# Efficiently Navigating a Bounded Degree and Light Planar Spanner Without a Map

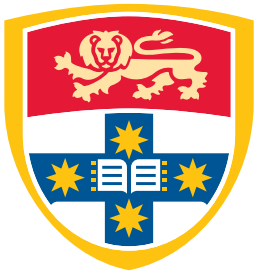
VIKRANT ASHVINKUMAR
SID: 440591085

Supervisor: Dr. Joachim Gudmundsson
Associate Supervisor: Dr. André van Renssen

This thesis is submitted in partial fulfillment of
the requirements for the degree of
Bachelor of Computer Science and Technology (Honours)

School of Information Technologies
The University of Sydney
Australia

6 November 2018



THE UNIVERSITY OF SYDNEY

## Student Plagiarism: Compliance Statement

I certify that:

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

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

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

**Name**: Vikrant Ashvinkumar

**Signature**: vash7242 **Date**: November 6, 2018

# Abstract

For a finite point set $V$ in the plane, are there networks connecting $V$ that have low degree and light weight that we can moreover navigate efficiently without a map? Yes, there are. We show how to construct a constant degree and light planar spanner on $V$, and show an $O(1)$-memory deterministic 1-local routing algorithm on these spanners with a routing ratio of not more than $5.90(1 + 1/r)\max(\pi/2, \pi\sin(\theta/2) + 1)$, where $0 < \theta < \pi/2$ and $r > 0$ are adustable parameters. The degree of our construction is bounded above by $5\lceil 2\pi/\theta \rceil$, which gives a least upper bound of 25. The weight is bounded above by $1.998(2r + 1)\max(\pi/2, \pi\sin(\theta/2) + 1)$ times that of a minimum spanning tree on $V$; and, if $\theta < \pi/3$, the weight is then bounded above by just a $2r + 1$ factor instead.

Previously, online routing algorithms have been studied on graphs such as, notably, the Delaunay Triangulation under different metrics. However, the graphs known to be efficiently routable have so far not been cheap; the Delaunay Triangulation, while planar, has neither constant degree nor is it light. Similarly, other graphs shown to admit an online routing algorithm with a constant routing ratio do not have all of planarity, a constant degree, and lightness. To the best of our knowledge, this is the first time online routing has been attempted on a constant degree and light planar spanner.

# Acknowledgements

Thanks to my collaborators Joachim Gudmundsson and André van Renssen, for letting me use your noses to navigate a little part of this little corner of Computational Geometry. When I think about your contributions, which I try to avoid for the following reason, I invariably end up asking myself: so what did I really do other than show up?

# Contents

**Student Plagiarism: Compliance Statement** ii

**Abstract** iii

**Acknowledgements** iv

**List of Figures** viii

**Chapter 1 Introduction** 1

1.1 Our Contribution . . . 2

1.2 Outline . . . 3

**Chapter 2 Preliminary Notions** 5

2.1 Spanners . . . 5

2.1.1 The $\Theta$-Graph . . . 6

2.1.2 Bounded Degree and Light Planar Spanners Exist . . . 7

2.2 Online Routing . . . 7

2.2.1 Routing in the $\Theta$-Graph . . . 8

2.2.2 Greedy Routing . . . 8

2.2.3 Compass Routing . . . 9

2.2.4 Right-Hand Routing . . . 9

2.3 Delaunay Triangulations . . . 9

**Chapter 3 Everything But Lightness** 12

3.1 Description of $\mathcal{BDG}(V)$ . . . 12

3.2 Degree Bound of $\mathcal{BDG}(V)$ . . . 14

3.3 Spanning Properties of $\mathcal{BDG}(V)$ . . . 14

3.3.1 A Separation of Neighbours of Neighbours . . . 14

3.3.2 Notion of Protection . . . 15

3.3.3 Lesser Protection Buys Neighbours in the Same Cone . . . 15

3.3.4 Penultimate and Middle Edges are Bought Full Protection . . . . . 16
3.3.5 Hull Paths Within Cones are Almost Complete . . . . . 16
3.3.6 Proof of the Spanning Ratio . . . . . 17
3.4 Algorithmic Construction of $\mathcal{BDG}(V)$ . . . . . 20
3.5 $\mathcal{BDG}(V)$ Defeats Both Greedy and Compass Routing . . . . . 21

**Chapter 4 Marking the Terrain** **22**
4.1 Everything is Protected . . . . . 22
4.2 Signposts . . . . . 23
4.3 Algorithmic Construction of $\mathcal{MBDG}(V)$ . . . . . 26

**Chapter 5 Routing** **28**
5.1 Chew's Routing Algorithm . . . . . 28
5.2 Compromised Chew's Routing Algorithm . . . . . 29
5.2.1 Termination . . . . . 31
5.2.2 Routing Ratio . . . . . 31
5.3 Simulating Compromised Chew's Routing Algorithm on $\mathcal{MBDG}(V)$ . . . . . 35
5.3.1 Preliminaries . . . . . 35
5.3.2 The Simulation . . . . . 38

**Chapter 6 Lightness, and Routing Redux** **45**
6.1 The Levcopoulos and Lingas Protocol . . . . . 45
6.1.1 Preliminaries . . . . . 46
6.1.2 How the Polygon $P$ Grows . . . . . 46
6.1.3 Condition For Including an Edge . . . . . 48
6.2 $weight$ Refers to Face Paths . . . . . 55
6.3 Signposts II . . . . . 56
6.4 Routing on $\mathcal{LMBDG}(V)$ . . . . . 56

**Chapter 7 Conclusion** **59**
7.1 Further Work . . . . . 59
7.1.1 Tighter Bounds on the Routing Ratio of $\mathcal{LMBDG}(V)$ . . . . . 60
7.1.2 Navigating Unmarked Terrain . . . . . 60
7.1.3 Pushing the Degree Down . . . . . 60
7.1.4 Memoryless Navigators . . . . . 60

7.1.5 Tighter Weight Bound . . . . . 61
7.1.6 Simulation of Alternative Algorithms . . . . . 61

**References** **62**

# List of Figures


2.1 Points arranged in a star give high degree and heavy $\Theta$-Graphs. 6

2.2 The hull of $u$ coloured in red. 11

2.3 $u$ is in $\odot(t, v, w)$. 11

3.1 Two examples of the vertices in some cone $C$ with apex $u$. On the left, $m = 4$, and on the right, $m = 3$. 13

3.2 Extreme, penultimate, and middle are mutually exclusive properties taking precedence in that order. Red, blue, and brown edges are extreme, penultimate, middle edges of the Delaunay Triangulation, respectively. Gray unlabeled edges are edges which are neither extreme, penultimate, nor middle, and can only occur in a cone with more than five neighbours (bottom right). 13

3.3 Example placement of $u, v_l, v, v_r$ in in the circle $\odot(v_l, u, v_r)$. 15

3.4 Observation 2, pictorially. 16

3.5 The red path from $v_1$ to $v_m$ is assuredly in $\mathcal{BDG}(V)$. $uv_0$ and $uv_{m+1}$ are extreme, $uv_1$ and $uv_m$ penultimate, and $uv_j$ middle. 17

3.6 Details about the spanning path from $u$ to $v$ (in this example $v = v_9$). 18

3.7 Greedy and Compass Routing dithers between $s$ and $p$. $te_1$ and $te_2$ are extreme, $tp_1$ and $tp_2$ penultimate, and $tm$ middle. 21

4.1 Spanning path from $u$ to $v$. 23

4.2 When routing from $v$ to $u$, how can we know which way to take to give a spanning path? 24

4.3 Bits at non-graphical edges indicating whether to take the clockwise or counterclockwise edge to it to get a spanning path from $v$ to $u$. 24

5.1 Routing to $p$ (left) and $q$ (right). 29

5.2 $A_{i-1}$ and $A_i$ (with $A_i$ not necessarily being the same as $T_i$). 30

5.3 Routing to $p$ (left) and $q$ (right). 30

5.4 Example of a Type $A_1$ Worst Case Circle (outlined in black). 32

5.5 Example of a Type $A_2$ Worst Case Circle (outlined in black). 33

5.6 Example of a Type $B$ Worst Case Circle (outlined in black). 33

5.7 $v_i$ is above $l_i$ and below $w_i$. 34

5.8 Observation 5 does not hold since a clockwise walk around $C_i$ from $v_i$ to $r_i$ encounters neither vertex of the Delaunay Triangle. 35

5.9 Face of $\mathcal{MBDG}(V)$. 36

5.10 Unguided Face Walk from $v$ to $p$. 36

5.11 $vp$ is a chord of some face, with a guiding bit at $v$. 37

5.12 Guided Face Walk from $v$ to $p$. 38

5.13 $A_0$ contained between $su_1$ and $su_m$ 39

5.14 Example depiction of $\overline{v_i f}$. 40

5.15 $A_i$ contained between $v_i u_1$ and $v_i u_m$ 41

5.16 $A_{i-1}$ and $A_i$ contained between $v_i u_1$ and $v_i u_m$ 42

5.17 $A_i$ contained between $v_i u_1$ and $v_i u_m$ 43

6.1 $\partial P(u, v)$ has a part visible to $uv$. The dotted edge is a convex hull edge of $V$. 47

6.2 $\partial P(u, v) \cup uv$ is subdivided into $k$ cells ($k = 3$ in this picture). Each gray edge is an edge in $\mathcal{MBDG}(V)$. 47

6.3 $c_1$ coincides with part of $\partial P(u, v)$ except its one unsettled edge. $c_2$ and $c_3$ are not candidates for expansion. 48

6.4 Expansion of $P$ into $c_1$. 48

6.5 We can replace $uv_2$ with $v_1 v_2$ to get a lighter tree. 52

6.6 We can replace $uv$ with $uw$ to get a tree no heavier. 53

6.7 We can replace $uv$ with $vw$ to get a lighter tree. 53

6.8 Excluded edges along $\partial P(u, v)$ have face paths inductively. 55

6.9 Routing an Unguided Face Path in $\mathcal{LMBDG}(V)$. The orange paths are $(1 + 1/r)$-paths of their corresponding edges. 57

6.10 Routing a Guided Face Path in $\mathcal{LMBDG}(V)$. The orange paths are $(1 + 1/r)$-paths of their corresponding edges. 58

CHAPTER 1

# Introduction

---

Given a finite point set $V \subset \mathbb{R}^2$, can we lay down a good planar network connecting these points, that is also cheap to build?

Well, firstly, what is 'good'? We say a network is good if, for every pair of points $u, v \in V$, the length of the shortest path in the network where edges are weighted by the Euclidean distance between their endpoints is relatively short – it is at most a constant times the length of the straight line between $u$ and $v$. We refer to these good networks as 'spanners'.

And what about 'cheap'? A network can be cheap in many ways; in this work, we consider two senses in which a network is cheap:

(1) The network has a bounded degree, which is to say that each node has a constant degree.
(2) The network is light, which is to say the weight of the network is at most a constant times the weight of the minimum spanning tree of $V$.

A bounded degree network says that we do not need fancy and expensive nodes in our network; they do not need to support too many ports, channels, or roads. A light network says that the links of the network, in sum, are not so many times more expensive than the bare minimum required to achieve connectivity.

So now we ask again: Given a finite point set $V \subset \mathbb{R}^2$, can we lay down a good planar network connecting these points, that is also cheap to build? The answer, for quite some time, has been known to be in the affirmative. The first result of a bounded degree and light planar spanner was by Bose, Gudmundsson, and Smid in [**4**] with a degree bound of 27. Shortly thereafter, Li and Wang in [**10**] discovered such spanners with a degree bound of 24. There have since been numerous results on bounded degree and light planar spanners.

But what now, if we find ourselves in the situation where entities are to navigate these networks efficiently?

These entities are simple creatures; they do not have access to a map of the network, but, situated at any node $v$, they can see its neighbours $N(v)$ (called 1-locality) and perhaps a small amount of information associated with $v$; they have a limited memory which can only hold $O(1)$ information at once, including a source $s$ and a destination $t$; and, with these limitations, they make forwarding decisions deterministically, to arrive at $t$ when started at $s$. In the parlance, the algorithm these entities use to route the network is an $O(1)$-memory deterministic 1-local routing algorithm.

When we say 'efficiently', we mean that the paths found by these entities have a length not more than a constant $c$ times the straight line distance between its source and destination. We call $c$ the routing ratio of the online routing algorithm.

The central question now is then: Are there good planar networks that are cheap to build, and that can also be navigated efficiently without a map? To the best of our knowledge, this has heretofore been left unanswered. Some networks have been shown to be efficiently routable, such as the Delaunay Triangulation as shown in [**2**] and a Dynamic Geometric Spanner shown in [**8**], for example. The former construction, which we will take a closer look at in the next chapter, is planar but does not have constant degree and is heavy. The latter construction has only a constant degree. In this paper, we show that the answer to the central question is in the affirmative. Given a finite point set $V \subset \mathbb{R}^2$, we can indeed lay down a good planar network connecting these points, that is also cheap to build, and efficiently navigable without a map.

## 1.1 Our Contribution

Given a finite pointset $V \subset \mathbb{R}^2$ and two adjustable parameters $0 < \theta < \pi/2$ and $r > 0$, we show how to construct in $\mathrm{O}(n \lg n)$ time a planar $1.998(1 + 1/r)\max(\pi/2, \pi\sin(\theta/2) + 1)$-spanner with degree at most $5\lceil 2\pi/\theta \rceil$ (giving a least upper bound of 25), and weight at most $1.998(2r+1)\max(\pi/2, \pi\sin(\theta/2)+1)$ times the weight of a minimum spanning tree of $V$ (and if $\theta < \pi/3$, the weight is bounded by only the $2r + 1$ factor). This construction admits an $O(1)$-memory deterministic 1-local routing algorithm with a routing ratio of $5.90(1+1/r)\max(\pi/2, \pi\sin(\theta/2)+1)$ for the same adjustable parameters $\theta$ and $r$. To the best of our knowledge, online routing has not been done on bounded degree and light planar spanners.

We consider this contribution to be in three parts.

(1) We give a new, simple, and efficient construction for a $1.998\max(\pi/2, \pi\sin(\theta/2)+1)$-spanner with a $5\lceil 2\pi/\theta\rceil$ upper bound on its degree. This construction, we show, is very amenable towards an $O(1)$ memory marking scheme on the vertices which facilitates online routing.

(2) We unpack the algorithm presented in **[9]** by Levcopoulos and Lingas and examine some further properties thereof which are known but not explicitly mentioned in the paper. We use a slightly modified version of the algorithm to bound the weight of our bounded degree spanner, yielding a $1.998(1+1/r)\max(\pi/2, \pi\sin(\theta/2)+1)$-spanner with the same degree bound as before and a weight of at most $1.998(2r+1)\max(\pi/2, \pi\sin(\theta/2)+1)$ times that of a minimum spanning tree of $V$ (and if $\theta < \pi/3$, the weight is bounded by only the $2r+1$ factor).

(3) We present an $O(1)$-memory deterministic 1-local routing algorithm on the bounded degree and light spanner constructed that achieves a $5.90(1+1/r)\max(\pi/2, \pi\sin(\theta/2)+1)$ routing ratio.

