the designated vertices. Every two of these paths form a simple polygon along with the boundary of the obstacle and the enclosing container. However, the resultant container P may not have a 3-connected triangulation, meaning that P might contain a chord. In that case the subroutine HANDLESUBPROBLEM is invoked, which further divides P into smaller polygons $\{P_1, P_2, \ldots, P_k\}$ such that a polygon P_i does not have a chord. Hence, a triangulation of P_i along with the boundary of P_i forms a 3-connected graph. To have 3 vertices on the boundary of a polygon P_i that have

three vertex-disjoint paths to C, it is enough to show that the boundary of P_i contains portions of at least two vertex-disjoint paths. Initially the polygon P has this property. Hence there are three vertices on P with vertex-disjoint paths to the triangular container C. To ensure that the smaller polygons generated by the subroutine HANDLESUBPROBLEM also contain portions of at least two vertex-disjoint paths, we use path shortening subroutine. The subroutine SHORTENPATH ensures that the boundary of P on either side of any chord does not comprise of a single vertex-disjoint path. If it were to comprise of a single path, the path would have been shortened in the first place. Hence, the procedure AUGMENT is always invoked such that a triangulation of P is 3-connected, and the vertices v_R, v_G, v_B have three vertex-disjoint paths to the triangular container C. □

Theorem 1 *The augmentation algorithm adds edges such each obstacle o has three vertex-disjoint paths to the container C.*

Proof. According to Lemma 1, whenever an obstacle o is about to be augmented the enclosing container P has a 3-connected triangulation and three vertices on its boundary with vertex-disjoint paths to C. Using a max-flow algorithm on a triangulation, we can find three vertex-disjoint paths from the obstacle o to the designated vertices on P, and hence to the triangular container C. □

Lemma 2 *Given three vertex-disjoint paths from an obstacle to v_R, v_G, and v_B, the path to v_R cannot touch the boundary of the polygon P between the vertices v_G and v_B.*

Proof. The lemma follows from the fact that the three paths are vertex disjoint. □

Theorem 2 *For k convex polygonal obstacles, the augmentation adds at most $3k$ edges.*

Proof. As a result of the path-shortening algorithm, a vertex-disjoint path enters and leaves an obstacle at most once. We don't add the edges along the boundaries of the obstacles, or the boundaries of the containers (see Lemma 2), hence we can charge each obstacle for each outgoing edge (if we consider the paths as directed paths that start from o). Thus each edge is charged to some obstacle. Since, we stop adding edges along a path when we reach an obstacle that is already

connected to the destination of the path, each obstacle is charged for at most three outgoing edges. Hence, the augmentation adds only $3k$ edges. □

2.5 Our Results

In this chapter, we presented the 3-connectivity-augmentation problem for polygonal obstacles inside a triangular container. We then showed that this is not always possible with non-convex obstacles. Then, we proved upper and lower bounds for the case where the obstacles are convex. For k convex obstacles, where k can be arbitrarily large, we showed that $3k-1$ edges are sometimes necessary. We then showed that for k convex obstacles, where k can be arbitrarily large but each obstacle has at most s sides, $3k - \frac{k-1}{s-1}$ edges are sometimes necessary. As for the upper bound, we presented an algorithm which gives an augmentation that adds only $3k$ edges, where k is the number of convex obstacles.

We conjecture that there exists an augmentation that adds only $3k - \frac{k-1}{s-1}$ edges, where k is the number of obstacles, and s is the upper-bound on the size of each one.

3 Locally-Geodesic Paths

This chapter presents the problem of finding 3 locally-geodesic paths around line-segment obstacles, from any point p inside a triangular container to the vertices of its container. This problem is motivated by the problem of augmenting convex obstacles inside a triangular container such every obstacle has 3 locally-geodesic vertex-disjoint paths to the container's vertices. In this chapter, we present an example where using geodesic shortest paths leads the paths overlap. We also conjecture that it is always possible to find 3 locally-geodesic paths from p to its container's vertices and present four attempts that we made to come up with an algorithm that produces the desired paths. Then, we finally describe a fifth line of attack to this problem which we believe to be a promising one, and one that we intend to pursue further.

3.1 Problem Definition

Given a set of line-segment obstacles inside a triangular container, do there always exist 3 disjoint locally-geodesic paths from any point p in the free space, to the three vertices of the container?

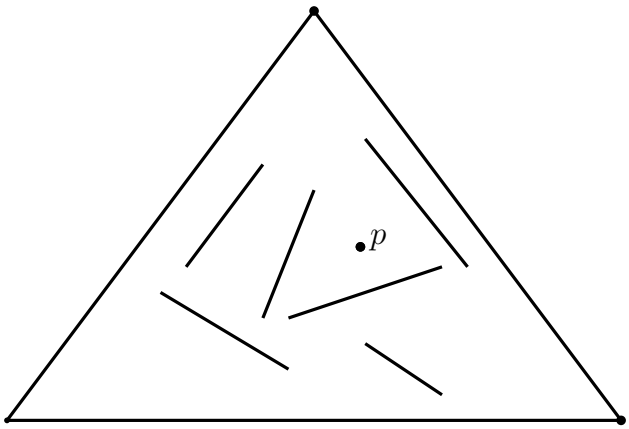


Figure 3.1: A triangular container with a set of disjoint line-segment obstacles in its interior, and a point p.

3.2 Geodesic Shortest Paths

For a point p in the free space, the 3 geodesic shortest paths (shortest Euclidean paths in the free-space) to the vertices of the container may not always be disjoint. Figure 3.2 demonstrates one such example where the geodesic shortest paths (shortest rubber-band paths) to the container's three vertices are not disjoint (see Figure 3.2).

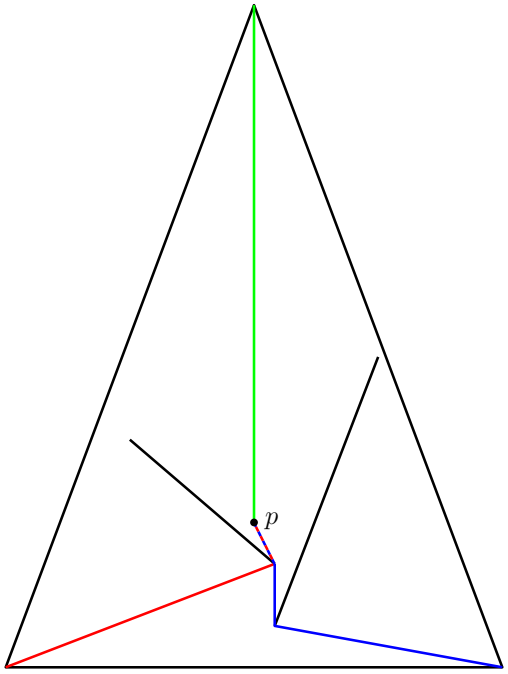


Figure 3.2: A triangular container with a set of disjoint line-segment obstacles in its interior, and a point p. In this figure, the shortest paths from p to its container's vertices are not disjoint.

Since geodesic shortest paths are not always disjoint, we focus our attention on locally-geodesic paths (where the rubber-band is fully stretched but is not necessarily the shortest possible path). We conjecture that the answer is always 'yes' to the question of whether for any point in free space, surrounded with disjoint line segments, there always exist 3 disjoint locally-geodesic paths from that point to the three vertices of a triangular container. In the rest of this chapter, we describe four attempts we made to come up with algorithms that produce three disjoint locally-geodesic paths, and present counterexamples where these approaches fail. We conclude with a promising line of attack, the Blocking Triangle approach, which could possibly lead to a proof of existence.

3.3 Monotone Algorithm Attempt

Recall that a path is called *monotone* with respect to a line ℓ, if it intersects every line perpendicular to ℓ at most once.

