Stony Brook University

OFFICIAL COPY

The official electronic file of this thesis or dissertation is maintained by the University Libraries on behalf of The Graduate School at Stony Brook University.

© All Rights Reserved by Author.

Analysis of Three Geometric Optimization Problems: Local Greedy Routing in Triangulations, Touring Sequences of Polygons, and Hitting Sets of Segments

A Dissertation Presented

by

Kan Huang

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics & Statistics

Stony Brook University

August 2015

Stony Brook University

The Graduate School

Kan Huang

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

Joseph S. B. Mitchell – Dissertation Advisor
Professor, Department of Applied Mathematics and Statistics

Esther M. Arkin – Chairperson of Defense
Professor, Department of Applied Mathmatics and Statistics

Jiaqiao Hu
Associate Professor, Department of Applied Mathmatics and Statistics

Jie Gao
Associate Professor, Department of Computer Science

This dissertation is accepted by the Graduate School.

Charles Taber
Dean of the Graduate School

Contents

List of Figures

List of Tables

Acknowledgements

I'd like to thank my advisor, Joseph S. B. Mitchell, who is supportive and considerate anywhere and anytime. He teaches me not only how to solve problems, but how to treat people, love research and balance life and work. He indeed sets up a perfect model for my future life.

I also appreciate the help from my co-authors and people who give invaluable suggestions to my research: Chien-Chun Ni, Rik Sarkar, Jie Gao, Francisc Bungiu,Michael Hemmer, John Hershberger, Alexander Kröller, Gui Citovsky, Sándor P. Fekete, Ojas Parekh, Cynthia A. Phillips, Mark Daniel Rintoul, Mayank Goswami, Michael Biro.

At last I'd like to thank my parents, Weibing Huang and Rongzhen Li, for their unconditional support and love.

Chapter 1

Local Routing in Triangulation

1.1 Introduction

The path planning problem in an initially unknown environment has received a lot attention from the communities of computational geometry, robotics and on-line algorithms. Given an on-line problem P, the cost of solving an instance i of this problem optimally is $opt(i)$. If for any instance $i \in P$, an on-line strategy σ gives a solution whose cost, $\sigma(i)$ is not great than $c \times opt(i)$, where c is a certain constant factor, then the strategy σ is said to be a *c-competitive* solution of P. And we define the *competitive ratio* of strategy σ by

$$\rho_\sigma = \sup_{i \in P} \frac{\sigma(i)}{opt(i)}$$

In path planning problems, the cost is the length of path between the starting point and the destination or the time of travel.

Considering an on-line problem P, there are three essential questions we need to answer in sequence. First, does a competitive strategy exist? If yes, what's the competitive ratio of this strategy gives? Finally, what is the smallest ratio ρ that can be achieved. That ratio is defined as the *competitive complexity* of problem P.

On-line navigation problems have various models. There are four main aspects in a model:

1. **Target**
    - Is it known initially?
    - Is it a point, an infinite line, level in a layered graph etc?
2. **Environment**

- Is the searching space a graph, several rays, a polygon, etc?
- What kind of obstacles does it have?

3. **Robot** (conventionally, we call the moving agent robot.)
   - The representation of a robot, a point, a polygon or others?
   - A single robot or a group?
   - What senses does it have, unbounded or bounded vision, touch, position?
4. **Metric** (in the paper, the L_2-metric is applied if no metric is specifically declared.)

The competitive complexities of several problems have been known. Icking et al. [1] show that the competitive complexity of streets walking problem is $\sqrt{2}$. López-ortiz and Schuierer [2] show that if the location of target is known, the competitive complexity for searching in generalized streets in L_1-metrics is 9.

Some competitive strategies for several problems have been found, but the complexities of those problems are unknown. [3] provides a lower bound of 9 for the competitive ratio of searching in a star-shaped polygon and propose an algorithm with a ratio, 11.52. It also present a strategy with a competitive ratio of 28.85 and give a lower bound of $\sqrt{82}$ for the decision problem of whether the searching space is star-shaped or not.

Furthermore, there are more problems that have no constant factor competitive strategy. For such a problem P, to evaluate the performance of a strategy, σ, it's common to define a competitive ratio function, $\rho_\sigma(n)$, where $n = d_2(s,t)$ here denotes the Euclidean distance between s and t and $\rho_\sigma(n)$ is defined by

$$\rho_\sigma(n) = \sup_{p \in P; s,t: d_2(s,t)=n} \frac{d_\sigma(s,t;p)}{d(s,t;p)}$$

where $d(s,t;p)$ and $d_\sigma(s,t,p)$ are the length of the shortest path and the length of the $s-t$ path produced by strategy σ in p respectively. [4, 5] show that if the target is an infinite vertical lime ("wall"), at distance n from s, and the obstacles are aligned rectangles, then $\rho(n) = \Theta(\sqrt{n})$. [6] proposes a strategy named CBUG for the searching problem of a robot with size D and provides a quadratic relation between $d(s,t;p)$ and $d_\sigma(s,t,p)$.

1.2 Problem Definition and Algorithm

Given a sequence of triangles as $\triangle_1, \triangle_2, \cdots, \triangle_n$, where $\triangle_i$ and $\triangle_{i+1}$ share a common edge. Without loss of generality we assume that the source s is a vertex of $\triangle_1$ and the destination t is a vertex of $\triangle_n$. We would like to deliver a message from s to t inside the sequence of triangles. But we do not have information about the entire list of triangles. Instead, the message is delivered with only local information — when the message stays inside $\triangle_i$ it knows the coordinates of the next triangle $\triangle_{i+1}$ but nothing beyond that. We wish to have an algorithm that uses the limited information to find a path that is at most constant factor longer than the shortest possible.

Definition 1.2.1 (Greedy Routing Algorithm). *For a message at point p inside $\triangle_i$ towards destination t,*

1. *It is routed along the shortest path towards the next triangle $\triangle_{i+1}$ in the sequence S.*
2. *If $\triangle_i$ is the last triangle in S, then it is routed along the shortest possible path from p to t.*

Such a path is shown in blue in Figure 1.1.

In the Euclidean plane, the *shortest path* between two points is the straight line, while that from a point to triangle is simply the straight line from the given point to the nearest point of the triangle. The greedy path from s to t therefore consists of such a sequence of segments through the triangles of S (shown in blue in Figure 1.1). We will prove the following theorem

Theorem 1.2.2. *Given a non-repeating sequence S of triangles, the greedy routing algorithm finds a path of length at most ρ times the length of the shortest path following the same sequence S, where ρ is a constant independent of the input.*

A non-repeating sequence means that no triangle appears more than once in S. Thus the shortest path never visits the same triangle again, and therefore does not self intersect. The theorem implies that while the algorithm operates greedily with very local information, it still produces good quality paths, not much longer than the shortest possible in its category.

We first introduce the terminology used in this chapter. Let us represent the greedy path and the shortest path between two points by $Q_{s,t}$. We use P and Q for short sometimes for conciseness. Any two successive triangles $\triangle_i$ and $\triangle_{i+1}$ in sequence S share a common edge, let us call it e_i, and the straight line containing it l_i. The intersection of P with e_i is denoted by v_i, while the intersection of Q with e_i is given by q_i. The segment between these two points is given by $r_i = q_i v_i$.

Some segments of P, such as v_1v_2 intersects the corresponding edge (here e_2) at an interior point, and at a right angle. We refer to such *orthogonal segments as o−segments.* Other segments, such as v_3v_4 intersect at a boundary of the edge, we refer to these as *boundary segments or b−segments.*

For points u and v on a path P, we will use $P_{u,v}$ to represent the part of P in between these two points, and $|P_{u,v}|$ to represent its length. For general points x and y, $|xy|$ will represent the Euclidean distance between these two points. For vertices v_i and v_j on P, we will use $P_{i,j}$ to denote the path between them.

These two paths may intersect each-other at points other than at s and t, but we need to only consider regions between intersections. To see why this is true suppose that the paths intersect at an intermediate point w. In this case, if $|P_{s,w}| \leq \rho|Q_{s,w}|$ and $|P_{w,t}| \leq \rho|Q_{w,t}|$, then by simply adding we have $|P_{s,t}| \leq \rho|Q_{s,t}|$. Thus we only need to prove the ρ stretch for each segment of P, Q between consecutive intersections.

If the line segment connecting s, t is inside the sequence of triangles, $Q_{s,t}$ is straight and its length equals $|st|$. Otherwise, $Q_{s,t}$ makes turns in the middle. We can divide the path $Q_{s,t}$ into subpaths where all the turns are in the same direction. More formally, a *spiral* is a directed simple path where every turn is in the same direction, clockwise or counter clockwise.

Lemma 1.2.3. *If v and v' are successive intersections of P and Q, then $Q_{v,v'}$: the shortest path between these two intersections, is a spiral.*

This lemma means that to estimate the stretch between intersections, which is our goal, **we can assume that the shortest path segment is a spiral.**

Before giving the proof of general cases, let's first look at a special case when the shortest path is a line segment(the simplest spiral). It will provide insights of the problem.

1.3 When Optimal Path is a Segment

In this section we prove the following theorem

Theorem 1.3.1. *For the simple case when the shortest path connecting s, t is a straight line segment, ρ is $\pi + 1$ and the bound is tight.*

We have already known that the only case needs to prove is the one when the greedy path is on "one-side", i.e., it does not intersect the shortest path st except at s and t.

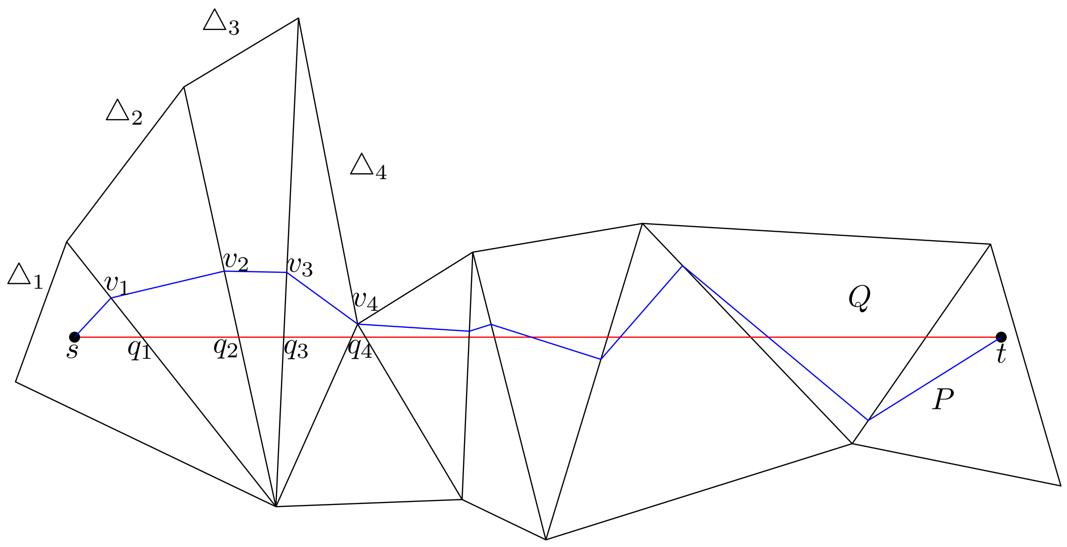


Figure 1.1: The optimal path and the greedy path of a triangulation connecting s and t

We first show an example in which the stretch of the greedy path can be arbitrarily close to $\pi + 1$. See Figure 1.2. A semicircle with diameter se is sliced into small sectors. Every triangle covers a sector and they have common vertex o. t is in the last sector. The dashed line is the greedy path. In the extreme case, i.e., the sectors are arbitrarily slim and t is arbitrarily close to o, then the greedy path will be arbitrarily close to $\widehat{se}$ and eo, approaching a stretch factor of $\pi + 1$.

Next we are going to prove that $\pi + 1$ is an upper bound of the path stretch.

1.3.1 Modification: Replace b segments

We will modify the greedy path P to a different path P', which is easier to analyze. This new path P' is necessarily longer than the original path, therefore an upper bound on its length is an upper bound on the length of P. Let us start with $P' = P$ and modify through the following steps. For each b-segment $v_{i-1}v_i$, we take $v_{i-1}u_i$ as the perpendicular line from v_{i-1} to l_i. Then we replace $v_{i-1}v_i$ by an o-segment $(v_{i-1}u_i)$ and a segment u_iv_i. This replacement is shown in Figure 1.3. The segment u_iv_i is tangential to the edge e_i and we call it an *l-segment.*

After all such replacements, P' has no b-segments, and only has o and l segments. Applying triangle inequality at each place shows $|P'| \geq |P|$. For

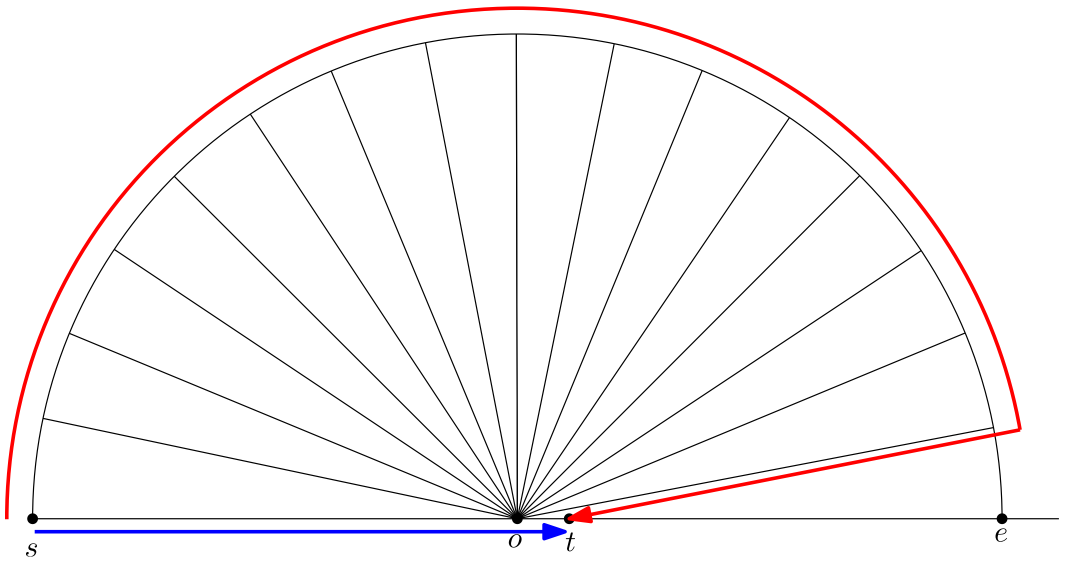


Figure 1.2: An example of greedy path achieving a stretch factor arbitrarily close to $\pi + 1$.

simplicity, let us refer to P' as P in the rest of the section. Note that v_i is now the point where the greedy path *leaves* the edge e_i.

It is important to observe that the greedy path only depends on the current position and the subsequent triangles. Thus if we change the triangles to the left side of a diagonal e_i but make sure that the greedy path still goes through the same entrance point v_i on e_i, the path to the right side of e_i will not be affected. Now given a triangle sequence and the (modified) greedy path, we could transform it from left to right. All o-edges will be repalced by arcs; and all l-edges will be eventually eliminated. We will show that each transformation can only increase the stretch of the greedy path. The details will be discussed below.

Step 1: Transform the first segment on the greedy path to an arc

As explained earlier, the first segment on the transformed greedy path is always an o-edge. Suppose that this segment ends at point v_1 on the first diagonal e_1 (or an extension of e_1), see Fig. 1.3. The intersection of e_1 with st is o_1. Take the arc centered at o_1 with radius $|so_1|$ that intersects e_1 (or the extension of e_1) at d. We replace the greedy path segment sv_1, by an arc $\overset{\frown}{sd}$ followed by an l-edge dv_1. Obviously $|\overset{\frown}{sd}| + |dv_1| > |sv_1|$. After this transformation new path starts with an arc.

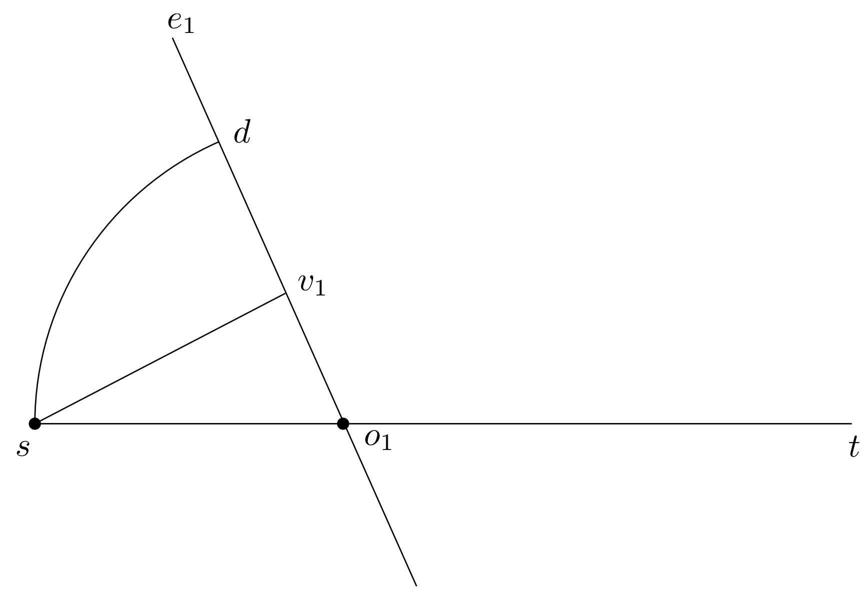


Figure 1.3: Transform the first segment of the greedy path.

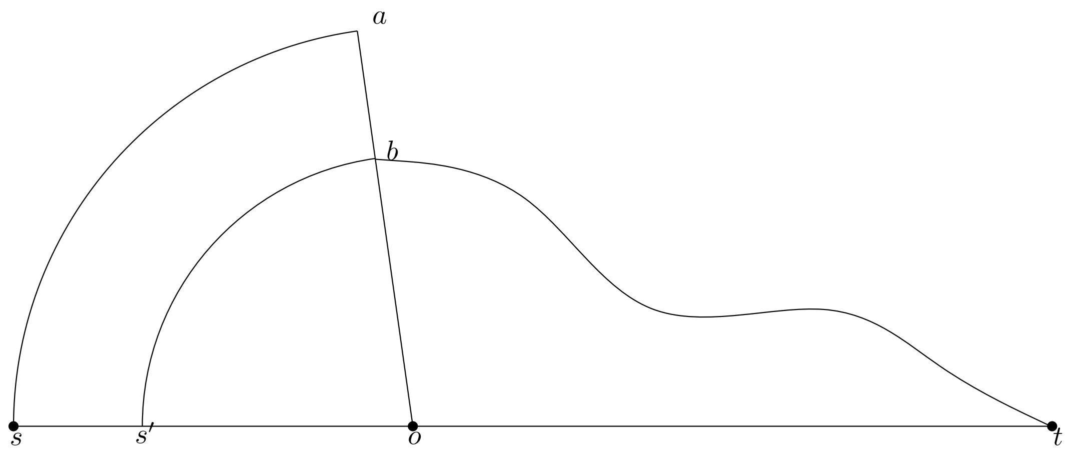


Figure 1.4: Handle the l-edge

Step 2: Transform the greedy path starting with an arc

We will continue to transform the greedy path, during which the worst case stretch is not made smaller. For an l-edge we will remove it. For an o-edge it will be merged into a new arc. The following discussion provides the details.

Case 1: remove an l-edge Take the first l-edge, ab, on diagonal e_i as in Figure 1.4. The greedy path to the left of the diagonal e_i is an arc.

What we want to do is to proportional shrink all the triangles to the left of the diagonal e_i, towards o. This will move the source along the line segment st to a new source s' and move the arc $\overset{\frown}{sa}$ as the arc $\overset{\frown}{s'b}$. Suppose that the greedy path after e_i is $P_{b,t}$. Notice that the operation before e_i will not affect $P_{b,t}$ as long as the entrance to $\triangle_{i+1}$ is at b. We will prove below that if the path after the operation has a worst case stretch of $\pi + 1$ then the orignal path also has a bounded stretch of $\pi + 1$.

Lemma 1.3.2. *If* $|\overset{\frown}{s'b}| + |P_{b,t}| \leq (\pi + 1)|s't|$, *then* $|\overset{\frown}{sa}| + |ab| + |P_{b,t}| \leq (\pi + 1)|st|$.

Proof: Let $p = |s'o|/|so|$, so $p < 1$. Then $p|\overset{\frown}{sa}| = |\overset{\frown}{s'b}|$.

$$\begin{aligned}
|\overset{\frown}{sa}| + |ab| + |P_{b,t}| &= p|\overset{\frown}{sa}| + (1-p)|\overset{\frown}{sa}| + |ab| + |P_{b,t}| \\
&= |\overset{\frown}{s'b}| + |P_{b,t}| + (1-p)|\overset{\frown}{sa}| + |ab| \\
&\leq (\pi + 1)|s't| + (1-p)|\overset{\frown}{sa}| + (1-p)|ao| \qquad (1.1)\\
&\leq (\pi + 1)|s't| + (1-p)(\pi + 1)|so| \\
&= (\pi + 1)|s't| + (\pi + 1)|ss'| \\
&= (\pi + 1)|st|
\end{aligned}$$

□

Case 2: handle an o-edge. Now assume that the greedy path starts with an arc $\overset{\frown}{se}$ arriving at a diagonal e_i. The next step of $P_{s,t}$ is an o-edge eb, in the following triangle with b on diagonal e_{i+1}. Suppose the diagonal e_i and e_{i+1} intersect st at o and o' respectively. Clearly o' is to the right of o. Thus the disk centered at o' with radius $|so'|$ completely includes the disk centered at o with radius $|so|$. We take the point d on e_{i+1} (or extension of e_{i+1}) such that sd is an arc with center at o'. Now we replace the path $\overset{\frown}{se}$ and eb by the arc $\overset{\frown}{sd}$ with an l-edge db. This operation will only make the path longer by the following proof.

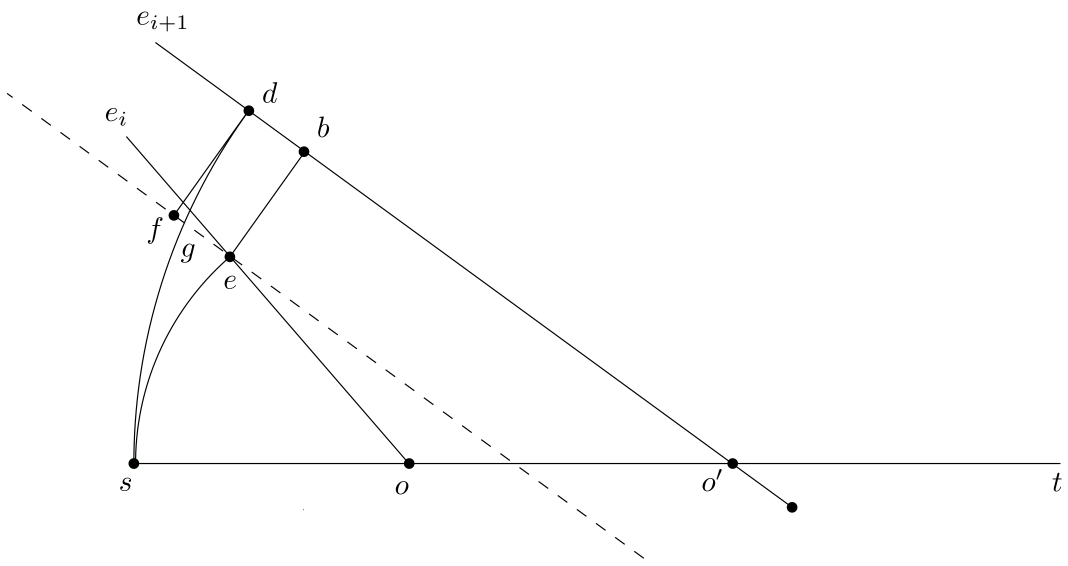


Figure 1.5: In the middle of the path

Lemma 1.3.3. $|\overset{\frown}{se}| + |eb| \leq |\overset{\frown}{sd}| + |db|$.

Proof: See Figure 1.5. We take a line ℓ through e, parallel to e_{i+1}. Take f on ℓ such that fd is perpendicular to e_{i+1} at d. ℓ intersects $\overset{\frown}{sd}$ at g. By triangle inequality, we have

$$\left.\begin{array}{l} |\overset{\frown}{se}| \leq |\overset{\frown}{sg}| + |ge| \leq |\overset{\frown}{sg}| + |db| \\ |eb| \leq |\overset{\frown}{gd}| \end{array}\right\} \Rightarrow |\overset{\frown}{se}| + |eb| \leq |\overset{\frown}{sd}| + |db|$$

□

Step 3: Find the last segment

According to the discussion above we can transform the original greedy path to an arc whose center o stays on st until it hits a point v_n on the last diagonal e_n. This part of the path has length at most $\pi|so| \leq \pi|st|$. Further we know that $|v_nt| \leq |st|$, as s is the furthest point to t among all points on the circle centered at o with radius $|so|$. Thus the total length is at most $(\pi + 1)|st|$.

Unitl now we have finished the proof of Theorem 1.3.1.

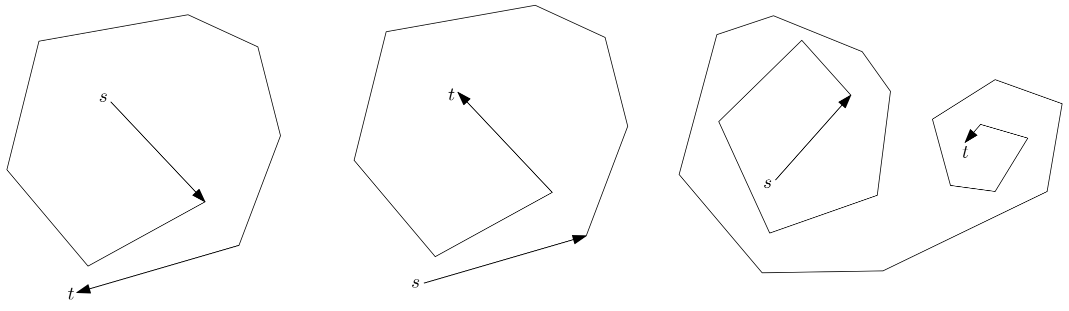


Figure 1.6: Examples of spirals

1.4 Proof of the General Case

Recall that in the general case both the shortest path Q in the sequence of triangles and the greedy, online path P are non-self intersecting polygonal curves. They are both directed, going from the source s to destination t.

Before we proceed to the proof, we first give some definitions.