## 1.2 Outline

In Chapter 2, we go through preliminary notions in more depth. These include laying down the general setting of things, stating key results in the field that are pertinent to the exposition of our result, giving examples of spanners and online routing algorithms, and, finally, covering some basic facts about Delaunay Triangulations $\mathcal{DT}(V)$ which we will be using. This chapter may be skipped if one is already familiar with all these terms, and in-the-know of important results around them.

In Chapter 3, we give a construction for $\mathcal{BDG}(V)$, a subset of the Delaunay Triangulation $\mathcal{DT}(V)$, that is a bounded degree planar spanner. We end on a sour note by showing a negative result: the simplest of online routing algorithms do not work on this construction!

In Chapter 4, we gear up towards designing a routing algorithm that uses subtler ideas. We make the observation that $\mathcal{BDG}(V)$ is the ruins of the Delaunay Triangulation $\mathcal{DT}(V)$, terrain we know to be efficiently navigable. By placing signposts on the nodes in $V$, we propound that a navigator in $\mathcal{BDG}(V)$ can recreate the Delaunay Triangulation $\mathcal{DT}(V)$ and thus efficiently navigate this network, which we call $\mathcal{MBDG}(V)$.

In Chapter $5$, we describe an efficient online routing algorithm on $\mathcal{MBDG}(V)$. We show how a navigator can interpret the signposts in $\mathcal{MBDG}(V)$, and act upon this interpretation to effectively simulate a path in the Delaunay Triangulation.

In Chapter $6$, we take a step back and show how to bound the weight of $\mathcal{MBDG}(V)$, to give the graph $\mathcal{LMBDG}(V)$ that satisfies our desiderata; it is a bounded degree and light planar spanner. With a slight modification to the online routing algorithm in Chapter $5$, we demonstrate an efficient online routing algorithm on $\mathcal{LMBDG}(V)$.

In Chapter $7$, we wrap up and suggest a few directions we can further take this work.

CHAPTER 2

# Preliminary Notions

---

The objects we work with are finite undirected graphs $G = (V, E)$ with a geometry. More specifically, we work with graphs whose vertices are points in the Euclidean Plane ($V \subset \mathbb{R}^2$), with a general position assumption on these points. That is to say, in any finite point set $V$, we assume that no three points are collinear and no four points are cocircular. The weight, or length, of an edge $uv$, is then the Euclidean distance between its endpoints $u$ and $v$. Finally, we use the terms 'vertex' and 'node' interchangeably to reference the elements of $V$.

With this setting now established, we give a brief on a few well-established concepts and key results that we use herein.

## 2.1 Spanners

For most of this section, we refer to the encyclopedic resource [**11**] by Narasimhan and Smid. For a survey and open problems on spanners, one may also consult [**6**].

Our main desideratum is to have a spanner. While we give the setting to be in $\mathbb{R}^2$, do note that the general definition of a spanner is in $\mathbb{R}^d$.

DEFINITION 1 (Spanner). *Let $V$ be a finite point set in $\mathbb{R}^2$. A $t$-spanner for $V$ is an undirected graph $G$ with vertex set $V$, such that for any pair of points $u, v$ in $V$, there is a path in $G$ from $u$ to $v$ whose length is at most $t$ times the Euclidean distance between $u$ and $v$. Any path satisfying this condition is referred to as a $t$-path from $u$ to $v$. A spanner is sparse if it has $O(n)$ edges.*

OBSERVATION 1. *Any planar spanner is a sparse spanner.*

In softer terms, a sparse spanner (which we from here on refer to plainly as a spanner) is a network that approximates the complete graph by $t$. The smallest number $t$, for a given graph $G$, is called the stretch factor or spanning ratio of $G$.

### 2.1.1 The $\Theta$-Graph

Let us now look at one classic example of a spanner, upon which we draw some inspiration from in our own construction.

Let $\kappa \geq 2$ be an integer, and $\theta = 2\pi/\kappa$, and let $V \subset \mathbb{R}^2$ be a finite point set. The graph $\Theta(V, \kappa)$, is a graph whose vertices are $V$ and whose edges are formed as follows:

For each point $v$ in $V$, partition the plane into $\kappa$ disjoint cones with a predetermined orientation and with angle measure $\theta$ and apex $v$. For each such cone $C$, let $l$ be an arbitrarily chosen ray emanating from $v$ and contained in $C$. Connect $v$ to the point in $C \cap V$ whose orthogonal projection onto $l$ is closest to $v$.

$\Theta(V, \kappa)$ clearly contains $\kappa n$ edges at the most and is thus sparse. Moreover, under some mild conditions, it is a spanner; if $\kappa \geq 9$, then $\Theta(V, \kappa)$ is a $1/(\cos\theta - \sin\theta)$-spanner [**11**]. However, $\Theta(V, \kappa)$ is neither planar, bounded degree, nor light. To see the latter two, consider points arranged in a star like that shown in Figure 2.1. $\kappa$ edges are determined from cones with apex at the centre vertex, but at each boundary vertex, an edge to the centre is also determined from a cone with apex at that boundary vertex.

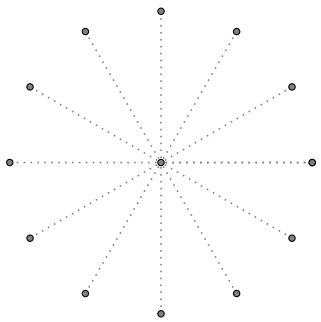


FIGURE 2.1: Points arranged in a star give high degree and heavy $\Theta$-Graphs.

The star graph will be embedded in $\Theta(V, \kappa)$, which has $O(n)$ degree and weight. The $\Theta$-Graph comes short, and we will have to look elsewhere to find the spanner we desire. But let us keep the idea in the periphery of out mind's eye; it will prove to come in handy.

### 2.1.2 Bounded Degree and Light Planar Spanners Exist

Bounded degree and light planar spanners have been shown to exist. The first two such constructions are detailed in [**4**] and [**10**] respectively, with the former having a degree bound of 27, and the latter having a least upper bound on the degree of 24 for some adjustable parameter.

## 2.2 Online Routing

For most of this section, we refer to [**5**] by Bose and Morin and [**3**] by Bonichon et al.

Algorithms where knowledge of the environment being routed on are not known beforehand are referred to as online routing algorithms. More specifically, when routing from $s$ to $t$ on a graph $G$, it is typical that an entity initially, situated at $s$, only knows $s$, $t$, and $N(s)$, where $N(\cdot)$ denotes the neighbourhood of $s$ in $G$. The entity then makes forwarding decisions using mainly $s$, $t$, and $N(v_i)$ where $v_i$ is the current node it is situated at, and sometimes, though less desirable, it is given more power such as the faculty to look at some auxiliary information associated to $v_i$. In contrast, if the whole of $G$ is known to the entity, routing from $s$ to $t$ would be a simple affair: just use any of the well-known shortest path algorithms, such as Dijkstra's Algorithm, and take the path it prescribes.

Online routing algorithms can be classified in numerous ways. We focus on the entity's use of lookahead, randomness, and memory.

(1) An online routing algorithm is said to be $k$-local if the forwarding decision at node $v_i$ is based on $s$, $t$, and the $k$-neighbourhood of $v_i$ in $G$, which is to say $N(v_i) \cup \cdots \cup N^k(v_i)$.

(2) An online routing algorithm is called randomised if random bits are required to make the forwarding decision of an entity at a node $v_i$; simply put, $v_{i+1}$ is chosen from $N(v_i)$ at random. Otherwise, an online routing algorithm is called deterministic.

(3) If the only information stored by the entity (in the message header), at any point in the online routing algorithm, is $s$ and $t$, we call it a memoryless algorithm. If, however, some amount $f(n)$ of additional information is stored, we call the algorithm an $O(f(n))$-memory algorithm.

The convention we use to interpret the output of $f(n)$ is in terms of words, instead of bits. For example, if at any node $v_i$, the entity needs to remember $v_{i-1}$ in order to make a forwarding decision, we say that this is an $O(1)$-memory algorithm. Had we interpreted the output in terms of bits, we would have called it an $O(\lg n)$-memory algorithm. We justify this by noting that

the message header needs to contain $s$ and $t$, which are each $\lg n$ bits of information. What we desire is that the message header not be so many times larger than what it minimally could be.

Moreover, a graph $G$ that is routed on can be classified by its use of additional information at each node. For example, the nodes can be coloured and have forwarding decisions of a navigator at each node depend partially on its colour. In general, the less information per node, the more attractive the routing algorithm is. In our main construction, we require $O(1)$ words per node.

Besides how online routing algorithms are categorised, we now also address the issue of their performance. For any online routing algorithm $\mathcal{A}$ and any graph $G$, we say that $G$ defeats $\mathcal{A}$ if there is a pair $s, t$ in $G$ such that $\mathcal{A}$ does not terminate when finding a path from $s$ to $t$. We say that $\mathcal{A}$ is $c$-competitive for $G$ if, for all pairs $s, t$ in $G$, the length of the path found by $\mathcal{A}$ from $s$ to $t$ is at most $c$ times the shortest path from $s$ to $t$ in $G$. If $\mathcal{A}$ is $c$-competitive for a spanner, then the paths found by $\mathcal{A}$ are bounded by a constant times the straight line between their endpoints. We will thus use the following notion of efficiency, which is more appropriate for geometric spanners.

DEFINITION 2 (Routing Ratio). *We say that $\mathcal{A}$ has a routing ratio of $c$ for $G$ if $c$ is the smallest number where all paths found by $\mathcal{A}$ from $s$ to $t$ are at most $c$ times the Euclidean distance between $s$ and $t$.*

Let us finally look at four online routing algorithms to help familiarise ourselves with these terms. We focus our attention on deterministic 1-local routing algorithms.

### 2.2.1 Routing in the $\Theta$-Graph

Routing from $s$ to $t$ in the $\Theta$-Graph is simple. Until $t$ has been reached, take the edge from the current vertex $v_i$ that is formed from the cone with apex $v_i$ that contains $t$. If $\kappa \geq 9$, this is a memoryless deterministic 1-local routing algorithm with a routing ratio of at most $1/(\cos\theta - \sin\theta)$. Vertices in the $\Theta$-Graph will need to store the orientation of cones and direction of rays in each cone, in order for the navigator to determine the correct edge to take at any point.

### 2.2.2 Greedy Routing

Greedy routing from $s$ to $t$, for any graph, is also simple. Until $t$ has been reached, take the edge from the current vertex $v_i$ to the vertex $v_{i+1}$ in $N(v_i)$ that minimises the distance $|v_{i+1}t|$. This is a memoryless deterministic 1-local routing algorithm. However, it is not generally competitive.

### 2.2.3 Compass Routing

Compass routing from $s$ to $t$, for any graph, is just as simple. Until $t$ has been reached, take the edge from the current vertex $v_i$ to the vertex $v_{i+1}$ in $N(v_i)$ that minimises the angle $\angle(t, v_i, v_{i+1})$. This is a memoryless deterministic 1-local routing algorithm. However, it is also not generally competitive.

### 2.2.4 Right-Hand Routing

Right-hand routing from $s$ to $t$ on convex subdivisions is only a little more complicated. It is based on the idea that a connected maze can be explored by placing one's right hand on the wall and never lifting it off while exploring the maze. For any convex subdivision $T$, let $T'$ be the planar subdivision obtained by removing all edges of $T$ intersecting the line segment $[st]$. $T'$ is connected by the convexity of $T$, and $s$ and $t$ are on the boundary of the same face in $T'$. Then, remembering the last edge a navigator has taken to arrive at $v_i$, right-hand routing can be implemented to walk along this face until $t$ is reached. This is an $O(1)$-memory deterministic 1-local routing algorithm. Moreover, no convex subdivision defeats right-hand routing. Unfortunately, it too is not generally competitive.

## 2.3 Delaunay Triangulations

Finally, we cover the basics of Delaunay Triangulations, an important piece we use to solve the puzzle at hand; not only is our construction a subgraph of the Delaunay Triangulation, we also simulate routing on the Delaunay Triangulation on our construction. First, we define a geometric primitive: the circumcircle.

DEFINITION 3 (Circumcircle). *The circumcircle of three points $u, v, w \in \mathbb{R}^2$ that are not collinear is the unique circle that passes through them. We denote it by $\odot(u, v, w)$.*

Having defined a circumcircle, we are ready to dive right into the definition of a Delaunay Triangulation.

DEFINITION 4 (Delaunay Triangulation). *For any finite point set $V \subset \mathbb{R}^2$, the Delaunay Triangulation of $V$, denoted $\mathcal{DT}(V)$, is the graph $(V, E \cup E')$ where $E = \{uv, vw, wu \in \binom{V}{2} \mid \forall t \in V \backslash \{u, v, w\}, t \notin \odot(u, v, w)\}$, and $E'$ are edges added to to the graph $(V, E)$ to give a maximal plane subdivision ($E'$ is empty when $V$ is in general position, which is to say no four points are cocircular).*

Note that the Delaunay Triangulation is planar (the definition given above already presupposes so). The Delaunay Triangulation also contains a minimum spanning tree of $V$ (see [**1**]). Moreover, the Delaunay Triangulation is a spanner. The following theorem is stated as the main result of [**12**].

THEOREM 1. *The stretch factor of the Delaunay Triangulation is at most* $1.998$.

Unfortunately, the Delaunay Triangulation does not have a bounded degree nor is it light, for the same example to show that a $\Theta$-Graph does not have either property suffices to show the same for Delaunay Triangulations (see Figure 2.1).

Simple routing algorithms such as greedy routing and compass routing are not defeated by the Delaunay Triangulation. While neither greedy nor compass have a constant routing ratio on the Delaunay Triangulation, as shown in [**5**], there are (non-trivial) memoryless deterministic 1-local routing algorithms on Delaunay Triangulations with a constant routing ratio. [**2**] shows such an algorithm.

THEOREM 2. *There is a memoryless deterministic* $1$*-local routing algorithm on the Delaunay Triangulation with a routing ratio of no more than* $3.56$.

We can say more: a theorem in [**3**] gives a lower bound to the routing ratio of deterministic $k$-local routing algorithms on Delaunay Triangulations.

THEOREM 3. *There exists no deterministic* $k$*-local routing algorithm on Delaunay Triangulations with routing ratio at most* $1.70$.

Finally, let us go through some properties and terms we use pertaining to Delaunay Triangulations.

DEFINITION 5 (Delaunay Triangle). *For any Delaunay Triangulation* $\mathcal{DT}(V)$*, we say that, for* $u, v, w \in V$*,* $\Delta u, v, w$ *is a Delaunay Triangle if* $\Delta u, v, w$ *is a face of* $\mathcal{DT}(V)$*. We extend this so say that for any finite point set* $V \subset \mathbb{R}^2$*,* $\Delta u, v, w$ *is a Delaunay Triangle if it is one in* $\mathcal{DT}(V)$.

DEFINITION 6 (Delaunay Neighbour). *For any Delaunay Triangulation* $\mathcal{DT}(V)$*, we say that the Delaunay Neighbours of* $u$ *are all vertices adjacent to* $u$*. We extend this to say that for any finite point set* $V \subset \mathbb{R}^2$*, and for* $u \in V$*, the Delaunay Neighbours of* $u$ *are all the vertices that are adjacent to* $u$ *in* $\mathcal{DT}(V)$.