We construct each one of the three monotone paths to follow the path that would be taken by the free-fall of a ball towards a gravity source at the destination vertex, starting at p. Formally, one can describe this as creating three directed paths where each one goes from p to its respective container vertex, C for instance, such that the projection onto the vector from p to C ($\overrightarrow{pC}$) of the vector $\overrightarrow{pt}$ for any point t on the path would be shorter than the projection of $\overrightarrow{pu}$ for any other later point u (See Figure 3.3). This would lead every path to be monotone with respect to the straight line connecting p to the path's corresponding container vertex. Then, once all the paths reach their targets, we would allow them to snap (like rubber-bands) in order for them become locally-geodesic. The locally-geodesic paths produced will remain monotone.

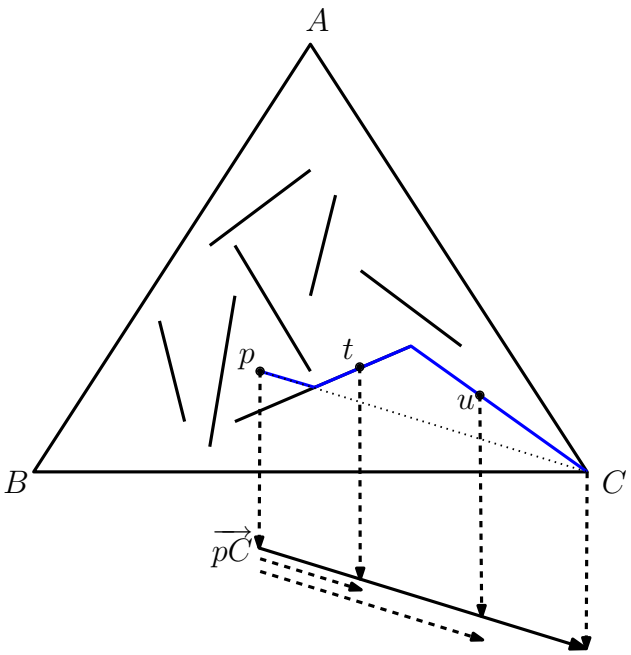
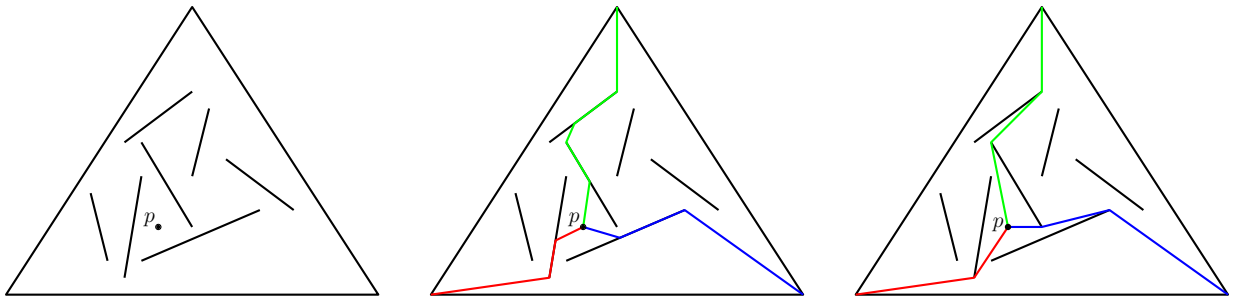


Figure 3.3: A figure that demonstrates how a path from p to a container vertex C is to be constructed. The projection of $\overrightarrow{pt}$ for a point t onto the vector $\overrightarrow{pC}$ would be shorter than the projection of $\overrightarrow{pu}$ for a later point u.

The monotone algorithm works very well in some cases (see Figure 3.4), but it also produces geodesic paths that are not disjoint in other cases (see Figure 3.5).

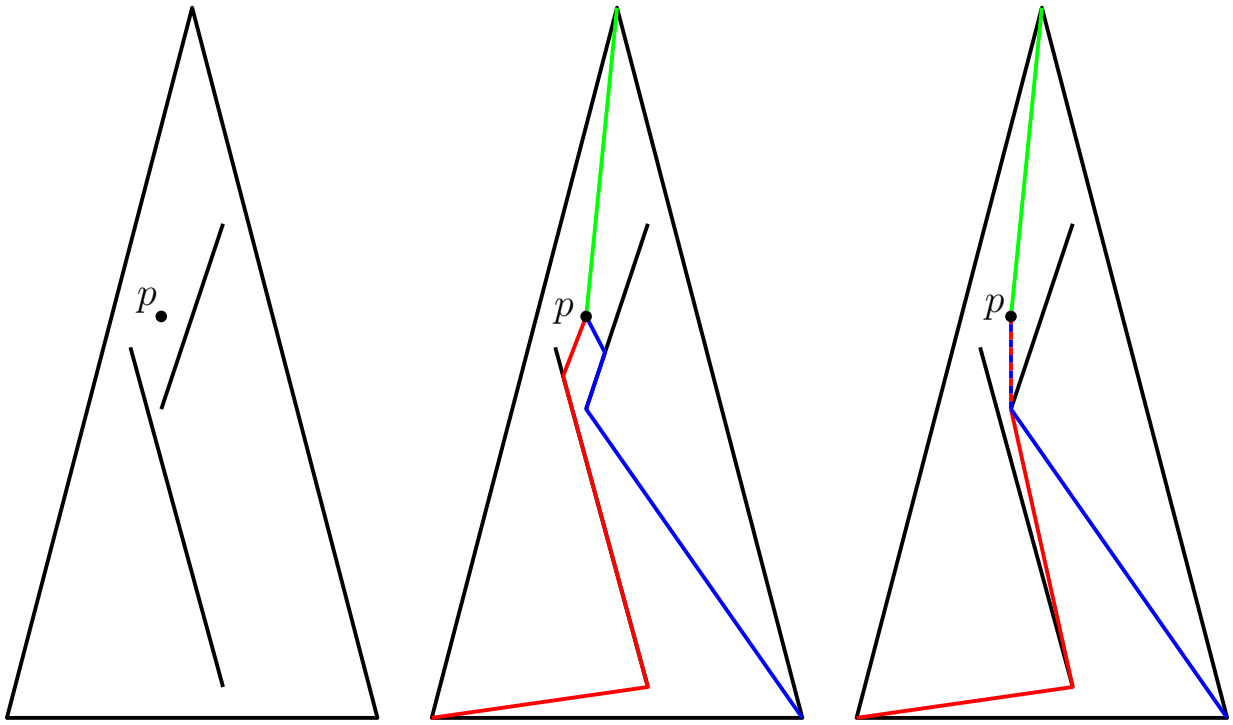


(a) Problem to be solved using the monotone algorithm

(b) Preliminary solution before releasing the paths and letting them snap

(c) Final solution

Figure 3.4: An example for the application of the monotone algorithm



(a) Problem to be solved using the monotone algorithm

(b) Preliminary solution before releasing the paths and letting them snap

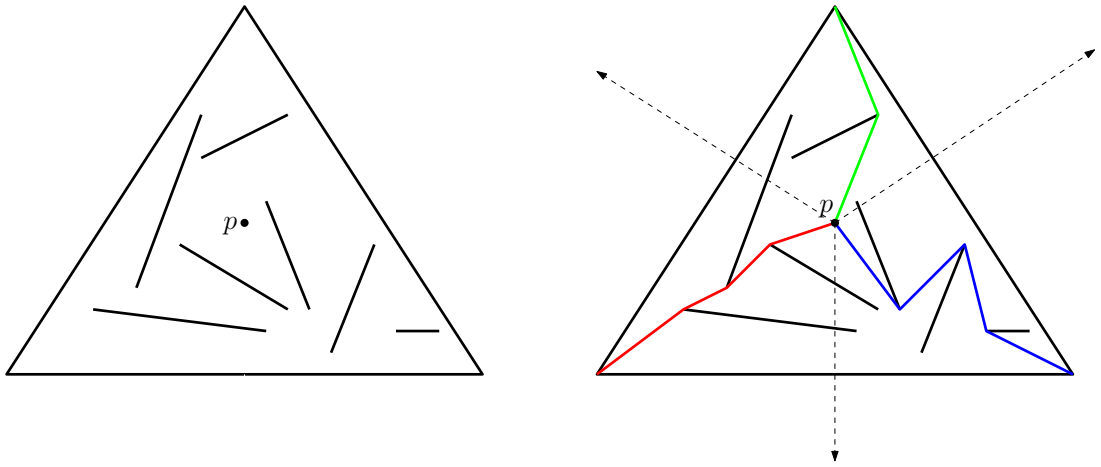
(c) Final (incorrect) solution

Figure 3.5: An example for which the monotone algorithm fails

3.4 Partitioning Approach

In the partitioning approach attempt, we divide the space around the point p into three partitions using rays that originate from p and act as boundaries for the partitions (we shall refer to those rays as *separators*). We will refer to a collection of rays (separators) as "proper" when each partition (the area of the container between two separators) includes exactly one of the container's vertices.