Definition 1.4.1. *A spiral is growing(outward) if for every segment, $q_i q_{i+1}$, on it, the preceding path before q_i is on the same side of the extension line of $q_i q_{i+1}$. A spiral is shrinking(inward) if and only if when its source and destination are switched, it is outward.*

Figure 1.6 gives three spirals, which are outward, inward and general respectively.

A spiral that is neither outward nor inward can be cut into one outward spiral and one inward spiral.

Therefore we can focus on the cases where the optimal path is a spiral and without loss of generality, we assume all spirals mentioned afterwards turn clockwise.

1.4.1 Bound Outward Spiral Case

The idea of modification that we used before is still valid here. The only difference is that in an outward spiral case, we transform the greedy path to a spiral arc not a circular arc.

1.4.2 Growing Spiral

Given a growing spiral $Q = \{s, q_1, q_2, \ldots, q_{m-1}, t\}$, we first introduce three definitions: *spiral arc, spiral sector, spiral sector plus.* To avoid being redundant, we also use $q_0(q_m)$ to represent $s(t)$.

Definition 1.4.2. *The spiral arc of Q consists of $m-1$ arcs; centered at q_i with radius $|Q_{s,q_i}|$, ith arc C_i spans the extensions of q_iq_{i-1} and $q_{i+1}q_i$, $i = 1, 2, \ldots, m-1$. We denote it by $SA(Q)$.*

Definition 1.4.3. *The spiral sector of Q is the region enclosed by Q, its spiral arc and the line of $q_{m-1}t$. If Q and its spiral arc doesn't intersect except at s, we say the spiral arc and the spiral sector are simple. We denote the spiral sector of Q by $SS(Q)$.*

Definition 1.4.4. *The spiral sector plus of Q is the union of $SS(Q)$ and a circular sector which is centered at t with a radius of $|Q|$; the circular sector attaches $SS(Q)$ along the line of $q_{m-1}t$. If that sector and $SS(Q)$ are disjoint and $SS(Q)$ is simple, we say the spiral sector plus is simple. We denote the spiral sector plus of Q by $SSP(Q)$.*

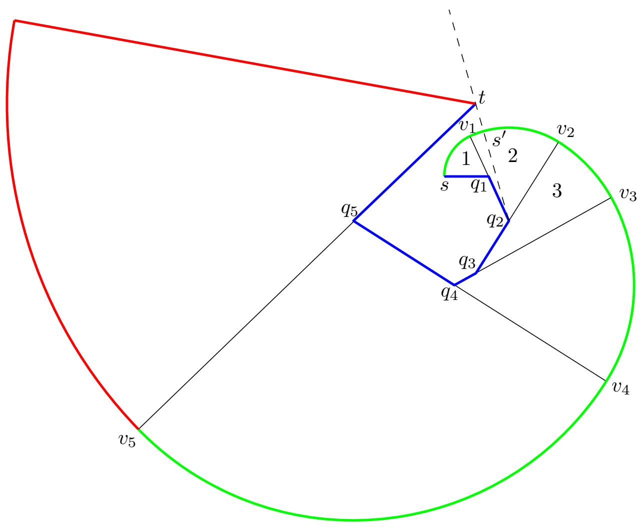


Figure 1.7: The blue path is a growing spiral Q(from s to t). The green path is $SA(Q)$. The whole figure is $SSP(Q)$.

Figure 1.7 illustrates the definitions above. Given a simple growing path Q and a simple spiral sector plus $SSP(Q)$, it's easy to see that $SSP(Q)$ represents a specail case of our problem, in which the optimal path is Q and the left part of the boundary of $SSP(Q)$ is the greedy path P. We now prove the following lemma:

Lemma 1.4.5. *In a simple spiral sector plus of a growing spiral Q, $|P| \leq (2.8\pi + 2)|Q|$*

Proof: Intuitively, the extra cost of the greedy path incurs at turns. The shortest path Q makes a sharp turn at q_i, while the greedy path P has to travel a longer distance along the outside of the turn.

Figure 1.7 shows two types of possibilities for such sectors. At the outer points of the spiral, the wedges defined by the sectors do not intersect the shortest path Q; we say they are of type $RN2$. At the inner points of a spiral, two rays of the resulting wedges intersect one or more outer layers of Q; we call these of type $RN1$, and denote by a_i and b_i the nearest intersections of Q with the two rays. If $\xi_i = Q_{a_i,b_i}$ is part of the shortest path between these points, then it can be shown that over the $RN1$ type wedges: $\sum\limits_{RN1} |r_i|\beta_{i+1} \leq \sum\limits_{RN1} \xi_i$. The right side is no more than $|Q|$. In Figure 1.7, sector 1 is type $RN1$; sector 3 is type $RN2$. Sector 2 is a little special, because a ray of it intersects Q and the other doesn't. For that case, we can cut this sectors into two by adding an infinitesimal edge at q_2. As Figure 1.7 shows, sector 2 is cut by the line joining t and q_2;we use s' to denote the intersection of q_2t and $\widehat{v_1v_2}$. Therefore it won't lose generality to assume there are only two types of sectors.

While for $RN2$ type sectors, the last angle θ_i is at most π, which means the part of P in the last sector(the red path in Figure 1.7) is at most $(\pi+1)|Q|$. The analysis of the left $RN2$ type sectors actually goes to a special simple spiral sector. Take Figure 1.7 as an example. The left $RN2$ type sectors are P_{s',v_5}, which can be regarded as the spiral arc of a spiral, $s'q_2$ plus $Q_{q_2,t}$. We denote the spiral arc of the left $RN2$ type sectors by P'. Let the correspondent growing spiral be $Q' = \{s, q_1, q_2, \ldots, q_{m-1}, t\}$. We know that Q' turns through at most 2π and the first two vertices and the last vertex of which are colinear.

We will prove that $|P'|$ is at most 1.8π times $|Q'|$. We can put Q' into a bounding box such that s, q_1 and t are on an edge of that box as Figure 1.8 shows. We denote $|q_{j-1}q_j|$ by l_j. We also denote the angle Q' turns over at q_j by θ_j. Then we have

$$|P'| = \sum_{i=1}^{m} \theta_i \sum_{j=1}^{i} l_j = \sum_{j=1}^{m} l_j \sum_{i=j}^{m} \theta_i$$

$$|Q'| = \sum_{i=1}^{m} l_i$$

$$\frac{|P'|}{|Q'|} = \sum_{j=1}^{m} (\sum_{i=j}^{m} \theta_i) \frac{l_j}{\sum_{k=1}^{m} l_k}$$

From the above equations, we know that even s and t are not necessarily the same, $\frac{|P'|}{|Q'|}$ will increase if we extend s to t. Since what we are interested in is an upper bound of $\frac{|P'|}{|Q'|}$, we only have to consider the case where s and t are the same. See Figure 1.8. Let q_i be any rightmost point of Q'. We know

$$|Q'_{q_i,t}| \geq |sq_i| \geq \frac{|sb| + |bq_i|}{\sqrt{2}} \geq \frac{|Q'_{s,q_i}|}{\sqrt{2}}$$

Since the angle $Q'_{q_i,t}$ turns over is at most $\frac{3\pi}{2}$, we have

$$\frac{|P'|}{|Q'|} \leq \frac{2\pi|Q'_{s,q_i}| + \frac{3\pi}{2}|Q'_{q_i,t}|}{|Q'_{s,q_i}| + |Q'_{q_i,t}|} = \pi\frac{2 + \frac{3}{2}x}{1 + x}$$

where $x = \frac{|Q'_{q_i,t}|}{|Q'_{s,q_i}|} \geq \frac{\sqrt{2}}{2}$.

It's easy to see that $\frac{|P'|}{|Q'|}$ is at most $\frac{5-\sqrt{2}}{2}\pi (\approx 1.8\pi)$.

Adding up three parts of P, we prove the lemma. □

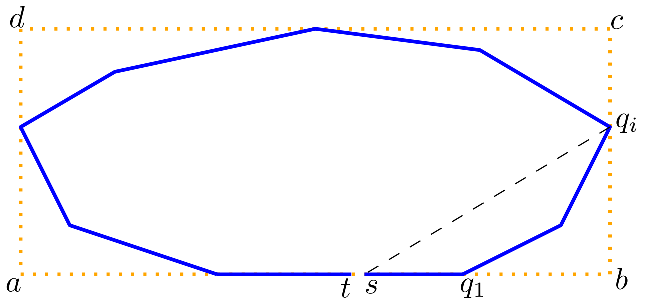


Figure 1.8: From s to t, path Q turns over 2π. If t is s, Q becomes the boundary of a convex polygon.

Let c be $2.8\pi + 2$. The following lemma shows that the spiral sectors are the "worst" cases in growing spirals.

Lemma 1.4.6. *For any growing spiral case with n triangles, $|P| \leq c|Q|$*

Proof: We use induction on m, which means triangle sequence before diagonal $q_{n-m}v_{n-m}$ is a simple SSP.

The base case, when $m = 0$, is just Lemma 1.4.5. Now we assume the portion before diagonal $q_k v_k$ is a simple SSP, i.e. $m = n - k$. As Figure 1.9 shows, first we replace $P_{k-1,k+1}$ by $\overset{\frown}{v_{k-1}b}$ and bv_{k+1}. It's easy to show that the new path is longer than P. We call it $\bar{P}$. Then we scale down all the triangles to the left of the diagonal $q_{k+1}v_{k+1}$, towards q_{k+1} until b matches v_{k+1}. It's easy to be shown that the shrinked SSP is fully contained in the green region, i.e. it will not overlap triangles $\triangle_i$, $i = k+2, \ldots, n$. We call the greedy path obtained after scaling $\tilde{P}$.

Let $\frac{|q_{k+1}v_{k+1}|}{|q_{k+1}b|}$ be α. We have

$$\begin{aligned} |\tilde{P}_{s',v_{k+1}}| &= \alpha|\bar{P}_{s,b}| \\ |Q_{s',q_{k+1}}| &= \alpha|Q_{s,q_{k+1}}| \\ |bv_{k+1}| &= (1-\alpha)|q_{k+1}b| \end{aligned}$$

By the induction hypothesis, we have

$$|\tilde{P}| = |\tilde{P}_{s',v_{k+1}}| + |P_{v_{k+1},t}| \leq c|Q_{s',t}| \tag{1.2}$$

Then we bound the original greedy path by

$$\begin{aligned} |P| \leq |\bar{P}| &= \alpha|\bar{P}_{s,b}| + (1-\alpha)|\bar{P}_{s,b}| + (1-\alpha)|q_{k+1}b| + |P_{v_{k+1},t}| \\ &\leq c|Q_{s',t}| + (1-\alpha)(|\bar{P}_{s,b}| + |q_{k+1}b|) \\ &\leq c(|Q_{s',q_{k+1}}| + |Q_{q_{k+1},t}|) + c(1-\alpha)|Q_{s,q_{k+1}}| \\ &= c|Q| \end{aligned} \tag{1.3}$$

When $m = n-k$, the statement is true. Therefore the lemma is proved. ☐

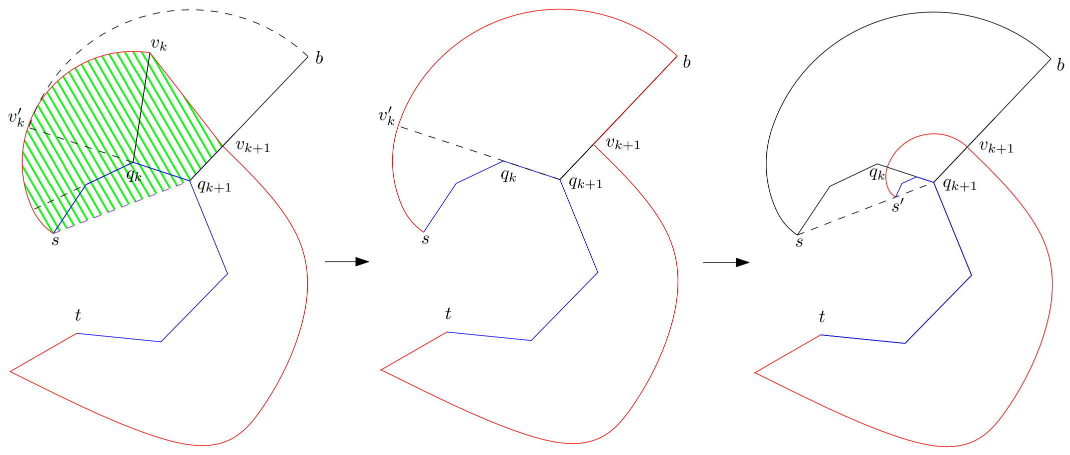


Figure 1.9: steps to transform a growing spiral case to a SSP

1.4.3 Shrinking Spiral

The induction above doesn't work for shrinking spiral cases, because scaling down preceding triangles may cause conflict with the following triangles, i.e. non-repeating property doesn't hold. As a result we adopt a charging scheme to measure how much every portion of Q "contributes" to $|P|$.

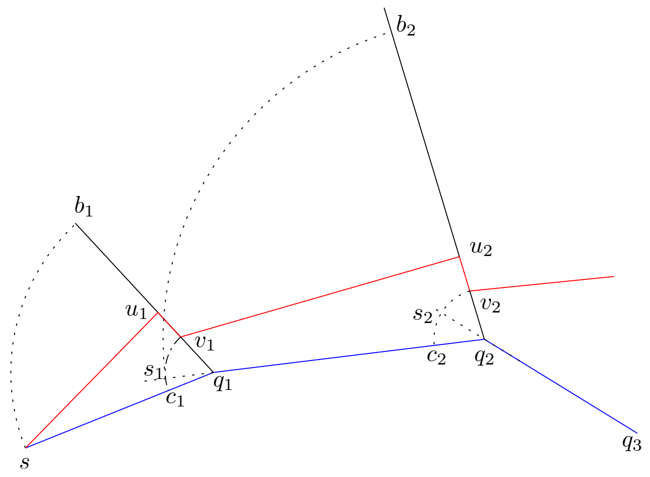


Figure 1.10: $\widehat{sb_1}$ is centered at q_1. $\widehat{c_1v_1}$ with center q_1 intersects the extension of q_1q_2 at s_1. $\widehat{s_1b_2}$ is centered at q_2. $\widehat{c_2v_2}$ whose center is q_2 intersects the extension of q_2q_3 at s_2.

See Fig 1.10. The red path is Q(after replacing b-segments). Let $\angle b_1q_1s$ and $\angle s_1q_1c_1$ be θ_1 and α_1 respectively. We use θ_i and α_i analogously. Then we have

$$|su_1| + |u_1v_1| \leq |\widehat{sb_1}| + |b_1v_1| = (1+\theta_1)|sc_1| + \alpha_1|c_1q_1| + |\widehat{s_1v_1}|$$

We charge $(1+\theta_1)|sc_1|$ to sc_1. Nothing will be charged to sc_1 afterwards, i.e. sc_1 has no contribution to $|P|$ after the first triangle. Left two terms are relevant to c_1q_1. We charge $\alpha_1|c_1q_1|$ to c_1q_1 and leave the last term to the next step.

During the second step, we have

$$\begin{aligned} |\widehat{s_1v_1}| + |v_1a_2| + |a_2v_2| \leq |\widehat{s_1b_2}| + |b_2v_2| \\ = (1+\theta_2)|s_1c_2| + \alpha_2|c_2q_2| + |\widehat{s_2v_2}| \\ = (1+\theta_2)(|c_1q_1| + |q_1c_2|) + \alpha_2|c_2q_2| + |\widehat{s_2v_2}| \end{aligned} \quad (1.4)$$

Similarly, we charge $(1+\theta_2)|c_1q_1|$, $(1+\theta_2)|q_1c_2|$ and $\alpha_2|c_2q_2|$ to c_1q_1, q_1c_2

and c_2q_2 respectively. At the moment, c_1q_1 stop contributing and it's charged by $1+\alpha_1+\theta_2$ times its length. q_1c_2's total contribution is $1+\theta_2$ times its length, which all happens in Step 2. c_2q_2 contributes $\alpha_2|c_2q_2|$ during Step 2 and it will continue to contribute. We can keep doing this. In the perspective of this charge scheme, some portion of a growing spiral may contribute arbitrarily huge, which is an easy observation from a SSP. Fortunately, in a shrinking spiral case, it can't happen because of the following crucial observation:

For any point p on the optimal path $Q_{s,t}$, $Q_{s,p}$ has stopped contributing before q_iv_i if $|q_iv_i| \leq |Q_{p,q_i}|$

δ denotes a portion of q_lq_{l+1}. Suppose it contributes until v_kq_k, we can claim that its contribution is $|\delta|$ times $1+\gamma_\delta$ which is at most $(1+\sum_{i=l+1}^{k-1}\alpha_i+\theta_k)$, where the second term is the sum of angles that $Q_{l,k}$ turns through and the third term is $\angle q_{k-1}q_kv_k$. We denote the second term by A_{l+1}^{k-1}.

We proceed to prove a lemma that gives an upper bound of γ_δ.

Lemma 1.4.7. *For any portion δ on sq_1, $\gamma_\delta \leq 3\pi$.*

Other portion can be proved in the same way.

Proof: Suppose δ doesn't stop contribution after v_kq_k and $A_1^{k-1}+\theta_k > 3\pi$. That implies the extension of v_kq_k will intersect $Q_{q_1,t}$ at some point p. See Figure 1.11. Hence we have

$$|v_kq_k| \leq |q_kp| < |Q_{p,t}| < |Q_{q_1,t}|$$

which means δ has stopped contribution before v_kq_k.

□

With this charge scheme, we prove the following lemma:

Lemma 1.4.8. *For a shrinking spiral, $|P| \leq (3\pi+1)|Q|$*

1.4.4 Combined Spiral

Now We are going to discuss the last case: combined spirals. In this case the optimal path is growing first then shrinking. See Fig.1.12. Q_{s,q_k} is a growing spiral, whose greedy path is a concatenation of P_{s,v_k}, $\overset{\frown}{v_kt'}$ and $t'q_k$. $Q_{s',t}$ is a shrinking spiral, whose greedy path is $\overset{\frown}{s'v_k}$ followed by $P_{v_k,t}$.

Before bounding $|P|$, we proceed to prove a lemma.

Lemma 1.4.9. $|v_{k+1}q_{k+1}| \leq |v_ko_k| + |o_kq_{k+1}|$*, for $k = 2, \cdots, n$.*

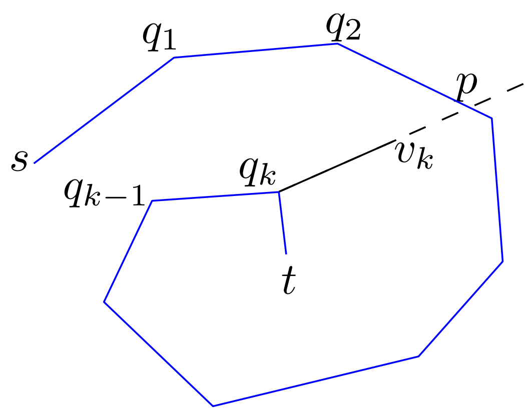


Figure 1.11: Only the optimal path is showed.

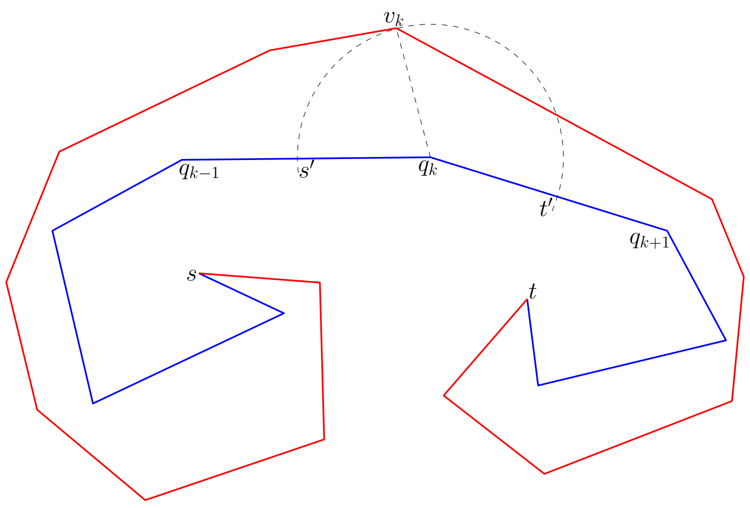


Figure 1.12: The blue path is Q and the red one is P. The arc centered at q_k with radius v_kq_k intersects two rays of q_kq_{k-1} and q_kq_{k+1}, at s' and t' respectively.

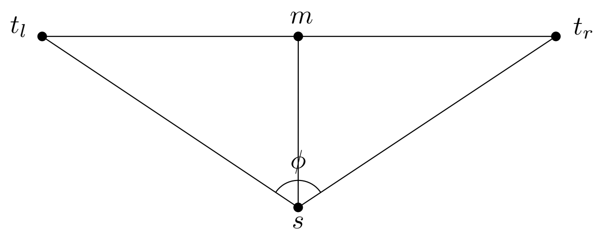


Figure 1.13: Establishing a generalized lower bound.

Proof: See the first part of Figure 1.9. We have $|v_{k+1}q_{k+1}| \leq |bq_{k+1}| = |v_kq_k| + |q_kq_{k+1}|$

$\square$

By Lemma 1.4.9, it's easy to see $|v_kq_k| \leq |Q_{s,q_k}|$. We denote $\angle s'q_kv_k$, $\angle v_kq_kt'$ and $|v_kq_k|$ by θ_1, θ_2 and r respectively. We have

$$\begin{aligned}|P_{s,v_k}| + (\theta_2+1)r &\leq (2.8\pi+2)|Q_{s,q_k}| \\ |P_{v_k,t}| + \theta_1 r &\leq (3\pi+1)(r+|Q_{q_k,t}|) \\ \theta_1+\theta_2 &\geq \pi\end{aligned}$$

Therefore we obtain

$$\begin{aligned}|P| = |P_{s,v_k}| + |P_{v_k,t}| &\leq (2.8\pi+2)|Q_{s,q_k}| + (3\pi+1)|Q_{q_k,t}| + (3\pi-\theta_1-\theta_2)r \\ &\leq (4.8\pi+2)|Q_{s,q_k}| + (3\pi+1)|Q_{q_k,t}| \\ &\leq (4.8\pi+2)|Q| \qquad (1.5)\end{aligned}$$

Here is the final theorem we get:

Theorem 1.4.10. *The length of the greedy path is at most* $(4.8\pi+2)$ *times the length of the shortest path.*

1.5 Lower Bound for all On-line Algorithms

First we prove that the lower bound for all on-line algorithms when t is not known is $\sqrt{2}$.

Lemma 1.5.1. *No on-line algorithm can guarantee a competitive ratio smaller than* $\sqrt{2}$ *if the destination* t *is not known initially.*

The proof is from [1].

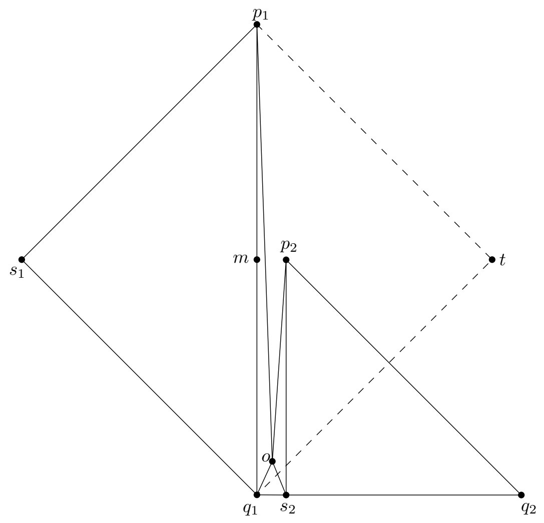


Figure 1.14: Going to m is the best choice for the worst situtin, if the robot is at s_1.

Proof: We take an isosceles triangle with an angle ϕ at vertex s; the first segment is $t_l t_r$; see Fig. 1.13. The target becomes visible only when the line segment $t_l t_r$ is reached. If this happens to the left of the midpoint m, the target may be to the right, and vice versa. In any case the path length is at least the distance from s to m plus a half of $t_l t_r$. We obtain the ratio by simple trigonometry

$$\rho \geq cos\frac{\phi}{2} + sin\frac{\phi}{2} = \sqrt{1 + sin\phi}$$

When $\phi = \frac{\pi}{2}$, we have the well-known lower bound of $\sqrt{2}$ stemming from a rectangular isosceles triangle.

$\square$

We can have the following theorem.

Theorem 1.5.2. *No on-line algorithm can guarantee a competitive ratio smaller than $\sqrt{2}$ even the destination t is known initially.*

Proof: See Figure 1.14. First five triangles are $s_1p_1q_1$, p_1q_1o, oq_1s_2, os_2p_2 and $s_2p_2q_2$; $\triangle s_ip_iq_i$ is an isosceles right triangle whose hypotenuse has a length of $2^{-\frac{i}{2}}$. The start is s_1. $s_1p_1tq_1$ is a square; t is the destination. According to Lemma 1.5.1, going to m, the middle point of p_1 and q_1 is the best decision when the robot is at s_1. We have

$$|P_{s_1 s_2}| = s_1 m + m q_1 + \varepsilon_1 \tag{1.6}$$
$$|Q_{s_1 s_2}| = s_1 q_1 + \varepsilon_2 \tag{1.7}$$

where $P_{s_1 s_2}$ is the path given by any algorithm between s_1 and diagonal os_2; $Q_{s_1 s_2}$ is the optimal path between s_1 and diagonal os_2; $\varepsilon_1, \varepsilon_2 \to 0$ when $p_2 s_2$ and o are moving to $p_1 q_1$ and q_1 respectively.

We can also find that if $\triangle o q_1 s_2$ shrinks to point q_1, three vertex of $\triangle s_2 p_2 q_2$ and t constitute another rectangle; at the same time, any path should leave diagonal os_2 from s_2, which means it is as same as the original problem, where the only difference is that the distance between the start and target is shortened by a scale of $\sqrt{2}$. We can do this things iteratively and the robot can be arbitrarily close to t. In this example, it's easy to see that no on-line strategy can give a competitive ratio less than $\sqrt{2}$. $\square$

Chapter 2

Variations of Greedy Routing

2.1 Routing in a Tetrahedron Sequence(3D Version)

Our algorithm is easy to generalized to higher dimensions. Given a tetrahedron sequence, we just route the message to the closest point on the face that's shared by the next tetrahedron. However, unlike triangle sequence on a plane, greedy routing in a tetrahedron won't give a competitive path. Figure 2.1 is an example. But this scheme still works if the tetrahedrons are 'fat'.