DEFINITION 7 (Hull). *For any Delaunay Triangulation* $\mathcal{DT}(V)$*, we say that the hull of* $u$ *is the cycle induced by the Delaunay neighbours of* $u$*. We extend this to say that for any finite point set* $V \subset \mathbb{R}^2$*, for* $u \in V$*, the hull of* $u$ *is that with* $\mathcal{DT}(V)$ *considered. See Figure 2.2.*

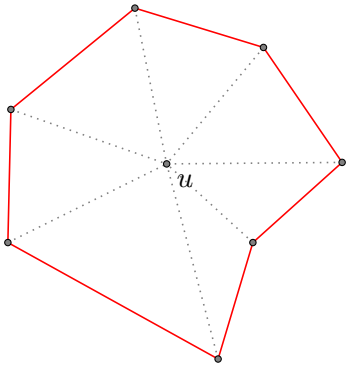



FIGURE 2.2: The hull of $u$ coloured in red.

The following two are crucial properties that we will use.

LEMMA 1 (Empty Circle Property). *For any Delaunay Triangulation $\mathcal{DT}(V)$, and for any of its Delaunay Triangles $\Delta u, v, w$, no point in $V$ is contained in the interior of the circumcircle $\odot(u, v, w)$.*

PROOF. This follows from the definition of the Delaunay Triangulation. □

LEMMA 2. *For any Delaunay Triangulation $\mathcal{DT}(V)$, let $\Delta t, u, v$ and $\Delta u, v, w$ be a pair of Delaunay Triangles. If the quadrilateral $\Delta t, u, v, w$ is in convex position, then $u \in \odot(t, v, w)$ (see Figure 2.3).*

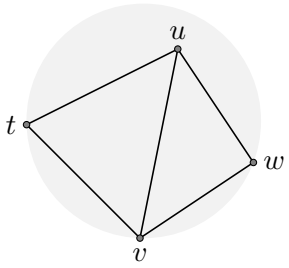



FIGURE 2.3: $u$ is in $\odot(t, v, w)$.

PROOF. Suppose for a contradiction, that $u \notin \odot(t, v, w)$. Then, since opposite angles of a cyclic quadrilateral sum to $\pi$, $\angle(t, u, w) + \angle(t, v, w) < \pi$. This implies that $\angle(u, t, v) + \angle(u, w, v) > \pi$ and thus $w \in \odot(t, u, v)$, which, by the Empty Circle Property (Lemma 1), is inconsistent with $\Delta t, u, v$ being a triangle.

We conclude that therefore $u \in \odot(t, v, w)$. □

CHAPTER 3

# Everything But Lightness

---

Recall our ultimate goal: we want to give an answer to the question: Are there good planar networks that are cheap to build, and that can also be navigated efficiently without a map? Before we can think about navigation of any kind, with or without a map, we first need a spanner to navigate on.

In this chapter, we aim to show a construction of a network that is cheap in the first sense we sought: A palanar spanner network with a bounded degree. We call such a graph a Bounded Degree Graph on $V$, or $\mathcal{BDG}(V)$ for short.

For clarity of exposition, we withhold bounding the weight (our second desideratum) until Chapter 6. Without any more ado, let us begin!

## 3.1 Description of $\mathcal{BDG}(V)$

Given a finite point set $V \subset \mathbb{R}^2$, $\mathcal{BDG}(V)$ is a subgraph of the Delaunay Triangulation $\mathcal{DT}(V)$. The idea behind the construction is slightly reminiscent to that of the $\Theta$-Graph:

For an adjustable parameter $0 < \theta < \pi/2$, let $\kappa = \lceil 2\pi/\theta \rceil$, and let $\mathcal{C}_{u,\kappa}$ be a set of $\kappa$ disjoint cones partitioning the plane, with each cone having angle measure at most $\theta$ at apex $u$.

Let $v_0, \ldots, v_m$ be the clockwise-ordered Delaunay Neighbours of $u$ within some cone $C \in \mathcal{C}_{u,\kappa}$ (see Figure 3.1).

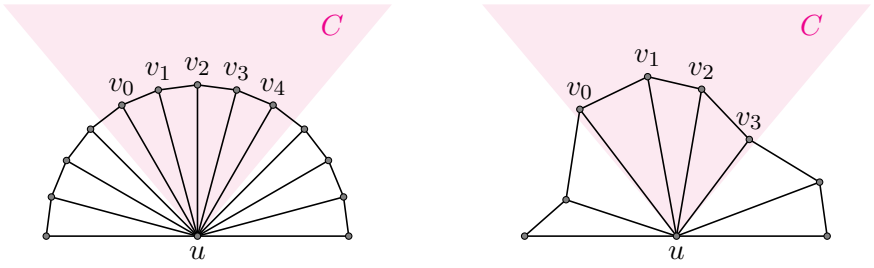



FIGURE 3.1: Two examples of the vertices in some cone $C$ with apex $u$. On the left, $m = 4$, and on the right, $m = 3$.

Call edges $uv_0$ and $uv_m$ *extreme* at $u$. Call edges $uv_1$ and $uv_{m-1}$ *penultimate* at $u$ if there are two distinct extreme edges at $u$ induced by $C$. If there are two distinct edges that are extreme at $u$ induced by $C$ and two distinct edges that are penultimate at $u$ induced by $C$, then, of the remaining edges incident to $u$ and contained in $C$, the one that is the shortest is called a *middle* edge at $u$. We emphasise that:

(1) If there are fewer than three neighbours of $u$ in the cone $C$, then there are no penultimate edges induced by $C$.
(2) If there are fewer than five neighbours of $u$ in the cone $C$, then there is no middle edge induced by $C$.

Figure 3.2 shows the classification of edges when there are few neighbours of $u$ in the cone $C$.

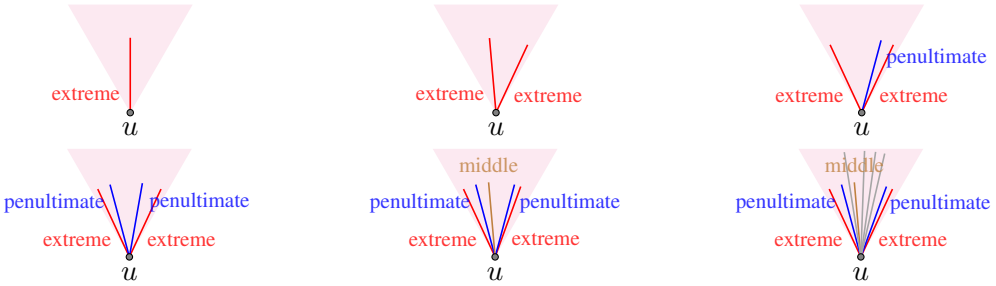



FIGURE 3.2: Extreme, penultimate, and middle are mutually exclusive properties taking precedence in that order. Red, blue, and brown edges are extreme, penultimate, middle edges of the Delaunay Triangulation, respectively. Gray unlabeled edges are edges which are neither extreme, penultimate, nor middle, and can only occur in a cone with more than five neighbours (bottom right).

The construction removes every edge except for the extreme, penultimate, and middle ones in every $C \in \mathcal{C}_{u,\kappa}$, for every point $u$, in any order. The edges present in this construction are thus the ones which are either extreme, penultimate, or middle at both of their endpoints (not necessarily the same at each endpoint). In Section 3.4, we provide the running time of an efficient construction.

The resulting graph is denoted by $\mathcal{BDG}(V)$. We now work towards proving that $\mathcal{BDG}(V)$ has a constant degree (Theorem 4) and spanning ratio (Corollary 1).

## 3.2 Degree Bound of $\mathcal{BDG}(V)$

We can now upper bound the degree of $\mathcal{BDG}(V)$.

THEOREM 4. *The maximum degree of $\mathcal{BDG}(V)$ is at most $5\kappa$, where $\kappa = \lceil 2\pi/\theta \rceil$ for an adjustable parameter $0 < \theta < \pi/2$.*

PROOF. Each of the $\kappa$ cones $C \in \mathcal{C}_{u,\kappa}$ can induce at most two extreme edges, two penultimate edges, and one middle edge at $u$. There are thus at most $2\kappa$ extreme edges, $2\kappa$ penultimate edges, and $\kappa$ middle edges at $u$. Adding them up, we get the degree bound of $5\kappa$, where $\kappa = \lceil 2\pi/\theta \rceil$ for the adjustable parameter $0 < \theta < \pi/2$. □

## 3.3 Spanning Properties of $\mathcal{BDG}(V)$

The network has a bounded degree, but does it satisfy the principal requirement that it is a spanner? The answer is in the positive, but first, we need a few lemmas and definitions to help us make such an assertion.

### 3.3.1 A Separation of Neighbours of Neighbours

LEMMA 3. *Let $C$ be a cone with apex $u$ and angle measure $0 < \theta < \pi/2$. Let $v_l, v, v_r$ be consecutive clockwise-ordered Delaunay Neighbours of $u$ contained in $C$. The interior angle $\angle(v_l, v, v_r)$ must be at least $\pi - \theta$.*

PROOF. There are two cases:

(1) When $\angle(v_l, v, v_r)$ is reflex in the quadrilateral $\Delta u, v_l, v, v_r$.

(2) When $\angle(v_l, v, v_r)$ is not, in which case the quadrilateral $\Delta u, v_l, v, v_r$ is convex.

The result follows trivially for the first case. Let us thus just examine the second case.

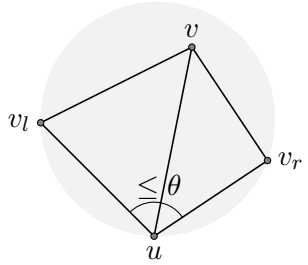



FIGURE 3.3: Example placement of $u, v_l, v, v_r$ in in the circle $\odot(v_l, u, v_r)$.

Since $v_l$ and $v_r$ lie in a cone of angle measure $\theta$, $\angle(v_l, u, v_r)$ is at most $\theta$. By Lemma 2, $\angle(v_l, u, v_r) + \angle(v_l, v, v_r)$ must be at least $\pi$ (see Figure 3.3). Hence, $\angle(v_l, v, v_r)$ is at least $\pi - \theta$. □

This essentially means that $\angle(v_l, v, v_r)$ is wide, and will help us to say more about the cone with apex $v$ that $vu$ lies in; can $v_l$ or $v_r$ be in such a cone? Can both $v_l$ and $v_r$ be in such a cone? We answer these presently.

### 3.3.2 Notion of Protection

DEFINITION 8. *An edge $uv$ is* protected *at $u$ (with respect to some fixed $\mathcal{C}_{u,\kappa}$) if it is extreme, penultimate, or middle at $u$. An edge $uv$ is* fully protected *if it is protected at both $u$ and $v$.*

In view of the above definition, it is clear that an edge is contained in the $\mathcal{BDG}(V)$ if and only if it is fully protected.

### 3.3.3 Lesser Protection Buys Neighbours in the Same Cone

The following observation follows immediately from the defintions of extreme and penultimate edges.

OBSERVATION 2. *If an edge $uv_i$ is not extreme at $u$, then $u$ must have consecutive clockwise-ordered Delaunay Neighbours $v_{i-1}, v_i, v_{i+1}$, all in the same cone $C \in \mathcal{C}_{u,\kappa}$ (see Figure 3.4 (left)). Similarly, if*

*$uv_i$ is neither extreme nor penultimate at $u$, then $u$ must have consecutive clockwise-ordered Delaunay Neighbours $v_{i-2}, v_{i-1}, v_i, v_{i+1}, v_{i+2}$, all in the same cone $C \in \mathcal{C}_{u,\kappa}$ (see Figure 3.4 (right)).*

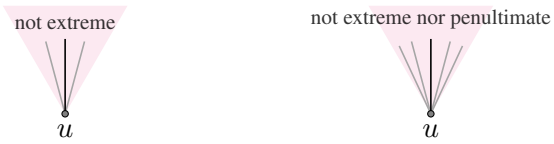



FIGURE 3.4: Observation 2, pictorially.

### 3.3.4 Penultimate and Middle Edges are Bought Full Protection

LEMMA 4. *Every edge that is penultimate or middle at one of its endpoints is fully protected.*

PROOF. Consider an edge $uv$ that is penultimate or middle at $u$. It is obviously protected at $u$. We are thus left with showing that it is protected at $v$. Since $uv$ is not extreme at $u$, $u$ must have consecutive clockwise-ordered Delaunay Neighbours $v_l, v, v_r$ in the same cone by Observation 2.

We show that $uv$ must be extreme at $v$. Suppose for a contradiction that $uv$ is not extreme at $v$. Then, by Observation 2, $v_l v$ and $v v_r$ are contained in the same cone with apex $v$ and angle at most $\theta < \pi/2$. By Lemma 3, $\angle(v_l, v, v_r) \geq \pi - \theta > \theta$ since $\theta < \pi/2$, which is impossible. Thus, $uv$ must be extreme at $v$ and therefore protected at $v$. The edge is fully protected. □

### 3.3.5 Hull Paths Within Cones are Almost Complete

LEMMA 5. *Let $v_0, \ldots, v_{m+1}$ be the clockwise-ordered Delaunay Neighbours of $u$ contained in some cone $C \in \mathcal{C}_{u,\kappa}$. The edges in the path $v_1, \ldots, v_m$ are all fully protected (see Figure 3.5).*

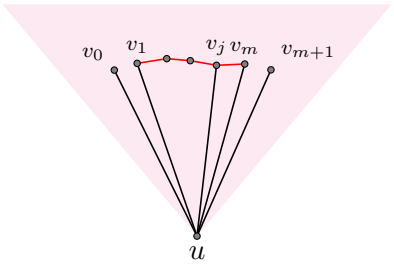



FIGURE 3.5: The red path from $v_1$ to $v_m$ is assuredly in $\mathcal{BDG}(V)$. $uv_0$ and $uv_{m+1}$ are extreme, $uv_1$ and $uv_m$ penultimate, and $uv_j$ middle.

PROOF. Let $v_iv_{i+1}$ be an edge along this path. Suppose for a contradiction that $v_iv_{i+1}$ is not protected at $v_i$. It is thus neither extreme nor penultimate at $v_i$. Then, by Observation 2, $v_iu$ and $v_iv_{i-1}$ must be contained in the same cone with apex $v_i$ as $v_iv_{i+1}$. By Lemma 3, $\angle(v_{i-1}, v_i, v_{i+1}) \geq \pi - \theta > \theta$, contradicting that $v_{i-1}$, $v_i$, and $v_{i+1}$ lie in the same cone. Such an edge must therefore be either extreme or penultimate, and thus protected, at $v_{i \geq 1}$. An analogous argument shows that the edge is either extreme or penultimate at $v_{i+1 \leq m}$. It is thus fully protected. □

This says that such paths $v_1, \ldots, v_m$ are included in $\mathcal{BDG}(V)$, and taking such paths proves useful in finding spanning paths.