After obtaining three partitions using proper separators, we then attempt to find — within each one of those partitions — a geodesic shortest path from the point p to the endpoint contained in the partition. Figure 3.6 demonstrates how this could work.



(a) Problem to be solved using the partitioning approach

(b) Possible solution

Figure 3.6: An example for the application of the partitioning approach

We prove Lemma 3, which demonstrates an important property of this approach.

Lemma 3 *For any proper combination of separators, the boundary edge between any two neighboring container vertices must intersect exactly one ray.*

Proof. Assuming that the boundary between two neighboring container vertices intersects two or more separating rays would imply that the partition between two of those rays would not contain any of the container's vertices, which would imply that the separators are not proper.

Similarly, assuming that the boundary between two neighboring container vertices intersects no separating rays implies that the partition that contains one of those vertices must also contain the other as well, which would imply that the separators are not proper.

Therefore, for any proper combination of separating rays, the segment between every two neighboring container vertices must intersect exactly one ray. □

One implication of Lemma 3 is that when we are dealing with proper separators, we can use the names r_{AB}, r_{BC} and r_{AC} to refer to the separators that intersect the boundary edges AB, BC and AC respectively.

The major problem with this solution attempt was that, sometimes, an obstacle would intersect two of the three separators that define the partitions, such an obstacle would be called a chord with respect to the affected partition. This leads to a situation where finding a geodesic from p to the corresponding container endpoint in the affected partition impossible (see Figure 3.7).

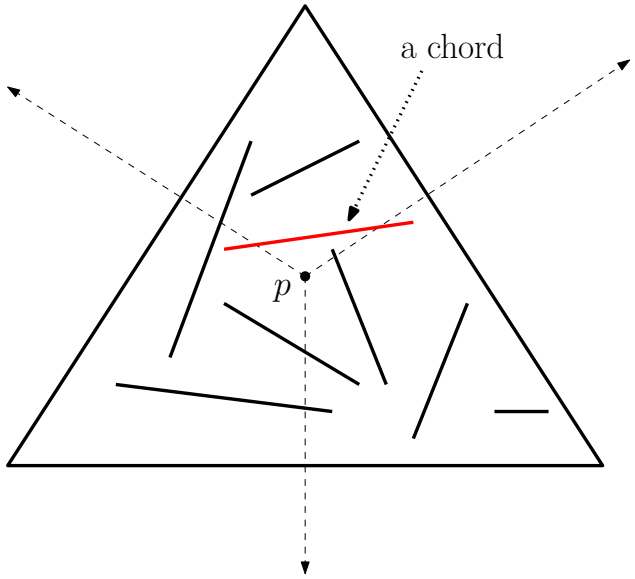


Figure 3.7: The introduction of chords (segments that intersect two separating rays) is problematic for the partitioning approach

The goal of this approach was to choose the separating rays so that the partitions they form would not have any chords so that every partition would contain a geodesic that connects p to the corresponding endpoint in the partition, solving the problem.

However, an adversary can produce a set of segments for which every proper combination of rays would produce partitions blocked by chords, which makes this approach fail. This set of segments is illustrated in Figure 3.8.

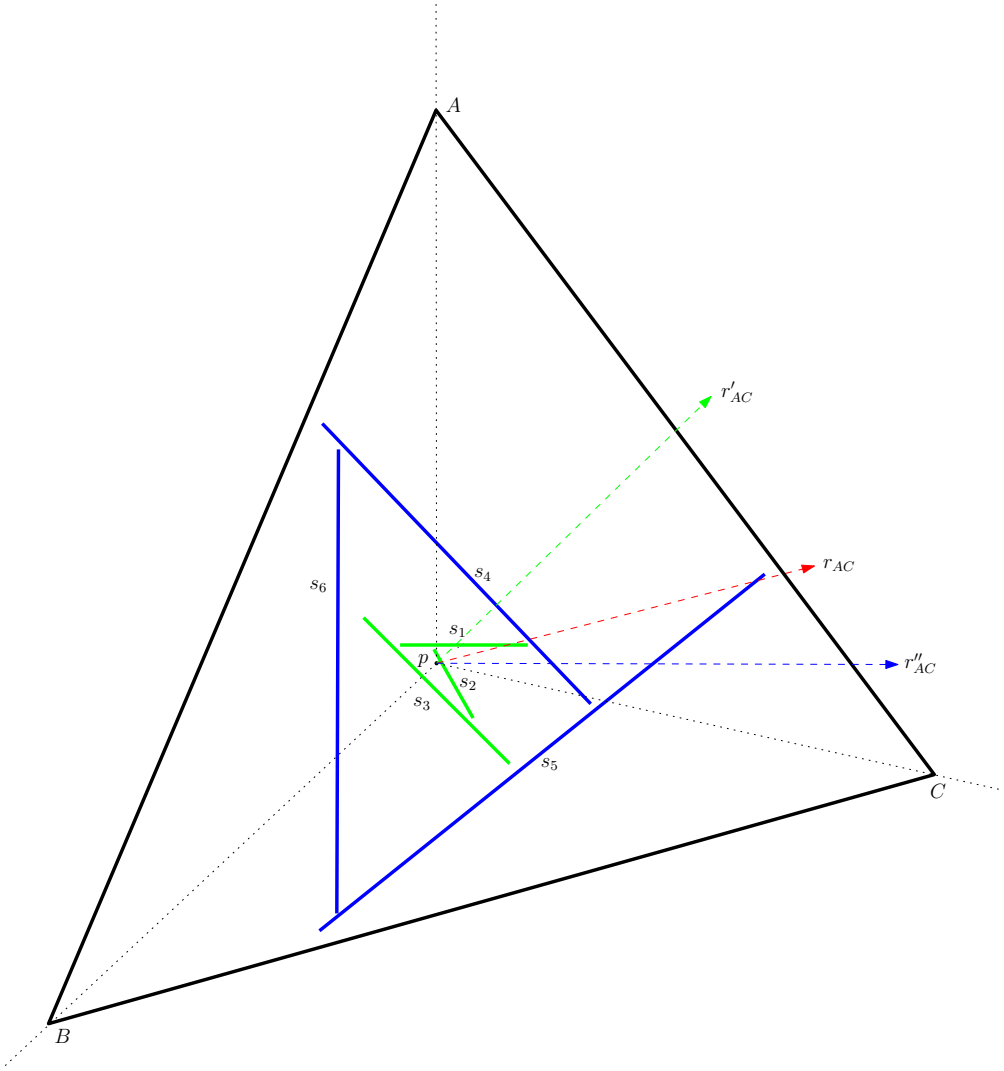


Figure 3.8: A triangular container with a set of disjoint line-segment obstacles in its interior, and a point p. Since r'_{AC} intersects both s_1 and s_2, to avoid s_1 and s_2, then r_{BC} and r_{AB} would both have to intersect s_3. Similarly, since r''_{AC} intersects both s_4 and s_5, to avoid s_4 and s_5, then r_{BC} and r_{AB} would both have to intersect s_6. This means that, for this figure, there is no proper combination of rays that would not introduce chords. Therefore, for this figure, the partitioning approach fails to produce a 3 disjoint locally-geodesic paths from p to A, B and C.