Definition 2.1.1. *A triangle is α-fat if its inner angles are all at least α. A triangle sequence is α-fat if every triangle in it is α-fat. A tetrahedron is α-fat if its faces are α-fat.*

Theorem 2.1.2. *To any sequence of n α-fat triangles in Euclidean Space, the greedy routing algorithm is $cot(\frac{\alpha}{2})$-competitive.*

Proof: By induction. Simple for $n = 1$. Suppose it is true for $n = k$: in a sequence of k α-fat triangles, $|P_{s,t}| \leq \cot(\frac{\alpha}{2})|Q_{s,t}|$. See Figure 2.2. We have

$$\frac{|sv_1|}{|sq_1| - |v_1q_1|} = \frac{\sin(\theta)}{1-\cos(\theta)} = \cot(\frac{\theta}{2}) \leq \cot(\frac{\alpha}{2}) \tag{2.1}$$

According to the assumption, we know $|P_{v_1,t}| \leq \cot(\frac{\alpha}{2})|Q_{v_1,t}|$. Therefore:

$$\begin{aligned}
\cot(\frac{\alpha}{2})|Q_{s,t}| &= \cot(\frac{\alpha}{2})(|sq_1| + |Q_{q_1,t}|) && (2.2)\\
&= \cot(\frac{\alpha}{2})(|sq_1| - |v_1q_1|) + \cot(\frac{\alpha}{2})(|v_1q_1| + |Q_{q_1,t}|) && (2.3)\\
&\geq |sv_1| + \cot(\frac{\alpha}{2})|Q_{v_1,t}| \geq |sv_1| + |P_{v_1,t}| = |P_{s,t}| && (2.4)
\end{aligned}$$

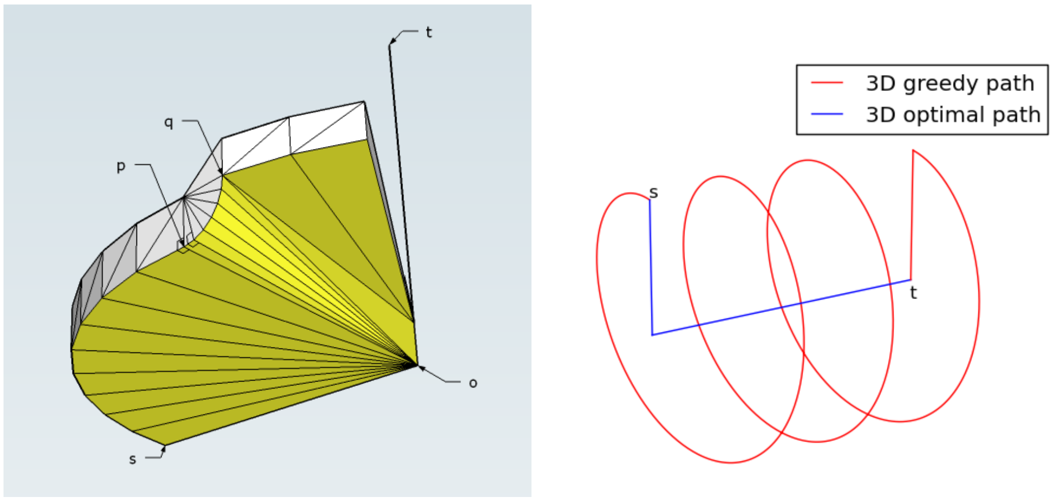


Figure 2.1: From s, the greedy path will reach p along the short edges of the yellow triangle faces, which are on the same plane orthogonal to line to. Then the path goes to next plane orthogonal to line to along circular curve pq. The right figure shows what optimal path and greedy path look like at last.

$\square$

The following theorem is an analog of Theorem 2.1.2 in Euclidean space.

Theorem 2.1.3. *To any sequence of n α-fat tetrahedron, the greedy routing algorithm is $cot(\frac{\alpha}{2})$-competitive.*

The proof is analogy to that of Theorem 2.1.2.

2.2 Apply to a Simple Polygon Sequence

In the section we discuss the application of different routing scheme to a simple-connected sequence of simple polygons.

A simple-connected sequence of simple polygons means in the sequence two ajacent polygons share only one edge and the inner of two simple polygons are disjoint. For brevity, we refer to simple-conneted sequence of simple polygons as polygon sequence in the following content.

We will prove the following theorem:

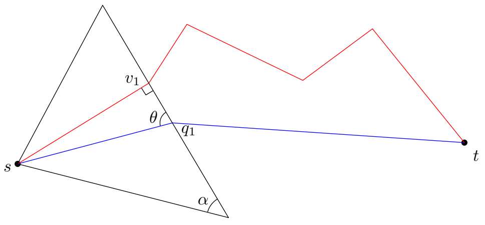


Figure 2.2: Routing in a fat triangle sequence. The blue path is the optimal one; the red path is the greedy one. $\angle sq_1v_1 = \theta \leq \alpha$.

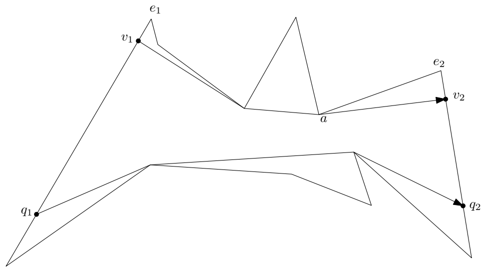


Figure 2.3: Optimal path and its one-sided greedy path

Theorem 2.2.1. *If a greedy algorithm is c-competitive for online routing problem in a non-repeating sequence of triangles, then it's also c-competitive for online routing problem in a simple connected sequence of simple polygons.*

2.2.1 Outline of Proof

Analogy to a triangle sequence, the lemma below indicates that we only need to consider the case that the greedy path is one-sided to the optimal path.

Lemma 1. *If the greedy path $P_{s,t}$ intersects the optimal path $Q_{s,t}$ at m, then there are two ordered polygon sequences, $\{S_i^1\}$ and $\{S_i^2\}$, such that $P_{s,m}$ and $Q_{s,m}$ are the greedy and optimal solutions for $\{S_i^1\}$ respectively, and $P_{m,t}$ and $Q_{m,t}$ are the greedy and optimal solutions for $\{P_i^2\}$ respectively.*

From now on we will only consider the one-sided case.

See Fig. 2.3. The optimal path enters the polygon at q_1 and exits at q_2. The greedy path enters at v_1 and exits at v_2. We know that the greedy path is the geodesic curve between v_1 and v_2; additionally, the optimal path is the geodesic curve between q_1 and q_2. We also find the angle between diagonal e_2 and the last segment of the greedy path (in this example, it's $\angle av_2q_2$) is at least $\frac{\pi}{2}$.

The follow lemma describes the shape of the polygon between the optimal path and the greedy path.

Lemma 2. *If the greedy path and optimal path doesn't intersect in the polygon, then both pathes are spirals with different spiral directions.*

In another way to say, the boundary of the polygon between the optimal path and the greedy path consists of four parts: one entrance edge, one exit edge,

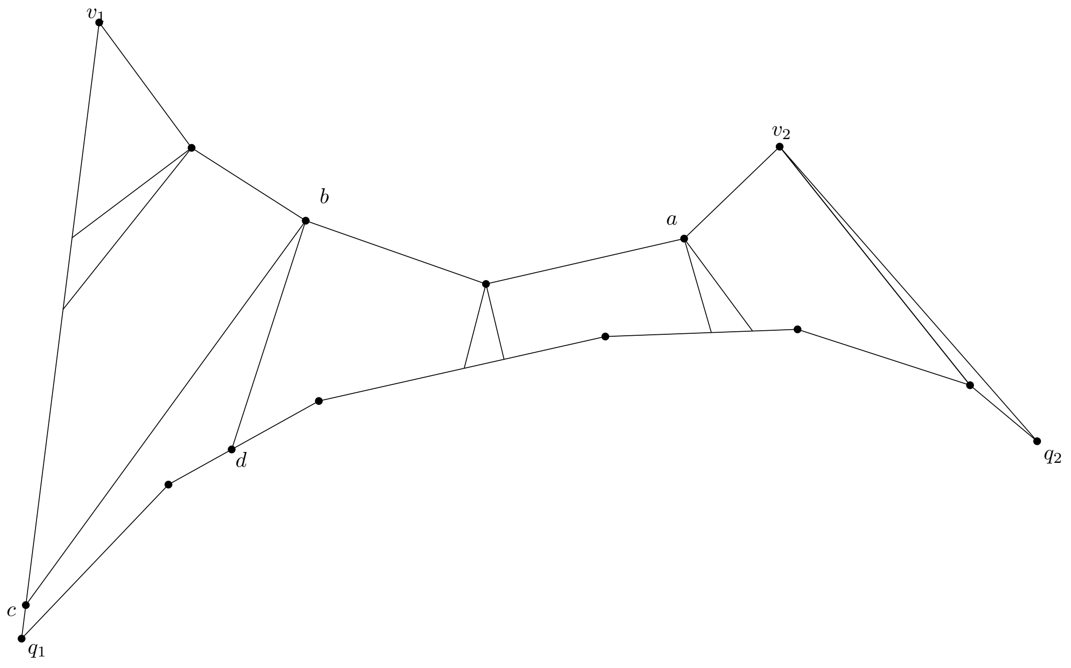


Figure 2.4: Cut of an hour-glass

two concave curves. We call it hour-glass polygon or hour-glass instead. We refer to those two edges as the heads of an hour-glass.

2.2.2 Cutting an Hour-glass

We will cut the hour-glass into triangles, then apply the greedy algorithm to those triangles to get a new greedy path, P_{tri}. The new greedy path will coincide with the original one. At the same time, the optimal path is still in the hour-glass, which means the optimal path will stay the same too. Finally from the competitiveness of greedy algorithm in a triangle sequence, we attain the competitiveness of greedy algorithm in a polygon sequence. Just remind that we will cut the hour-glass not triangulate it, because we may add new vertices on its boundary.

Fig. 2.4 is an hour-glass. The boundary between v_1 and v_2 is the greedy path. Each vertice between v_1 and v_2 has two normals correspondent to the edges it is incident to. Those normals will not intersect with each other in the hour-glass because all vertices are reflex. We can always do this because $\angle av_2q_2$ is at least $\frac{\pi}{2}$. There are three different kinds of polygons in the division of an hour-glass. The first two are showed in Fig.2.6.

- Type 1 is a fox-shape polygon. We cut it along the unique triangulation

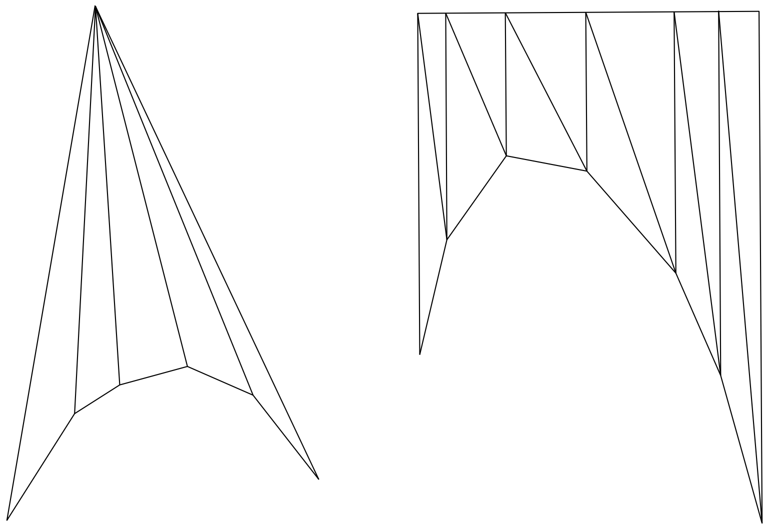


Figure 2.5: how to cut Type 1 and Type 2

it has.

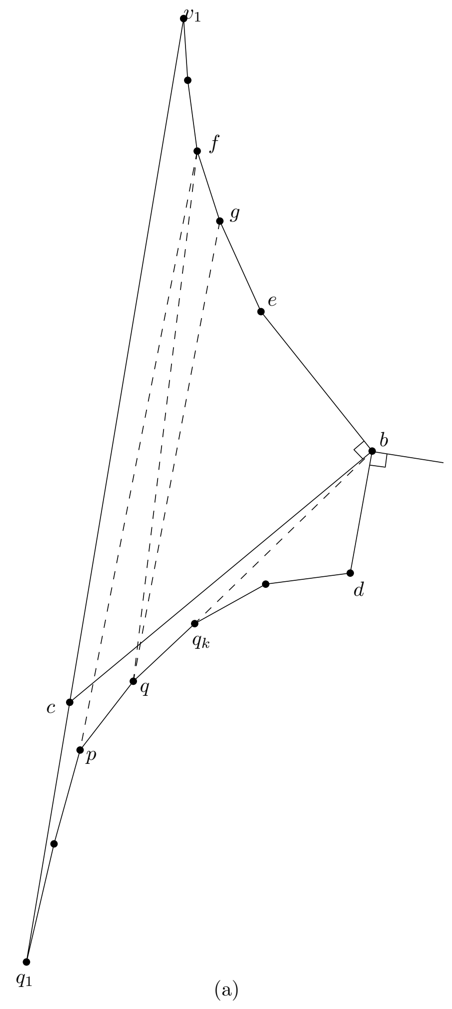
- Type 2 is an hour-glass whose heads are parallel and parts between them are an edge perpendicular to heads and a piece-wise linearly concave curve.

When the norm intersects v_1q_1, things become a little complicated. See bcq_1db in Fig.2.4 as an example.

Let the first normal that intersects the optimal path is bd. The hour-glass after bd consists of first two types of polygons. Now we consider the part before bd. There are two different cases.

1. The normal before bd still has b as an end.
   See Fig. 2.6(a). From o_1 to d along the optimal path, find the first vertex that is visible to b. Let it be q_k and connect it to b. The polygon is divided into two parts by q_kb: one is type 1; the other is enclosed by v_1q_1 and two concave curves; the angle between those concave curves, $\angle ebq_k$, is greater than $\frac{\pi}{2}$.

2. The normal before bd does not have b as an end.
   See Fig. 2.6(b). Let ec be the normal right before bd. From o_1 to d along the optimal path, find the first vertex, q_k, that is visible to e. Connect eq_k. The polygon is divided into two parts by eq_k. For the part after eq_k, it's easy to prove that the optimal curve q_kd is visible to e. The left



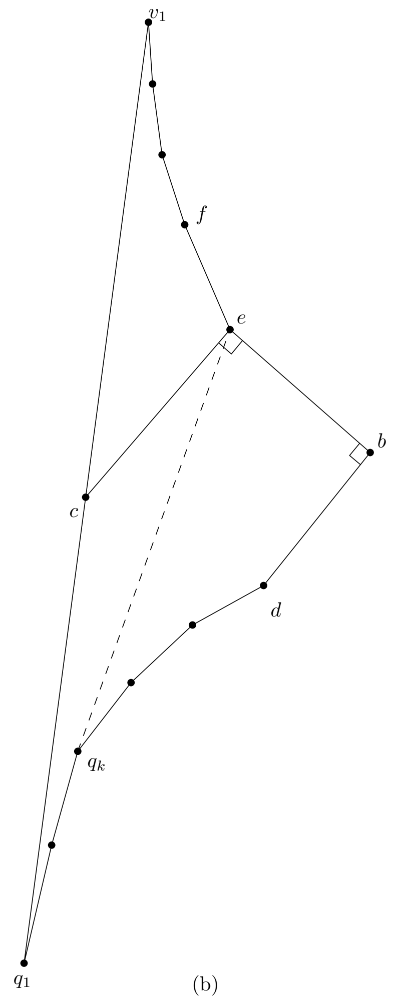


Figure 2.6: how to cut Type 3

part is enclosed by v_1o_1 and two concave curves; the angle between those concave curves, $\angle feq_k$, is greater than $\frac{\pi}{2}$.

The last question is how to triangulate the polygon that is bounded by a segment and two concave curve. The answer is every triangulation will work. See Fig. 2.6(a). When the greedy path arrives at f, the next triangle will have one or two vertices from curve o_1d. If it has two, like $\triangle fpq$, then the greedy path will stay at f. If it has one, like $\triangle fgq$, then the greedy path will go to g, because $\angle fgq > \angle ebq_k > \frac{\pi}{2}$.

2.3 Dual-direction Routing

When it comes to robot searching problem, it's a natural idea to use two robots that start routing at s and t simultaneously to reduce the searching time, i.e., one robot goes to the last polygon from point s in the first polygon; the other goes to the first polygon from point t in the last polygon with the same speed; when they find the other is in the same polygon, we will go directly to each other along the shortest path between them.

The following theorem implies that dual-direction routing will cut down the searching time to a half in the worst case, which observes the intuition.

Theorem 3. *Dual-direction greedy algorithm gives each robot a path which is at most $\frac{1}{2}\sigma P_{s,t}$, where ρ is the competitive ratio.*

Proof: See Figure 2.7. Let the robots at s and t be R_s and R_t respectively. W.l.o.g, we assume that R_s arrives at v_m when R_t reaches v_t on a diagonal e_t. At that moment, they know they are in the same polygon, so they will go along the shortest path between v_m and v_t to meet. e_s is the last diagonal that R_s passes through. q_s and q_t denote the optimal path's intersections with e_s and e_t respectively.

From the idea of cutting an hour-glass, we can find a point q_e on Q_{q_s,q_t} such that the connotation of P_{s,v_m} and v_mq_m is the greedy path correspondent to Q_{s,q_m} as the optimal path.

Therefore We have

$$\begin{aligned} |P_{s,v_m}| + |v_mq_m| &\leq \rho|Q_{s,q_m}| \\ |P_{t,q_t}| + |v_tq_t| &\leq \rho|Q_{t,q_t}| \\ |Q_{v_m,v_t}| &\leq |v_mq_m| + |Q_{q_m,q_t}| + |q_tv_t| \end{aligned} \tag{2.5}$$

Sum them up, then we have

$$|P_{s,v_m}| + |P_{t,q_t}| + |Q_{v_m,v_t}| \leq \rho|Q_{s,q_m}| + |Q_{q_m,q_t}| + \rho|Q_{q_t,t}| \leq \rho|Q_{s,t}| \tag{2.6}$$

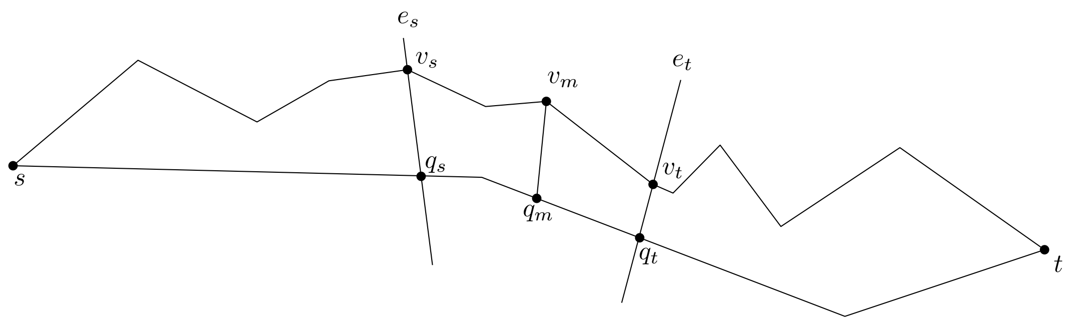


Figure 2.7: bounding the path that one robot goes along until it meets another robot.

Because $|P_{s,v_m}| = |P_{t,q_t}|$, the distance one robot goes is

$$\frac{1}{2}(|P_{s,v_m}| + |P_{t,q_t}| + |Q_{v_m,v_t}|) \leq \frac{1}{2}\rho|P_{s,t}| \tag{2.7}$$

□

2.4 Parallel Routing

The problem of greedy routing is that the flow of traffic will converge through every triangle. The worst result is all traffic will congest to a vertex, when routing from a shorter edge of an obtuse triangle to another shorter edge. See Figure 2.8 as an illustration. Congestion will deteriorate the performance of the network, like load balance, resilience and speed. An observation of Figure 2.8 suggests that we can control congestion by spreading the flow to the white region that is not used by greedy routing.

We have seen that α-fatness simplifies the proof of competitiveness of greedy routing a lot. In this section we will see that if the triangle sequence is α-fat, parallel routing will give competitive heuristic paths and avoid congestion. All triangles the following proofs are α-fat if there is no declaration. We know $\alpha \leq \frac{\pi}{3}$. If $\alpha = \frac{\pi}{3}$, all triangles are equilateral and it's easier to analyze. Therefore we always assume $0 < \alpha < \frac{\pi}{3}$.

2.4.1 Algorithm

Definition 2.4.1 (Parallel Routing). *For a message at point p inside $\triangle_i$ towards destination t,*

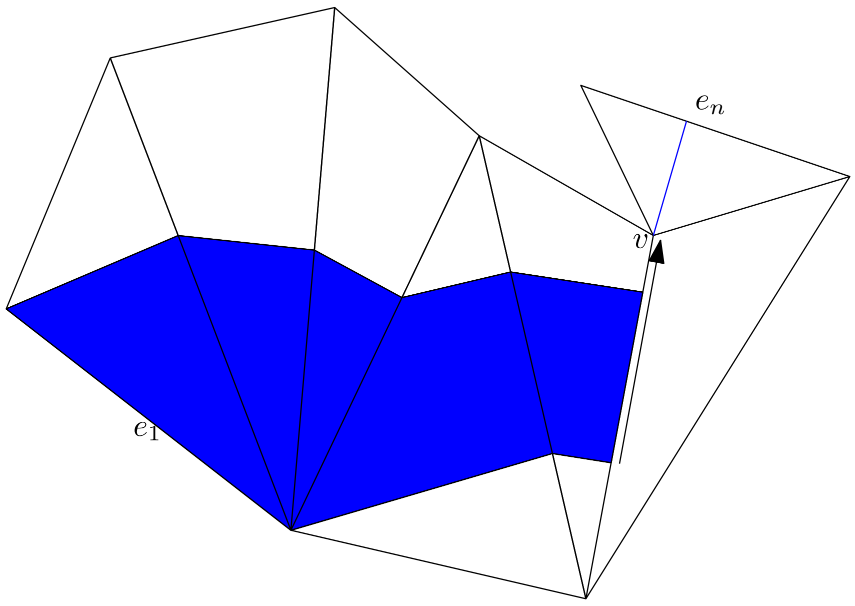


Figure 2.8: The blue polygon is the region that could be passed when routing from e_1 to e_n. It could be seen as a traffic flow from e_1 to e_n under greedy routing. The flow will congest to v.

1. *If t is not in $\triangle_i$, it is routed towards the exit edge along the direction parallel to the left edge of triangle $\triangle_{i+1}$ in the sequence S.*
2. *If $\triangle_i$ is the last triangle in S, then it is routed along the shortest possible path from p to t.*

When the p is s, the exit edge is known but the entrance edge is not given. Pick any one of those two edges as the entrance and the other one gives the routing direction.

Theorem 2.4.2. *Given a sequence S of α-fat triangles, the parallel routing algorithm finds a path of length at most ρ times the length of the shortest path following the same sequence S, where ρ is a constant independent of the input.*

We follow the most notation of chapter 1. We call the path given by parallel routing is the *parallel path*. The *parallel path* from s to t is represented by $P_{s,t}$. The shortest path is $Q_{s,t}$. For a triangle sequence S, vertices incident to the entrance edge and exit edge of the same triangle are called *pivot vertex*. The polygonal path between s and t, passing through pivot vertices orderly is called the *prime path* of S.

We preceed to bound the length of each portion of parallel path between two consecutive segments of the prime path by the optimal path.

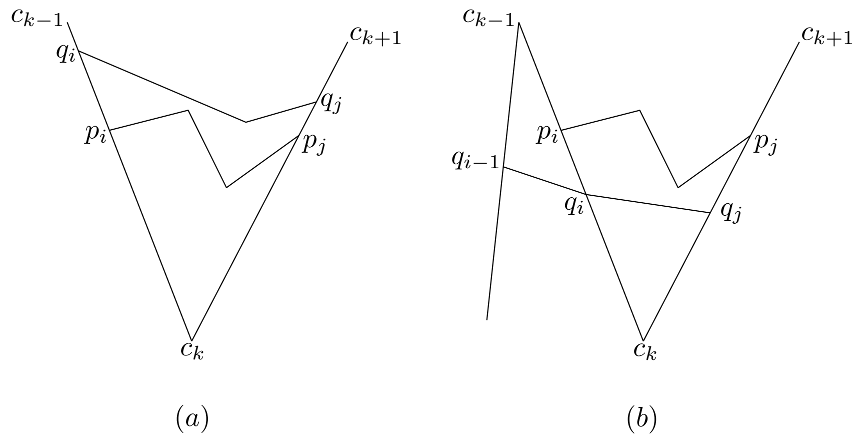


Figure 2.9: c_{k-1}, c_k, c_{k+1} are pivot vertices. In (a), $|q_i c_k \geq p_i c_k|$ so $|P_{i,j}|$ can be bounded by $|Q_{i,j}|$. In (b), $|q_i c_k < p_i c_k|$ so $|P_{i,j}|$ is bounded by $|Q_{i-1,j}|$.

See Figure 2.9. Let $|p_i c_k|$ and $\angle c_{k-1} c_k c_{k+1}$ be l and θ respectively. We denote the length of $P_{i,j}$ by $F(l, \theta)$. By α-fatness, we can prove the following lemma.

Lemma 2.4.3. *$F(l, \theta) \leq l\frac{(2\cos\alpha)^k - 1}{2\cos\alpha - 1}$, where $k = \lceil \frac{\theta}{\alpha} \rceil$.*

Now we bound the length of $p_i c_k$ by considering two cases.

Case 1: $|q_i c_k| \geq |p_i c_k|$. See (a) of Figure 2.9 for an example. If $\theta \leq \frac{\pi}{2}$,

$$|Q_{i,j}| \geq |q_i c_k| \sin\theta \geq |q_i c_k| \sin\alpha \tag{2.8}$$

If $\theta > \frac{\pi}{2}$, $|Q_{i,j}| \geq |q_i c_k|$. Therefore,

$$|p_i c_k| \leq |q_i c_k| \leq \frac{|Q_{i,j}|}{\sin\alpha} \tag{2.9}$$

Case 2: $|q_i c_k| < |p_i c_k|$. Similarly, we have

$$|p_i c_k| \leq |c_{k-1} c_k| \leq \frac{|Q_{i,j}|}{\sin\alpha} + \frac{|Q_{i-1,i}|}{\sin\alpha} = \frac{|Q_{i-1,j}|}{\sin\alpha} \tag{2.10}$$

Therefore we can finish the proof of Theorem 2.4.2.