### 3.3.6 Proof of the Spanning Ratio

We now show that $\mathcal{BDG}(V)$ is a spanner. This proof is heavily based on the proof of Theorem 3 in [**10**] by Li and Wang, modified slightly to suit our construction.

THEOREM 5. *$\mathcal{BDG}(V)$ is a $\max(\pi/2, \pi\sin(\theta/2) + 1)$-spanner of the Delaunay Triangulation $\mathcal{DT}(V)$ for an adjustable parameter $0 < \theta < \pi/2$.*

PROOF. We show that for any edge $uv$ in $\mathcal{DT}(V)$ that is not present in $\mathcal{BDG}(V)$, there is a spanning path in $\mathcal{BDG}(V)$ from $u$ to $v$. Refer to Figure 3.6 to supplement the following proof.

The edges in $\mathcal{BDG}(V)$ are exactly the edges in $\mathcal{DT}(V)$ that are fully protected. Accordingly and without loss of generality, let $uv$ be an edge in $\mathcal{DT}(V)$ that is not protected at $u$. Then, $uv$ is a chord of the face $u, v_0, \ldots, v_i = v, \ldots, v_m$ where $uv_0$ is a middle edge, $uv_m$ is a penultimate edge, and all edges on the boundary are included in the construction (Lemma 4 and Lemma 5). Moreover, $\angle(v_0, u, v_i) <$

$\angle(v_0, u, v_m) < \theta < \pi/2$. Consider the geodesic $S(v_0, v_i)$ from $v_0$ to $v_i$ contained in the polygon $u, v_0, \ldots, v_i = v$. Label $|uv_0|$ with $x$, $|uv_i|$ with $y$, and let $w$ be the point on the segment $uv_i$ with length $x$ so that $|wv_i| = y - x$.

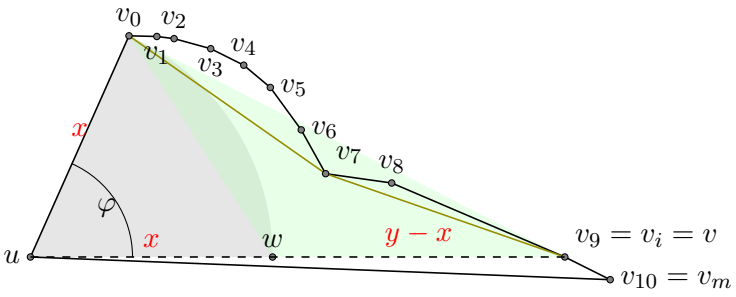



FIGURE 3.6: Details about the spanning path from $u$ to $v$ (in this example $v = v_9$).

We will show that $S(v_0, v_i)$ is contained in the triangle $\Delta v_0, w, v_i$. If none of $v_1 \ldots v_{i-1}$ are contained in the triangle, the claim must hold since all such vertices must be additionally outside the circle with centre $u$ and radius $x$ ($uv_0$ is the middle edge) and thus the line segment joining $v_0$ and $v_i$ is unobstructed. If there are some vertices in the triangle, then $v_0$ must connect directly to one of them along the geodesic, say $s$, and $v_i$ must connect directly to one of them, say $t$ possibly the same as $s$. $S(v_0, v_i)$ can be seen as the lower convex hull of $v_0 \ldots v_i$. Therefore, since $s$ and $t$ are in the triangle, the subpath connecting them in $S(v_0, v_i)$ must be too.

$S(v_0, v_i)$ is convex with base $v_0 v_i$ and contained in the triangle $\Delta v_0, w, v_i$; it must thus have a length not more than $|v_0 w| + |w v_i| = 2x \sin(\varphi/2) + y - x$ where $\varphi < \theta < \pi/2$ is the angle $\angle(v_0, u, v_i)$. Now consider an edge of $S(v_0, v_i)$, say $st$. The edge $st$ shortcuts the subpath $D(s, t)$ of $v_0, \ldots, v_i$. By Lemma 6 shown in **[4]** by Bose, Gudmundsson, and Smid, $|D(s, t)| \leq |st|\, \pi/2$ if the two conditions hold:

(1) $\angle(s, u, t) < \pi/2$, and
(2) $D(s, t)$ lies on one side of $st$.

Since both conditions hold in our case, $|D(v_0, v_i)| \leq |S(v_0, v_i)|\,\pi/2 \leq (|v_0w|+|wv_i|)\pi/2 = (2x\sin(\varphi/2)+y-x)\pi/2$. Putting everything together, we have that the path $u, v_0, \ldots, v_m$ has length at most

$$\begin{aligned} & x + (2x\sin(\varphi/2) + y - x)\pi/2 \\ = \; & y(\pi/2 + (\pi\sin(\varphi/2) + 1 - \pi/2)x/y) \\ \leq \; & y(\pi/2 + (\pi\sin(\theta/2) + 1 - \pi/2)x/y) \\ \leq \; & y\max(\pi/2, \pi\sin(\theta/2) + 1) \\ = \; & |uv|\max(\pi/2, \pi\sin(\theta/2) + 1). \end{aligned}$$

The last inequality can be shown to hold by considering the cases where $\pi\sin(\theta/2) + 1 \leq \pi/2$ and when $\pi\sin(\theta/2) + 1 > \pi/2$.

The right-hand-most side of the inequality shows that for any edge $uv$ in $\mathcal{DT}(V)$, there is a $\max(\pi/2, \pi\sin(\theta/2)+1)$-spanning path in $\mathcal{BDG}(V)$ between $u$ and $v$. $\mathcal{BDG}(V)$ is thus a $\max(\pi/2, \pi\sin(\theta/2) + 1)$-spanner of the Delaunay Triangulation $\mathcal{DT}(V)$ for an adjustable parameter $0 < \theta < \pi/2$. □

COROLLARY 1. *$\mathcal{BDG}(V)$ is a $1.998\max(\pi/2, \pi\sin(\theta/2) + 1)$-spanner for an adjustable parameter $0 < \theta < \pi/2$.*

PROOF. Using Theorem 5 that $\mathcal{BDG}(V)$ is a $\max(\pi/2, \pi\sin(\theta/2) + 1)$-spanner of the Delaunay Triangulation $\mathcal{DT}(V)$ for an adjustable parameter $0 < \theta < \pi/2$, and the main result in [**12**] that $\mathcal{DT}(V)$ is a $1.998$-spanner, the result follows. □

COROLLARY 2. *$\mathcal{BDG}(V)$ is a bounded degree planar $1.998\max(\pi/2, \pi\sin(\theta/2) + 1)$-spanner for an adjustable parameter $0 < \theta < \pi/2$.*

PROOF. Planarity follows from $\mathcal{BDG}(V)$ being a subset of the Delaunay Triangulation $\mathcal{DT}(V)$, a planar graph. The rest follows from Theorem 4 and Corollary 1. □

## 3.4 Algorithmic Construction of $\mathcal{BDG}(V)$

Finally, we state the construction of $\mathcal{BDG}(V)$ algorithmically and analyse its time complexity.

**Algorithm 1** $\mathcal{BDG}(V)$

**Require:** $V$
**Require:** $0 < \theta < \pi/2$
1: $E \leftarrow \{\}$
2: $\mathcal{DT} \leftarrow \mathcal{DT}(V)$
3: **for** $u \in V$ **do**
4: Compute $\mathcal{C}_{u,\kappa}$, where $\kappa = \lceil 2\pi/\theta \rceil$.
5: **for** $u \in V$ **do**
6: **for** $uv \in E(\mathcal{DT})$ **do**
7: Bucket $uv$ into $C \in \mathcal{C}_{u,\kappa}$.
8: **for** $u \in V$ **do**
9: **for** $C \in \mathcal{C}_{u,\kappa}$ **do**
10: Reset values of $e_1, e_2, p_1, p_2, m$.
11: **for** $e$ bucketed into $C$ **do**
12: $e_1 \leftarrow \arg\min_{angle}(e_1, e)$
13: $e_2 \leftarrow \arg\max_{angle}(e_2, e)$
14: **for** $e$ bucketed into $C \backslash \{e_1, e_2\}$ **do**
15: $p_1 \leftarrow \arg\min_{angle}(p_1, e)$
16: $p_2 \leftarrow \arg\max_{angle}(p_2, e)$
17: **for** $e$ bucketed into $C \backslash \{e_1, e_2, p_1, p_2\}$ **do**
18: $m \leftarrow \arg\min_{length}(m, e)$
19: Mark $e_1, e_2, p_1, p_2, m$, if their values are set, as protected by $u$.
20: **for** $uv \in E(\mathcal{DT})$ **do**
21: **if** $uv$ marked as protected by both endpoints **then**
22: $E = E \cup \{uv\}$.
**return** $(V, E)$.

THEOREM 6. *$\mathcal{BDG}(V)$ takes $O(n \lg n)$ time to construct. $\mathcal{BDG}(V)$ takes $O(n)$ time to construct if the input is a Delaunay Triangulation $\mathcal{DT}(V)$ on $V$.*

PROOF. The construction of the Delaunay Triangulation $\mathcal{DT}(V)$ at line `2` takes $O(n \lg n)$ time.

The loops at lines `3`, `5`, `8`, `20` are independent of each other. That starting at line `3` takes $O(n)$ time. `5` takes $O(n)$ time since there are a linear number of edges in $E(\mathcal{DT})$, which we look at twice (once for each endpoint), and the bucketing of each edge takes $\kappa$ time at most. The loop starting at line `8` takes $O(n)$ time since there are a linear number of edges in $E(\mathcal{DT})$, which we look at six times at most (thrice

for each endpoint). Finally, the loop at line `20` takes $O(n)$ time since there are a linear number of edges in $E(\mathcal{DT})$.

The result follows that $\mathcal{BDG}(V)$ takes $O(n \lg n)$ time to construct and $\mathcal{BDG}(V)$ takes $O(n)$ time to construct if the input is a Delaunay Triangulation $\mathcal{DT}(V)$ on $V$. □

## 3.5 $\mathcal{BDG}(V)$ Defeats Both Greedy and Compass Routing

Having done all this work, it is unfortunate to know that both Greedy Routing and Compass Routing are defeated by $\mathcal{BDG}(V)$. This is in constrast to the Delaunay Triangulation by which, albeit not competitive, neither of the algorithms are defeated. Figure 3.7 shows a portion of $\mathcal{BDG}(V)$, completely specified within the shaded region, that, when routing from $s$ to $t$, will result in both Greedy and Compass Routing dithering between $s$ and $p$, forever.

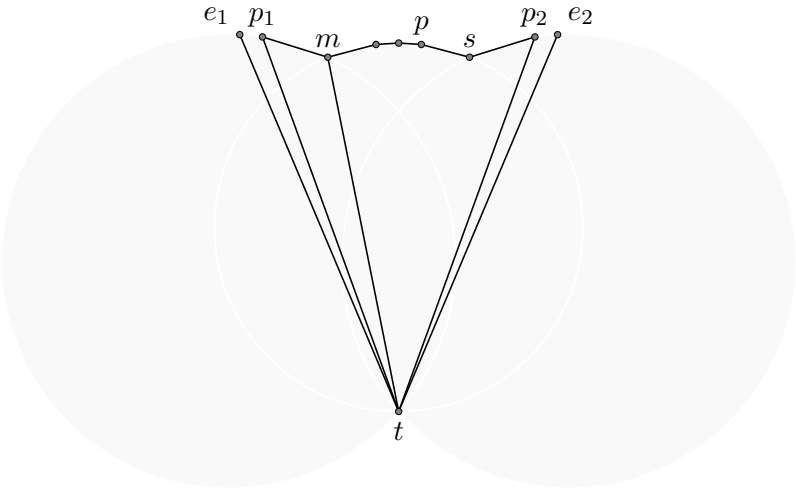



FIGURE 3.7: Greedy and Compass Routing dithers between $s$ and $p$. $te_1$ and $te_2$ are extreme, $tp_1$ and $tp_2$ penultimate, and $tm$ middle.

Have we made things worse? Is $\mathcal{BDG}(V)$ any good an idea?

CHAPTER 4

# Marking the Terrain

---

We have just seen that Greedy and Compass Routing algorithms are defeated by $\mathcal{BDG}(V)$. However, we have yet to lose all hope. Fundamentally, these algorithms only use the geometry of the network. Is the geometry alone enough to navigate such a network as $\mathcal{BDG}(V)$ without the aid of a map? Perhaps so, but what if, for now, we make things easier and mark the nodes with small signposts? If we can indirectly store the Delaunay Triangulation within these signposts, will this avail to the navigator a routing algorithm that is competitive on the Delaunay Triangulation that can then be adapted to routing on $\mathcal{BDG}(V)$?

In what follows, we discuss how to divide the information content of a Delaunay Triangulation among the nodes in $\mathcal{BDG}(V)$ such that each node stores $O(1)$ information.

## 4.1 Everything is Protected

We first state a lemma that allows us to divide the information content as we have just alluded to.

LEMMA 6. *For every edge $uv \in \mathcal{DT}(V)$, $uv$ is protected by at least one of its endpoints $u$ or $v$.*

PROOF. Suppose without loss of generality that $uv$ is not protected at $u$. Then $uv$ is not extreme at $u$ and thus by Observation 2, $u$ must have consecutive clockwise-ordered Delaunay Neighbours $v_l, v, v_r$. By Lemma 3, $\angle(v_l, v, v_r) \geq \pi - \theta > \theta$ since $0 < \theta < \pi/2$, and thus $v_l$ and $v_r$ cannot both belong to the same cone with apex $v$ and of angle measure at most $\theta$. Since $v_r, u, v_l$ are consecutive clockwise-ordered Delaunay Neighbours of $v$, and $vv_l$ and $vv_r$ cannot be in the same cone, it then follows that $vu$ must be extreme at $v$. $uv$ is therefore protected at $v$ when it is not at $u$, which shows that for every edge $uv \in \mathcal{DT}(V)$, $uv$ is protected by at least one of its endpoints $u$ or $v$. □

## 4.2 Signposts

So what happens if, at each node, we store the information of all its extreme edges? The edges this information corresponds to may not be contained in $\mathcal{BDG}(V)$, but nevertheless, in view of Lemma 6, all edges from $\mathcal{DT}(V)$ can be recovered from $\mathcal{BDG}(V)$ once this information has been added to the graph; if an edge is penultimate or middle, it is an edge of $\mathcal{BDG}(V)$ by Lemma 4, and otherwise, it will be extreme for at least one of its endpoints and thus stored as a *ghost-edge* at that endpoint.

Let us take this idea a little further. Consider as a thought experiment the issue of routing, using the edges of $\mathcal{BDG}(V)$, from some vertex $u$ to somewhere on its Delaunay Neighbourhood, say $v$, where $uv$ is in $\mathcal{DT}(V)$, and $uv$ is protected at $v$. If $uv$ is protected at $u$, we can then take the direct edge $uv$ from $u$ to $v$. If, on the other hand, $uv$ is not protected at $u$, then a spanning path from $u$ to $v$ can be extracted as an artefact from the proof of Theorem 5: find the middle edge and penultimate edge $uv$ lies between, and walk along the face towards $v$ from the middle edge (see Figure 4.1).