Theorem 3 *There exists a set of line-segment obstacles for which every proper combination of separating rays would produce partitions that introduce chords.*

Proof. Assume for the sake of a contradiction that there exists a proper combination of separating rays: $\{r_{AB}, r_{BC}, r_{AC}\}$, for Figure 3.8 which produce partitions that do not introduce chords.

According to Lemma 3, exactly one of those rays must lie between the endpoints A and C, namely r_{AC}. This ray, r_{AC}, must then lie either in the wedge between pA and pr or in the wedge between pr and pB.

r_{AC} cannot lie in the wedge between pA and pr. This is because if it did, then it must intersect both obstacles: s_1 and s_2. And since the separating rays cannot create partitions with chords, the two other rays: r_{AB} and r_{BC} must not intersect neither s_1 nor s_2. However, this means that they would both intersect s_3 (by construction), which would make s_3 a chord.

On the other hand, r_{AC} also cannot lie in the wedge between pr and pB. This is because it if did, then it must intersect both obstacles: s_4 and s_5. And since the separating rays cannot create partitions with chords, the two other rays: r_{AB} and r_{BC} must not intersect neither s_4 nor s_5. However, this means that they would both intersect s_6 (by construction), which make s_6 a chord.

If r_{AC} cannot lie anywhere in the wedge between pA and pB, this means that the separators cannot be proper, which leads to a contradiction.

□

3.5 Separating Path Attempt

Consider the case when the point p in the free space is visible to one of the vertices, say A. One possible solution would be to connect the three rubber bands such that one is from p to A, one is from p to B via A, and one is from p to C via A, and then let those rubber bands snap. pA will remain a straight line, and pB and pC will move away from each other with the path pA acting as a separator. Therefore, we will have three vertex disjoint locally-geodesic paths.

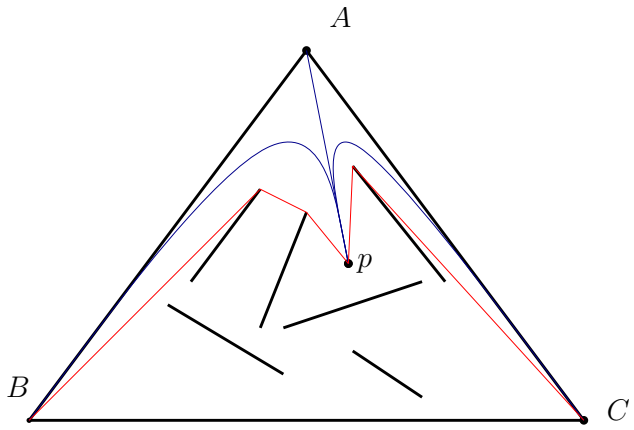
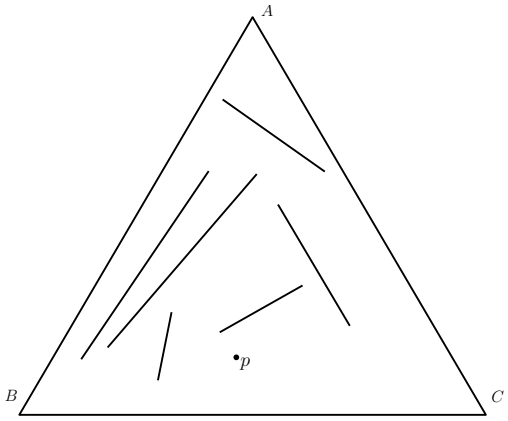
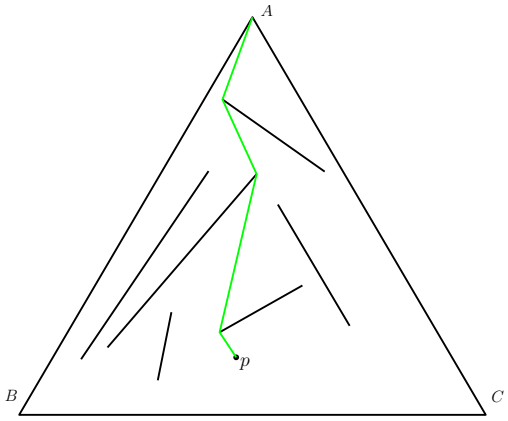


Figure 3.9: Three locally-geodesic paths for the case when p is visible to one of the container's vertices

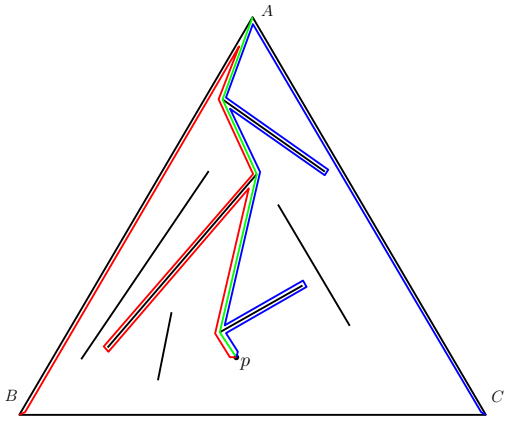
The goal of this attempt is to generalize this idea of having one path act as a separator while the other two paths move away from it. In order to do this, we first find a monotone geodesic path between p and one of the container's vertices (say A). Now, we find two more paths that go all the way up to A each on either side of the path pA, going around the other side of every segment that is incident on pA; this is done by modifying the path to move along the segment until it reaches the end of it, and then making it go all the way back on the other side. Those paths would then be made to move along either container boundary until each one reaches the respective container vertex at the end of either boundary edge. Ideally, once we let those paths snap, pA would then act as a separator between the pB and pC.



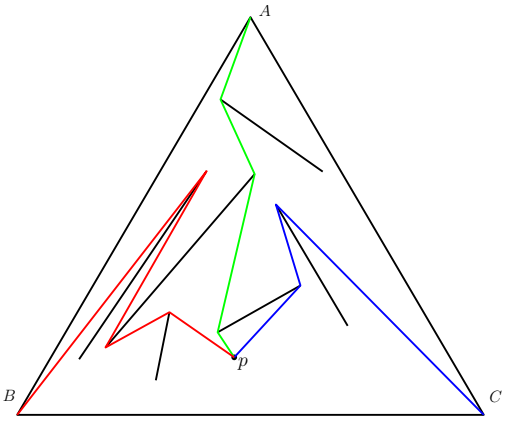
(a) The problem to be solved using the separating path attempt



(b) Finding a monotone path to A



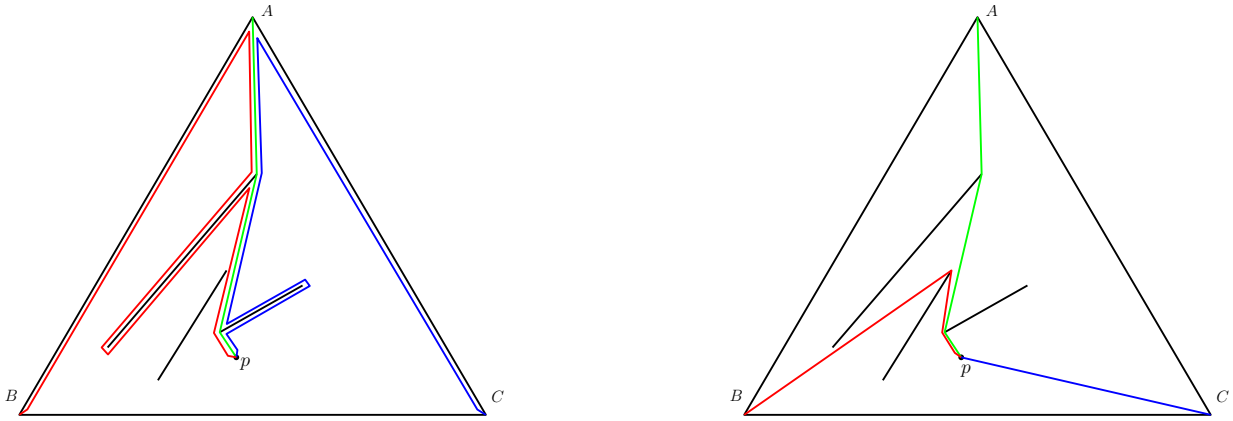
(c) Making the two other paths go around the path pA and its incident segments, each from its respective side



(d) The final solution

Figure 3.10: An example of the application of the separating path approach

Unfortunately, in some cases, the two new paths might still overlap with pA when we let them snap (see Figure 3.11).