Proof: Since $\theta \le 2\pi$,

$$\begin{aligned}
|P_{s,t}| &= \sum |P_{i,j}| \le \sum |p_i c_k| \frac{(2\cos\alpha)^k - 1}{2\cos\alpha - 1} \\
&\le \sum \frac{|Q_{i-1,j}|}{\sin\alpha} \frac{(2\cos\alpha)^k - 1}{2\cos\alpha - 1} \le \sum \frac{|Q_{i-1,j}|}{\sin\alpha} \frac{(2\cos\alpha)^{k^*} - 1}{2\cos\alpha - 1} \\
&\le \frac{2[(2\cos\alpha)^{k^*} - 1]}{(2\cos\alpha - 1)\sin\alpha} \sum |Q_{i,j}| = \rho |Q_{s,t}| \qquad (2.11)
\end{aligned}$$

where $k^* = \lceil 2\pi/\alpha \rceil$, $\rho = \frac{2[(2\cos\alpha)^{k^*}-1]}{(2\cos\alpha-1)\sin\alpha}$. $\square$

Chapter 3

Applications

3.1 Routing Scheme of Sensor Networks

The first application of local routing in a triangulation is the problem of routing in a wireless sensor network deployed in a complex geometric domain Σ with holes. This is a common situation for large-scale sensor networks, as the shape of the sensor deployment region (due to obstacles, terrain variation and other deployment forbidding factors) necessarily comes to play with the network design and management.

3.1.1 Introduction

The goal is to find short paths of a specific *homotopy type*, i.e., paths that go around holes in some specific ordering. In the example of Figure 3.1, there are many different ways to "thread" a route from s to t in the network with three holes. Observe that paths α, β, γ are all different in a global sense, in that, e.g., one can't deform α to β without "lifting" it "over" some hole. In contrast, paths γ and δ are only different in a local sense; one can deform γ to δ continuously through local changes, keeping δ within the domain. This difference is characterized by the *homotopy type* of a path. Two paths in a domain are *homotopy equivalent* if one path can be continuously deformed to the other, while staying within the domain.

For paths in a network, differences in homotopy types are differences at a global scale, and are crucial factors in adapting to large dynamic obstructions – such as fires or floods that gradually destroy sensors in a region. The obvious method of using shortest paths globally, and making local detours to get around faults is not a good strategy in such cases. The phenomenon will continue to destroy local detours, causing loss of messages, forcing repeated detours and eventually blocking all local paths, requiring reconstruction of large parts of routing table. Knowing the topology of the network, and using homotopy types, we can effortlessly switch to a completely different type of path when we notice persistent disturbances in a region.

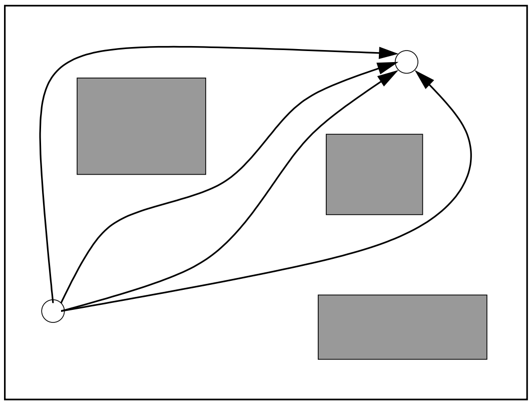


Figure 3.1: In a network of 3 holes (shaded), paths α, β, γ have distinct homotopy types; γ and δ are homotopy equivalent.

In Figure 3.1, a regional failure connecting the upper two holes may destroy both γ and δ, while paths α and β remain available. Despite the importance of homotopic routing in terms of improving load balancing and routing resilience, it is only very recently that homotopic routing has been explicitly addressed [7]. In Zeng *et al.* [7], greedy routing in a virtual coordinate space finds paths of different homotopy types. However, the algorithm has no theoretical guarantee on the *path stretch*, i.e., there is no bound on the path length in comparison with the length of a shortest path *of the same homotopy type*.

We [8] introduce a routing framework with a modest state per node that guarantees *constant worst-case stretch* for any given homotopy type. This is the first work that achieves a provable bound.

We assume that the sensor nodes are densely deployed in a geometric domain (e.g., campus map, floor plan) that is represented by a polygon (possibly with holes) Σ. While the network may have a huge number of nodes, the domain in which the nodes are deployed is often of a much smaller complexity. Let us denote by n the number of sensor nodes and k the number of vertices of Σ. In practice, $k \ll n$. Instead of building a structure on the network topology, we take the geometric domain in which the network is embedded and use a structure on Σ to encode the path homotopy type. Notice that this approximation gives us a number of benefits. The complexity of Σ is much smaller. Further, a structure operating on Σ is relatively stable, while network links can be volatile. As is common in geographical routing, we assume that nodes know their own geographical location through GPS or other localization schemes.

Our method operates on a two-level structure. On the top level, we use a coarse triangulation of the *geometric domain* to encode the network shape and to compute the path homotopy type. On the bottom level, we show that a simple, greedy geographic routing scheme within a sequence of triangles, which together with the top-level gives a constant (bounded) worst-case stretch compared to the shortest path of the same homotopy type.

We decompose Σ into triangles using certain diagonals connecting vertices of the polygon Σ. The dual graph of the triangulation is a planar graph $\mathcal{D}$. If Σ has h holes, we cut the domain along h diagonals (*cut edges*) that interconnect the holes, thereby obtaining a simply connected domain whose corresponding dual graph is a tree T. For a particular homotopy type, the shortest path stays inside a sequence of triangles $S = \{\triangle_1, \triangle_2, \cdots, \triangle_m\}$ in the triangulation, whose dual is a simple path in T. Since the simple path connecting two nodes in a tree is unique, any existing method of routing on a tree, such as hyperbolic embedding [9] or compact routing labels [10], can be used to find S using a greedy algorithm on the tree T. Thus, the top-level greedy algorithm simply reveals the triangles of S, one at a time, which contains a shortest path of the required homotopy type. Our bottom-level algorithm will then realize a path within S, whose length is at most a constant times the shortest path inside S, i.e., the shortest path of the requested homotopy type. We remark that triangulation of Σ and the tree T can be computed at the network initialization phase. The corners of each triangle are pre-loaded at the sensor nodes that are inside the triangle, along with the corners of the (at most 3) triangles that are adjacent to it. This way, finding the "global path" in T is done at runtime in the network using local, greedy information.

To summarize, both the top-level and bottom-level routing algorithms are of a greedy nature. The top level computes a sequence of geometric triangles that the homotopic routing path should visit; the bottom level uses a greedy algorithm *in the sensor network* to find a network routing path realizing the requested homotopy type. The top level uses virtual coordinates for finding the homotopy type; the bottom level uses the nodes' true geographical coordinates to realize one such path. After the initial preprocessing, each node only stores the Euclidean coordinates of the corners of the triangle containing it and the adjacent triangles (for bottom-level routing), the virtual coordinates of its own and adjacent triangles (for top-level routing), and the h cut edges of the dual graph (for specifying the path homotopy types). The storage requirement for each node is of size $O(h)$, where h is the number of holes in Σ, and is independent of the network size or the complexity of the geometric domain Σ. The preprocessing involves computing a triangulation of Σ and the top-level embedding (e.g., by [9]) or computation of routing labels (e.g., by [10]), whose complexity is roughly linear in the complexity, k, of the geometric domain Σ and independent of the number, n, of sensor nodes, which can potentially be much larger.

Related Work. Our scheme is in the family of geometric routing schemes [11] that use the nodes' coordinates to guide a message to the destination. In particular, the simplest geographical greedy routing [12] delivers the message to the neighbor whose distance to the destination is the smallest. One well known limitation of this approach is that a message can get stuck at a node that does not have any neighbor closer to the destination; this often happens when the domain Σ is not convex. To resolve this issue, a number of schemes (such as GLIDER [13]) first decompose the network into pieces such that simple greedy routing can be carried out inside each piece and the adjacency of the pieces are extracted and propagated to the network on top of which global routing is performed. In some papers (e.g., [14, 15]), the sensor network domain is partitioned into convex or nearly convex pieces, and, again, a similar routing scheme is designed by first finding the sequence of pieces to visit and then using a local, greedy algorithm to deliver the message to the next piece in the sequence. However, none of these prior local routing schemes guarantees bounded stretch (compared to the shortest path through the same sequence of cells). Further, none of the global routing schemes explicitly considers path homotopies.

In the geometric setting, computing the shortest homotopic path inside a polygon Σ, when we have global knowledge of Σ, can be done in almost linear time. A simple polygon with k edges can be triangulated in $O(k)$ time [16, 17]. A polygon with k vertices and h holes can be triangulated in $O(k + h\log^{1+\epsilon} h)$ time [18], or in $O(k\log^* k + h\log k)$ expected time [19]. Given a triangulated polygon, computing the shortest path with a given homotopy type can be done in time linear in the number of times the path crosses a diagonal of the triangulation [20]. All of the above schemes assume full knowledge of the entire triangulation Σ and compute only a geometric path; they do not route within a sensor network deployed inside Σ.

There has been a number of related papers about online navigation for robot motion planning, which uses models similar to our bottom-level routing algorithm. In one of the most investigated models [21], we are given a planar straight line graph H with n vertices, whose edges are weighted by their Euclidean lengths, the source s and destination t are vertices of H, and a packet can only move on edges of H. A packet only knows s, t, $N(v)$ (the set of neighbors of v), and the location of the packet. Various studies have been done under this model [21–25]. It has been established that deterministic oblivious (i.e., "memoryless") algorithms can be found for triangulations, but no algorithm has constant competitive ratio. For triangulations that have the "diamond property", a constant competitive algorithm exists if the algorithm can use $O(1)$-memory within the packet being routed [24]. In particular, if the triangulation has exactly two ears (as does a sequence of triangles), there is a simple algorithm with competitive ratio 9. This setting is different from ours. In our setting, the triangulation is coarser than the network resolution. The routing path does not need to follow the edges of the triangulation; it can pass through interiors of triangles, potentially allowing the path to be significantly shorter.

The problem of robot motion planning and online navigation has also been considered in a geometric domain with obstacles under various models of the robots' vision; refer to the survey [26]. A *tactile* robot learns the boundary of an obstacle only when it encounters it and then moves along it [27]. A *vision-based* robot only learns the obstacles when it can see them. For tactile robots in an environment with square obstacles, constant competitive ratios can be achieved; when the obstacles can have unbounded aspect ratio, no constant competitive algorithm exists [28, 29]. If the domain is a special simple polygon called a "street", in which s, t split the polygon boundary into two chains such that any point on one chain is visible to some point on the other chain, then online algorithms exist in which a robot with a vision sensor can search for the destination t with a constant competitive ratio [30]. Additional results can be obtained for star-shaped polygons, etc.; refer to [31]. Note that a sequence of triangles, as in our setting, is a street. Note, though, that in our model the "vision" of the robot (message) is restricted to the current containing triangle.

3.1.2 Bounded Stretch Homotopic Routing

We assume a dense collection of sensor nodes deployed inside a given polygonal domain Σ with $h \geq 0$ holes. The number of vertices of Σ is k. The number of sensors inside Σ is n. Typically, $h \ll k \ll n$. Our objective is to prepare the sensor nodes with minimal information such that one can easily answer the query of *homotopic routing* using local greedy algorithms. We start by explaining how to define path homotopy in the query.

Path Homotopy

First we triangulate the polygon Σ, i.e., adding diagonals (edges connecting visible vertices of Σ) to decompose Σ into triangles. Note that no Steiner points are added. An example is shown in Figure 3.2.

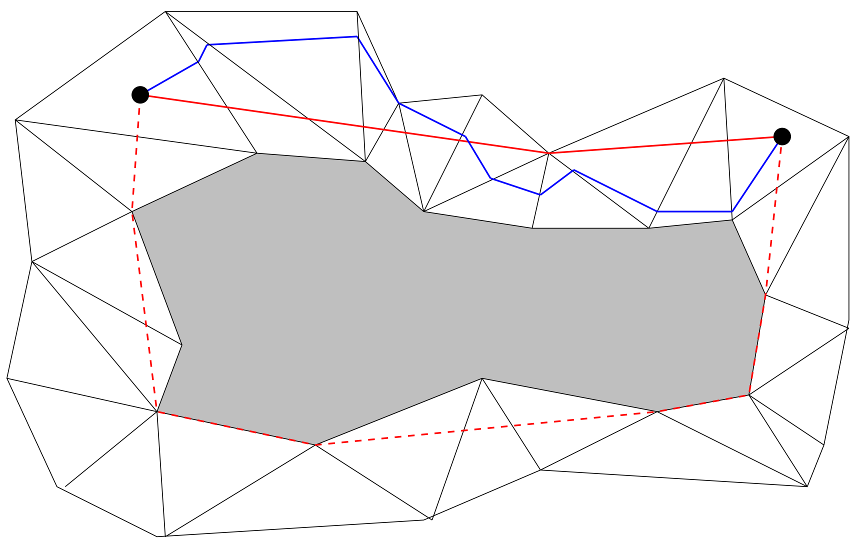


Figure 3.2: A triangulated region and routing paths inside. The shaded part is a hole with no wireless nodes while the rest of the domain is densely covered by sensors (not shown in the figure). The red path shows the shortest path from point s to point t, while the blue path is created by following our greedy routing strategy. The dashed path is a shortest path with a different homotopy type.

Any path from a source s to a destination t must go through a sequence of triangles. We say a path follows a sequence S, if it visits the triangles in the order S. The homotopy type of a path is captured by S. Figure 3.2 shows (in solid red and dashed red) the shortest geometric paths from s to t of two different homotopy types.

To define a path of a certain homotopy type, we work with the dual graph ($\mathcal{D}$) of the triangulation, in which each triangle is represented by a vertex, and vertices corresponding to adjacent triangles are connected by edges. See Figure 3.3 for an example.

A path in a domain with holes can go around holes in different ways; e.g., consider the red and green paths connecting z and d in Figure 3.3. To characterize these differences formally, we introduce *cut edges* to the triangulation, which are diagonals (or edges in the dual graph) that connect holes to the outer boundary (possibly via other holes). The cut edges are chosen one per hole arbitrarily . Removing the cut edges makes Σ a simple polygon and makes

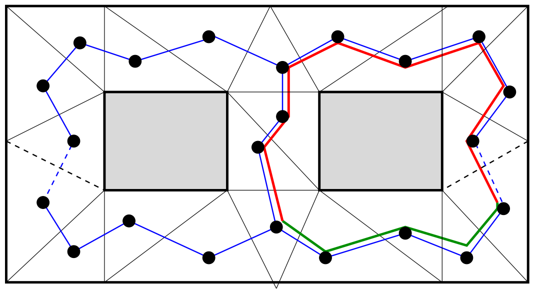


Figure 3.3: A triangulated polygon. Dual graph shown with blue edges. Cut edges are shown in dashed lines. Red and green curves show paths of different homotopy types. The solid edges in the dual constitute the tree T.

the dual graph $\mathcal{D}$ a tree, denoted by T. In Figure 3.3 the cut edges (shown dashed) are ab and cd in the dual; they cross corresponding triangulation edges (also shown dashed).

This demarcation of cut edges suffices to distinguish paths of different homotopy types. Any two paths that cross the same sequence of cut edges are of the same homotopy type. Note that we are interested only in some *basic* homotopy types. In general, a path may go around a hole many times, but such paths are not of practical interest in network applications. Therefore we are only interested in the homotopy types in which the shortest path only visits each triangle once. This means that the path winds around a hole at most once, i.e., any cut appears at most once in the type description. Thus, the length of a description for a practical path is $O(h)$, where h is the number of holes of Σ. The homotopy type definition is only with respect to the geometric domain Σ and is independent of the sensor network within Σ.

Our goal is, for a given homotopy type (sequence of cut edges), to find a path of that type *in the sensor network*, from the source to the destination, by using a local greedy decision rule. Our routing protocol follows a two-level structure, and adopts greedy decisions to make progress at each level:

1. **Top level:** Determines the sequence S of triangles that the shortest path with the specified homotopy type goes through.
2. **Bottom level:** Routes the message in the sensor network, following the sequence of triangles S.

The top-level procedure is concerned with finding the sequence of triangles that the shortest path of the given homotopy type should follow. Our hope of achieving short stretch paths rests on the bottom-level procedure, which must be designed with care. We will first describe our algorithm and prove its performance in the geometric case of a continuous Euclidean metric, and then consider its adaptation to a discrete network setting in Section 3.1.2.

Greedy Routing at the Bottom Level

In this subsubsection we design and analyze the performance of the bottom-level protocol for greedy routing in a sequence of triangles $S = \triangle_1, \triangle_2, \cdots$.

Definition 3.1.1 (Bottom-Level Greedy Routing). *For a message at point p, inside $\triangle_i$ and destined for t,*

1. *It is routed along the shortest path to (any point in) the next triangle $\triangle_{i+1}$ in the sequence S.*
2. *If $\triangle_i$ is the last triangle in S, then it is routed along the shortest possible path from p to t.*

Such a path is shown in blue in Figure 3.2.

Theorem 1.2.2 makes sure that the above algorithm yields a competitive path at the bottom level.

Greedy Routing at the Top Level

At the top level, we have a tree T, dual to the triangulation of Σ after cut edges are removed. Routing at this level deals with finding a sequence of triangles that contains the shortest path of the specified homotopy type. Routing in a tree using a greedy algorithm can be done in various ways. We describe two such methods below and show how to add homotopy types on top of it.

Homotopic routing by hyperbolic embedding. To support greedy routing in the tree T, we can embed T in the hyperbolic plane using the method of Kleinberg [9]. Here we explain how to augment it for routing of a given homotopy type.

To support routing of a given homotopy type, we need to attach copies of the tree along cut edges such that routes through cut edges can be found in a greedy manner. Each cut edge connects two vertices of T that are always leaves. We can attach a copy of T to the open end of each duplicate cut edge. The continuity that was lost in removing the cut edges to create T is now restored.

For example, the edge ab in Figure 3.3 maps to two different leaves b and a. Let us attach a copy of T by connecting edge ab at a and b, respectively, to maintain continuity of the original $\mathcal{D}$. That is, a and b are now neighbors once

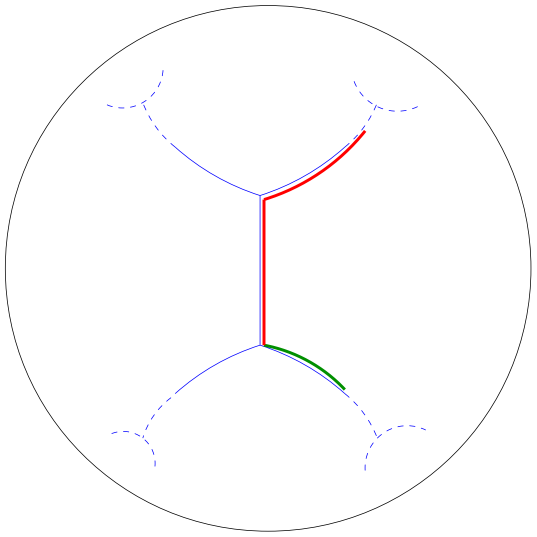


Figure 3.4: The tree T is shown in solid blue inside the Poincare disk. The red and green paths from Figure 3.3 are shown as red and green paths to different images of d. Together they represent a loop.

again, while maintaining all other neighbor relations. However, this creates a new set of cut leaves with discontinuity where we need to attach copies of T for continuity. This can be carried out indefinitely to get a tree $\mathcal{T}$ with infinitely number of copies of T. This tree $\mathcal{T}$ is the *Universal Covering Space* of the graph $\mathcal{D}$. Each node of T maps to many copies in the universal covering space $\mathcal{T}$. Paths of different types are obtained by simply routing to different images of the destination in $\mathcal{T}$. In particular, the shortest path for a given homotopy type maps to a simple path in the $\mathcal{T}$ connecting the source to the proper image of the destination. This simple path can be found by using a greedy routing algorithm using a embedding of $\mathcal{T}$ in hyperbolic virtual coordinates, as explained below.

The universal covering space of any graph is a tree [32], and an infinite tree of bounded degree has a greedy, $(1+\varepsilon)$-distortion embedding in the hyperbolic plane that can be computed by a distributed algorithm [9, 33]. Figure 3.4 shows an example of the embedding in the Poincare disk.

To perform the top-level routing, we need to identify this suitable image of the destination corresponding to the given homotopy type. Recall that a homotopy type is determined by the sequence in which a path crosses cut edges. For example, the red path above crosses the cut edge cd in direction $\overrightarrow{cd}$. We define a corresponding transformation in the hyperbolic plane: the transformation g_{cd} maps the cd edge at the bottom right to the cd edge at the top right. Then the two destinations d and $g_{cd}(d)$ corresponds to different homotopy types shown by the green and red paths.

A path to $g_{cd}(d)$ must cross the cut edge cd in direction $\overrightarrow{cd}$, and thus is equivalent to that homotopy type. Transformations such as g_{cd} form a group called the group of *deck transformations*. The generators of this group can be composed together: $g_{ab}g_{cd}(w) = g_{ab}(g_{cd}(w))$. Therefore, a homotopy type can be represented by a sequence of generators, $g = g_1 g_2 g_3 \ldots$, which represents crossing the corresponding cut edges in the given order and directions. Thus, for our purposes, given a homotopy type g and a final destination d, we will select as our destination $g(d)$ and route to it by greedy routing in the hyperbolic metric. Each generator is an isometry of the hyperbolic plane and can be written as a Möbius transformation: $z \to e^{i\theta} \frac{z - z_0}{1 - \bar{z_0} z}$, with only two parameters: a complex number z_0 and an angle θ. A composition g can be written in the same form, and therefore needs only two parameters as well.

An analogous method of universal covering spaces used in [7] embeds the *entire* network in the hyperbolic plane using an expansive Ricci Flow computation. In contrast, we need to embed only a coarse tree.

Routing with compact labels of T. An alternative method of finding routing paths in $\mathcal{D}$ is to use compact routing labels for the finite tree T. In a tree of size m – for example, as described in [10] – this method assigns label $\zeta(p)$ of size $O(\log m)$ to each node p. Given a destination label t, the source s can find a neighbor p that is closer to t by merely comparing $\zeta(p)$ and $\zeta(t)$. Using ζ, we can perform homotopic routing as follows. Suppose $ab, cd, \ldots$ is the sequence of cut edges for the type. We route from s to a using ζ, route from a to b locally, and route from b to c using ζ again, and so on.

Implementation in a Sensor Network

In this section we describe the implementation issues in a sensor network. This includes preprocessing to prepare the necessary information for both top-level and bottom-level routing algorithms. We also elaborate on how to implement the bottom-level routing in a network setting.

Preprocessing. We assume that the polygon Σ (e.g., a map of the deployment domain) in which the network is deployed is already known before the deployment/initialization of the network. The triangulation of Σ is done using a centralized algorithm offline by the network owner/operator. A polygon with k vertices and h holes can be triangulated in $O(k + h \log^{1+\epsilon} h)$ time [18], or in $O(k \log^* k + h \log k)$ expected time [19]. h cut edges are selected to cut the holes open. We also prepare the coordinates for top-level routing in the tree T (and the universal covering space $\mathcal{T}$) obtained from the triangulation after the removal of the cut edges in a centralized manner. The running time is dependent on k, h, the complexity of Σ instead of the size of the sensor network.

Information about the triangles as well as the coordinates of nodes of $\mathcal{T}$ are then disseminated to the network using a network-wide broadcast such that each sensor node only stores the following information: 1) the Euclidean coordinates of the triangle containing itself and the (at most 3) adjacent triangles; 2) the virtual coordinates of the triangle for top-level routing and that of (at most 3) adjacent triangles; and 3) the cut edges for issuing homotopic routing. With such information each node is able to perform top-level greedy routing given a specified homotopy type using only local information. The storage requirement for each node is $O(h)$. The message size in the broadcast procedure is $O(k)$, the size of Σ. Notice that this procedure is only done once for the entire network lifetime.

To specify a destination, we need to specify the triangle containing it and its Euclidean coordinates. The former is for top-level routing and the later is for bottom-level routing when the message gets to the final triangle. Such information for the destination can be obtained through standard location services.

Bottom-level routing in a sensor network. To perform bottom-level routing, we implement the greedy strategy, described in the continuous setting in Section 3.1.2. There are two cases. In the first case, the destination is not in the current triangle $\triangle$. The top-level routing suggests the next triangle $\triangle'$ to be visited. We find a path in the network to the geometric diagonal shared by $\triangle$ and $\triangle'$. We simply use geographical greedy routing towards the diagonal, i.e., forwarding the message to the neighbor whose Euclidean distance is closest to the next triangle. Since we assumed that sensors are densely deployed, geographical greedy routing inside a triangle-shape domain should have high delivery rate. In the rare situation that the message gets stuck, for example due to small and temporary routing holes, we simply flood it inside the triangle $\triangle$. In the second case, the destination is inside the current triangle $\triangle$; again, simple geographical greedy routing is employed and, in rare situations, flooding is used when a message gets stuck.

3.1.3 Simulations

In this section, we implemented our algorithm and run simulations to investigate the observed stretch factor and load balancing. Here, we are especially interested in evaluating the quality of the bottom-level routing of our algorithm; notice that the top-level simply produces the unique sequence of triangles to be used at the bottom-level. We compare to three network routing algorithms within a sequence of geometric triangles: shortest path (SP) routing, the landmark-based greedy routing algorithm GLIDER used in [13], and the greedy algorithm using hyperbolic virtual coordinates in [7].

GLIDER: A landmark-based 2-level routing method. GLIDER [13] starts by selecting some nodes as $l$andmarks. Then, these landmarks flood the network such that every node in the network can identify its nearest landmark. Nodes with the same nearest landmark are said to be in the same Voronoi cell. Each node is informed of the dual graph of the Voronoi decomposition and thus knows the path in the dual to any cell. When a message needs to move from cell A to cell B to cell C, the message moves along a shortest path toward the landmark of B, until it enters B. At that point, it switches to a shortest path toward the landmark of C, etc.

Hyperbolic routing using Ricci flow. The hyperbolic routing algorithm in [7] applies hyperbolic Ricci flow to compute an embedding of the network in the hyperbolic space. For any given domain, a triangulation is extracted from the connectivity graph. If there are holes (represented by non-triangular faces), they are cut open so that the network becomes simply connected. Then, the network is embedded, using hyperbolic Ricci flow, as a convex polygon in the hyperbolic plane. Similarly, the universal covering space is embedded so that homotopic routing is supported. We remark that our method uses a very coarse triangulation on the top level, rather than embedding the entire network. In addition, our method guarantees bounded stretch, while hyperbolic routing has no worst case guarantee.

Setting. While both the GLIDER scheme and our algorithm utilize network decompositions, the decompositions are different in nature. Thus, for consistent comparison, we slightly modify the GLIDER scheme to utilize the partition of the given triangulation (instead of the Voronoi partition), treating the triangles as cells, and setting the landmark of a triangle to be the node nearest to the centroid of that triangle. Further, all paths in comparison have the same homotopy type.