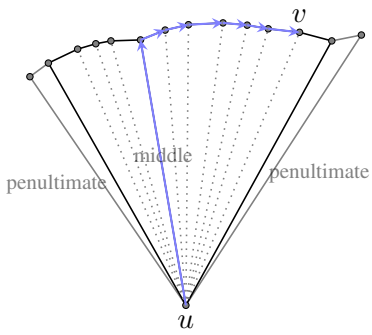



FIGURE 4.1: Spanning path from $u$ to $v$.

The catch here is we ought to have known which edge was the middle edge (really, we could just take the shorter of the two edges, but let us just play along). So we might as well store two bits at each edge of $\mathcal{BDG}(V)$ at each endpoint, indicating whether it is extreme, penultimate, or middle at that endpoint. Having fixed that, what if we flip the question slightly? Let us say that ceteris paribus we were at $v$ and wanted to find a way to $u$. We can route along the same face; like before, we know there is a spanning path along this face from Theorem 5. But which direction along this face should we take?

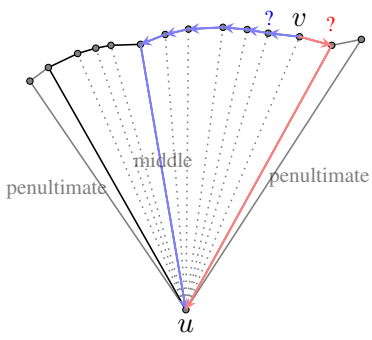



FIGURE 4.2: When routing from $v$ to $u$, how can we know which way to take to give a spanning path?

If we had a map, this would be easy; but without a map, routing backwards from an unprotected Delaunay Neighbour to a node is not as straightforward as routing from the node to its unprotected Delaunay Neighbour (see Figure 4.2). As a reminder, we are considering the case that $uv$ is protected at $v$ but not at $u$. This means that $uv$ is extreme at $v$ (for if it were penultimate or middle, Lemma 4 tells us that it would be protected at $u$). We thus have $uv$ stored as a ghost-edge at $v$ in spite of it not being an edge of $\mathcal{BDG}(V)$. What if, at this signpost, we could point in the direction to move towards if routing to $u$? We can! We just place a bit at this ghost-edge: 1 to take the neighbour of $v$ clockwise to $vu$, and 0 to take the neighbour of $v$ counterclockwise to $vu$. Storing these bits at each vertex, for each of their extreme edges, we can then use them to navigate to $u$ (see Figure 4.3).

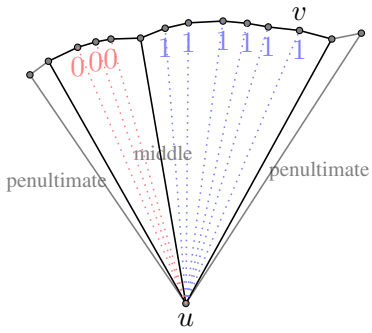



FIGURE 4.3: Bits at non-graphical edges indicating whether to take the clockwise or counterclockwise edge to it to get a spanning path from $v$ to $u$.

To recapitulate, at each node $u$, we store:

(1) Edges $uv$ (which are the fully protected edges), with two additional bits to denote whether it is extreme, penultimate, or middle at $u$.

(2) Ghost-edges $uv$ (which will be extreme at $u$), with one additional bit to denote whether to take the edge $uv_r$ clockwise to $uv$, or $uv_l$ counterclockwise to $uv$, as the first edge in a spanning path from $v$ to $u$.

When we augment $\mathcal{BDG}(V)$ with this information, we will denote such a graph as a Marked Bounded Degree Graph or $\mathcal{MBDG}(V)$ for short.

THEOREM 7. *$\mathcal{MBDG}(V)$ stores $O(1)$ information at each of its nodes.*

PROOF. There are at most $5\kappa$ edges, each with two additional bits, and $2\kappa$ ghost-edges, each with one additional bit, stored at each node, where $\kappa$ is a fixed constant. □

THEOREM 8. *$\mathcal{MBDG}(V)$ has a degree of at most $5\lceil 2\pi/\theta \rceil$, and is a planar $1.998\max(\pi/2, \pi\sin(\theta/2)+1)$-spanner for an adjustable parameter $0 < \theta < \pi/2$.*

PROOF. This follows trivially from Corollary 2 since the edges of $\mathcal{MBDG}(V)$ are the same as those of $\mathcal{BDG}(V)$. □

## 4.3 Algorithmic Construction of $\mathcal{MBDG}(V)$

We now state a modification to the construction of $\mathcal{BDG}(V)$ to construction of $\mathcal{MBDG}(V)$. Added lines to the former construction have been inserted at lines `20,21,22` and coloured red.

**Algorithm 2** $\mathcal{BDG}(V)$

**Require:** $V$
**Require:** $0 < \theta < \pi/2$
1: $E \leftarrow \{\}$
2: $\mathcal{DT} \leftarrow \mathcal{DT}(V)$
3: **for** $u \in V$ **do**
4: Compute $\mathcal{C}_{u,\kappa}$, where $\kappa = \lceil 2\pi/\theta \rceil$.
5: **for** $u \in V$ **do**
6: **for** $uv \in E(\mathcal{DT})$ **do**
7: Bucket $uv$ into $C \in \mathcal{C}_{u,\kappa}$.
8: **for** $u \in V$ **do**
9: **for** $C \in \mathcal{C}_{u,\kappa}$ **do**
10: Reset values of $e_1, e_2, p_1, p_2, m$.
11: **for** $e$ bucketed into $C$ **do**
12: $e_1 \leftarrow \arg\min_{angle}(e_1, e)$
13: $e_2 \leftarrow \arg\max_{angle}(e_2, e)$
14: **for** $e$ bucketed into $C \backslash \{e_1, e_2\}$ **do**
15: $p_1 \leftarrow \arg\min_{angle}(p_1, e)$
16: $p_2 \leftarrow \arg\max_{angle}(p_2, e)$
17: **for** $e$ bucketed into $C \backslash \{e_1, e_2, p_1, p_2\}$ **do**
18: $m \leftarrow \arg\min_{length}(m, e)$
19: Mark $e_1, e_2, p_1, p_2, m$, if their values are set, as protected by $u$.
20: **for** $uv$ bucketed into $C \backslash \{e_1, e_2, p_1, p_2, m\}$ **do**
21: Store $uv$ at $v$ as a ghost-edge, marked with 1 if it's to the right of $m$, and 0 otherwise.
22: Mark $e_1, e_2, p_1, p_2, m$ as extreme, penultimate, or middle at $u$, if their values are set.
23: **for** $uv \in E(\mathcal{DT})$ **do**
24: **if** $uv$ marked as protected by both endpoints **then**
25: $E = E \cup \{uv\}$.
**return** $(V, E)$.

THEOREM 9. *$\mathcal{MBDG}(V)$ takes $O(n \lg n)$ time to construct. $\mathcal{MBDG}(V)$ takes $O(n)$ time to construct if the input is a Delaunay Triangulation $\mathcal{DT}(V)$ on $V$.*

PROOF. The running time of the loop at line `8` remains unchanged; it is $O(n)$ since there are a linear number of edges, each looked at at most eight times (four times per endpoint). For the same reasons that justify the construction time of $\mathcal{BDG}(V)$, we can then conclude that $\mathcal{MBDG}(V)$ takes $O(n \lg n)$

time to construct and $\mathcal{MBDG}(V)$ takes $O(n)$ time to construct if the input is a Delaunay Triangulation $\mathcal{DT}(V)$ on $V$. □

THEOREM 10. *$\mathcal{MBDG}(V)$ takes $\Omega(n \lg n)$ time to construct in the Algebraic Decision Tree (see* **[11]***) model of computation.*

PROOF. The construction of the Delaunay Triangulation $\mathcal{DT}(V)$ in the Algebraic Decision Tree model of computation is $\Omega(n \lg n)$. We show how to construct $\mathcal{DT}(V)$ in linear time given $\mathcal{MBDG}(V)$.

Given $\mathcal{MBDG}(V)$, we can construct $E(\mathcal{DT}(V))$ by iterating through each node and adding any edge or ghost-edge. By Theorem 7 there are only a constant number of edges to add at each vertex iterated through, thus taking linear time, and by Lemma 6 this is indeed $E(\mathcal{DT}(V)$. It then follows that $\mathcal{MBDG}(V)$ takes $\Omega(n \lg n)$ time to construct in the Algebraic Decision Tree model of computation. □

We have now given the navigator a new hope. We have left clues (ghost-edges) in the vertices of $\mathcal{MBDG}(V)$ to allow for the recreation of the Delaunay Triangulation. How could the navigator now use these signposts to actually route on $\mathcal{MBDG}(V)$?

CHAPTER 5

# Routing

Having now defined $\mathcal{MBDG}(V)$, we are ready to show how a navigator can route on it. To approach this, we will first look at routing on the Delaunay Triangulation $\mathcal{DT}(V)$. Then, we work our way towards simulating a routing algorithm on $\mathcal{MBDG}(V)$.

## 5.1 Chew's Routing Algorithm

Let us consider Chew's Routing Algorithm for Delaunay Triangulation presented in [**3**].

Given a source $s$ and a destination $t$ on the Delaunay Triangulation $\mathcal{DT}(V)$, we may assume without loss of generality that the line segment $[st]$ is horizontal with $s$ to the left of $t$. Chew's Routing Algorithm then works as follows: when we are at a vertex $v_i$ ($v_0 = s$), set $v_{i+1}$ to $t$ and terminate if $v_it$ is an edge in $\mathcal{DT}(V)$. Otherwise, consider the rightmost Delaunay Triangle $T_i = \Delta v_i, p, q$ at $v_i$ that has a non-empty intersection with $[st]$. Denote the circumcircle $\odot(v_i, p, q)$ with $C_i$. Denote the leftmost point of $C_i$ with $w_i$ and the rightmost intersection of $C_i$ and $[st]$ with $r_i$. Then,

- If $v_i$ is encountered in the clockwise walk along $C_i$ from $w_i$ to $r_i$, set $v_{i+1}$ to $p$, the first vertex among $\{p, q\}$ to be encountered in the clockwise walk along $C_i$ starting from $v_i$. See Figure 5.1 (left).
- Otherwise, set $v_{i+1}$ to $q$, the first vertex among $\{p, q\}$ to be encountered in the counterclockwise walk along $C_i$ starting from $v_i$. See Figure 5.1 (right).

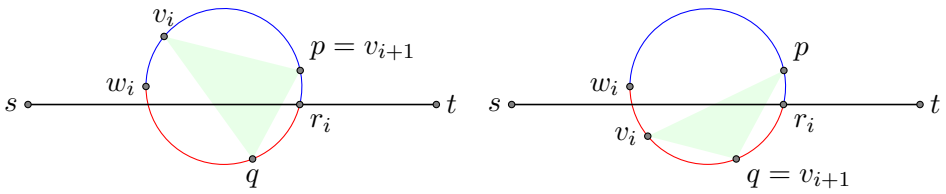



FIGURE 5.1: Routing to $p$ (left) and $q$ (right).

The following is a result from [**3**].

THEOREM 11. *Chew's Routing Algorithm on the Delaunay Triangulation has a routing ratio of at most* $(1.185043874 + 3\pi/2) \approx 5.90$.

## 5.2 Compromised Chew's Routing Algorithm

Let us now consider a different routing algorithm on the Delaunay Triangulation, heavily based on Chew's Routing Algorithm.

Given a source $s$ and a destination $t$ on the Delaunay Triangulation $\mathcal{DT}(V)$, we may assume without loss of generality that the line segment $[st]$ is horizontal with $s$ to the left of $t$. The Compromised Chew's Routing Algorithm then works as follows: when we are at a vertex $v_i$ ($v_0 = s$), set $v_{i+1}$ to $t$ and terminate if $v_it$ is an edge in $\mathcal{DT}(V)$. Otherwise, at $v_0$, consider the Delaunay Triangle $T_0 = \Delta v_0, p, q$ at $v_0$ that has a non-empty intersection with $[st]$, and make the same routing decision as Chew's Routing Algorithm. At $v_{i>0}$, instead of considering the rightmost Delaunay Triangle $T_i$ intersecting $[st]$ at $v_i$, we will find a Delaunay Triangle $A_i$ based on the Delaunay Triangle $A_{i-1} = \Delta v_{i-1}, v_i, f$ used in the routing decision at $v_{i-1}$ ($A_0 = T_0$).

$A_i = \Delta v_i, p, q$ is a Delaunay Triangle with a non-empty intersection with $[st]$ to the right of the intersection of $A_{i-1}$ with $[st]$. Moreover, if $v_i$ is above $[st]$, then, when making a counterclockwise sweep starting at the Delaunay edge $v_iv_{i-1}$ centred at $v_i$, the Delaunay edge $v_iq$ is encountered before $v_ip$, with $v_iq$ having a non-empty intersection with $[st]$ and $v_ip$ not intersecting $[st]$. If, on the other hand, $v_i$ is below $[st]$, then, when making a clockwise sweep starting at the Delaunay edge $v_iv_{i-1}$ centred at $v_i$, the Delaunay edge $v_ip$ is encountered before $v_iq$, with $v_ip$ having a non-empty intersection with $[st]$ and $v_iq$ not intersecting $[st]$. See Figure 5.2 where $A_i \neq T_i$.

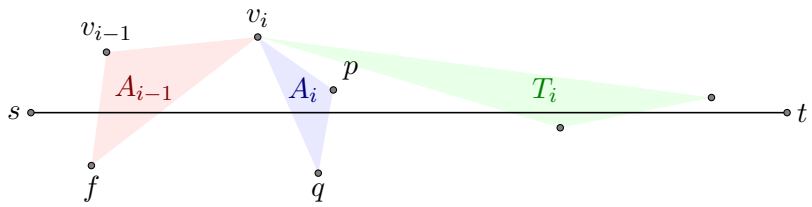



FIGURE 5.2: $A_{i-1}$ and $A_i$ (with $A_i$ not necessarily being the same as $T_i$).

Denote the circumcircle $\odot(v_i, p, q)$ with $C_i$ and its centre with $O_i$. Denote the leftmost point of $C_i$ with $w_i$ and the rightmost intersection between $C_i$ and $[st]$ with $r_i$. Then,

- If $v_i$ is encountered in the clockwise walk along $C_i$ from $w_i$ to $r_i$, set $v_{i+1}$ to $p$, the first vertex among $\{p, q\}$ to be encountered in the clockwise walk along $C_i$ starting from $v_i$. See Figure 5.3 (left).
- Otherwise, set $v_{i+1}$ to $q$, the first vertex among $\{p, q\}$ to be encountered in the counterclockwise walk along $C_i$ starting from $v_i$. See Figure 5.3 (right).