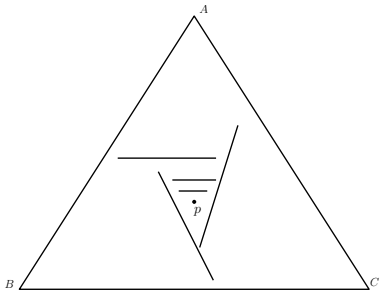


(a) An example for a possible problematic situation

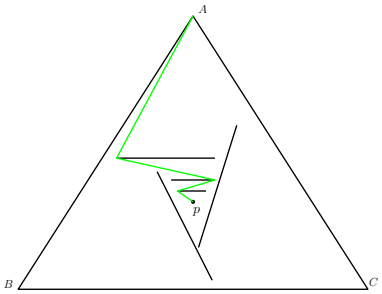
(b) The resulting (incorrect) solution

Figure 3.11: An example where the separating path approach fails

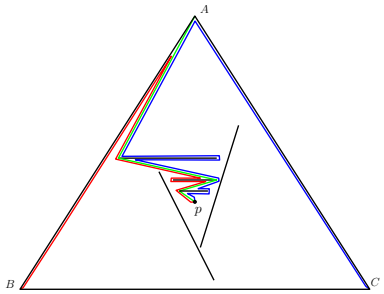
One approach to overcome this situation would be to go around more segments, even those that are not incident on the path pA, but in that case there's no way to guarantee that pB and pC would not intersect below p. In fact, one can find constructions that lead to this problem, as demonstrated by Figure 3.12.



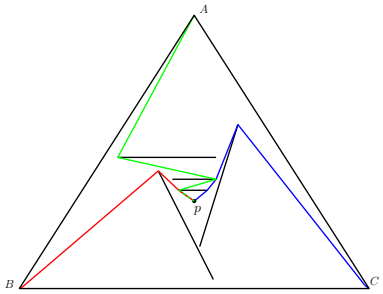
(a) The problem to be solved using the separating path attempt



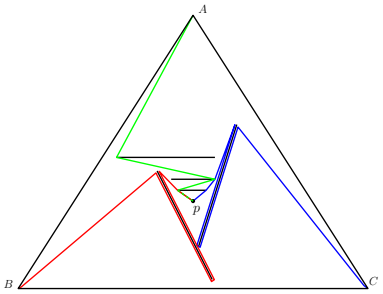
(b) Finding a monotone path to A



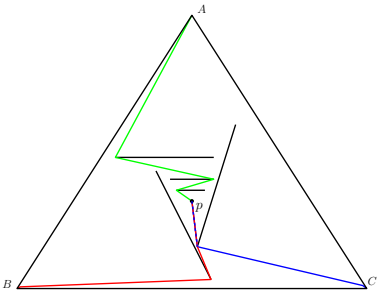
(c) Making the two other paths go around the path pA and its incident segments, each from its respective side



(d) The final (incorrect) solution



(e) Attempting to fix the solution by making the two paths go around the obstacles that are forcing them to intersect with pA, each from its respective side



(f) The modified paths intersect below p, which makes the solution incorrect still

Figure 3.12: An example where attempting to make the solution provided by the separating path approach avoid additional obstacles fails in fixing it

3.6 Online Algorithm Attempt

The idea behind the online algorithm attempt was to find an algorithm that solves the online version of this problem. The intuition behind this approach was that if we start with a correct solution for a set of obstacles, obtaining a correct solution after adding an extra obstacle should only require minor local modifications. Therefore, to find the solution for a set, O, of obstacles, we first start with the triangle T with no obstacles inside it, and generate the (trivial) solution for that case by connecting straight lines from p to each one of T's vertices. Then proceeding to add the obstacles from S one by one, and each time an obstacle is added, we locally modify the paths accordingly in order to maintain the solution's correctness.

The Algorithm's Invariants

The following invariants are necessary for this algorithm to work:

- The paths should always represent a correct solution for the obstacles added so far.
- for any obstacle x, that is compatible with all the obstacles added so far, the number of paths that intersect with x is less than 3.

  This is necessary, because inserting an obstacle that intersects more than 2 paths would lead to a dead-end (Although, heuristics which can resolve such situations have not been explored).

The Algorithm

For any triangle $T = \triangle ABC$, and a point p inside it, and a set, O, of disjoint objects in its interior, we call FIND-PATHS$(T = \{A, B, C\}, O, p)$ to obtain three disjoint geodesic paths from p to T's vertices.

Algorithm 4 FIND-PATHS$(T = \{A, B, C\}, O, p)$

Start with no obstacles.

Initially, the 3 paths from p to the vertices of T are simply the straight lines from p to those vertices.

while O is not empty **do**

Remove an obstacle s from the set O.

Call Add-Obstacle(s).

end while

return the paths pA, pB, pC.

Algorithm 5 ADD-OBSTACLE(s)

Insert s.

Let $pathCount$ be the number of paths that s intersects.

if $pathCount == 0$ **then**

The paths are still correct, so there's nothing to be done.

return

else if $pathCount == 1$ **then**

Choose one of the sides of s, and make the path that it intersects go around it from that side.

Allow the modified path to take the form of a locally-geodesic path (i.e: release the rubber band).

else if $pathCount == 2$ **then**

Adjust the two paths that s intersects to go around it, each from its corresponding side.

Allow the two modified paths to take the form of locally-geodesic paths (i.e: release the rubber band).

else

Output "Fail".