The simulation results reported in here are for unit-disk graphs. The results for quasi-unit-disk graphs and other relaxations of radio models are similar (and thus omitted from this extended abstract). The nodes are distributed as a perturbed grid.

Our main observations from the simulations are:

- Our greedy routing method almost always finds routes with a stretch of at most 3, which is much better than the theoretical bounds, GLIDER and hyperbolic routing. Its performance is consistent and is not sensitive to the actual triangulation of Σ used.

- Our algorithm distributes traffic load more evenly than the GLIDER 2-level routing method and causes fewer hotspots.
- Although there is no theoretical guarantee on the stretch for the GLIDER algorithm and the hyperbolic routing scheme, the paths they generate have stretch typically no more than 5 in all of the examples we have tested. This is in agreement with results reported in [13] and [7].

Path Length

We first consider a network of 8713 sensors deployed in a rectangle with three concave holes. The average degree is 23. We first triangulate this region as shown in Figure 3.5. We then take random sequences of triangles from this network, select random source and destination from them, execute routing using different methods and measure the stretch and node loads. Each experiment is repeated $10,000$ times.

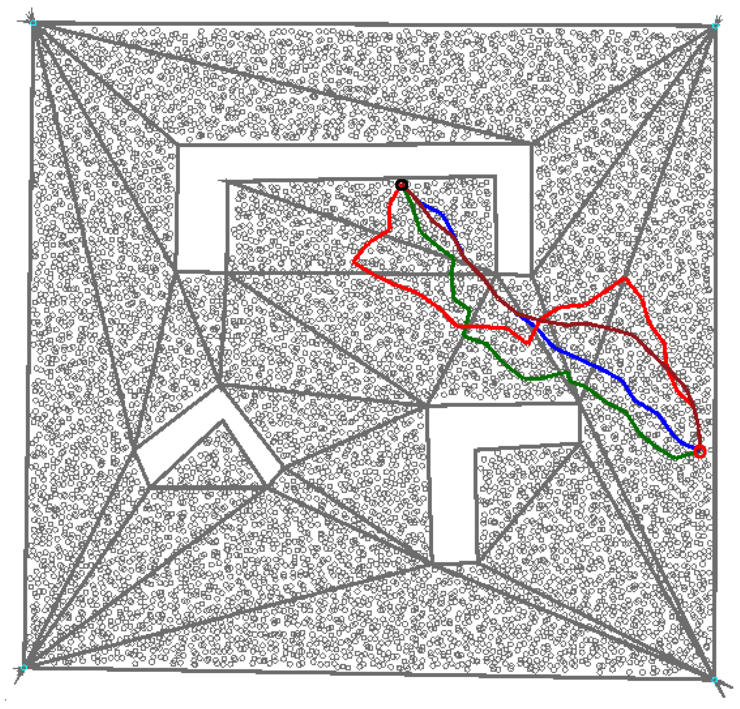


Figure 3.5: The rectangular domain with 8713 nodes distributed on a perturbed grid. Comparing a shortest path (blue), the GLIDER path (red), the hyperbolic routing path (brown), and our greedy route (green). The starting point is circled by red, the end point is circled by black.

Figure 3.6 demonstrates the distribution of the stretch of our algorithm, GLIDER, and hyperbolic routing over the shortest path algorithm. Of the three algorithms, 90% of the paths have stretch factor below 1.5, which is far better than the theoretical bound. Moreover, in this experiment our algorithm produces no stretch greater than 3, which outperforms the other two algorithms.

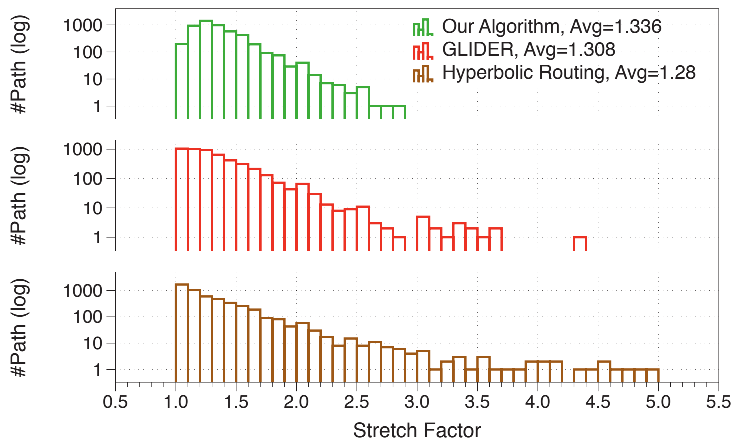


Figure 3.6: Histogram of stretch factors of paths over the shortest path algorithm produced by our algorithm, GLIDER, and hyperbolic routing. Our algorithm has a typical stretch less than 1.5, and is always less than 3, while GLIDER and hyperbolic routing have some paths of length more than 3 times the shortest path length.

Load Balancing

Geographical routing using face routing to get around holes is known to have a tendency to congregate near hole boundaries, creating traffic hotspots that slow down routing and drain sensor batteries. It has been shown that both GLIDER and hyperbolic routing improved load balancing in this aspect [7, 13], since both of them take long de-tours around holes.

We show the load distribution as a histogram in Figure 3.7 for different algorithms by running them on 10000 random source and destination pairs. All three algorithms perform reasonably well in this respect. Our algorithm is slightly better than GLIDER and slightly worse than hyperbolic routing. This is because by allowing paths longer than shortest paths, and not using face routing that explicitly traverses boundaries, our methods can distribute the load away from hole boundaries and hence ease the maximum load.

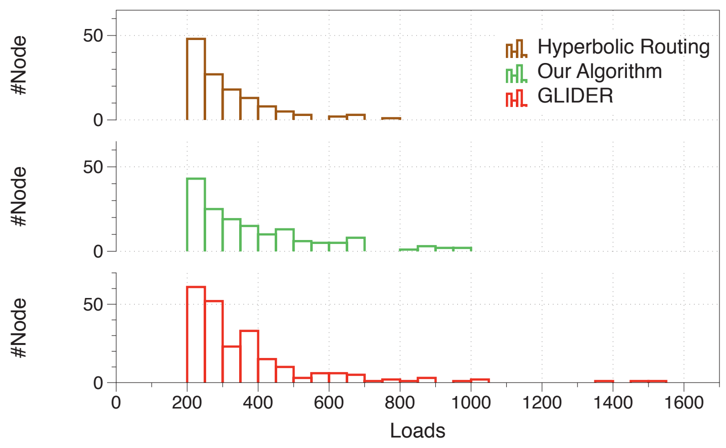


Figure 3.7: A load comparison between paths produced by hyperbolic routing, our algorithm and GLIDER. The load is captured by running three algorithms on 10000 random source and destination pairs. Only nodes with load above 200 are shown here.

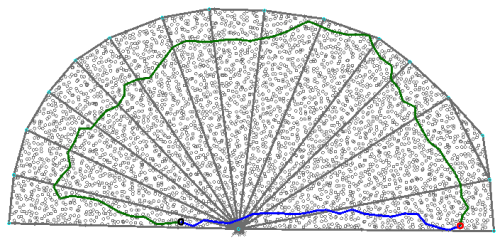
Various Domain

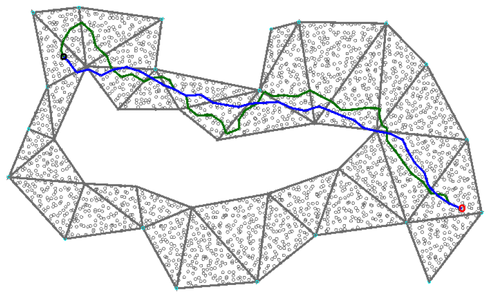
We tested the performance of our algorithm in the six different domains shown in Figure 3.8. In Figure 3.8a, we tested the fan shape, in which it is expected that our algorithm performs close to the worst case stretch bound – our route approximately follows a semi-circle, resulting in a stretch factor of about $\pi/2$. In Figure 3.8c, because of the nature of the spiral shape, all algorithm performs the same. We also tested various domains with obstacles, in Figure 3.8d, a concave hole is present within the domain. The equivalent states are as in Figure 3.8b, Figure 3.8e, and Figure 3.8f. All of the obstacles within the domain are treated as part of the triangulation.

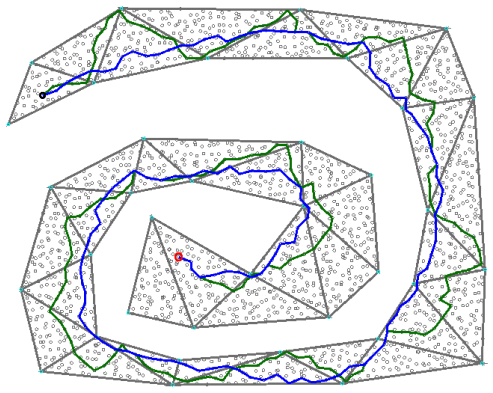
We chose $10,000$ random routing paths within each network. Table 3.1 shows that the average stretch of the greedy routing method is always less than 1.5, irrespective of the network geometry.

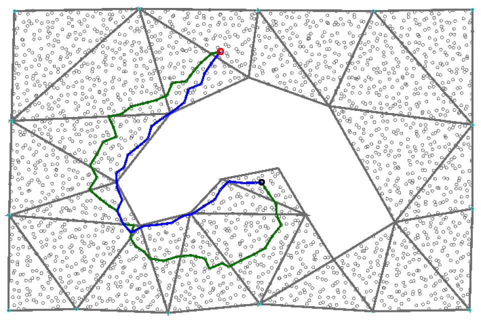
Dependency on Triangulation

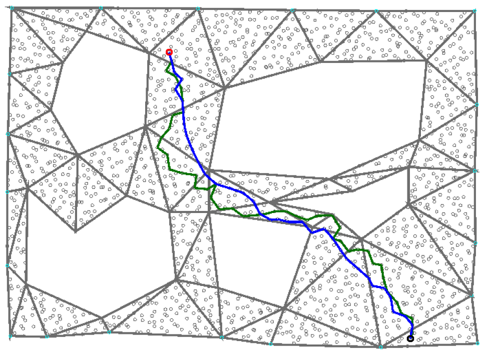
To check the dependence of the quality of results on the nature of triangulations, we repeated the experiment for different triangulations with randomly distributed interior vertices in a rectangle domain as Figure 3.8e. For each experiment, we randomly place 20 additional vertices for triangulation, and found that the standard deviation of the stretch of our method and GLIDER are both small, about 0.013 and 0.018, respectively. This implies that the per-


(a) Fan shape domain, 3965 nodes, avg degree 13


(b) Polygon domain with 1 hole, 2588 nodes, avg degree 6


(c) Spiral shape domain, 2227 nodes, avg degree 6


(d) Rectangular domain with 1 hole, 2403 nodes, avg degree 6


(e) Rectangular domain with 5 holes, 1883 nodes, avg degree 6

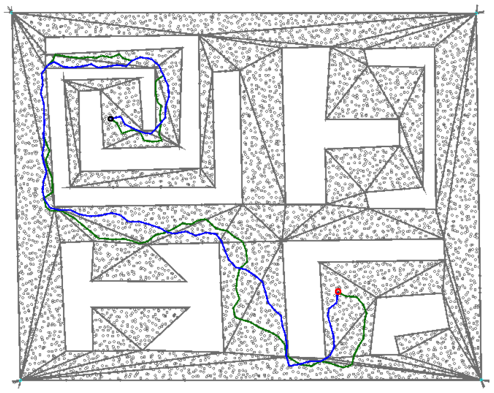

(f) Rectangular domain with 7 holes, 5432 nodes, avg degree 7

Figure 3.8: Six different domains and the shortest path (blue) and the path (green) obtained using our algorithm, the starting point is circled by red, the end point is circled by black.

Networks	Stretch
Fan shape domain	1.412191984
Spiral shape domain	1.271866423
Polygon domain with 1 hole	1.338248974
Rectangular domain with 1 hole	1.328246153
Rectangular domain with 5 holes	1.313282006
Rectangular domain with 7 holes	1.319227400

Table 3.1: Stretch in different networks of Figure 3.8

formances of these methods are largely independent from the triangulation.

3.2 Routing In Fat Triangulations

Definition 3.2.1. *Given a path P which threads a triangle sleeve $\mathcal{S}$, a vertex v of a triangle T is the pivot of T with respect to $\mathcal{P}$ if two edge incident to p in T is crossed by $\mathcal{P}$. Triangles of $\mathcal{S}$ are in the same (pivot) group if they have the same pivot.*

Figure 3.9 shows a triangle sleeve with two pivots.

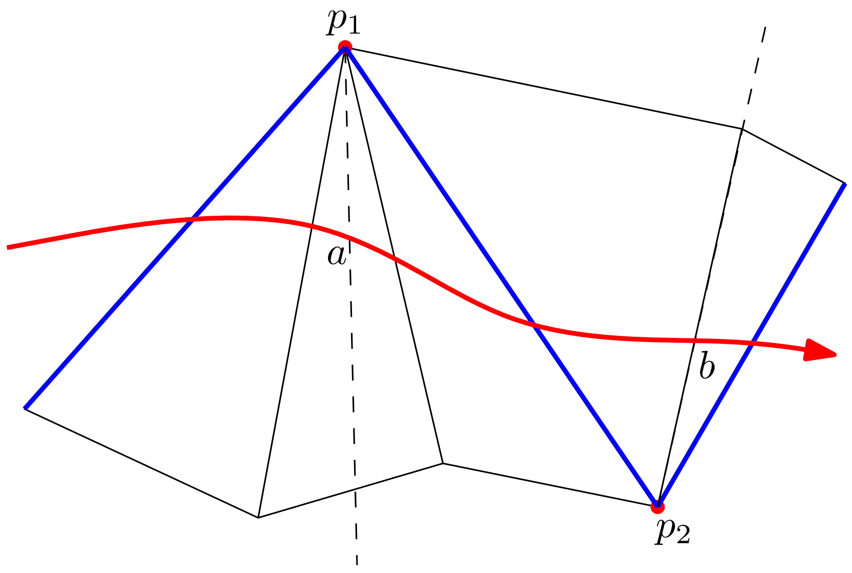


Figure 3.9: pivots of a triangle sleeve and its partition according to the pivots

Definition 3.2.2. *Let $\mathcal{T}$ be a triangulation of a planar region $\mathcal{R}$, with vertex set V. $\mathcal{T}$ is (ρ, α)-*fat*, if it satisfies the following properties:*

- *the ratio r_{max}/r_{min} of the longest to the shortest edge in $\mathcal{T}$ is bounded by some positive ρ.*

- *All angles in $\mathcal{T}$ have size at least α.*

Theorem 3.2.3. *Consider a (ρ, α)-fat triangulation $\mathcal{T}$ of a planar region $\mathcal{R}$, with vertex set V, and maximum and mininum edge lengths r_{max} and r_{min}. Let s, g be points in $\mathcal{R}$ that the triangles $\triangle_s, \triangle_t$ in $\mathcal{T}$ that contain s and g do not share a vertex, i.e. there are at least two pivot groups. Let $\overline{ab}$ be a shortest polygonal path joining two points s and t in $\mathcal{R}$. Let $P_{\mathcal{T}}(s,t)$ be a $\mathcal{T}$-greedy path between s and t. Then $P_{\mathcal{T}}(s,t) \leq c|\overline{st}| + 2r_{max}$, for $c = 2\rho/\min\left\{\frac{\sin(\frac{\alpha}{2})}{\lfloor\frac{2\pi}{\alpha}\rfloor}, \frac{1}{2(\lfloor\frac{2\pi}{\alpha}\rfloor-1)}\right\}$.*

Proof: Assume $\overline{st}$ threads $\triangle_1, \ldots, \triangle_{l'}$ between $\triangle_s$ and $\triangle_t$. Let l donetes number of triangles between $\triangle_s$ and $\triangle_t$ in the triangle sleeve given by $P_{\mathcal{T}}(s,t)$; by assumption, $l' \geq l \geq 1$.

We first show a lower bound of $|\overline{st}|$. With the angle bisectors incident to pivots, we divide l' triangles into $2l'$ triangles whose minimum angle is at least $\frac{\alpha}{2}$. The following analysis is to the new triangle sleeve. An edge is the bisector of a pivot group if there are same number of triangles on both sides.

Look at Figure 3.9. p_1 and p_2 are consecutive pivot points. Let $a(b)$ be the intersection of $\overline{st}$ and the bisector of the pivot group on the left(right) side of p_1p_2. Let $x(y)$ be the number of triangle between $p_1a(p_2b)$ and p_1p_2. We charge the length of $\overline{ab}$ equally to the $x+y$ triangles. The following function is a lower bound of length will be charged to each triangle, where m is $\min\{x,y\}$:

$$f_\alpha(m) = \begin{cases} \frac{\sin(\frac{m\alpha}{2})}{m+\lfloor\frac{2\pi}{\alpha}\rfloor-1} \cdot r_{min}, & 1 \leq m \leq \frac{\pi}{\alpha} \\ \frac{1}{m+\lfloor\frac{2\pi}{\alpha}\rfloor-1} \cdot r_{min}, & \frac{\pi}{\alpha} < m \leq \lfloor\frac{2\pi}{\alpha} - 1\rfloor \end{cases} \tag{3.1}$$

It's easy to prove that the minimum of $f_\alpha(m)$ is $r_{min} \cdot \min\left\{\frac{\sin(\frac{\alpha}{2})}{\lfloor\frac{2\pi}{\alpha}\rfloor}, \frac{1}{2(\lfloor\frac{2\pi}{\alpha}\rfloor-1)}\right\}$. Let it be c_0. Since the first half triangles in the first pivot group and the second half triangles in the last pivot group don't belong to any pivot switch, they may not be charged by any length. The number of these triangles is $u+v(\leq l')$. Therefore we have

$$|\overline{st}| \geq (2l' - u - v)c_0 \geq l'c_0 \tag{3.2}$$

On the other hand, it's straightforward to see that $|P_{\mathcal{T}}(s,t)| \leq 2(l+2)r_{max}$. Comparing it and the lower bound of $|\overline{st}|$ yields the theorem.

□

3.3 Greedy Routing In Perfect Triangulations

Definition 3.3.1. *A triangulation is perfect if all triangles it has are equilateral, i.e. it's* $(1, \frac{\pi}{3})$*-fat.*

We will show the stretch of greedy routing in perfect triangulations. Similarly to the previous section, we denote an optimal path and greedy path between two points a and b by $\overline{ab}$ and $\widetilde{ab}$ respectively. The following two observations will simplify our case analysis:

1. Look at Figure 3.10(2). t is the intersection of $\overline{st'}$ and a_2b_2. We have:

$$\frac{|\widetilde{st'}|}{|\overline{st'}|} \leq \frac{|sp_1| + |p_1p_2| + |p_2t| + |tt'|}{|\overline{st}| + |tt'|} < \frac{|sp_1| + |p_1p_2| + |p_2t|}{|\overline{st}|} \tag{3.3}$$

   which means we only have to consider the case when the destination t is on the last exit edge.

2. If the interior of $\overline{st}$ interests that of $\widetilde{st}$ at a point p, the greater stretch of the subpathes before and after p is not smaller than the original stretch. That's to say one can assume that $\overline{st}$ and $\widetilde{st}$ intersect only at s and t.

We now proceed to analyze the stretches of different cases.

1. Stick cases: in this case, each pivot group has one triangle; the sleeve looks like a stick.

   (a) two triangles. Look at Figure 3.10(1). It's easy to see the stretch is at most $\sqrt{2}$.

   (b) three triangles.

   Case 1: t is on p_2b_2. Look at Figure 3.10(2). We can scale all triangles such that s is on the boundary of the first triangle. This special case will be discussed in the case of four triangles.

   Case 2: t is on a_2p_2. Look at Figure 3.10(3). c is on the line of a_2b_2 such that $sc \perp a_2c$. Since $|\widetilde{st}| \leq |sc| + |ct|$, it's straightforward to get an upper bound $\sqrt{2}$ of stretch.

   (c) four triangles.

   Case 1: t is on p_3b_2. In Figure 3.10(4), c is the intersection of the extensions of sp_1 and tp_3. We have $|\widetilde{st}| \leq |sc| + |ct| \leq \sqrt{2}|\overline{st}|$.

   Case 2: t is on a_2p_3. In Figure 3.10(5), c is the intersection of a_2b_3 and the extension of sp_1; s' is on sp_1. It's an easy observation that to achieve a bigger stretch, $\angle stb_3$ is at most $\frac{\pi}{2}$. s' is on sp_1. When $|ss'| \to 0$, we have

$$\frac{|\widetilde{st}| - |\widetilde{s't}|}{|\overline{st}| - |\overline{s't}|} = \frac{1}{\cos \angle ts'c} \leq \frac{2}{\sqrt{3}} \tag{3.4}$$

Therefore we can assume s is at p_1 without decreasing the stretch if it's greater than $\sqrt{2}$. Let $|tc|$, $|cp_3|$ and $|a_1b_1|$ be x, y and 2 respectively. We have

$$\frac{|\widetilde{st}|}{|\overline{st}|} = \frac{3 + \sqrt{3}x + (1 + \sqrt{3})y}{\sqrt{9 + 3x^2}}, \;\; where \; y \leq \frac{3(1-x)}{4} \tag{3.5}$$

It's easy to show when $x = 0$ and $y = 0.75$, the biggest possible stretch 1.69 is achieved. Case 1 of three triangles is also closed.

(d) five or more triangles. We only discuss the cases of five triangles and the proof for more triangles is similar.

Case 1: t is on a_3p_4. $\angle sa_3b_4$ is smaller than $\frac{\pi}{2}$, which indicates the interiors of $\widetilde{st}$ and $\overline{st}$ are not disjoint. Due to observation 2, we don't have to consider this case.

Case 2: t is on p_4b_3, $\angle sta_3$ is greater than $\frac{\pi}{2}$, which means the maximum stretch is achieved when t is at b_3. Look at Figure 3.10(6). Let $|ss'|$ and $\angle s'sp_1$ be Δ and θ. We have

$$|p_iq_i| = 2^{1-i}|ss'| \sin\theta \tag{3.6}$$

$$|\widetilde{st}| - |\widetilde{s't}| = sp_1 - s'q_1 + \frac{|p_1q_1| + |p_4q_4|}{\sqrt{3}} + |p_4q_4| \tag{3.7}$$

$$= \Delta(\cos\theta + \frac{9 + \sqrt{3}}{8\sqrt{3}} \sin\theta) \leq \sqrt{2}\Delta \tag{3.8}$$

It indicates that we can assume s is at c. Therefore it's reduced to the case of four triangles. This method will be wildly used in the following discussion.

(e) infinite stick. If a stick has k triangles, $|\overline{st}|$ is in $[k-2, k+1]$; $|\widetilde{st}|$ is in $[\frac{2(k-4)}{\sqrt{3}}, \frac{2(k-4)}{\sqrt{3}} + 4]$. Therefore the stretch is $\frac{2}{\sqrt{3}}$ asymptotically.

2. Spiral cases: in these cases, one pivot vertex might have multiple(at most four) triangles; since we assume the optimal path only intersects the greedy path at two ends, the sleeve looks like a spiral. We call a

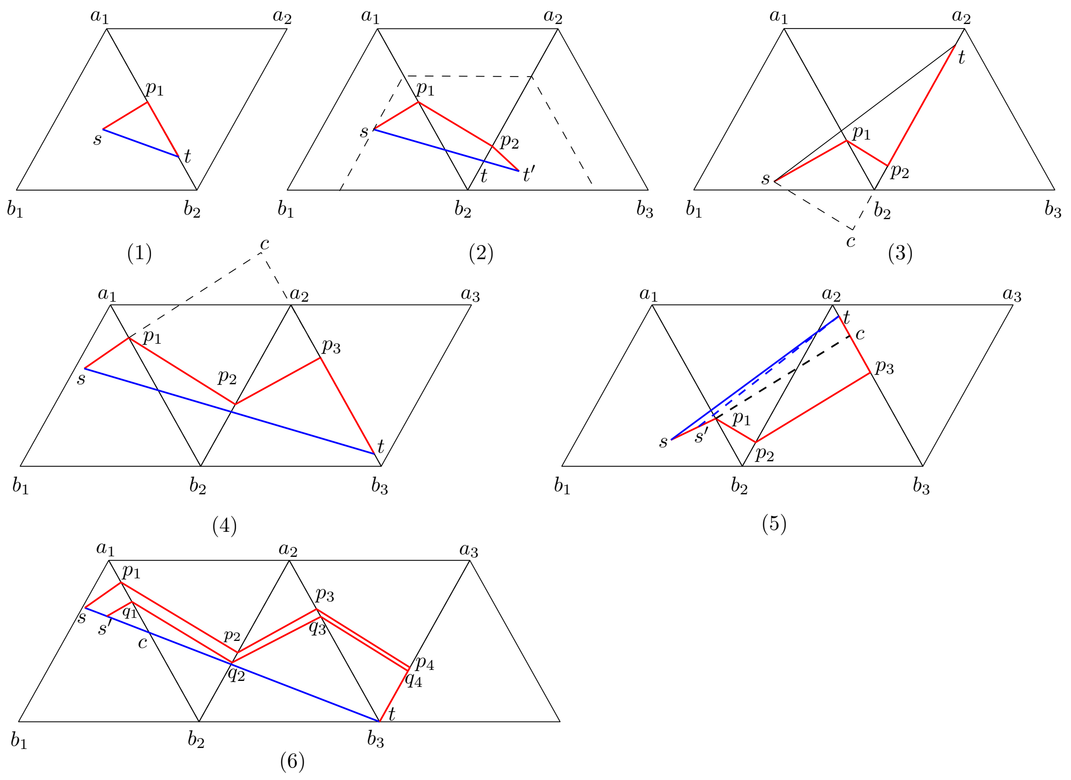


Figure 3.10: Case analysis of routing in equilateral triangle sequences

pivot that has $k(> 1)$ triangles a k-turn.

In Figure 3.11, s' is on $\overline{st}$ and $\{s', q_1, q_2, \dots\}$ is the greedy path starting from s'. We use Δ and θ to denote $|ss'|$ and $\angle s'sp_1$. Comparing the lengths of st and $s't$, we find the difference comprises of two parts: the first part is $|ss'|\cos\theta$; the second is proportional to $|p_1q_1|$, whose coefficient will determine the stretch. Let the maximum possible coefficient of triangle spirals with k turns be C_k. We proceed to explain the relationship between C_{k+1} and C_k by discussing sleeves with different first turns.