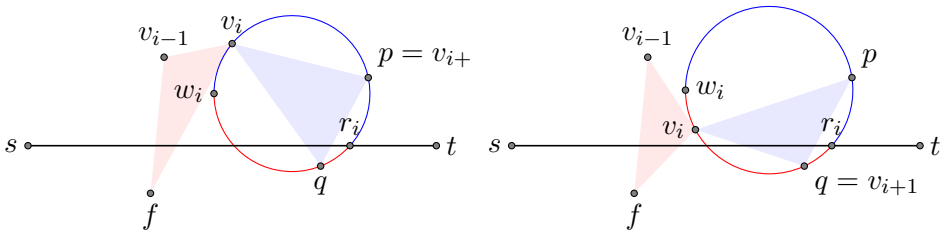



FIGURE 5.3: Routing to $p$ (left) and $q$ (right).

Can we always find some $A_i$ knowing $A_{i-1}$? Yes, we can indeed.

OBSERVATION 3. *Delaunay Triangles $A_i$ always exist, since the rightmost Delaunay Triangle $T_i$ is always a candidate.*

We can see that this is essentially Chew's Routing Algorithm without necessarily using the rightmost triangle to make the routing decision at any point in the path, but instead choosing any triangle $A_i$

from a set of suitable candidates (see Figure 5.2). Chew's Routing Algorithm is just a special case of Compromised Chew's Routing Algorithm.

This algorithm has not been analysed in [**3**], so we are thus left to show that, like Chew's Routing Algorithm, the Compromised Chew's Routing Algorithm has a routing ratio of about $5.90$.

### 5.2.1 Termination

Can Compromised Chew's Routing Algorithm end up in a non-terminating loop, never reaching $t$? The answer is in the negative, as we now show.

OBSERVATION 4. *The Delaunay Triangles $A_i$ are ordered along $[st]$ by definition.*

THEOREM 12. *Compromised Chew's Routing Algorithm terminates in a path from $s$ to $t$.*

PROOF. At any point $v_i$ with no direct edge to $t$, we can always find a Delaunay Triangle $A_i$ to continue the algorithm since at least $T_i$ is a candidate. In view of Observation 4, it then follows that Compromised Chew's Routing Algorithm terminates in a path from $s$ to $t$. □

### 5.2.2 Routing Ratio

Let us finally say the word on the routing ratio of Compromised Chew's Routing Algorithm.

THEOREM 13. *Compromised Chew's Routing Algorithm on the Delaunay Triangulation has a routing ratio of at most* $(1.185043874 + 3\pi/2) \approx 5.90$.

PROOF. The proof for the routing ratio of Chew's Routing Algorithm on the Delaunay Triangulation in [**3**] goes through for Compromised Chew's Routing Algorithm on the Delaunay Triangulation since the only parts of the proof of the latter using the property that $T_i$ is rightmost are:

(1) The termination of the algorithm (which we have shown).
(2) The categorisation of the Worst Case Circles of Delaunay Triangles $T_i$ into three mutually exclusive cases (we will presently expound this and show that the Worst Case Circles of $A_i$ are categorised into the same three cases).

Thus, Compromised Chew's Routing Algorithm on the Delaunay Triangulation has a routing ratio of at most $(1.185043874 + 3\pi/2) \approx 5.90$. □

#### 5.2.2.1 Worst Case Circles Associated With $T_i$

In the analysis of the routing ratio of Chew's Routing Algorithm in [**3**], the notion of Worst Case Circles is introduced whereby the length of the path yielded by the algorithm is bounded above by some path consisting of arcs along these Worst Case Circles; it is this arc-path along Worst Case Circles that is directly shown to have a routing ratio of no more than $5.90$ thus giving a bound on the algorithm's routing ratio.

Given a Delaunay Triangle $T_i$, we denote its circumcircle by $C_i$. Let the centre of $C_i$ be denoted by $O_i$. Then, the Worst Case Circle $C_i'$ is a circle that goes through $v_i$ and $v_{i+1}$, whose centre $O_i'$ is obtained by starting at $O_i$ and moving it along the perpendicular bisector of $[v_iv_{i+1}]$ until either $C_i'$ is tangent to $[st]$ or $v_i$ is the leftmost point of $C_i'$, whichever occurs first. The direction $O_i'$ is moved in depends on the routing decision at $v_i$: if $v_i$ is encountered on the clockwise walk from $w_i$ to $r_i$, then $O_i'$ is moved towards this arc, and otherwise, $O_i'$ is moved in the opposite direction.

Letting $w_i'$ be the leftmost point of $C_i'$, we can categorise the Worst Case Circles into the following three mutually exclusive types:

(1) Type $A_1$ : $v_i \neq w_i'$, and $[v_iv_{i+1}]$ does not cross $[st]$, and $C_i'$ is tangent to $st$.
(2) Type $A_2$ : $v_i = w_i'$ and $[v_iv_{i+1}]$ does not cross $[st]$.
(3) Type $B$ : $v_i = w_i'$ and $[v_iv_{i+1}]$ crosses $[st]$.

Figures 5.4, 5.5, and 5.6 show examples of Type $A_1$, $A_2$, and $B$ Worst Case Circles respectively.

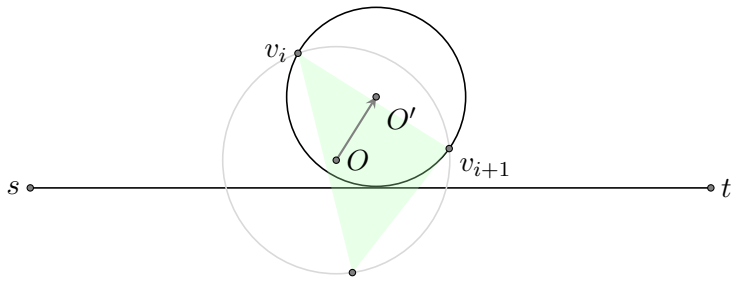



FIGURE 5.4: Example of a Type $A_1$ Worst Case Circle (outlined in black).

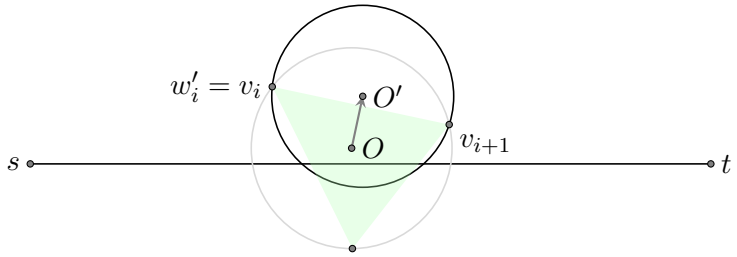



FIGURE 5.5: Example of a Type $A_2$ Worst Case Circle (outlined in black).

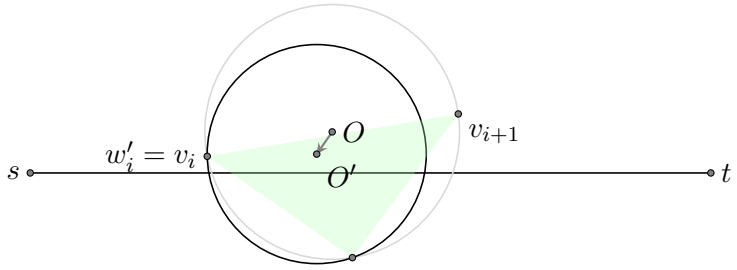



FIGURE 5.6: Example of a Type $B$ Worst Case Circle (outlined in black).

#### 5.2.2.2 Worst Case Circles Associated With $A_i$

What remains to be shown, is whether the Worst Case Circles of Delaunay Triangles $A_i$ fall into exactly one of the above categories. If we can show that it is indeed the case, we are done. Let $C_i$ be the circumcircle of $A_i$ centred at $O_i$, let $w_i$ be the leftmost point of $C_i$, and let $r_i$ be the right intersection of $C_i$ with $[st]$. We begin with the following observation which follows from how the criteria forces $A_i$ to intersect $[st]$:

OBSERVATION 5. *Let $p$ and $q$ be the other vertices of $A_i$. Taking a clockwise walk along $C_i$ from $v_i$ to $r_i$, exactly one of $p$ or $q$ will be encountered. Taking a counterclockwise walk along $C_i$ from $v_i$ to $r_i$, exactly one of $p$ or $q$ will be encountered.*

This observation captures the necessary property of $T_i$ that allows the categorisation into the three cases $A_1, A_2, B$ to go through. Define the Worst Case Circles associated with $A_i$ similarly to that of those

associated with $T_i$. Denote the Worst Case Circle of $A_i$ with $C_i'$, its centre with $O_i'$, and its leftmost point with $w_i'$.

LEMMA 7. *$C_i'$ can be categorised into the following three mutually exclusive types:*

(1) *Type $A_1$ : $v_i \neq w_i'$, and $[v_iv_{i+1}]$ does not cross $[st]$, and $C_i'$ is tangent to $st$.*
(2) *Type $A_2$ : $v_i = w_i'$ and $[v_iv_{i+1}]$ does not cross $[st]$.*
(3) *Type $B$ : $v_i = w_i'$ and $[v_iv_{i+1}]$ crosses $[st]$.*

PROOF. If $[v_iv_{i+1}]$ does not cross $[st]$, $C_i'$ is clearly of type $A_1$ or $A_2$.

Let us then consider when $[v_iv_{i+1}]$ crosses $[st]$. Without loss of generality, let $v_i$ be above $[st]$ and $v_{i+1}$ be below $[st]$. By Observation 5, we can deduce that $v_i$ must occur on the counterclockwise walk around $C_i$ from $w_i$ to $r_i$, for if not, neither of the vertices of $A_i$ occur on the clockwise walk around $C_i$ from $v_i$ to $r_i$. Since $v_i$ is above $[st]$, it is located above $l_i$, the leftmost intersection of $C_i$ with $[st]$, and below $w_i$ (see Figure 5.7).

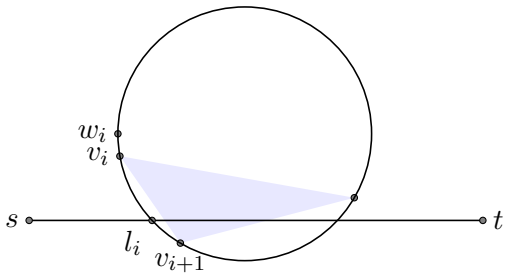



FIGURE 5.7: $v_i$ is above $l_i$ and below $w_i$.

Since $O_i'$ is moved along the perpendicular bisector of $[v_iv_{i+1}]$ in the direction towards the counterclockwise arc from $v_i$ to $v_{i+1}$, it must be that $w_i'$ (which starts at $w_i$ when $O_i'$ starts at $O_i$) moves onto $v_i$ eventually. Thus, $C_i'$ is Type B. □

Before we move on, let us note that if $[v_iv_{i+1}]$ intersects $[st]$ and $v_i$ and $v_{i+1}$ are on $C_i'$, then $C_i'$ can never be tangent to $st$. So why then, does this not show that the alternative termination condition ($w_i' = v_i$) holds? To wit, since $C_i'$ cannot be tangent, should we not get $w_i' = v_i$ for free? The force of Lemma 7 is to say that the $C_i'$ are well defined (circles of finite radius). Consider the situation when Observation 5 does not hold (see Figure 5.8).

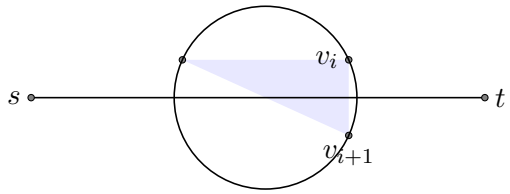



FIGURE 5.8: Observation 5 does not hold since a clockwise walk around $C_i$ from $v_i$ to $r_i$ encounters neither vertex of the Delaunay Triangle.

$C_i'$ would then grow interminably and is thus not well defined. We have chosen the criteria $A_i$ is to meet so that this can never happen. Since $C_i'$ is indeed well defined, the proof in [**3**] goes through.

Recall that we said in Theorem 13 Compromised Chew's Routing Algorithm on the Delaunay Triangulation has a routing ratio of at most $(1.185043874 + 3\pi/2) \approx 5.90$ if the following two can be shown:

(1) The termination of the algorithm.
(2) The categorisation of the Worst Case Circles of Delaunay Triangles $T_i$ into three mutually exclusive cases.

Having show both, it is indeed the case that Compromised Chew's Routing Algorithm on the Delaunay Triangulation has a routing ratio of at most $(1.185043874 + 3\pi/2) \approx 5.90$.

## 5.3 Simulating Compromised Chew's Routing Algorithm on $\mathcal{MBDG}(V)$

It is Compromised Chew's Routing Algorithm on a Delaunay Triangulation that we simulate on $\mathcal{MBDG}(V)$. At a high level, the simulation searches for a suitable candidate $A_i$ at $v_i$. This search is done by taking a walk from $v_i$ with no particular destination in consideration, ending eventually at some vertex of $A_i$. Once the routing decision is made about which vertex of $A_i$ to route towards, another walk is taken towards that vertex. Miraculously, the concatenation of walks (the first to find $A_i$ and the second to move to $v_{i+1}$) turns out to be a spanning path of the Delaunay edge $v_iv_{i+1}$. We now get into the details.

### 5.3.1 Preliminaries

First, let us make some provisionary notions to simplify the upcoming exposition.

#### 5.3.1.1 Unguided Face Walks

Suppose $vu_1$ and $vu_m$ are a middle edge and a penultimate edge and suppose that the shorter of the two, say $vu_1$, is the immediately counterclockwise protected edge to the other. These edges belong to a particular face of $\mathcal{MBDG}(V)$ (see Figure 5.9).

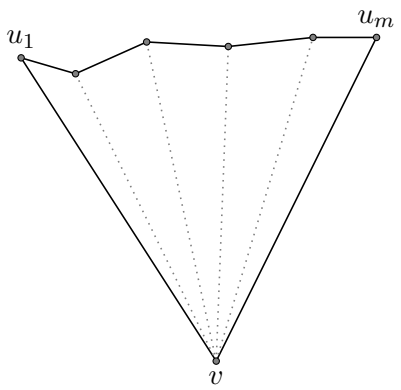



FIGURE 5.9: Face of $\mathcal{MBDG}(V)$.

For any vertex $p$ on this face, we refer to the spanning path from $v$ to $p$ beginning with the shorter of the two, say $vu_1$, as an Unguided Face Walk from $v$ to $p$ (see Figure 5.10).

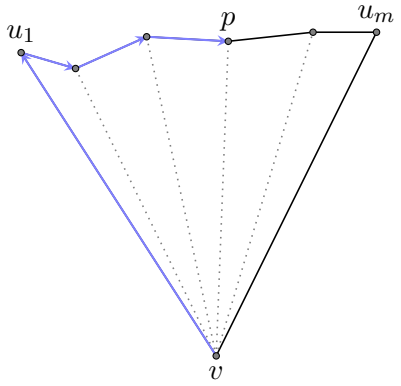



FIGURE 5.10: Unguided Face Walk from $v$ to $p$.

In the simulation of Compromised Chew's Routing Algorithm, we will be using Unguided Face Walks in a sense where $p$ is undetermined until it is reached; we will take an Unguided Face Walk from $v$ and test at each vertex along this walk if it satisfies some property, ending the walk if it does. Routing in this manner from $v$ to an undetermined $p$ can easily be done locally:

Suppose $vu_1$ was counterclockwise to $vu_m$ and the shorter of the two. Then, at any intermediate vertex $u_i$, we take the edge immediately counterclockwise to $u_iu_{i-1}$ ($v = u_0$). The procedure when $vu_1$ is clockwise to $vu_m$ is similar modulo orientation changes.