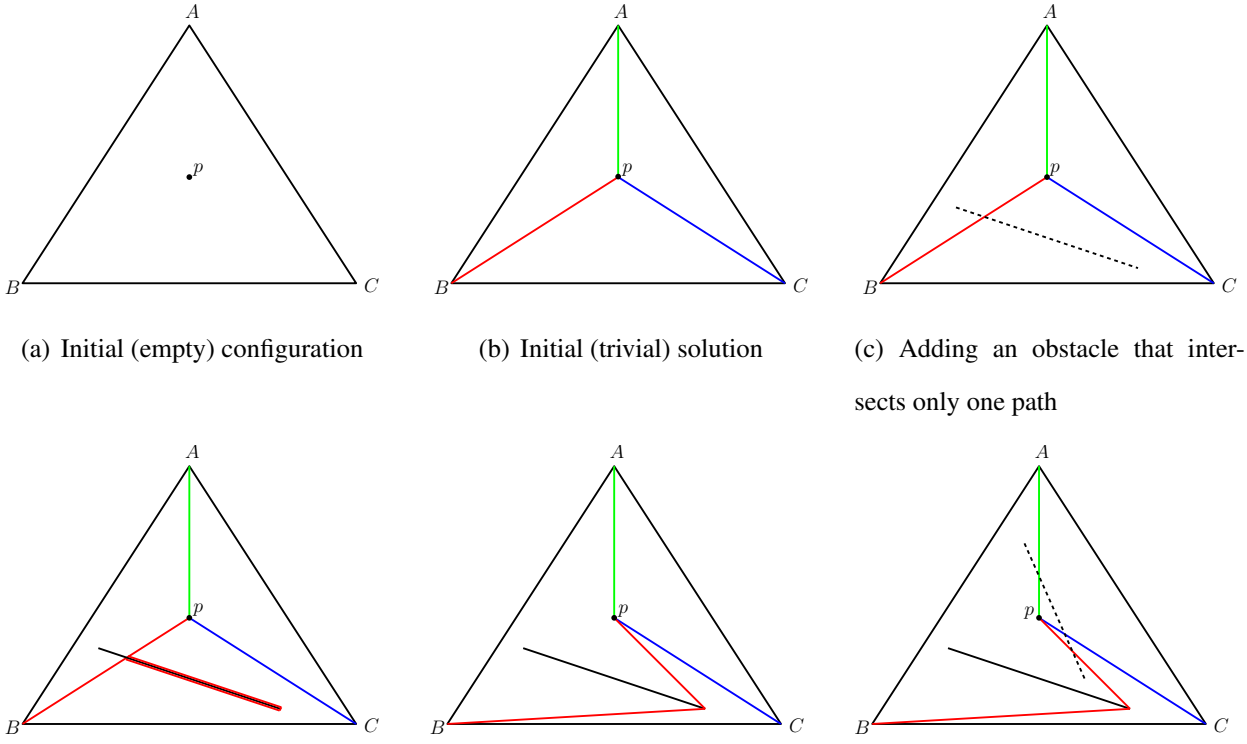
end if

We use the phrase "going around" an obstacle to refer to modifying a path by introducing to it a

detour along the boundary of the obstacle (in the direction of the chosen side). This is important to define to ensure that modifying the path would not cause it to intersect already existing segments in the space.

Further Investigation

One of this algorithm's potential problematic areas is the fact that that the obstacle endpoints that the paths choose to go around during 1-path-intersections were essential to maintaining the invariant, and that making the wrong choice can break it, leading to possible situations which cannot be handled by the algorithm. This is demonstrated by Figure 3.13.



(a) Initial (empty) configuration

(b) Initial (trivial) solution

(c) Adding an obstacle that intersects only one path

(d) Going around that obstacle, but making a bad choice for the endpoint

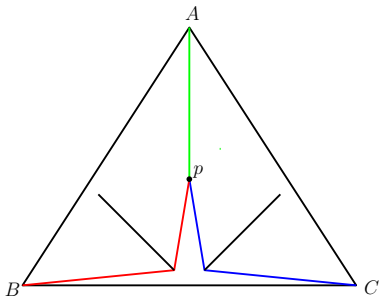
(e) Allowing the path to take shape as a geodesic

(f) The invariant is now no longer true

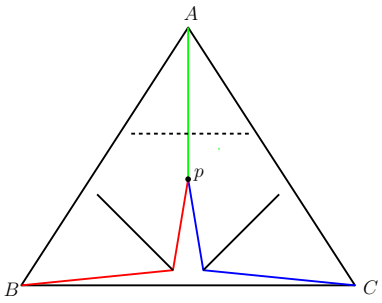
Figure 3.13: An example for breaking the invariant of the online algorithm by choosing the wrong endpoint

What's more, there is no simple heuristic for choosing which endpoint to go around that would

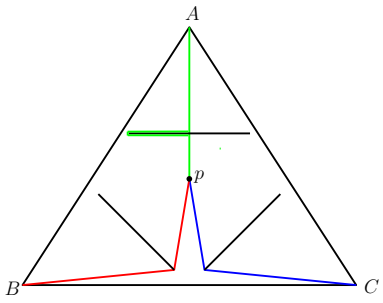
guarantee that the invariant will remain true. This is because there are examples where choosing to go around either endpoint resulted in paths which break the invariant for both cases, as demonstrated by Figure 3.14.



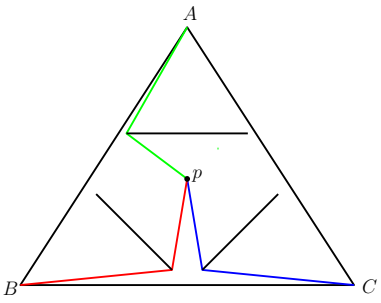
(a) A valid configuration



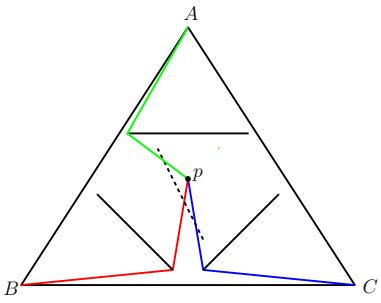
(b) Adding a new obstacle, that intersects only one path



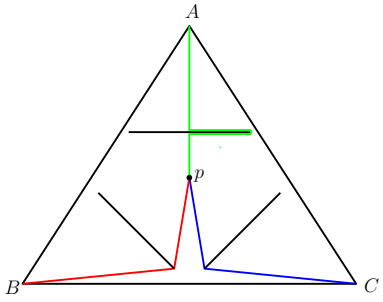
(c) Going around that obstacle's left endpoint



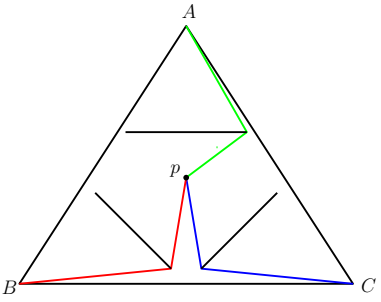
(d) Allowing the path to take shape as a geodesic



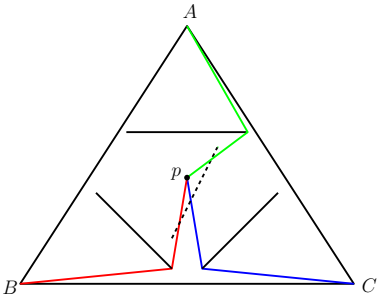
(e) The invariant is now no longer true



(f) Going around that obstacle's right endpoint



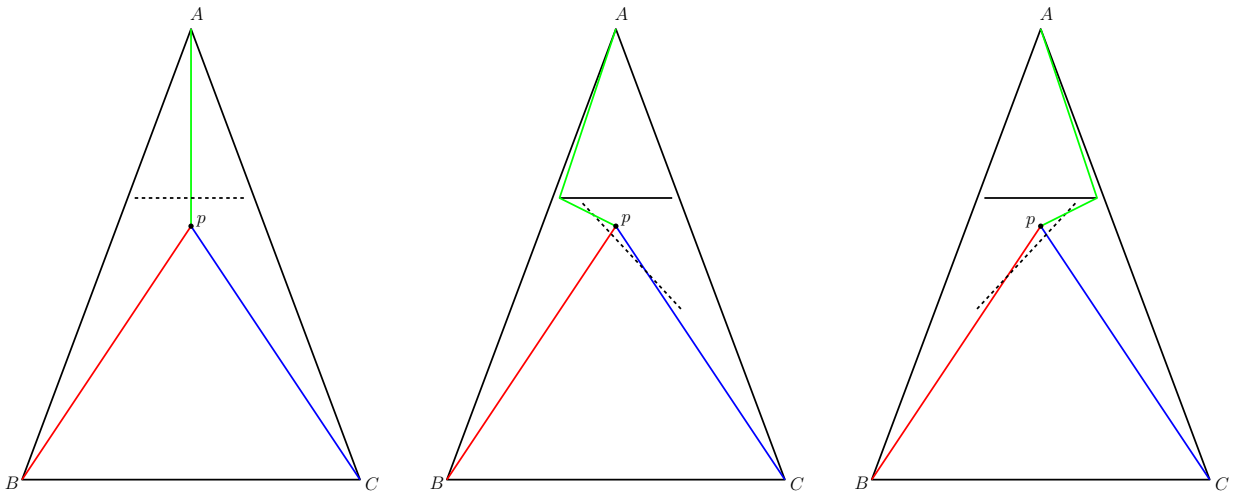
(g) Allowing the path to take shape as a geodesic



(h) The invariant is now no longer true

Figure 3.14: An example that breaks the online algorithm's invariant regardless of the choice of the endpoint to go around

In fact, as Figure 3.15 demonstrates, there are cases where no possible solution satisfies the proposed invariant (see Figure 3.15).