- 2-turn: look at Figure 3.11(1). The first turn v_i is a 2−turn; $\triangle_i$ is the first triangle in the group of v_i. Let d_i be $|p_iq_i|$; we know $d_i = 2^{1-i}d_1$. Let the coefficient be c^i_{k+1}. Since $i \geq 2$, we have

$$c^i_{k+1} \leq \frac{\frac{1}{\sqrt{3}}(d_1 + 2d_i - d_{i+2}) + C_k d_{i+2}}{d_1} \tag{3.9}$$

$$\leq \frac{5\sqrt{3}}{8} + \frac{C_k}{8} \tag{3.10}$$

- 3-turn: in Figure 3.11(2), the first turn is a 3-turn. With a similar analysis of the case before, we have

$$c^i_{k+1} \leq \frac{\frac{1}{\sqrt{3}}(d_1 + 2d_i + 2d_{i+1} - d_{i+3}) + C_k d_{i+3}}{d_1} \tag{3.11}$$

$$\leq \frac{13\sqrt{3}}{16} + \frac{C_k}{16} \tag{3.12}$$

- 4-turn: It's a straightforward observation that a 4-turn should be the last turn of a sequence and C_1 is achieved by a sequence having a 4-turn. If there are an odd number of triangle(s) between $\triangle_1$ and $\triangle_i$, we can assume the number is 1 as Figure 3.11(3) shows; otherwise $\widetilde{st}$ will cross $\overline{st}$, which case we don't have to consider. Hence we know

$$c^{odd}_1 = \frac{\frac{1}{\sqrt{3}}(-d_1 + 2d_i + 2d_{i+1} + d_{i+2}) + d_{i+2}}{d_1} = \frac{1-\sqrt{3}}{16} \tag{3.13}$$

If there are an even number of triangle(s) between $\triangle_1$ and $\triangle_i$, we

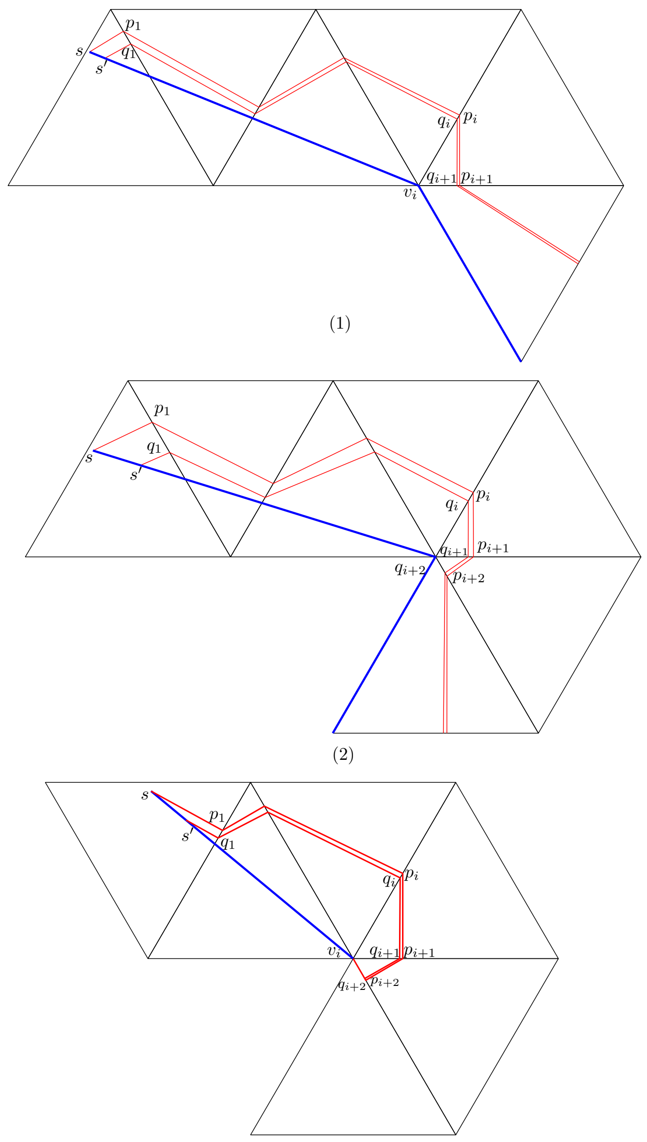


Figure 3.11: spiral cases

have

$$c_1^i = \frac{\frac{1}{\sqrt{3}}\left(d_1 + 2d_i + 2d_{i+1} + d_{i+2}\right) + d_{i+2}}{d_1} \leq \frac{7\sqrt{3}+1}{8} \approx 1.766 \tag{3.14}$$

To sum up, we know

$$C_{k+1} \leq \max\left\{\frac{5\sqrt{3}}{8} + \frac{C_k}{8}, \frac{13\sqrt{3}}{16} + \frac{C_k}{16}\right\}, \qquad k > 0 \tag{3.15}$$

$$C_1 = 1.766 \tag{3.16}$$

It's easy to prove that $\max\{C_k | k > 0\}$ is C_1. We finally get

$$\frac{|\widetilde{st}| - |\widetilde{s't}|}{|ss'|} \leq \cos\theta + C_1 \sin\theta \leq \sqrt{1 + C_1^2} = 2.03 \tag{3.17}$$

The discussion above finishes the proof of the following theorem.

Theorem 3.3.2. *Given a perfect triangle sleeve, i.e. a triangle sequence of equilateral triangles, the stretch of the greedy algorithm is at most* 2.03, *and this bound is tight.*

Chapter 4

Touring Polygons

Travelling Salesman Problem(TSP) may be the most famous travelling problem in computational geometry. Even in Euclidean plane the problem is quite hard. One difficulity comes from the order of points to visit is not given. Dror et al. [34] first studyed the touring a sequence of polygons problem(TPP), in which the order of polygons is given. They gives a polynomial optimal algorithm to find the shortest path that tours the polygons in the given order, when the polygons are disjoint and convex. They also shows that when the polygons are non-convex and nesting, the problem is NP-hard. One open problem in their paper is whether the problem is NP-hard when the polygons are non-convex but disjoint.

In this chapter, we give a positive answer to this open problem. [1]

4.1 Touring Disjoint Non-convex Polygons

For completeness we describe the hardness proof from [34] below: given an instance of 3-SAT with n variables $v_1, \ldots, v_n$ and m clauses $C_i = (l_{i1} \vee l_{i2} \vee l_{i3})$, one constructs polygons, $(P_1, \ldots, P_{O(n+m)})$, along with point s and t such that by solving the TPP on this instance, one can determine if the 3-SAT formula $\wedge_i C_i$ has a satisfying truth assignment. They construct four kinds of gadgets:

- 2-way path splitters that double the number of shortest path classes
- 3-way path splitters that triple the number of shortest path classes, keeping their ordering

[1][35] and us independently found the hardness proof. In their proof, they asks whether there is proof without exponentially growing gadgets, which is just what we have in our proof.

- path shufflers that perform a perfect shuffle of the input path classes
- literal filters that that "select" paths that have a particular bit equal to 0 or 1

They use n 2-way splitters to create 2^n distinct path classes that form a "bundle" of parallel paths, of which each path class encodes a truth assignment for the n variables. Then the parallel paths will go through a sequence of clause filters, each consisting of three literal filters sandwiched between two 3-way splitters, to filter out those path classes whose corresponding variable assignments fail to satisfy each clause. The literal filter has the property that only those path classes that have a particular bit set to 0 or 1 are able to pass through the filter without having a detour.

We adopt the same idea and use their 2-way splitter gadgets and 3-way splitter gadgets. Look at Figure 4.1.

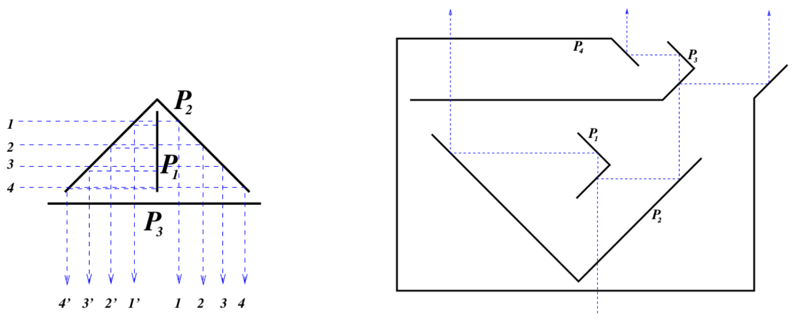


Figure 4.1: Left: 2-way splitter gadget. Right: 3-way splitter gadget

To avoid nesting polygons, we redesign the path shuffler, the literal filter.

The new path shuffler consists three parts: a bundle separater, a shuffle component and a half-reverse component. Recall that 2^n rays will go through a 3-way splitter and one will have three copies of them before shuffling. In each copy, the first(second) half of rays encode truth assignments in which one specific variable is true(false). First we adjust the distance between two bundles in one copy by a bundle separater. Figure 4.2 shows how it works. It's easy to connect three copies of bundle separaters such that two bundles in a copy of 2^n rays are separated by one bundle separater.

Figure 4.4 is a shuffle component and three copies of it are connected by red segments. Figure 4.5a shows more details. The blue and green rectangles denote two bundles. Let d be the distance between two rays in a bundle. l_5 denotes the distance between P_3 and the symmetric axis of the gadget. The

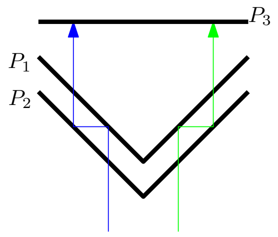


Figure 4.2: beam adjuster

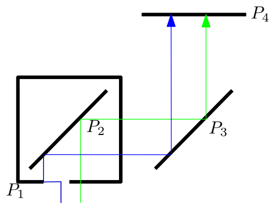


Figure 4.3: bit filter

gadget has following properties(other trivial distance relations are not listed here): (1)$l_1 - l_2 = \frac{d}{2}$; (2) $l_3 > 2l_4$; (3)$l_5 = \frac{d}{4}$.

A shuffle component doesn't do a perfect shuffle. That's why we need a half-reverse component before it. Figure 4.5b shows a half-reverse component, which has the following properties: (1) $l_2 = 2l_4 < l_3$; (2) $l_1 > l_2$.

Refer to Figure 4.5b for more details. The blue bundle directly reflects back and the order doesn't change; while the green bundle reflects twice and the order is reversed. Assuming there are 2^3 rays$(0, 1, 2, \ldots, 7)$, after passing through a half-reverse component, the order of rays becomes $(0, 1, 2, 3, 7, 6, 5, 4)$. The order of the rays will be $(0, 4, 1, 5, 2, 6, 3, 7)$ after touring through a shuffle component. With this permutation, the rays whose third bit is 1 are all in the first half.

Recall that the property of the literal filter is that only those path classes that have a particular bit set to 0 or 1 are able to pass through the filter without having a detour, force the path to be longer than the others. One bundle will be forced to detour like the blue ray in Figure 4.3; the left five bundle won't have to detour like the green ray.

4.2 Deeply Touring

In this section we will introduce deeply touring a sequence of polygons problem(DTPP). In TPP the objective is to find the path as short as possible, which means the visitor may touch a polygon then leave immediately. However, under some circumstances, we need the visitor to stay in polygons for specfic time, that's what 'deeply' means. For example, a data-gethering robot can collect data from servers within a range via wireless connection. It takes time to transfer data. Therefore the robot has to stay within a region defined by the server long enough to finish the data transferation.

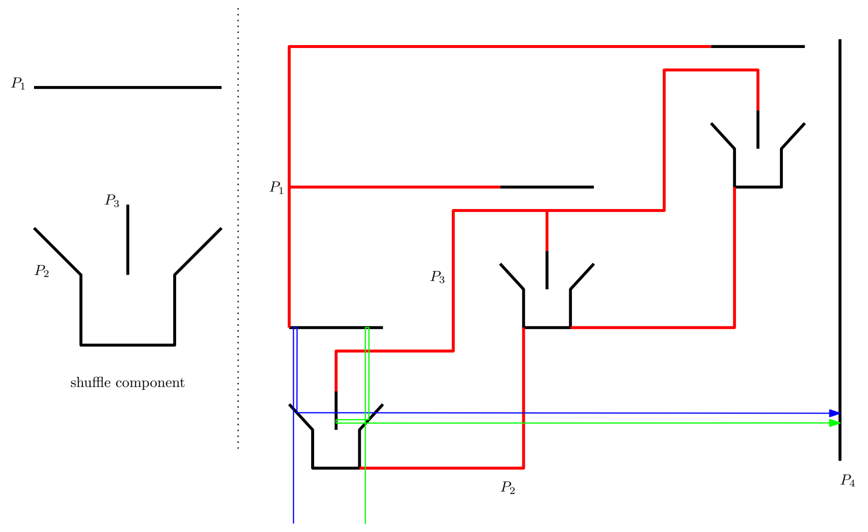
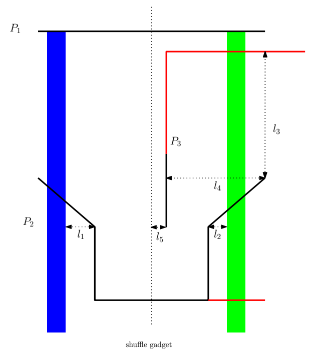
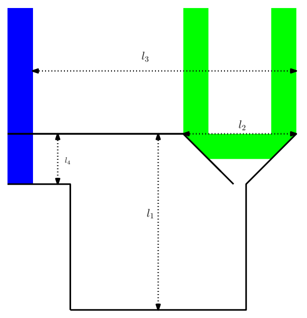


Figure 4.4: a shuffle component and connection



(a) shuffle component



(b) half-reverse component

Figure 4.5: two bundles passing through a shuffle component and a half-reverse component

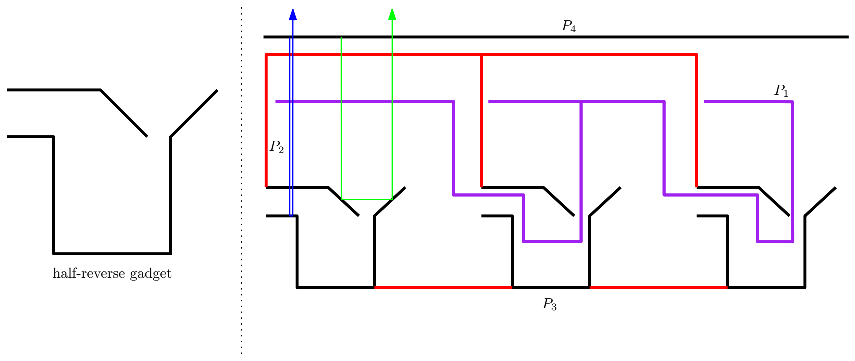


Figure 4.6: half-reverse gadget and connection

4.2.1 Problem Definition

Given a sequence of polygons $(P_1, P_2, \ldots, P_n)$, a set of numbers $(T_1, T_2, \ldots, T_n)$ and two points, s and t, find a tour such that one can start from s, visit those polygons in the order given and stay in polygon P_i for T_i time, at last arrive t as fast as possible. One can treat s and t as degenerate polygons. In the following formats, we denote them by P_0 and P_{n+1}.

Let a_i be the point where the route enters P_i and b_i be the point where it leaves P_i. The optimization problem becomes:

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{i=0}^{n}(x_i + y_i) & \\ \text{subject to} & x_i = |b_i a_{i+1}| & i = 1, ..., n \\ & y_i = \max\{|a_i b_i|, T_i\} & i = 1, ..., n \end{array}$$

We have several straightforward observations:

- In L_1 metric, the problem can be solved optimally by Linear Programming as long as all polygons are convex(don't have to be disjoint).
- If all polygons are convex, LP solution above in L_1 ia a $\sqrt{2}$-approximation for L_2 metric.
- Since $\displaystyle\sum_{i=1}^{n} T_i$ and the solution of optimal algorithms like [34] are two lower bounds of the solution of DTPP. That means any optimal algorithm for TPP yields a 2-approximation algorithm for DTPP and that algorithm is to follow the path of TPP and stay as long as what's required.

L_2 metric	C, D	$C, \overline{D}$	$\overline{C}, D$	$\overline{C}, \overline{D}$
TPP	$knlog(n/k)$	$knlog(n/k)$	n^4/ε^2	n^4/ε^2
DTPP	n^4/ε^2	n^5/ε^2	n^4/ε^2	n^5/ε^2

Table 4.1: C means all polygons are convex; $\overline{C}$ means any polygon could be nonconvex. D means all polygons are disjoint; $\overline{D}$ means that some polygons could overlap. The NP-hardness of red problems is not known.

We still don't know when the polygons are disjoint and convex, the $DTPP$ problem is NP-hard or not. However, known techniques to obtain an approximately shortest path amony obstacles in thress dimensions(e.g. [36–39]), which work for *TPP* still work for *DTPP*. Table 4.1 shows the collection of complexity results.

Chapter 5

Geometric Hitting Set for Segments of Few Orientations

5.1 Introduction

[This chapter is based on the paper "Geometric Hitting Set for Segments of Few Orientations," by Sándor P. Fekete, Kan Huang, Joseph S. B. Mitchell, Ojas Parekh, and Cynthia Phillips; submitted, 2015.]

The set cover problem is fundamental in combinatorial optimization. The problem is known to be NP-hard and to have an $O(\log n)$-approximation algorithm, which is, in general, best possible (unless $P = NP$, [40]). Equivalently, set cover can be cast as a hitting set problem, in which we are given a set S of *objects* and we desire a smallest cardinality set, H, of points such that every object in S contains at least one point of H. Numerous special instances of set cover/hitting set have been studied. Our focus in this paper is on geometric instances that arise in covering (hitting) sets of (possibly overlapping) line segments using the fewest points ("hit points"). A closely related problem is the "Guarding a Set of Segments" (GSS) problem [41–44], in which the segments are allowed to cross arbitrarily, but do not overlap. Since this problem is strongly NP-complete [42] in general, our focus is on special cases, primarily that in which the segments come from a small number of orientations (e.g., horizontal, vertical). We provide a variety of hardness results and approximation algorithms.

Besides being a fundamental geometric instance of set cover, our problem arises in the following scenario: Consider a set of paths/trajectories that lie along a road network (e.g., a Manhattan network of axis-parallel roads). Each path corresponds to a route of a potential customer; in the Manhattan case, the paths are (possibly overlapping) horizontal/vertical segments. Our goal is to place the fewest vendors ("hit points") to meet all customer paths.

Our results. We give complexity and approximation results for several problems all in the family of geometric hitting set problems on inputs S of line "segments" of special classes, mostly of fixed orientations. The segments are allowed to overlap arbitrarily. We consider various cases of "segments" that may be bounded (line segments), semi-infinite (rays), or unbounded in both directions (lines). Our main results are:

(1) Hitting lines of 3 slopes in the plane is NP-hard. (Hitting lines of 2 slopes is easy to solve optimally.) Naively, one obtains a 3-approximation for hitting lines of 3 slopes (since one can trivially hit (parallel) lines of one slope optimally). The standard analysis of the greedy algorithm gives an approximation factor of $H(3) = 1 + (1/2) + (1/3) = (11/6)$. We prove that the greedy algorithm in this special case gives an approximation factor of 7/5.

(2) Hitting vertical lines and horizontal (leftwards/rightwards) rays is polynomially solvable.

(3) Hitting vertical lines and horizontal (even unit-length) segments is NP-hard. A consequence of our proof is that hitting horizontal and vertical unit-length segments is also NP-hard.

(4) Hitting vertical lines and horizontal segments has a (5/3)-approximation algorithm. (This problem has a straightforward 2-approximation.)

(5) Hitting pairs of horizontal/vertical segments has a 4-approximation. Hitting pairs having one vertical line and one horizontal segment has a (10/3)-approximation. These results are based on LP-rounding. More generally, hitting sets of k segments from r orientations has a $(k \cdot r)$-approximation algorithm.

(6) We give a fast (linear-time) combinatorial 3-approximation algorithm for hitting triangle-free (girth at least 4) sets of segments. A 3-approximation was recently obtained by [44] using linear programming (LP) methods. Our result (Theorem 5.7.1) appears as Section 7 in the Appendix, due to lack of space.

Related Work. There is a wealth of related work on geometric set cover and hitting set problems; we do not attempt here to give an exhaustive survey. The *point line cover* (PLC) problem (see [45, 46]) asks for a smallest set of lines to cover a given set of points; it is equivalent, via point-line duality, to the hitting problem for a set of lines. The PLC (and thus the hitting problem for lines) was shown to be NP-hard [47]; in fact, it is APX-hard [48] and Max-SNP Hard [49]. The problem has an $O(\log OPT)$-approximation (e.g., greedy – see [50]); in fact, the greedy algorithm for PLC has worst-case performance ratio $\Omega(\log n)$ [51].

Hassin and Megiddo [52] considered instances of geometric hitting set problems for hitting objects with the fewest lines having a small number of distinct slopes. They observed that, even for covering with axis-parallel lines, the greedy algorithm has an approximation ratio that grows logarithmically. They gave approximations for the problem of hitting horizontal/vertical segments with the fewest axis-parallel lines (and, more generally, with lines of a few slopes). Gaur and Bhattacharya [53] consider covering points with axis-parallel lines in d-dimensions; they give a $(d-1)$-approximation based on rounding the corresponding linear program (LP) formulation. Many other stabbing problems (find a small set of lines that stab a given set of objects) have been studied; see, e.g., [50, 54–58].

A recent paper [44] gives a 3-approximation for hitting sets of segments that are "triangle-free". Brimkov et al. [41–43] have studied the hitting set problem on line segments, including various special cases; they refer to the problem as "Guarding a Set of Segments", or GSS. The GSS problem is a special case of the "art gallery problem" in which one is to place a small number of "guards" (e.g., points) so that every point within a geometric domain is "seen" by at least one guard [59, 60]. Brimkov et al. [61] provide experimental results, for three heuristics, including two variants of "greedy", on the GSS, showing that in practice the algorithms perform well and are often optimal or very close to optimal. (They prove, however, that, in theory, the methods do not provide worst-case constant-factor approximation bounds.) For the special case that the segments are "almost tree (1)" (a connected graph is an *almost tree (k)* if each biconnected component has at most k edges not in a spanning tree of the component), a $(2-\epsilon)$-approximation is known [43].

An important distinction between the GSS and the problems we study here is that the set S of segments we allow includes *overlapping* (or partially overlapping) segments (rays, and lines), while, in the GSS, we assume that each line segment is maximal in the input set of line segments (the union of two distinct input segments is not a segment). In particular, a special case of our problem is that of *interval stabbing* on a line: Given a set of segments (intervals), arbitrarily overlapping on a line, find a smallest hitting set of points that hit all segments. (A simple sweep along the line solves the problem optimally.)

If the objects we are trying to hit are such that there is no point that lies within three or more objects, then the hitting set problem is readily solved as an edge cover problem in the intersection graph of the objects. In particular, if no three segments pass through a common point, the problem can be solved optimally in polynomial time. (This implies that in an arrangement of "random" segments, the GSS problem is almost surely polynomially solvable; see [43].)

Related to our problem of hitting horizontal and vertical segments is the geometric hitting set problem for axis-aligned rectangles. Aronov, Ezra, and Sharir [62] provide an $O(\log\log OPT)$-approximation for hitting set for axis-aligned rectangles (as well as axis-aligned boxes in 3D), by proving a bound of $O(\epsilon^{-1}\log\log(\epsilon^{-1}))$ on the ϵ-net size of the corresponding range space. The connection between hitting sets and ε-nets [63–66] implies a c-approximation for hitting set if one can compute an ε-net of size c/ε; recent major advances [67, 68] on lower bounds on ε-nets imply that associated range spaces (rectangles and points, lines and points, points and rectangles) have ε-nets of size superliner in $1/\varepsilon$. Remarkably, improved approximation algorithms, with factor $(1+\varepsilon)$ (i.e., PTASs), for certain geometric hitting set and set cover problems are possible with simple local search, as shown by Mustafa and Ray [69]; for example, they obtain a local search PTAS for computing a smallest subset, of a given set of disks, that cover a given set of points. (Hochbaum and Maas [70] used a grid shifting method to obtain a much earlier PTAS for the minimum unit disk cover problem when disks can be placed anywhere in the plane, not restricted to a discrete input set.)

5.2 Hitting Segments

First, consider the case in which S is a set of line segments in the plane.

If all segments are horizontal, then we can compute an optimal hitting set easily: we compute a hitting set for the segments (intervals) along each of the horizontal lines determined by the input.

If the segments are of two different orientations (slopes), then the problem becomes significantly harder. Without loss of generality, we can assume the segments of two slopes are horizontal and vertical.

We first state that even if the axis-parallel segments are all of the same length, the problem is hard. As it turns out, this result is a consequence of an even stronger result, Theorem 5.5.1, which we establish in Section 5.5.

Corollary 5.2.1. *Deciding if there exists a set of k points in the plane that hit a given set S of unit-length axis-parallel segments is NP-complete.*

We get an immediate 2-approximation algorithm by solving optimally each of the two orientations, and using the union of the hitting points for both. (This generalizes to approximating hitting sets for segments of k orientations, yielding a k-approximation.)

5.3 Hitting Lines

We now consider the case in which S is a set of n lines in the plane. It is known that greedy gives an $O(\log OPT)$ approximation factor; any approximation factor better than logarithmic would be quite interesting. (See [46, 51].)

If the lines are all axis-parallel (horizontal or vertical), then the problem is easily solved in polynomial time. This holds also in the case that the lines are of any two orientations; by an affine transformation, any two orientations is equivalent to horizontal and vertical.

5.3.1 Hardness of Hitting Lines of 3 Slopes in 2D

If there are more than two orientations, then we prove that the hitting set problem is NP-hard. Consider the problem in the dual formulation: (3-Slope-Line-Cover, 3SLC) Find a minimum-cardinality set of non-vertical lines to cover a set P of points (duals to the set S of lines), which are known to lie on three vertical lines.

If a line ℓ covers i points, we say that ℓ is an *i-line*. Let P_ℓ denote the set of points of P that are covered by line ℓ. If $P_{\ell_1} \cap P_{\ell_2} = \emptyset$, then we say that lines ℓ_1 and ℓ_2 are *independent*. A set L of lines is *independent* if the lines are pairwise independent; i.e., $\ell_i \in L$ and $\ell_j \in L$ are independent for every $\ell_i \neq \ell_j$.

Theorem 5.3.1. *The problem 3SLC is NP-complete.*

The proof can be found in the Appendix.