OBSERVATION 6. *An Unguided Face Walk needs $O(1)$ memory since at $u_i$, the previous vertex along the walk $u_{i-1}$ must be stored in order to determine $u_{i+1}$.*

OBSERVATION 7. *An Unguided Face Walk from $v$ to $p$ has at most a $\max(\pi/2, \pi\sin(\theta/2) + 1)$ stretch factor from the proof of Theorem 5.*

#### 5.3.1.2 Guided Face Walks

Suppose $vp$ is extreme at $v$ but not protected at $p$ (it is a ghost-edge stored at $v$). Then, $vp$ is a chord of some face determined by $pu_1$ and $pu_m$ where the former is a middle edge and the latter a penultimate edge. Moreover, recall that there is a bit on the ghost-edge $vp$ indicating whether to take the edge clockwise or counterclockwise to $vp$ to eventually reach $p$ from $v$. See Figure 5.11.

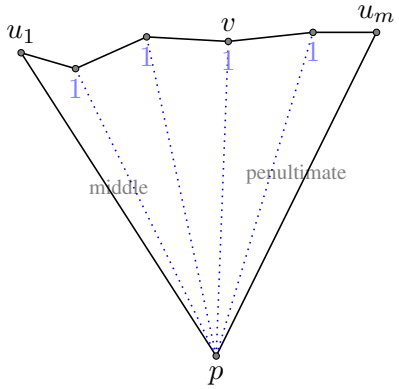



FIGURE 5.11: $vp$ is a chord of some face, with a guiding bit at $v$.

We refer to the face path from $v$ to $p$ following the direction pointed to by these bits as the Guided Face Walk from $v$ to $p$. To wit, it is the face path from $v$ to $p$ ending with the middle edge $u_1p$ (see Figure 5.12).

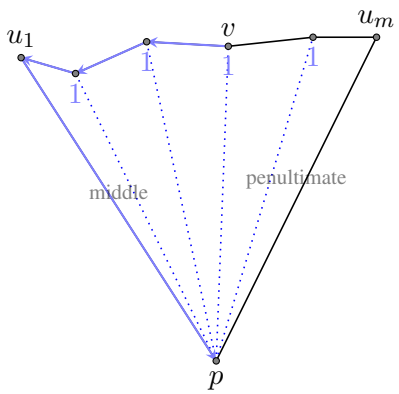



FIGURE 5.12: Guided Face Walk from $v$ to $p$.

Routing in this manner from $v$ to $p$ can easily be done:

(1) At $v$, store $p$ in memory.
(2) Until $p$ is reached, if there is an edge to $p$, take it. Otherwise, take the edge pointed to by the ghost-edge with $p$ at its opposite endpoint.

OBSERVATION 8. *A Guided Face Walk needs $O(1)$ memory since $p$ needs to be stored in memory for the duration of the walk.*

OBSERVATION 9. *A Guided Face Walk from $v$ to $p$ has at most a $\max(\pi/2, \pi\sin(\theta/2) + 1)$ stretch factor from the proof of Theorem 5.*

### 5.3.2 The Simulation

Finally, let us dive into the details of simulating Compromised Chew's Routing Algorithm. We split off the discourse into two parts:

(1) How to simulate the first step of Compromised Chew's Routing Algorithm, which does not require any prior Delaunay Triangle to find the Delaunay Triangle $A_0$ wherefrom the routing decision is made.
(2) How to simulate subsequent steps of Compromised Chew's Routing Algorithm, which requires, at step $i$, the Delaunay Triangle $A_{i-1}$ used in the previous step to find the next Delaunay Triangle $A_i$ wherefrom the routing decision is made.

#### 5.3.2.1 Simulating the First Step

If $st$ is an edge, take it and terminate. Otherwise, at $s = v_0$, we want to show how to find the vertices of $A_0 = T_0$ in $\mathcal{MBDG}(V)$ and make a routing decision. Look at all edges protected at $s$, and let $su_1$ be the first such edge encountered in a counterclockwise sweep of $[st]$ centred at $s$, and $su_m$ be the first such edge encountered in a clockwise sweep of $[st]$ centred at $s$. There are two subcases.

(I) In the first case, both $su_1$ and $su_m$ are not middle edges at $s$. $\Delta s, u_1, u_m$ is then a Delaunay Triangle $A_0$. Denote the circumcircle $\odot(s, u_1, u_m)$ with $C_0$. Denote the leftmost point of $C_0$ with $w_0$ and the rightmost intersection between $C_0$ and $[st]$ with $r_0$. Then,

- If $s$ is encountered in the clockwise walk along $C_0$ from $w_0$ to $r_0$, move along $su_1$ if it is fully protected, and otherwise take the Guided Face Walk from $s$ to $u_1$. Then, set $v_1$ to $u_1$, the first vertex among $\{u_1, u_m\}$ to be encountered in the clockwise walk along $C_0$ starting from $s$.
- Otherwise, move along $su_m$ if it is fully protected, and otherwise take the Guided Face Walk from $s$ to $u_m$. Then, set $v_1$ to $u_m$, the first vertex among $\{u_1, u_m\}$ to be encountered in the counterclockwise walk along $C_0$ starting from $s$.

Either way, the current memory used for the Face Walks should be cleared; and, furthermore, before leaving $s$, $A_0 = \Delta s, u_1, u_m$ should be stored as the last triangle used.

(II) In the second case, one of $su_1$ and $su_m$ is a middle edge at $s$. The other edge must then be a penultimate edge. Then, $A_0 = \Delta s, p, q$ must be contained in the cone with apex $s$ sweeping clockwise from $su_1$ to $su_m$ (see Figure 5.13).

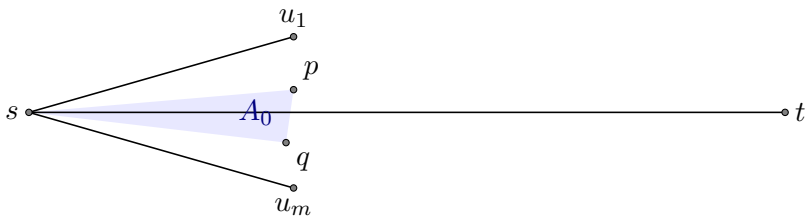



FIGURE 5.13: $A_0$ contained between $su_1$ and $su_m$

We may assume that $su_1$ is shorter than $su_m$. Take the Unguided Face Walk from $s$ until some $u_i$ such that $u_i = p$ is above $[st]$ and $u_{i+1} = q$ is below $[st]$. At $p$, we have now found $A_0 = \Delta s, p, q$. Denote the

circumcircle $\odot(s, p, q)$ with $C_0$. Denote the leftmost point of $C_0$ with $w_0$ and the rightmost intersection between $C_0$ and $[st]$ with $r_0$. Then,

- If $s$ is encountered in the clockwise walk along $C_0$ from $w_0$ to $r_0$, set $v_1$ to $p$, the first vertex among $\{p, q\}$ to be encountered in the clockwise walk along $C_0$ starting from $s$.
- Otherwise, complete the Unguided Face Walk from $s$ to $q$ by moving along $pq$. Then, set $v_1$ to $q$, the first vertex among $\{p, q\}$ to be encountered in the counterclockwise walk along $C_0$ starting from $s$.

Finally, clear the current memory used for Face Walks and store $A_0 = \Delta s, p, q$ as the last triangle used.

#### 5.3.2.2 Simulating the $i$th Step

Suppose $v_i$ is above $[st]$, and that $A_{i-1}$ is in memory. If $v_i t$ is an edge, take it and terminate. Otherwise, let $v_i f$ be rightmost edge of $A_{i-1}$ that intersects $[st]$, and $\overline{v_i f}$ be the edge extended to a line (see Figure 5.14).

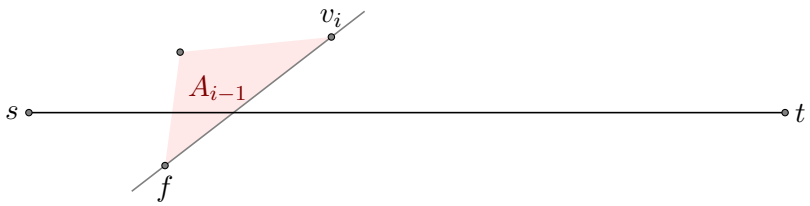



FIGURE 5.14: Example depiction of $\overline{v_i f}$.

Make a counterclockwise sweep, centred at $v_i$ and starting at $v_i f$, through all edges that are protected at $v_i$ and contained in the side of $\overline{v_i f}$ that contains $t$. Note that this region must have at least one such edge, otherwise $v_i f$ is a convex hull edge, which cannot be the case since $s$ is on one side and $t$ on the other.

(I) If there is some edge that does not intersect $[st]$ in this sweep, let $v_i u_1$ be the first such edge encountered in the sweep and let $v_i u_m$ be the protected edge immediately clockwise to $v_i u_1$ at $v_i$. There are two cases to consider.

(I.I) If $A_{i-1}$ is not contained in the cone with apex $v_i$ sweeping clockwise from $v_iu_1$ to $v_iu_m$, there are two subcases to consider.

(I.I.I) In the first subcase, both $v_iu_1$ and $v_iu_m$ are not middle edges. This is handled the same way the first step is simulated; decide whether to move to $u_1$ or $u_m$ and take a direct edge if it exists and a Guided Face Walk there otherwise.

(I.I.II) In the second subcase, one of $v_iu_1$ and $v_iu_m$ is a middle edge at $v_i$. The other edge must be a penultimate edge. Then, $A_i = \Delta v_i, p, q$ must be contained in the cone defined by $v_iu_1$ and $v_iu_m$ (see Figure 5.15).

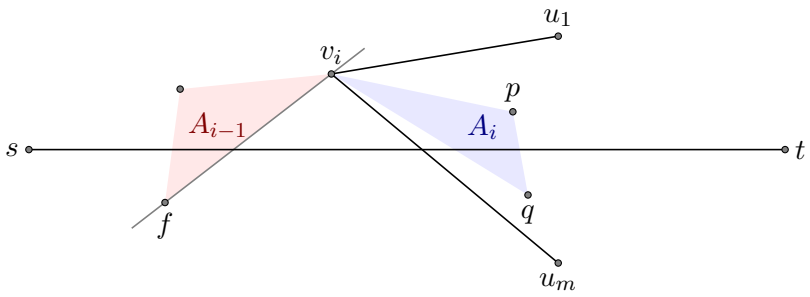



FIGURE 5.15: $A_i$ contained between $v_iu_1$ and $v_iu_m$

This is handled the same way the first step is simulated; we take an Unguided Face Walk until, assuming $v_iu_1$ is the first edge in the walk, some $u_i$ is reached such that $u_i = p$ is above $[st]$ and $u_{i+1} = q$ is below $[st]$, and make the decision to complete the Unguided Face Walk to $q$ or not.

(I.II) If $A_{i-1}$ is contained in the cone with apex $v_i$ sweeping clockwise from $v_iu_1$ to $v_iu_m$, then one of $v_iu_1$ and $v_iu_m$ must be a middle edge and the other a penultimate edge, since the edge $v_if$ is contained in the interior of this cone. Then, $A_i = \Delta v_i, p, q$ must be contained in the cone with apex $v_i$ sweeping clockwise from $v_iu_1$ to $v_if$. See Figure 5.16.

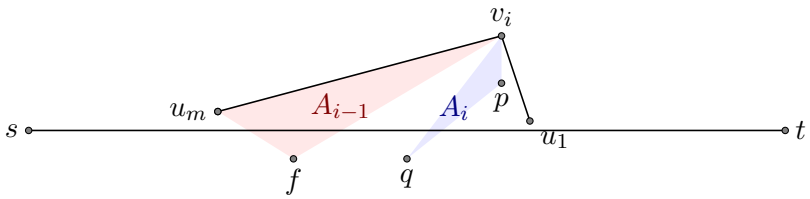



FIGURE 5.16: $A_{i-1}$ and $A_i$ contained between $v_iu_1$ and $v_iu_m$

If $v_iu_1$ is shorter than $v_iu_m$, then finding $A_i$ is handled the same way the first step is simulated; we take an Unguided Face Walk until some $u_i$ is reached such that $u_i = p$ is above $[st]$ and $u_{i+1} = q$ is below $[st]$, and make the decision to complete the Unguided Face Walk to $q$ or not. However, if $v_iu_1$ is longer than $v_iu_m$, we store $f$ in memory, and take an Unguided Face Walk until we pass $f$, and once we have passed $f$, continue until some $u_i$ is reached such that $u_i = q$ is below $[st]$ and $u_{i+1} = p$ is above $[st]$, finally making the decision to complete the Unguided Face Walk to $p$ or not. Lastly, clear the current memory and store $A_i = \Delta v_i, p, q$ in memory.

(II) If, on the other hand, all of the edges in the sweep have a non-empty intersection with $[st]$, let $v_iu_m$ be the last edge encountered in the sweep, and $v_iu_1$ be the protected edge immediately counterclockwise to it. Note that $\angle(u_1, v_i, u_m) < \pi$, for otherwise $v_iu_m$ is a convex hull edge which cannot be the case; this means that $A_{i-1}$ cannot be contained in the cone with apex $v_i$ sweeping clockwise from $v_iu_1$ to $v_iu_m$. Notwithstanding, there are two subcases to consider.

(II.I) In the first case, both $v_iu_1$ and $v_iu_m$ are not middle edges. This is handled the same way the first step is simulated; decide whether to move to $u_1$ or $u_m$ and take a direct edge if it exists and a Guided Face Walk there otherwise.

(II.II) In the second case, one of $v_iu_1$ and $v_iu_m$ is a middle edge at $v_i$. The other edge must be a penultimate edge. Then, $A_i = \Delta v_i, p, q$ must be contained in the cone with apex $v_i$ sweeping clockwise from $v_iu_1$ to $v_iu_m$ (see Figure 5.17).

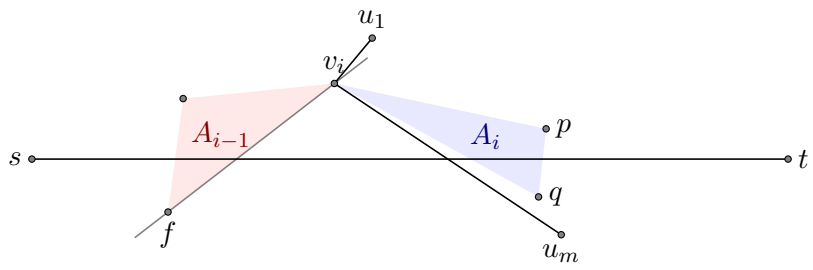



FIGURE 5.17: $A_i$ contained between $v_iu_1$ and $v_iu_m$

This is handled the same way the first step is simulated; we take an Unguided Face Walk until either $p$ or $q$ is reached, and make the decision to complete the Unguided Face Walk to the other of $p$ and $q$ or not.