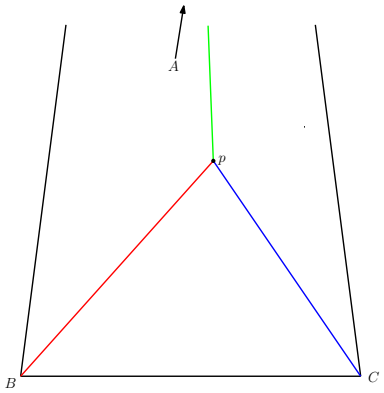
(a) The problematic configuration

(b) Choosing the left endpoint breaks the invariant

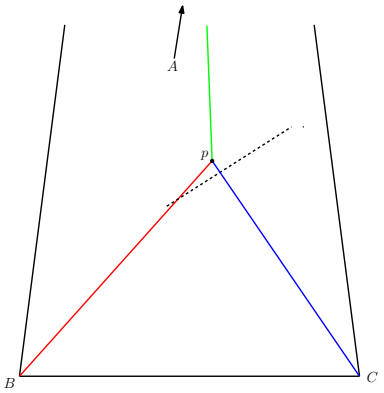
(c) Choosing the right endpoint also breaks the invariant

Figure 3.15: A triangular container with a set of disjoint line-segment obstacles in its interior, and a point p. In this figure, none of the possible solutions for three disjoint geodesic paths from p to the container's vertices satisfies the proposed invariant

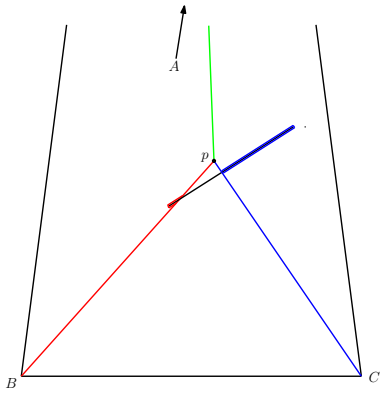
Another interesting problem with this approach is that not only segments that intersect one path can be problematic, but also segments that intersect two paths when added can lead to configurations that break the invariant. In fact, an adversary can break the invariant by repeatedly adding segments that *only* intersect two paths. This is demonstrated in Figure 3.16.



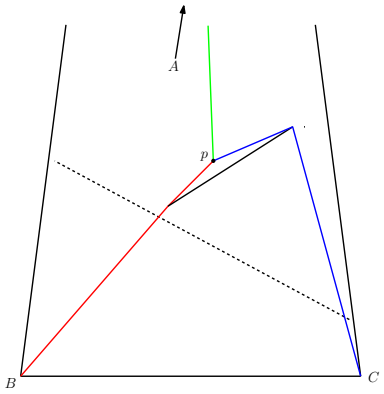
(a) Initial configuration



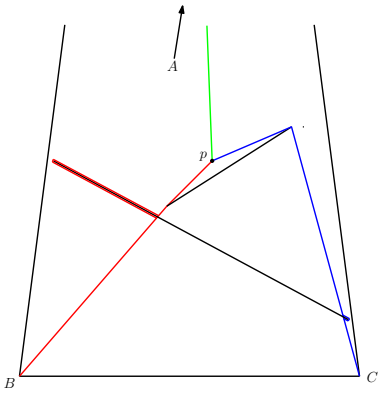
(b) Adding an obstacle that intersects two paths



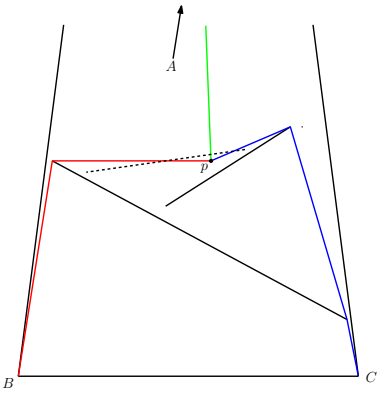
(c) Going around that obstacle, each path from its respective side



(d) Allowing the paths to take shape as geodesics, then adding another obstacle that intersects two paths



(e) Going around that obstacle, each path from its respective side



(f) Allowing the paths to take shape as geodesics, and observing that the invariant is now no longer true

Figure 3.16: An example for breaking the online algorithm's invariant by inserting obstacles that intersect two paths

3.7 Blocking Triangles Attempt

Given a triangular container with vertices A, B and C that contains disjoint line-segment obstacles in the interior, and point p in the free space inside it, draw a directed line ℓ from an arbitrarily chosen vertex, A, through p, as demonstrated by Figure 3.17

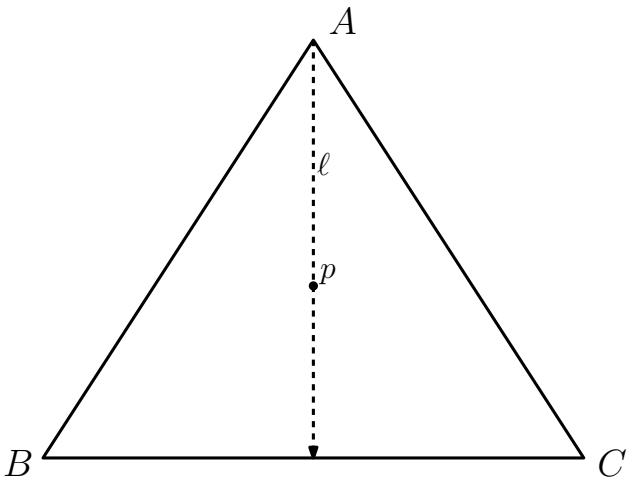


Figure 3.17: A triangular container $\triangle ABC$, and a point p in its interior. This figure demonstrates the directed line ℓ which is drawn from an arbitrarily chosen vertex, A, through p.

We define two structures with respect to the line ℓ: blocking pair, and blocking triangle.

Definition 1 *A pair of segments $\{s_1, s_2\}$ is called a* **blocking pair** *in the left (right) halfplane, if s_1 and s_2 intersect the line ℓ on opposite sides of p (i.e: one above and one below) and the left (right) endpoints of s_1 and s_2 cannot be separated by a line through p.*

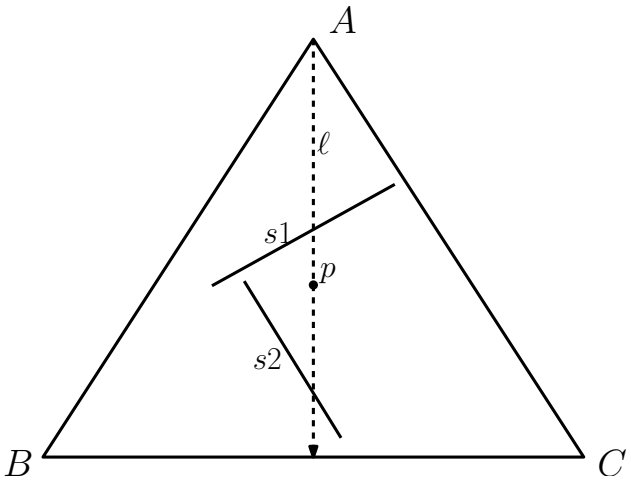


Figure 3.18: A triangular container $\triangle ABC$, and a point p in its interior, and a directed line ℓ that starts at an arbitrary vertex A and goes through p. In this figure, $\{s_1, s_2\}$ form a "blocking pair" in the right halfplane (w.r.t. ℓ).