5.3.2 Analysis of the Greedy Hitting Set Algorithm for Lines of 3 Slopes in 2D

The greedy algorithm is well known to yield a logarithmic factor approximation for general instances of hitting set (or set cover); more precisely, if no point lies in more than k sets, the greedy algorithm yields an approximation factor of $H(k) = \sum_{i=1}^{k}(1/i)$, the kth harmonic number [71]. In the case of hitting sets for which no point lies in more than 3 of the sets (as is the case for lines of 3 slopes), the greedy algorithm has approximation factor $H(3) = 11/6$. In the Appendix, we show that the geometric structure of the hitting set problem for lines of 3 slopes yields a better (7/5) approximation factor.

5.3.3 Axis-Parallel Lines in 3D

For three-dimensional space, we state another hardness result; the proof is in the Appendix.

Theorem 5.3.2. *Hitting set for axis-parallel lines in 3D is NP-complete.*

5.4 Hitting Rays and Lines

It is apparent that hitting rays is "harder" than hitting lines, since any instance of hitting lines has a corresponding equivalent instance as a hitting rays problem (just place the apices of the rays far enough away that they are effectively lines),

Unlike the case with lines, though, there can be many different rays that are collinear. Collinear rays can be divided into two groups according to their orientations; within a group, one ray is contained in the others of the group, so we need to keep only it for the purpose of solving the hitting set problem.

We now show that the special case in which the rays come from two orientations (say, horizontal and vertical), and in one of those orientations (say, vertical) the rays are effectively lines (i.e., all apices of the vertical rays are above or below all horizontal rays), the results problem (abbreviated HRVL) is exactly solvable in polynomial time. The proof is rather involved; it appears in the Appendix:

Theorem 5.4.1. *The hitting set problem for vertical lines and horizontal rays can be solved in polynomial time.*

5.5 Hitting Lines and Segments

5.5.1 Hardness

Theorem 5.5.1. *Hitting set for horizontal unit segments and vertical lines is NP-complete.*

Proof: The reduction is from 3SAT. See Figure 5.1.

Each variable is represented by a collinear connected sets of horizontal unit segments; these are connected by two vertical lines and additional sets of collinear unit segments into loops of even cardinality, producing a set of N horizontal unit segments and vertical lines, which we call variable components. Each clause is represented by a vertical line that intersects appropriate pairs of variable segments (if that variable occurs in a clause) or just single segments (in case a variable does not occur in a clause). Setting appropriate parities for the literals in a clause is achieved by appropriate horizontal shifting of the segments, as shown in the figure.

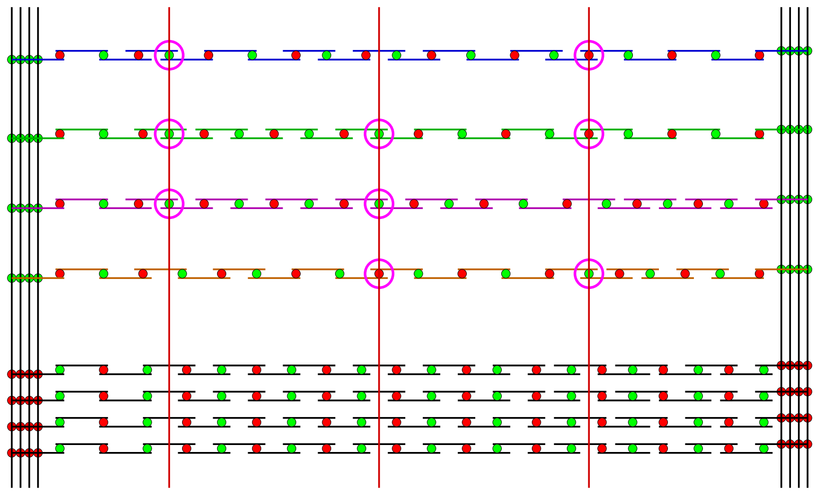


Figure 5.1: A set of horizontal unit segments and vertical lines that represents the 3SAT instance $I = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4)$. For better visibility, collinear segments are slightly shifted vertically, with red and green points indicating overlapping segments. Truth assignments for a variable correspond to the set of green or red points, respectively. Literals occuring in clauses are indicated by magenta circles; these are the only places where a point can stab three segments or lines at once.

This results in a construction in which the only place where three of the elements (segments or lines) can be stabbed involves a vertical line representing a clause; this correspond to literals occuring in the repective clauses. (These are indicated by magenta circles in the figure.) Thus, there is no point that stabs more than two of the variable components at once. Therefore, stabbing all N of them requires at least $N/2$ points. Clearly, $N/2$ of them suffice only if no two of them stab the same variable component; therefore, we must either pick all “green” or all “red” points for each variable. It follows that any such feasible set of $N/2$ points that stab all variable components induces a truth assignment, and vice versa. We get an overall stabbing set if and only if the points also stab the vertical clause lines, corresponding to a satisfying truth assignment. □ □

After appropriate vertical scaling, we can replace the vertical lines by vertical unit segments, so we immediately get Corollary 5.2.1 from Section 5.2.

5.5.2 Approximation

We give an improved approximation for hitting vertical lines and horizontal segments. We start by looking at the lower bounds: $v = |V|$ is the number of vertical lines (it is a lower bound). Let h be the lower bound on hitting horizontal segments only. We can compute h exactly; it is the minimum number of hit points for the horizontal segments (computed on each horizontal line). At any stage of the algorithm, we let h and v be the current values of the lower bounds that come from the sets H and V.

In stage 1, we place two kinds of points:

(a) We place maximally productive points on vertical lines that reduce h (and v) by one. Let $k_1 \geq 0$ be the number of points placed in this stage. Doing so reduced h and v each by k_1. As vertical lines are hit, we remove them from V. Similarly, as horizontal segments are hit, we remove them from H.

(b) Look for pairs (if any) of points, on the same horizontal line and on two vertical lines (from among the current set V), that succeed in decreasing h by one. Let $k_2 \geq 0$ be the number of points placed. Therefore the lower bound h decreased by $k_2/2$, and v decreased by k_2.

In stage 2, we now have a set of vertical lines V and horizontal segments H such that no single point at the intersection of a vertical line and a horizontal segment (or segments) reduces h, and no pair of points on two (adjacent) distinct vertical lines succeeds in reducing h.

Lemma 5.5.2. *For such sets V and H, an optimal hitting set has size at least $v + h$, where $v = |V|$ and h is the minimum number of points to hit H.*

Proof: The hit points we place on V (one per line) might conceivably decrease h. We claim that this cannot happen. Assume to the contrary that it happens. Let $\{q_1, q_2, \ldots, q_K\}$ be a minimum set such that each of them is on a line of V and h is decreased by 1 after placing the set.

Since we have found productive points and pairs of points in stage 1, K should be at least 3. Then consider the hit point q_2 on the middle among the 3 vertical lines, ordered left to right. The segments of H that are not hit by p are either completely left or right of p; let H_l and H_r be the corresponding sets. Since points on left and right sides of q_2 won't affect each other, if adding q_1 decrease H, that means q_1 and q_2 is a productive pair, which should be found in step (2); otherwise this means that the point q_1 is unnecessary, contradicting the minimality of K. □

Theorem 5.5.3. *There is a polynomial-time* $5/3$*-approximation algorithm for geometric hitting set for a set of vertical lines and horizontal segments.*

Proof: The total number of points selected by our algorithm is $k_1 + k_2$ from the first stage and $h - k_1 - k_2/2 + v - k_1 - k_2$ for the second stage. By the above lemma we have that the latter is lower bound on the cost of an optimal solution:

$$h - k_1 - k_2/2 + v - k_1 - k_2 \leq OPT, \tag{5.1}$$

and we also have that $h \leq OPT$ and $v \leq OPT$. We proceed with two cases.

(i) $k_1 + k_2 \leq 2/3 \cdot OPT$: In this case we select at most $2/3 \cdot OPT$ points in Stages 1, and we use (5.1) to bound the number of points selected in Stage 2. We conclude that our algorithm selects at most $5/3 \cdot OPT$ points.

(ii) $k_1 + k_2 > 2/3 \cdot OPT$: The total number of points selected by our algorithm is $h - k_1 - k_2/2 + v \leq 2 \cdot OPT - (k_1 + k_2/2)$. We have that $k_1 + k_2/2 \geq k_1/2 + k_2/2 > 1/3 \cdot OPT$, hence we obtain a 5/3-approximation in this case as well.

□

In fact, the approximation applies even in the case that the vertical lines are replaced with downwards rays.

Theorem 5.5.4. *There is a polynomial-time* $5/3$*-approximation algorithm for geometric hitting set for a set of vertical (downward) rays and horizontal segments.*

Proof: The 3-stage approximation algorithm described above works for this case as well. The key observation here is among any set of collinear downward rays, we may remove all but the one with the lowest apex from the instance. Therefore after Stage 1 and Stage 2, the hitting points we place on the rays not yet hit will not decrease h. The argument is analogous to that in Lemma 5.5.2. $\square$

5.6 Hitting Pairs of Segments

We consider now the hitting set problem for inputs that are *unions* of two segments, one horizontal and one vertical. The two segments might meet to form an "L" shape, a "+", or a "T" shape, or they may be disjoint. This hitting set problem is easily seen to be NP-hard, since it generalizes the case of horizontal and vertical segments. We give approximation results.

Theorem 5.6.1. *For objects that are unions of a horizontal and a vertical segment, the hitting set problem has a polynomial-time 4-approximation.*

Proof: Suppose we have l unions of segments as described above, and let P be the set of points serving as our hitters. We assume that $|P|$ is polynomial in l by preprocessing the instance, if necessary, so that we only consider points at endpoints and crossings of segments. For each such union i, we let S_i be the set of points covering the union, while H_i and V_i are the sets of points covering the horizontal segment and vertical segment respectively. We employ the standard set cover linear program (LP) relaxation specialized to our problem:

$$\begin{aligned} &\min \sum_{p \in P} x_p \\ \sum_{p \in H_i} x_p + &\sum_{q \in V_i} x_q \geq 1, \ \forall\ 1 \leq i \leq l \\ &0 \leq x_p \leq 1, \ \forall\ p \in P. \end{aligned} \tag{5.2}$$

We use an optimal LP solution, x^*, to construct a new instance of the problem in which each union contains either a vertical segment or a horizontal segment, but not both. This new instance is easier to approximate but no longer provides a lower bound on the original optimum value, OPT; however, we show that it provides a bound that is within a constant factor of OPT.

For each union of segments i, we set

$$S'_i = \begin{cases} H_i, & \text{if } \sum_{p \in H_i} x^*_p \geq 1/2 \\ V_i, & \text{otherwise.} \end{cases} \tag{5.3}$$

Now each S'_i corresponds to either a horizontal or vertical segment. Let $H' = \{i \mid S'_i \text{ represents a horizontal segment}\}$, and let $V' = \{1, \ldots, l\} \setminus H'$. Our algorithm is as follows:

1. Solve the LP, and let x^* be an optimal solution.
2. Construct S'_i for each union of segments i as described above.
3. Solve the hitting set problems for all the horizontal segments, H', and all the vertical segments, V', independently. Return the union of the points, X selected by optimal solutions to each instance.

This algorithm returns a feasible solution since it selects some point in $S'_i \subseteq S_i$ for each union of segments. The first two steps run in polynomial time. Hitting segments of a single orientation is solvable in polynomial time; in fact the corresponding set cover LP relaxation in this case has the consecutive ones property and is totally unimodular, hence the optimum LP value equals the optimum integer solution value.

To see that it is a 4-approximation, let $y^*_p = \min\{2x^*_p,\ 1\}$ for all p. By (5.2) and (5.3) we see that the fractional vector y^* is feasible for the LP instance defined by the segments corresponding to the S'_i. Now we modify the latter LP instance by taking each point p and replacing the variable x_p with variables $x_{p,h}$ and $x_{p,v}$, where $x_{p,h}$ appears only in horizontal segment constraints where x_p formerly appeared, and $x_{p,v}$ appears only in such vertical segment constraints. The resulting LP decouples the horizontal and vertical segments and captures precisely the problem from Step 3 of the algorithm. Since this LP is totally unimodular, we have that the number of chosen points, $|X|$, is at most the cost of any feasible fractional solution. In particular we see that the fractional vector z^* with $z^*_{p,h} = z^*_{p,v} = y^*_p$ is feasible for the decoupled LP, and so:

$$|X| \leq \sum_q z^*_q = 2 \sum_p y^*_p. \tag{5.4}$$

To obtain our desired result we note that $\sum_p y^*_p \leq 2 \sum_p x^*_p$ by the definition of y^*, yielding $|X| \leq 4 \sum_p x^*_p \leq 4 \cdot OPT$ by (5.4). □ □

The above idea naturally extends to and yields a 4-approximation for the weighted version of the problem as well. Another extension is to unions consisting of at most k segments drawn from r orientations, for which the approach yields a $(k \cdot r)$-approximation.

The LP-rounding technique in the proof above was introduced by Carr et al. [72] to obtain a 2.1-approximation for the weighted edge-dominating set problem. A similar idea was introduced independently by Gaur et al. [56] to obtain a 2-approximation for stabbing axis-aligned rectangles with horizontal and vertical lines. By using the approach above in conjunction with our approximation algorithm for Theorem 5.5.3, we obtain an improved approximation factor in the case that the vertical segments are lines. Before describing this result, we need a slightly stronger version of Theorem 5.5.3:

Lemma 5.6.2. *There is a polynomial-time 5/3-approximation algorithm for hitting a set of vertical lines and horizontal segments that always returns a solution of cost within 5/3 that of an optimal solution to the natural set cover LP relaxation.*

Proof: Given an instance of geometric hitting set over vertical lines and horizontal segments, let LP^* be the optimum value achieved by the natural set cover LP relaxation. We show that the algorithm used to establish Theorem 5.5.3 satisfies stronger versions of the bounds used in the proof of Theorem 5.5.3:

$$h \leq LP^*,\ v \leq LP^*, \text{ and } h - k_1 - k_2/2 + v - k_1 - k_2 \leq LP^*.$$

Since the vertical lines are disjoint, by summing the corresponding LP constraints, we see that $\sum_p x_p \geq v$ for any feasible x. Taking x to be an optimal solution, x^*, we have that $LP^* = \sum_p x_p^* \geq v$. As noted before, the natural set cover LP relaxation is totally unimodular in the case of hitting only horizontal segments. Thus by dropping the constraints corresponding to the lines from the LP, we may conclude that $LP^* \geq h$.

For the final bound, we need to show that for the type of instance obtained by our 5/3-approximation in Stage 3, LP^* is equal to OPT', the optimum size of a hitting set. Lemma 5.5.2 shows that for such instances, $OPT' = v' + h'$, where v' and h' are the individual vertical and horizontal lower bounds for the instance.

Consider a collection of collinear horizontal segments from a Stage-3 instance, and remove all points that lie on some vertical lie along with all the horizontal segments hit by such points. The proof of Lemma 5.5.2 shows that such a deletion does not increase the optimal number of points required to hit such an instance. Hence, appealing to the integrality of such LP instances when dropping the vertical line constraints, we have that $\sum_{p \in P' \setminus P'_V} x_p \geq h'$, where P' is the set of points of a Stage 3 instance, and P'_V is the set of points that line on some vertical line. Considering only the vertical line constraints, as above, gives us $\sum_{p \in P'_V} x_p \geq v'$. Together, these inequalities yield the desired bound, $\sum_{p \in P'} x_p \geq h' + v'$.

We may substitute these bounds in the proof of Theorem 5.5.3 to conclude that our algorithm selects at most $5/3 \cdot LP^*$ points instead of $5/3 \cdot OPT$ points. $\square$

Theorem 5.6.3. *For objects that are unions of a horizontal segment and a vertical line, the hitting set problem has a polynomial-time 10/3-approximation.*

Proof: Our algorithm is essentially the same as the 4-approximation of Theorem 5.6.1, with a different last step:

1. Solve the LP, and let x^* be an optimal solution.
2. Construct S'_i for each union i of a horizontal segment and a vertical line.
3. Now each S'_i is either a horizontal segment or a vertical line, and we find a feasible solution X for this instance using our 5/3-approximation.

We construct y^* just as in the proof of our 4-approximation; however, now we observe that y^* is feasible for the set cover LP relaxation for the instance defined by the S'_i. This is just an instance of hitting horizontal segments and vertical lines, and so:

$$|X| \leq 5/3 \cdot LP^* \leq 5/3 \cdot \sum_p y^*_p.$$

Since $\sum_p y^*_p \leq 2 \sum_p x^*_p$ as before, we have that $|X| \leq 10/3 \cdot \sum_p x^*_p \leq 10/3 \cdot OPT$ as desired. $\square$

Appendix

Details for Section 3: Proof of Theorem 5.3.1

Proof: Our reduction is from 3-SAT. The input points P are distributed on three vertical lines, denoted l_1, l_2 and l_3 from left to right. In Figure 5.2 we show a variable gadget: points in the first two columns can be hit by two sets of independent lines, red or blue; these represent the "True" or "False" setting of the variable u.

We can add the variable gadgets one by one onto the three columns so that all 3-lines are within variable gadgets. After that we use the following clause gadget to link variables. Figure 5.3 is a gadget represents $\bar{u} \cup v \cup \bar{w}$. The idea is we pick the location of $c[i]$ such that the addition of $c[i]$ and the intersections of $c[i]u[i]$, $c[i]\bar{v}[i]$ and $c[i]w[i]$ doesn't create new 3-lines except the green lines.

Let n and m denote the numbers of variables and clauses. Each variable gadget has $2m$ points on l_3. If the 3-SAT formula is satisfiable, the size of a maximum independent set of 3-lines in the variable and clause gadgets is $mn + m$, and there are $2n$ points on l_2 and $mn - m$ points on l_3 not covered. The last step is garbage collection: we put $mn + m$ points on l_1 under the condition that no new 3-line is created.

Thus, the 3-SAT formula is satisfiable if and only if the corresponding C3L can be covered by $2nm + 2m$ non-vertical lines($nm + m$ 3-lines and $nm + m$ 2-lines). $\square$

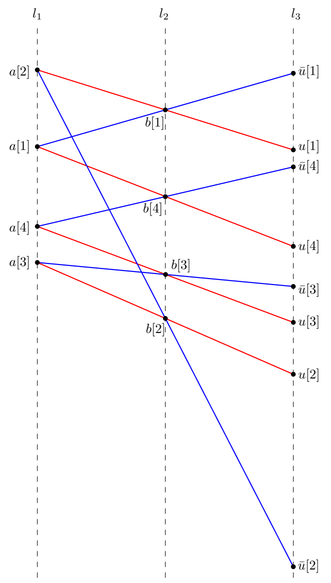


Figure 5.2: variable gadget

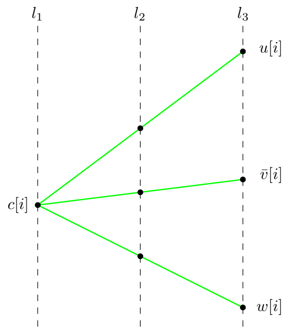


Figure 5.3: clause gadget

Details for Section 3: Analysis of the Greedy Hitting Set Algorithm for Lines of 3 Slopes in 2D

Let the numbers of lines of three orientations in a plane are x, y and z. WLOG we assume $x \geq y \geq z$.

If there is no 3-intersection, we know about OPT_2, the cardinality of an optimal solution

1. If $x \geq y + z$, $OPT_2(x, y, z) = x$.
2. If $x < y + z$, $OPT_2(x, y, z) = \lceil \frac{x+y+z}{2} \rceil = x + \lceil \frac{y+z-x}{2} \rceil$.

Statement 2 is true, because we can always pair up lines(all except one leftover) in the following way: pair two lines from two biggest groups currently; if three orientations have the same number of lines, pick from any two groups. After one pairring, one can show that the inequation that the cadinality of the biggest group is less than the sum of other groups still holds. Therefore the rightness can be proved by mathematical induction.

We propose a greedy algorithm: first pick 3-intersections(in any order) and remove the lines hit; then solve the problem without 3-intersections optimally.

Theorem 5.6.4. *The greedy algorithm yeilds a $\frac{7}{5}$-approximation.*

Proof: Let I be a maximum independent set of 3-intersections. Let $K = |I|$ be the cardinality of I. Let the set of 3-intersections given by greedy be J_3. We denote $|J_3|$ by G. We know that $3G \geq K \geq G$.

The optimal solution is

$$OPT = K + OPT_2(x - K, y - K, z - K) \tag{5.5}$$

The greedy solution is

$$GRE = G + OPT_2(x - G, y - G, z - G) \tag{5.6}$$

First consider a special case: $x = K$, which means $x = y = z = K$. Thus $OPT = K$ and $GRE = K + \lceil \frac{K-G}{2} \rceil$.

1. $K = 3l$, where l is an integer.

$$\frac{GRE}{OPT} \leq 1 + \lceil \frac{3l - l}{2} \rceil / 3l = \frac{4}{3} \tag{5.7}$$

2. $K = 3l + 1$

$$\frac{GRE}{OPT} \leq 1 + \lceil \frac{3l + 1 - (l + 1)}{2} \rceil / (3l + 1) < \frac{4}{3} \tag{5.8}$$

3. $K = 3l + 2$.

$$\frac{GRE}{OPT} \leq 1 + \lceil \frac{3l + 2 - (l + 1)}{2} \rceil / (3l + 2) \leq \frac{7}{5} \tag{5.9}$$

In the following discussion, we assume that $x \geq K + 1$.

1. If $x - G \geq y - G + z - G$, we have $x - K \geq y - K + z - K$. Thus

$$OPT = K + x - K = x$$
$$GRE = G + x - G = x$$

2. If $y + z - K \le x < y + z - G$,

$$OPT = x$$
$$GRE = x + \lceil \frac{y+z-x-G}{2} \rceil$$
$$\frac{GRE}{OPT} = 1 + \lceil \frac{y+z-x-G}{2} \rceil / x \le 1 + \lceil \frac{K-G}{2} \rceil / (K+1) \le \frac{4}{3}$$

   The detailed analysis is similar to the case $K = x$.

3. If $x < y + z - K$,

$$OPT = x + \lceil \frac{y+z-x-K}{2} \rceil \ge K + 1 + 1 = K + 2$$
$$GRE - OPT \le \frac{y+z-x-G}{2} + 1 - \frac{y+z-x-K}{2} \le 1 + \frac{K}{3}$$
$$\frac{GRE}{OPT} = 1 + \frac{GRE - OPT}{OPT} \le 1 + \frac{1 + K/3}{K+2}$$

   - when $K = 1$, $OPT = GRE$
   - when $K = 2, 3, 4$, $GRE - OPT = 1$. We have

$$\frac{GRE}{OPT} \le 1 + \frac{1}{4} = 1.25 \tag{5.10}$$

   - when $K \ge 5$,

$$\frac{GRE}{OPT} \le 1 + \frac{8}{21} \approx 1.381$$

□

Details for Section 3: Axis-Parallel Lines in 3D

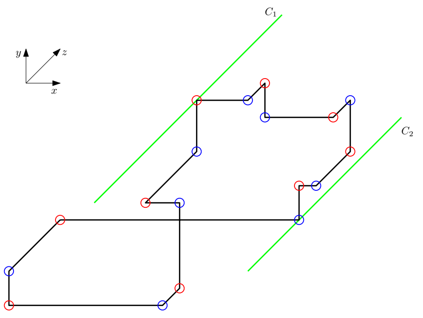


Figure 5.4: an example of variable loop

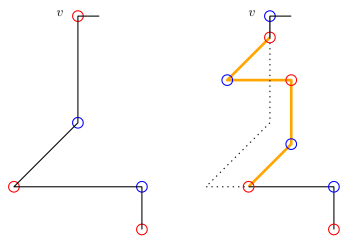


Figure 5.5: The insertion of the orange detour changes the color of v from red to blue.

Proof: We give a reduction of 3SAT. Let a d-line be a line parallel to d-axis. Let an ab-plane be a plane parallel to the plane spanned by a-axis and b-axis. A clause is represented by a z-line. A variable is represented by a loop of axis-parallel lines with the following properties:

1. No four edges of a loop are coplanar.
2. There are an even number of edges in each loop.
3. Lines from two different loops won't intersect.
4. A loop intersects a clause z-line iff the variable represented by that loop is in the clause represented by the z-line. The intersection actually is a literal in the corresponding clause.
5. There are two optimal hitting sets for a loop, odd vertices and even vertices. All positive literals should be in the same hitting set. So do the negative literals.

Figure 5.4 shows a part of an instance in which clause C_1 has x_1 and C_2 has $\overline{x_1}$.

In 3D space, it's not hard to take detours to avoid any unwanted intersection. The number of vertices on a loop can be adjusted by inserting a tiny detour as Figure 5.5 shows. At the end we argue that all clause z-lines can be hit for free by the optimal hitting sets of variable loops iff there ia a satisfying truth assignment for the corresponding 3SAT instance. □

Details for Section 5: Hitting Rays and Lines

Here, we give the proof of Theorem 5.4.1.

We refer to a horizontal ray to the left (resp., right) as an *l-ray* (resp. *r-ray*). Similarly, we speak of a horizontal (resp., vertical) line as an *h-line* (resp., *v-line*). If two collinear rays are disjoint, we shift one ray slightly, so that no two disjoint rays are collinear.

If a line only contains a ray, we add a ray to pair with it. For example, an h-line l has an r-ray. We put an l-ray on l such that the vertex of the ray is on the right of all v-lines. This additional ray won't change the optimal solution. We call the intersection of two h-rays a segment of them.

Let H and V denotes the numbers of segments and v-lines. We know that V points should be placed on V v-lines. Those points can help to hit segments. There are two ways to 'hit' a segment:

- place a point on the segment. In this case, we call the corresponding line a 3-hitter. We say the segment is 3-hit by the line.
- hit two extensions of the segment, which needs two points. In this case, we call the left(right) line l-hitter(r-hitter). We say the segment is dual-hit by those two lines.

Let v_1 and v_2 be the numbers of segments hit by the V points in first and second ways. Then the number of points to be placed is $H + V - v_1 - v_2$. Now the problem becomes to put a point on each lines to 'hit' segments as many as possible. From a case of HRVL, We construct a graph: each v-line and each segments are nodes; there is an edge between two node iff the v-line and segment they represent intersect. We know this is a bipartite graph and the maximum matching problem in a bipartite graph is polynomial solvable. A matching in the graph represents a set of intersections in the corresponding HRVL. The following lemma shows that in some sense, it is better to adopt the first way to hit segments. This is consistant with the intuition.

Lemma 5.6.5. *There is a maximum matching between lines and segments that can be augmented to be an optimal solution of HRVL.*

Proof: We prove this by contradiction. Let v_1^* be the biggest v_1 that an minimum hitting set can achieve. We assume v_1^* is less than m, which is the cardinality of the maximum matching between v-lines and segments. Thus there is an augmenting path.

An augmenting path in graph G matches to a sequence of alternate segments and v-lines in HRVL. In Figure 5.6 the green path is an example of an augmenting path. Because of the optimality of the current solution, any augmenting path cannot improve it. It is easy to prove that the following two things should both happen, because otherwise after augmenting, the sum of v_1 and v_2 will stay the same, but v_1^* would be increased by 1:

- e_1 is dual-hit by other lines;
- l_n is helping to dual-hit another segment,

where the augmenting path is $\{e_1, l_1, e_2, l_2, \ldots, e_n, l_n\}$.

Without loss of generality, we assume the intersection of e_1 and l_1 is on the left of l_n.

1. If l_n is the l-hitter of e_t, then the l-hitter of e_1 can take its job. One can do the augmenting and assign the l-hitter of e_1 to l-hit e_t. Therefore the solution is optimal and v_1^* increases.
2. If l_n is an r-hitter and the r-hitter of e_1 is on the right of l_n, then that r-hitter can take the job of l_n.
3. If l_n is an r-hitter and the l_r, the r-hitter of e_1, is on the left of l_n, we know that l_r will intersect e_t, where $1 \le t \le n$ because, we assume l_n is on the right of the intersection of e_1 and l_1. l_r leads us to a 'better' augmenting path, $\{e_1, l_1, \ldots, e_t, l_r\}$. Adopting the new augmenting path, we keep the hitting set optimal and increase v_1^* by 1.

□

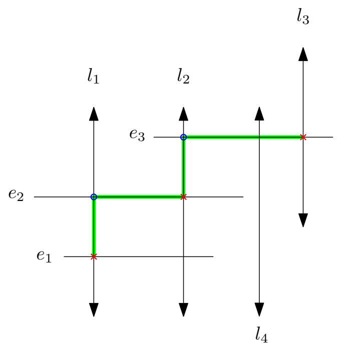


Figure 5.6: In this augmenting path, the size of a matching is increased by replacing the blue circles with red crosses.

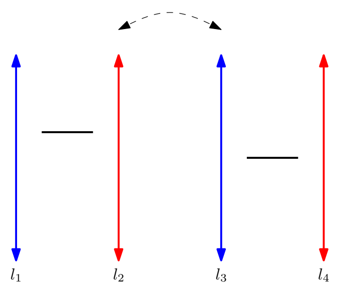


Figure 5.7: Swapping l_2 and l_3 makes both of them more useful.

Lemma 5.6.6. *Given an optimal solution $\mathcal{S}$, there is an optimal solution $\mathcal{S}'$ that has the same set of 3-hitters as $\mathcal{S}$ and its l-hitters are all on the left side of its r-hitters.*

Proof: In Figure 5.7 two segments are dual-hit by two pairs of v-lines; the blue lines are l-hitter and the red lines are r-hitter. When we pair l_1 to l_3 and pair l_2 to l_4, two segments are still dual-hit, because this swap makes the l-hitter more left and r-hitter more right. □

Having this lemma, we can easily find a sweeping algorithm for the case which has no 3-intersections.

```
Data: disjoint segments and $n$ v-lines($l_0, l_1, \dots, l_{n-1}$)
Result: maximum matching between segments and dual hitters
$k \leftarrow \lfloor n/2 \rfloor$, $i \leftarrow 0$, $j \leftarrow k$;
while $i < k$ and $j < n$ do
    if there is a segment on the right of $l_i$ then
        $e \leftarrow$ the leftmost segment that's on the right of $l_i$;
        while $j < n$ and $l_j$ is on the left of $e$ do
            $j \leftarrow j + 1$
        end
        if $j < n$ then
            match $e$ to $l_i$ and $l_j$;
            remove them;
        else
            exit;
        end
    else
        exit;
    end
    $i \leftarrow i + 1$;
end
```

Algorithm 1: Sweeping algorithm for HRVL without 3-intersections

For HRVL with 3-intersections, we propose the Algorithm below. The idea of this algorithm is to maximize the number of 3-intersections and meanwhile 'balance' the lines left as much as possible. In the algorithm, we test the vitality of a line: a line is vital means if the line is not hit by a 3-intersection, one has no chance to maximize the number of 3-intersections of that HRVL instance.

Let S be the solution given by Algorithm 2 and S' be an optimal solution with maximum set of 3-hitters. We know that S and S' have the same number of 3-hitters. Let D and D' denote the v-lines left behind in S and S' respectively. We order lines in D and D' from left to right. Let k be $\lfloor \frac{|D|}{2} \rfloor$. Hence there are at most k pairs of dual-hitters in S and S'. Let $lh_i(lh'_i)$ be the ith line of $D(D')$.

Given a solution P and a line l, we use $E(l, P)$ to denote the number of segments that are on the left side of l and not hit by 3-hitters in P. It's easy to see that a line having more segments on its right side has a bigger chance to be an l-hitter. Another way to say it is as an l-hitter, lh_i is at least as capable as lh'_i.

$$E(lh_i, S) \leq E(lh'_i, S'), \quad i = 1, 2, .., k \tag{5.11}$$

We split the proof of the statement above into two lemmas.

Lemma 5.6.7. *lh_i cannot be on the right side of lh'_i, $i = 1, 2, .., k$*

```
H ← [0, 0]   //H counts 2-hitters at left and right sides;
$I_3$ ← {}   //$I$ stores 3-intersections of the solution;
SD←0;  //SD stands for sweep direction. 0 is from left to right; 1 is
reverse;
step A: if there is any 3-intersection left then
    sweep along the direction indicated by SD;
    if the event is a line then
        step B: remove the line;
        H[SD]++;
        toggle SD;
    else
        the event is a segment $e_1$;
        if $e_1$ crosses some line(s) then
            $l$ ← the line hit $e_1$ and is closest along SD;
        else
            remove $e_1$;
            go to step A;
            if $l$ is vital then
                //For example, if SD is 0, look at the right endpoints of
                segments crossed by $l$. pick the nearest one to $l$;
                $e_2$ ← the segment crosses $l$ and has the closest endpoint
                along SD from $l$;
                put the intersection of $e_2$ and $l$ into $I_3$;
                go to step A;
            else
                go to step B;
            end
        end
    end
else
    The left problem can be solved optimally as an edge cover problem;
end
```
Algorithm 2: Bidirectional Sweeping algorithm for HRVL with 3-intersections

Proof: We prove it by induction.

Because of the vitality test, we have an important observation: in S if a 3-hitter is on the left side of an l-hitter, the segment hit by the 3-hitter will not intersect that l-hitter,

When $i = 1$, we assume that lh_1 is on the right side of lh'_1. It means in S lh'_1 is a 3-hitter. Suppose e_1 is the corresponding segment hit by lh'_1 in S. We know in S' lh'_1 doesn't hit e_1(since it's not a 3-hitter in S', so e_1 must be hit by a 3-hitter in S', say l_3. Again in S, l_3 hits e_2, which means in S', e_2 should be hit by another line which is l_4. Because of the first observation, the backtracking process totally happens on the left side of lh_1. It will stop eventually, because there are only finite number of lines on the lh_1's left and contradiction is found.

Now we assume k is the smallest integer such that lh_k is on the right side of lh'_k. We again start backtracking from lh_k and we know the process can only end at $lh_j(j <)$ or a contradiction exists as the base case. Let the backtracking sequence be $lh_k, lh'_k, e_1, l_3, e_2, l_4, \dots, lh_j$. Since lh'_j is on the right side of lh_j or lh_j itself, lh'_j will intersect some e_l in the backtracking sequence. That means we find an augment path by which we can increase the number of 3-hitters in S' by 1. The contradiction is found. □

Let $N(l)$ be the number of segments on the left of line l. An immediate result from this lemma is

$$E(lh_i, S') \le E(lh'_i, S') \tag{5.12}$$

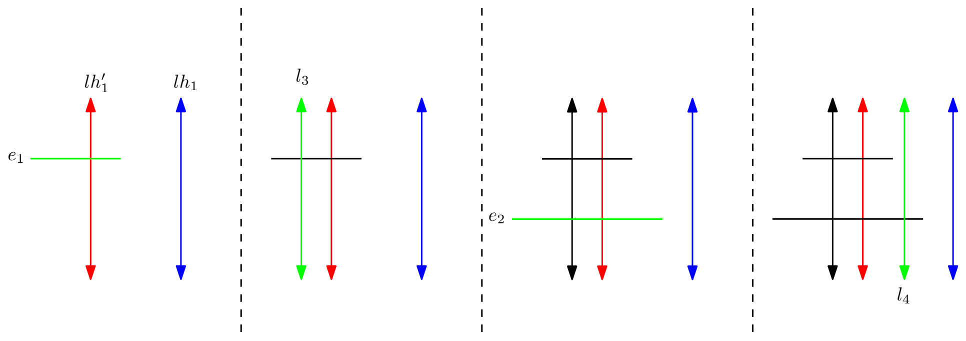


Figure 5.8: The backtracking sequence is lh_1, lh'_1, e_1, l_3, e_2, l_4.

Given a solution P and a line l, We use $C(l, P)$ to denote the number of segments that are on the left side of l and have been 3-hit in P. The following lemma is to say S leaves the segments which are easier to dual-hit for 2-hitters.

Lemma 5.6.8. $C(lh_i, S) \ge C(lh_i, S')$, $i = 1, 2, .., k$

Proof: We have already known that in S, all segments hit by 3-hitters on the left side of lh_i are also on the left side of lh_i. If $C(lh_i, S) < C(lh_i, S')$, then we replace the part of S that is on the left side of lh_i by the corresponding part of S'. What we get is a solution which has more 3-hitters than S, which contradicts to the assumption that S has the maximum set of 3-hitters.

□

Therefore we obtain

$$E(lh_i, S) = N(lh_i) - C(lh_i, S) \leq N(lh_i) - C(lh_i, S') = E(lh_i, S') \leq E(lh'_i, S') \quad (5.13)$$

.

5.7 Hitting Triangle-Free Sets of Segments

Consider the arrangement graph, $G = (V, E)$, induced by the set S of n input segments. Assume that G is triangle-free.

Define the following clipping/shortening process:

(i) Pick a vertex $v \in V$ of degree at most 3 (it will necessarily be a segment endpoint); such a vertex must exist, by the triangle-free property.

(ii) Remove the vertex v, and shrink the incident segments with endpoint v to the next adjacent vertex. (In particular, if v is a T-junction, where two of the edges incident to v lie on a common segment, then only the one segment with endpoint at v is shrunk, leaving the other two edges connected.)

(iii) When shortening a segment s results in segment s becoming a single point (vertex), u, establish a hitting point at u and remove all segments that pass through u.

Invariants at any stage of the process:

(1) There is at most one remaining subsegment of an input segment (i.e., the portion of an original segment s that remains is connected);

(2) All segments that have been removed are hit by the hitting points that have been established;

(3) Any hitting set of the remaining segments, together with the established hitting points already found, forms a hitting set for the original set of input segments;

(4) The graph G remains triangle-free.

Claim 4. *The number of hitting points established by the algorithm is at most 3 times the number, $|H|$, of points in any hitting set H for S.*

Proof: Place tokens on the vertices H and consider running the clipping/shortening process on G, with the following actions on the tokens:

When there is a token on the vertex v that is about to be clipped, replace the token with at most 3 clones of it, one on each of the segments that meet at v, allowing each clone to slide along with the endpoint of a clipped segment s as the segment is shrunk, leaving the clone at a new vertex u, the new endpoint of segment s. (There might also be a token at u already; we allow two or more tokens/clones to accumulate at a vertex.) We never clone a clone; if a clone associated with a segment s exists at a vertex v that is being clipped, it remains on segment s, and slides along it as it shrinks. Thus, associated with each point of H there is either a single token or up to 3 clones of the token (but not both).

This ensures that the tokens/clones continue to hit all segments, at all stages of the clipping/shortening process. (Here, we are using the degree-3 property, which allows us to make sure that two edges incident on v that lie on the same segment s are not cut apart at v in our process; thus, a point of H that lies on s continues to hit the shrunk version of segment s. If we had split s at v, with no point of H at v, then no clones are generated at v, and the point(s) of H on segment s may no longer be a valid hitting set for the new arrangement after splitting s at v.) In particular, when a segment shrinks to a point u, there is at least one token/clone present there. Thus, the number of hitting points established by our algorithm is at most $3|H|$, for any hitting set H of S. □

Theorem 5.7.1. *The algorithm yields a 3-approximation and runs in time $O(m)$, where m is the number of edges in the original (planar) arrangement graph G.*

Proof: Immediate, since we only have to maintain the graph G in a standard planar network data structure (e.g., the Doubly Connected Edge List (DCEL)) that allows us to know vertex degrees and perform elementary operations in constant time. □

Bibliography

[1] C Icking, R Klein, E Langetepe, S Schuierer, and I Semrau. An optimal competitive strategy for walking in streets. *SIAM JOURNAL ON COMPUTING*, 33(2):462–486, 2004. ISSN 0097-5397. doi: {10.1137/S0097539702419352}.

[2] Alejandro López-Ortiz and Sven Schuierer. Lower bounds for streets and generalized streets. *International Journal of Computational Geometry and Applications*, 11(4), 2001.

[3] Alejandro López-Ortiz and Sven Schuierer. Searching and on-line recognition of star-shaped polygons. *Inf. Comput.*, 185(1):66–88, August 2003. ISSN 0890-5401. doi: 10.1016/S0890-5401(03)00081-6. URL `http://dx.doi.org/10.1016/S0890-5401(03)00081-6`.

[4] Avrim Blum, Prabhakar Raghavan, and Baruch Schieber. Navigating in unfamiliar geometric terrain. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, STOC '91, pages 494–504, New York, NY, USA, 1991. ACM. ISBN 0-89791-397-3. doi: 10.1145/103418.103419. URL `http://doi.acm.org/10.1145/103418.103419`.

[5] Peter Eades, Xuemin Lin, and Nicholas C. Wormald. Performance guarantees for motion planning with temporal uncertainty. *Australian Computer Journal*, 25(1):21–28, 1993. URL `http://dblp.uni-trier.de/db/journals/acj/acj25.html#EadesLW93`.

[6] Y. Gabriely and E. Rimon. Cbug: A quadratically competitive mobile robot navigation algorithm. *Robotics, IEEE Transactions on*, 24(6):1451 –1457, dec. 2008. ISSN 1552-3098. doi: 10.1109/TRO.2008.2006237.

[7] Wei Zeng, Rik Sarkar, Feng Luo, Xianfeng David Gu, and Jie Gao. Resilient routing for sensor networks using hyperbolic embedding of universal covering space. In *Proc. of the 29th Annual IEEE Conference on Computer Communications (INFOCOM'10)*, pages 1694–1702, March 2010.

[8] Kan Huang, Chien-Chun Ni, Rjk Sarkar, Jie Gao, and Joseph SB Mitchell. Bounded stretch geographic homotopic routing in sensor networks. In *INFOCOM, 2014 Proceedings IEEE*, pages 979–987. IEEE, 2014.

[9] Robert Kleinberg. Geographic routing using hyperbolic space. In *Proceedings of the 26th Conference of the IEEE Communications Society (INFOCOM'07)*, pages 1902–1909, 2007.

[10] James Newsome and Dawn Song. GEM: graph embedding for routing and data-centric storage in sensor networks without geographic information. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 76–88, 2003. ISBN 1-58113-707-9.

[11] Jie Gao and Leonidas Guibas. Geometric algorithms for sensor networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 370(1958):27–51, 2012. doi: 10.1098/rsta.2011.0215. URL `http://rsta.royalsocietypublishing.org/content/370/1958/27.abstract`.

[12] B. Karp and H. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254, 2000.

[13] Qing Fang, Jie Gao, Leonidas Guibas, Vin de Silva, and Li Zhang. GLIDER: Gradient landmark-based distributed routing for sensor networks. In *Proc. of the 24th Conference of the IEEE Communication Society (INFOCOM)*, volume 1, pages 339–350, March 2005.

[14] Guang Tan, Marin Bertier, and Anne-Marie Kermarrec. Convex partition of sensor networks and its use in virtual coordinate geographic routing. In *INFOCOM*, pages 1746–1754, 2009.

[15] Xianjin Zhu, Rik Sarkar, and Jie Gao. Segmenting a sensor field: Algorithms and applications in network design. *ACM Trans. Sen. Netw.*, 5(2): 12:1–12:32, April 2009. ISSN 1550-4859. doi: 10.1145/1498915.1498918. URL `http://doi.acm.org/10.1145/1498915.1498918`.

[16] Nancy M. Amato, Michael T. Goodrich, and Edgar A. Ramos. A randomized algorithm for triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, pages 245–265, 2001.

[17] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, August 1991. ISSN 0179-5376. doi: 10.1007/BF02574703. URL `http://dx.doi.org/10.1007/BF02574703`.

[18] R. Bar-Yehuda and Bernard Chazelle. Triangulating disjoint Jordan chains. *Internat. J. Comput. Geom. Appl.*, 4(4):475–481, 1994.

[19] Raimund Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1(1):51–64, July 1991. ISSN 0925-7721. doi: 10.1016/S0925-7721(99)00042-5. URL `http://dx.doi.org/10.1016/S0925-7721(99)00042-5`.

[20] John Hershberger and Jack Snoeyink. Computing minimum length paths of a given homotopy class. *Comput. Geom. Theory Appl.*, 4(2):63–97, June 1994. ISSN 0925-7721. doi: 10.1016/0925-7721(94)90010-8. URL `http://dx.doi.org/10.1016/0925-7721(94)90010-8`.

[21] Prosenjit Bose and Pat Morin. Online routing in triangulations. In *Proceedings of the 10th International Symposium on Algorithms and Computation (ISAAC '99)*, pages 113–122, 1999.

[22] Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia. Compass routing on geometric networks. In *Proc. 11th Canadian Conference on Computational Geometry*, pages 51–54, 1999.

[23] Prosenjit Bose and Pat Morin. Online routing in triangulations. *SIAM J. Comput.*, 33(4):937–951, 2004.

[24] Prosenjit Bose and Pat Morin. Competitive online routing in geometric graphs. *Theor. Comput. Sci.*, 324(2-3):273–288, 2004.

[25] Prosenjit Bose, Andrej Brodnik, Svante Carlsson, Erik D. Demaine, Rudolf Fleischer, Alejandro López-Ortiz, Pat Morin, and J. Ian Munro. Online routing in convex subdivisions. *Int. J. Comput. Geometry Appl.*, 12(4):283–296, 2002.

[26] Joseph S.B. Mitchell. Geometric shortest paths and network optimization. In *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, 1998.

[27] Vladimir J. Lumelsky and Alexander A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[28] Peter Eades, Xuemin Lin, and Nicholas C. Wormald. Performance guarantees for motion planning with temporal uncertainty. *Australian Computer Journal*, 25(1):21–28, 1993. URL `http://dblp.uni-trier.de/db/journals/acj/acj25.html#EadesLW93`.

[29] Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theor. Comput. Sci.*, 84(1):127–150, July 1991. ISSN 0304-3975. doi: 10.1016/0304-3975(91)90263-2. URL `http://dx.doi.org/10.1016/0304-3975(91)90263-2`.

[30] Rolf Klein. Walking an unknown street with bounded detour. In *Proceedings of the 32nd annual symposium on Foundations of computer science*, SFCS '91, pages 304–313, Washington, DC, USA, 1991. IEEE Computer Society. ISBN 0-8186-2445-0. doi: 10.1109/SFCS.1991.185383. URL `http://dx.doi.org/10.1109/SFCS.1991.185383`.

[31] Piotr Berman. On-line searching and navigation. In *Online Algorithms*, pages 232–241, 1996.

[32] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, November 2001.

[33] Rik Sarkar. Low distortion delaunay embedding of trees in hyperbolic plane. In *Proceedings of the 19th international conference on Graph Drawing*, GD'11, pages 355–366, 2011.

[34] Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S. B. Mitchell. Touring a Sequence of Polygons. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 473–482, New York, NY, USA, 2003. ACM. ISBN 1-58113-674-9. doi: 10.1145/780542.780612. URL `http://doi.acm.org/10.1145/780542.780612`.

[35] Arash Ahadi, Amirhossein Mozafari, and Alireza Zarei. Touring a sequence of disjoint polygons: Complexity and extension. *Theoretical Computer Science*, 556:45–54, 2014.

[36] Joonsoo Choi, Juergen Sellen, and Chee-Keng Yap. Approximate Euclidean Shortest Paths in 3-Space. *International Journal of Computational Geometry & Applications*, 07(04):271–295, August 1997. ISSN 0218-1959. doi: 10.1142/S0218195997000181. URL `http://www.worldscientific.com/doi/abs/10.1142/S0218195997000181`.

[37] Kenneth L. Clarkson. Approximation Algorithms for Shortest Path Motion Planning (Extended Abstract). In *In Proc. 19th Annu. ACM Sympos. Theory Comput*, pages 56–65, 1987.

[38] Jürgen Sellen, Joonsoo Choi, and Chee-Keng Yap. Precision-Sensitive Euclidean Shortest Path in 3-Space. In *11TH ACM SYMP. ON COMP. GEOM*, pages 350–359, 1995.

[39] Tetsuo Asano, David Kirkpatrick, and Chee Yap. Pseudo Approximation Algorithms with Applications to Optimal Motion Planning. *Discrete & Computational Geometry*, 31(1):139–171, November 2003. ISSN 0179-5376, 1432-0444. doi: 10.1007/s00454-003-2952-3. URL `http://link.springer.com/article/10.1007/s00454-003-2952-3`.

[40] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 475–484. ACM, 1997.

[41] Valentin E Brimkov, Andrew Leach, Jimmy Wu, and Michael Mastroianni. Approximation algorithms for a geometric set cover problem. *Discrete Applied Mathematics*, 160(7):1039–1052, 2012.

[42] Valentin E Brimkov, Andrew Leach, Michael Mastroianni, and Jimmy Wu. Guarding a set of line segments in the plane. *Theoretical Computer Science*, 412(15):1313–1324, 2011.

[43] Valentin E Brimkov. Approximability issues of guarding a set of segments. *International Journal of Computer Mathematics*, 90(8):1653–1667, 2013.

[44] Anup Joshi and NS Narayanaswamy. Approximation algorithms for hitting triangle-free sets of line segments. In *Algorithm Theory–SWAT 2014*, pages 357–367. Springer, 2014.

[45] Apichat Heednacram. *The NP-hardness of covering points with lines, paths and tours and their tractability with FPT-algorithms*. Griffith University, 2010.

[46] Stefan Kratsch, Geevarghese Philip, and Saurabh Ray. Point line cover: the easy kernel is essentially tight. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1596–1606. SIAM, 2014.

[47] Nimrod Megiddo and Arie Tamir. On the complexity of locating linear facilities in the plane. *Operations research letters*, 1(5):194–197, 1982.

[48] Björn Brodén, Mikael Hammar, and Bengt J. Nilsson. Guarding lines and 2-link polygons is apx-hard. In *Proceedings of the 13th Canadian Conference on Computational Geometry, University of Waterloo, Ontario, Canada, August 13-15, 2001*, pages 45–48, 2001. URL `http://www.cccg.ca/proceedings/2001/mikael-2351.ps.gz`.

[49] VS Anil Kumar, Sunil Arya, and Hariharan Ramesh. Hardness of set cover with intersection 1. In *Automata, Languages and Programming*, pages 624–635. Springer, 2000.

[50] Sofia Kovaleva and Frits CR Spieksma. Approximation algorithms for rectangle stabbing and interval stabbing problems. *SIAM Journal on Discrete Mathematics*, 20(3):748–768, 2006.

[51] Adrian Dumitrescu and Minghui Jiang. On the approximability of covering points by lines and related problems. *arXiv preprint arXiv:1312.2549*, 2013.

[52] Refael Hassin and Nimrod Megiddo. Approximation algorithms for hitting objects with straight lines. *Discrete Applied Mathematics*, 30(1):29–42, 1991.

[53] Daya Ram Gaur and Binay Bhattacharya. Covering points by axis parallel lines. In *Proc. 23rd European Workshop on Computational Geometry*, pages 42–45. Citeseer, 2007.

[54] Michael Dom, Michael R Fellows, and Frances A Rosamond. Parameterized complexity of stabbing rectangles and squares in the plane. In *WALCOM: Algorithms and Computation*, pages 298–309. Springer, 2009.

[55] Guy Even, Retsef Levi, Dror Rawitz, Baruch Schieber, Shimon Moni Shahar, and Maxim Sviridenko. Algorithms for capacitated rectangle stabbing and lot sizing with joint set-up costs. *ACM Transactions on Algorithms (TALG)*, 4(3):34, 2008.

[56] Daya Ram Gaur, Toshihide Ibaraki, and Ramesh Krishnamurti. Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilinear partitioning problem. In *Algorithms-ESA 2000*, pages 211–219. Springer, 2000.

[57] Panos Giannopoulos, Christian Knauer, Günter Rote, and Daniel Werner. Fixed-parameter tractability and lower bounds for stabbing problems. *Computational Geometry*, 46(7):839–860, 2013.

[58] Stefan Langerman and Pat Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005.

[59] J. O'Rourke. *Art Gallery Theorems and Algorithms.* The International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

[60] Jorge Urrutia et al. Art gallery and illumination problems. *Handbook of computational geometry*, 1(1):973–1027, 2000.

[61] Valentin E Brimkov, Andrew Leach, Michael Mastroianni, and Jimmy Wu. Experimental study on approximation algorithms for guarding sets of line segments. In *Advances in Visual Computing*, pages 592–601. Springer, 2010.

[62] Boris Aronov, Esther Ezra, and Micha Sharir. Small-size ε-nets for axis-parallel rectangles and boxes. *SIAM Journal on Computing*, 39(7):3248–3282, 2010.

[63] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete Comput. Geom.*, 14:263–279, 1995.

[64] Kenneth L Clarkson. Algorithms for polytope covering and approximation. In *Algorithms and Data Structures*, pages 246–252. Springer, 1993.

[65] Kenneth L. Clarkson and Kasturi Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37(1):43–58, 2007.

[66] Guy Even, Dror Rawitz, and Shimon Moni Shahar. Hitting sets when the vc-dimension is small. *Information Processing Letters*, 95(2):358–362, 2005.

[67] Noga Alon. A non-linear lower bound for planar epsilon-nets. *Discrete & Computational Geometry*, 47(2):235–244, 2012.

[68] János Pach and Gábor Tardos. Tight lower bounds for the size of epsilon-nets. *Journal of the American Mathematical Society*, 26(3):645–658, 2013.

[69] Nabil H Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

[70] D. S. Hochbaum and W. Maas. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32:130–136, 1985.

[71] Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[72] Robert D. Carr, Toshihiro Fujito, Goran Konjevod, and Ojas Parekh. A 2 1/10-approximation algorithm for a generalization of the weighted edge-dominating set problem. In *Algorithms-ESA 2000*, pages 132–142. Springer, 2000.