Definition 2 *A triplet of segments $\{s_1, s_2, s_3\}$ is called a* **blocking triangle** *if one pair of them forms a blocking pair in one halfplane, and another pair of them forms a blocking pair in the other halfplane.*

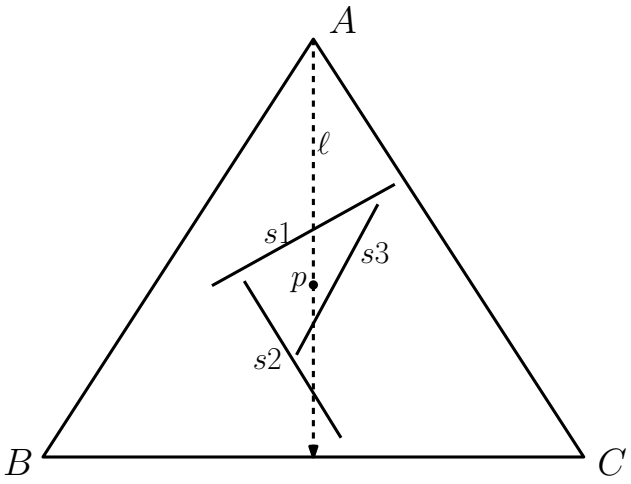


Figure 3.19: A triangular container $\triangle ABC$, and a point p in its interior, and a directed line ℓ that starts at an arbitrary vertex A and goes through p. In this figure, $\{s_1, s_2\}$ form a "blocking pair" in the right halfplane (w.r.t. ℓ), and $\{s_1, s_3\}$ form a "blocking pair" in the left halfplane (w.r.t. ℓ). Therefore, $\{s_1, s_2, s_3\}$ form a "blocking triangle" (w.r.t. ℓ).

The idea behind the blocking triangle approach is to find three disjoint geodesic paths from the container's vertices to the "endpoints" of the outermost blocking triangle, and then applying recursion. For this to work, we will need to show three things:

1. There exists three locally-geodesic paths, when there is no blocking triangle.

2. One can always find three locally-geodesic paths to the boundaries of the outermost blocking triangle.

3. Given three locally-geodesic paths to the boundary of the outermost blocking triangle Δ_0, the three paths can be extended to the boundary of the next blocking triangle Δ_1, while keeping the paths locally-geodesic and disjoint, by using recursion as well as, possibly, some minor local modifications.

We conjecture that the partitioning approach works well in the cases when there is only one blocking triangle, and that it only faces problems when p is surrounded by two or more nested blocking triangles. This would mean that one might possibly be able to use some modified version of the partitioning approach (or maybe a simpler heuristic) to find three geodesic paths from the container's endpoints to the "endpoints" of the first blocking triangle, or to p when no blocking triangle exists.

What's more, because of blocking triangles' ability to isolate what's inside them from what's outside them, it seems that it ought to be possible to define a recursive algorithm surrounding them.

3.8 Our Results

In this chapter, we presented the problem for connecting a point p, inside a triangular container, to its container's vertices around line-segment obstacles using locally-geodesic paths. This problem was motivated by the harder problem of augmenting convex obstacles to be 3-connected to their triangular container via locally-geodesic paths. We showed that it's not always possible to do this using geodesic shortest paths, and then presented four attempts that we made to create an

algorithm that produces the required paths, showing where they failed. Eventually, we presented a fifth approach which we believe to be a very promising one, and which we will be pursuing in the future to obtain a detailed proof of existence for 3 disjoint locally-geodesic paths around convex obstacles from any point p to its container's vertices.

We conjecture that for any point p, there exists 3 disjoint locally-geodesic paths around any set of convex obstacles from p to its container's vertices. We will attempt in our future work to prove this conjecture. After proving this conjecture to be true, our next goal will be to investigate whether there exists an augmentation for convex polygonal obstacles inside a triangular container, such that each obstacle has 3 disjoint and *locally-geodesic* paths to the container's vertices.

4 Conclusion

We have presented the problem of augmenting obstacles inside a triangular container to have 3 vertex-disjoint paths to the triangular container's vertices. We demonstrated that this is not always possible when the obstacles are allowed to be non-convex. We also proved both lower and upper bounds for the case when the obstacles are convex. We conjecture that the lower bound is the correct one for the problem. Therefore, for future work, we will attempt to prove this. Hence, we intend to find an augmentation algorithm that adds only $3k - \frac{k-1}{s-1}$ edges, where k is the number of obstacles, and where the size of those obstacles is bounded from above by s.

Also, motivated by the variant of this problem where the obstacles not only should have 3 vertex-disjoint paths to the triangular container's vertices, but where those paths are also required to be locally-geodesic, we investigated the problem of connecting a point p to the vertices of its triangular container, while avoiding convex obstacles. We presented a counter example that showed that this is not always possible using geodesic shortest paths. We also conjectured that three disjoint locally-geodesic paths from a point p to its triangular container's vertices always exist when the obstacles are convex. We then proceeded to present four attempts at creating an algorithm that would produce such paths and explained where they failed. Then we presented a final promising approach which we plan to pursue to obtain a detailed proof of existence in future work. Of course, our future work will also include investigating the problem that motivated much of this investigation, which is whether there exists an augmentation for convex polygonal obstacles inside a triangular container, such that each obstacle has 3 disjoint and *locally-geodesic* paths to the container's vertices.

The problems we investigated here are significant in many ways. Apart from the wide range of useful theoretical applications, they also serve a very important role in the field of building and maintaining fault-tolerant networks while minimizing costs. With the ever-growing demand for better, larger and more efficient networks, there is a critical need for efficient ways to solve connectivity-augmentation problems.

Bibliography

[ACM89] E. M. Arkin, R. Connelly, and J. S. Mitchell. On monotone paths among obstacles with applications to planning assemblies. In *SCG '89: Proceedings of the fifth annual symposium on Computational geometry*, pages 334–343, New York, NY, USA, 1989. ACM.

[AJIR+09] Marwan Al-Jubeh, Mashhood Ishaque, Kristóf Rédei, Diane L. Souvaine, and Csaba D. Tóth. Tri-edge-connectivity augmentation for planar straight line graphs. In *ISAAC*, pages 902–912, 2009.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, February 1993.

[BKB91] P. O. Bex, Goos Kant, and Hans L. Bodlaender. Planar graph augmentation problems. In *WADS*, pages 286–298, 1991.

[ET76] Kapali P. Eswaran and Robert Endre Tarjan. Augmentation problems. *SIAM J. Comput.*, 5(4):653–665, 1976.

[GHH+09] Alfredo García, Ferran Hurtado, Clemens Huemer, Javier Tejel, and Pavel Valtr. On triconnected and cubic plane graphs on given point sets. *Comput. Geom. Theory Appl.*, 42(9):913–922, 2009.

[JJ05] Bill Jackson and Tibor Jordán. Independence free graphs and vertex connectivity augmentation. *J. Comb. Theory Ser. B*, 94(1):31–77, 2005.

[Ple76] Ján Plesník. Minimum block containing a given graph. *Archiv der Mathematik*, 27(1):668–672, 1976.

[RW08] Ignaz Rutter and Alexander Wolff. Augmenting the connectivity of planar and geometric graphs. *Electronic Notes in Discrete Mathematics*, 31:53–56, 2008.

[TV09] Csaba D. Tóth and Pavel Valtr. Augmenting the edge connectivity of planar straight line graphs to three. In *XIII Spanish Meeting on Comput. Geom.*, 2009.