Finally, if $v_i$ is below $[st]$, simulating Compromised Chew's Routing Algorithm for this step is done analogously modulo orientation changes such as sweeping $v_if$ clockwise instead of counterclockwise.

Well, so what name do we give this simulation? Compromised Chew's Routing Algorithm on $\mathcal{MBDG}(V)$. How creative!

THEOREM 14. *Compromised Chew's Routing Algorithm on $\mathcal{MBDG}(V)$ has a routing ratio of at most* $5.90\max(\pi/2, \pi\sin(\theta/2) + 1)$.

PROOF. This follows from Theorem 11 and Observations 7 and 9 since, for every $A_i$, we are taking a direct edge, an Unguided Face Walk, or a Guided Face Walk to $v_{i+1}$. Compromised Chew's Routing Algorithm on $\mathcal{MBDG}(V)$ thus has a routing ratio of no more than $\max(\pi/2, \pi\sin(\theta/2) + 1)$ times that of Chew's Routing Algorithm on the Delaunay Triangulation $\mathcal{DT}(V)$, making it at most $5.90\max(\pi/2, \pi\sin(\theta/2) + 1)$. □

THEOREM 15. *Compromised Chew's Routing Algorithm on $\mathcal{MBDG}(V)$ uses $O(1)$ memory.*

PROOF. This follows from Observations 6 and 8, and that, additionally, when at $v_i$, the Delaunay Triangle $A_{i-1}$ is stored and one of its vertices possibly marked so as to know when its rightmost edge

is passed in an Unguided Face Walk (when $A_{i-1}$ is contained between $v_iu_1$ and $v_iu_m$). Compromised Chew's Routing Algorithm on $\mathcal{MBDG}(V)$ thus uses $O(1)$ memory. □

COROLLARY 3. *Compromised Chew's Routing Algorithm on $\mathcal{MBDG}(V)$ is an $O(1)$-memory deterministic $1$-local routing algorithm with a routing ratio of no more than $5.90\max(\pi/2, \pi\sin(\theta/2)+1)$.*

PROOF. This follows immediately from Theorems 14 and 15, and the description of Compromised Chew's Routing Algorithm on $\mathcal{MBDG}(V)$. □

CHAPTER 6

# Lightness, and Routing Redux

---

We have seen a bounded degree construction $\mathcal{MBDG}(V)$, and a routing algorithm on it with constant ratio, but we have yet to keep to our end of the deal fully. Let us address the elephant in the room: weight. We will describe a pruning algorithm that takes $\mathcal{MBDG}(V)$ and returns a graph (Light Marked Bounded Degree Graph) $\mathcal{LMBDG}(V) \subseteq \mathcal{MBDG}(V)$ that has a weight within a constant times that of the minimum spanning tree of $V$ at the trade-off of a slightly increased, but still constant, stretch factor. Then, with a slight modification to Compromised Chew's Routing Algorithm on $\mathcal{MBDG}(V)$, we show how to route on $\mathcal{LMBDG}(V)$ with a constant routing ratio and constant memory.

## 6.1 The Levcopoulos and Lingas Protocol

To bound the weight of $\mathcal{MBDG}(V)$, we will use the algorithm shown in [**9**] by Levcopoulos and Lingas with two slight modifications; one, to take in any planar graph as input as the original algorithm in [**9**] takes a Delaunay Triangulation as input; and two, a marking on endpoints of pruned edges to facilitate routing across these pruned edges. The first modification was proposed in [**9**] under their final remark (A), which says, "The method presented in Section 3 is quite general. It can be applied to any planar graphs $H, F$ embedded in the plane, where $H$ is a connected subgraph of $F$ and $F$ is triangulated outside $H$, in order to construct a subgraph $G$ of $F$ that approximates $F$ and has length $O(|H|)$."

At a high level, the algorithm works as follows: Given $\mathcal{MBDG}(V)$, we compute its minimum spanning tree and add all the minimum spanning tree edges to $\mathcal{LMBDG}(V)$. We then take an Euler Tour around the minimum spanning tree, treating it as a degenerate polygon $P$ enclosing $V$. Finally, we start expanding $P$ into the convex hull $CH(V)$. As edges of $\mathcal{MBDG}(V)$ enter the interior of $P$, we determine whether to include them in $\mathcal{LMBDG}(V)$, or exclude them from $\mathcal{LMBDG}(V)$. If an edge is excluded from $\mathcal{LMBDG}(V)$, we augment its endpoints with information to facilitate the routing process

should that edge be one that would have been used in the path found by Compromised Chew's Routing Algorithm on $\mathcal{MBDG}(V)$. Once $P$ has expanded into $CH(V)$, we return $\mathcal{LMBDG}(V)$.

### 6.1.1 Preliminaries

Let us first acknowledge and differentiate between a few kinds of edges which will play a part in the following discussion:

(1) Convex hull edges of $CH(V)$.
(2) Boundary edges of the polygon $P$ that encloses $V$.
(3) Included settled edges, which are edges of $\mathcal{MBDG}(V)$ in the interior of $P$ and included in $\mathcal{LMBDG}(V)$.
(4) Excluded settled edges, which are edges of $\mathcal{MBDG}(V)$ in the interior of $P$ and excluded from $\mathcal{LMBDG}(V)$.
(5) Unsettled edges, which are edges of $\mathcal{MBDG}(V)$ outside of $P$ and whose inclusion in $\mathcal{LMBDG}(V)$ have not yet been determined.

Note that while the last three kinds are mutually exclusive, there may be edges which are of more than one kind. For example, a boundary edge of $P$ can coincide with a convex hull edge of $CH(V)$.

### 6.1.2 How the Polygon $P$ Grows

For each iteration of the Levcopoulos and Lingas Protocol, $P$, a polygon without holes, grows, consuming more area and more edges of $\mathcal{MBDG}(V)$, until it coincides completely with the convex hull $CH(V)$. With more specificity, how would $P$ grow? Let us drill down and look at an iteration of the algorithm.

Consider any edge $uv$ on the convex hull $CH(V)$. If part of the boundary of $P$ coincides with $uv$, there is nothing to consider. However, if that is not the case, then consider the path $\partial P(u, v)$ from $u$ to $v$ along the boundary of $P$ which has a part visible to $uv$ (see Figure 6.1).

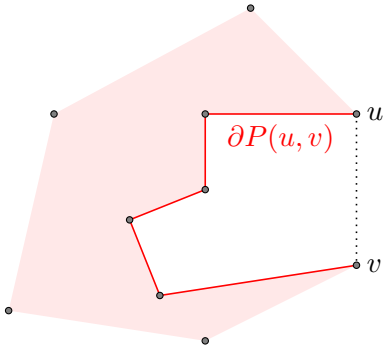



FIGURE 6.1: $\partial P(u,v)$ has a part visible to $uv$. The dotted edge is a convex hull edge of $V$.

$\partial P(u,v)$ concatenated with $uv$ then forms a closed curve $C$ on the plane that does not intersect the interior of $P$. $C$ is further subdivided by unsettled edges (non-crossing by planarity), with endpoints between vertices of $\delta P(u,v)$, into cells $c_1, \ldots, c_{k\geq 0}$ (see Figure 6.2).

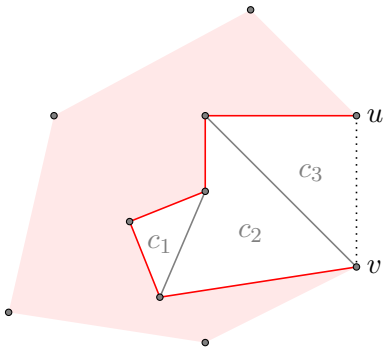



FIGURE 6.2: $\partial P(u,v) \cup uv$ is subdivided into $k$ cells ($k = 3$ in this picture). Each gray edge is an edge in $\mathcal{MBDG}(V)$.

If there are no unsettled edges, we expand $\partial P(u,v)$ into $uv$ by removing $\partial P(u,v)$ from $P$ and adding $uv$ to $P$. If, on the other hand, there is a non-zero number of unsettled edges, there must be some cell $c_i$ whose entire boundary, minus one unsettled edge $pq$, coincides with a part of $\partial P(u,v)$ (see Figure 6.3).

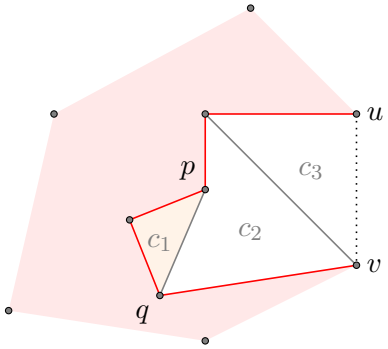



FIGURE 6.3: $c_1$ coincides with part of $\partial P(u,v)$ except its one unsettled edge. $c_2$ and $c_3$ are not candidates for expansion.

Then, we consider the addition of $pq$ into $\mathcal{LMBDG}(V)$, make it a settled edge, and expand $P$ into $c_i$ by removing the subpath from $p$ to $q$ along $\partial P(u,v)$ from $P$, and adding the edge $pq$ to $P$ (see Figure 6.4).

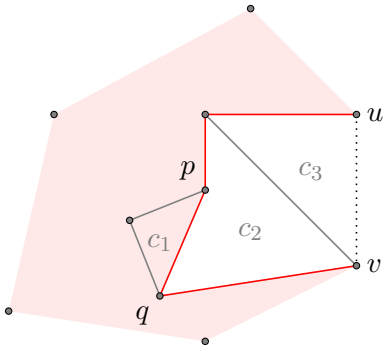



FIGURE 6.4: Expansion of $P$ into $c_1$.

Since the area of $P$ is increasing, this process must eventually terminate.

### 6.1.3 Condition For Including an Edge

The decision whether to include an edge in $\mathcal{LMBDG}(V)$ depends on an adjustable parameter $r > 0$, which causes an increase in the stretch factor by at most $1+1/r$ and ensures a weight of at most $(2r+1)$ times that of $|MST(\mathcal{MBDG}(V))|$.

All settled edges are assigned a $weight \geq 0$, which is the length of a short (but not necessarily shortest) path between their endpoints that uses only the currently included settled edges, which are by definition edges of $\mathcal{LMBDG}(V)$. Initially, $weight(pq) = |pq|$ for all edges $pq$ in the minimum spanning tree of

$\mathcal{MBDG}(V)$. Now, when considering whether to include the unsettled edge $uv$ into $\mathcal{LMBDG}(V)$, we take the sum $S$ of the $weight$ of edges in $\partial P(u, v)$. These edges have been settled and thus have $weight$ assigned. If $S$ is greater than $(1+1/r)\,|uv|$, add $uv$ to $\mathcal{LMBDG}(V)$ and assign it a $weight$ of $|uv|$ now that is has been settled. Otherwise, settle $uv$ but exclude it, and assign it a $weight$ of $S$; we can see that $S$ is the length of the path from $u$ to $v$ that is the concatenation of paths between the endpoints of edges in $\partial P(u, v)$.

We can now bound the stretch factor of $\mathcal{LMBDG}(V)$.

THEOREM 16. *$\mathcal{LMBDG}(V)$ is a $(1+1/r)$-spanner of $\mathcal{MBDG}(V)$ for an adjustable parameter $r > 0$.*

PROOF. Consider some edge $uv$ that is not in $\mathcal{LMBDG}(V)$ but in $\mathcal{MBDG}(V)$. The edge $uv$ was not added to $\mathcal{LMBDG}(V)$ because the sum $S$ of the $weight$ of edges in $\partial P(u, v)$ was less than $(1+1/r)\,|uv|$, so there must then be a path from $u$ to $v$ of length at most $(1+1/r)\,|uv|$ in $\mathcal{LMBDG}(V)$.

$\mathcal{LMBDG}(V)$ is thus a $(1 + 1/r)$-spanner of $\mathcal{MBDG}(V)$. □

COROLLARY 4. *$\mathcal{LMBDG}(V)$ is a $1.998(1+1/r)\max(\pi/2, \pi\sin(\theta/2)+1)$-spanner for the adjustable parameters $0 < \theta < \pi/2$ and $r > 0$.*

PROOF. This follows from Corollary 1 and Theorem 16. □

We can also bound the weight of $\mathcal{LMBDG}(V)$ now.

THEOREM 17. *$\mathcal{LMBDG}(V)$ has a weight of at most $2r + 1$ times the weight of the minimum spanning tree of $\mathcal{MBDG}(V)$.*

PROOF. Initially, give each edge $e$ of $P$, which are in a two-to-one correspondence with the edges of the minimum spanning tree of $\mathcal{MBDG}(V)$ since every such edge is part of the boundary of $P$ in two places, a starting credit of $r\,|e|$. Denote the sum of credits of edges in $P$ with $credit(P)$. The sum of $credit(P)$ and the weight of the initially included settled edges is then $(2r + 1)$ times the weight of the minimum spanning tree of $\mathcal{MBDG}(V)$.

As $P$ is expanded and edges are settled, we adjust the credits in the following manner:

- If an unsettled edge $uv$ is added into $\mathcal{LMBDG}(V)$ when settled, we set the credit of the newly added edge $uv$ of $P$ to $credit(\partial P(u,v)) - |uv|$, and, by removing $\partial P(u,v)$ from $P$, we effectively set the credit of edges along $\partial P(u,v)$ to 0.
- If an unsettled edge is excluded from $\mathcal{LMBDG}(V)$ when settled, we set the credit of the newly added edge $uv$ of $P$ to $credit(\partial P(u,v))$, and, by removing $\partial P(u,v)$, we effectively set the credit of edges along $\partial P(u,v)$ to 0.

We can see that the sum of $credit(P)$ and the weights of included settled edges, at any time, is at most $2r+1$ times the weight of the minimum spanning tree of $\mathcal{MBDG}(V)$.

It now suffices to show that $credit(P)$ is never negative, which we do by showing that for every edge $uv$ of $P$, at any time, $credit(uv) \geq r \cdot weight(uv) \geq 0$. Initially, when $P$ is the Euler Tour around the minimum spanning tree of $\mathcal{MBDG}(V)$, we have that $credit(uv) = r \cdot weight(uv)$. We now consider two cases.

(I) If $uv$ is added to $\mathcal{LMBDG}(V)$, then

$$\begin{aligned} credit(uv) &= credit(\partial P(u,v)) - |uv| \\ &\geq r \cdot weight(\partial P(u,v)) - |uv| \\ &\geq r(1+1/r)\,|uv| - |uv| \\ &= r\,|uv| \\ &= r \cdot weight(uv). \end{aligned}$$

The first inequality holds from the induction hypothesis, and the second inequality and last equality hold since $uv$ is added to $\mathcal{LMBDG}(V)$.

(II) If $uv$ is not added to $\mathcal{LMBDG}(V)$, then

$$\begin{aligned} credit(uv) &= credit(\partial P(u,v)) \\ &\geq r \cdot weight(\partial P(u,v)) \\ &= r \cdot weight(uv). \end{aligned}$$

The first inequality holds from the induction hypothesis, and the last equality holds since $uv$ is not added to $\mathcal{LMBDG}(V)$.

Since $credit(P)$ is never negative, and the sum of $credit(P)$ and the weights of included settled edges is at most $2r + 1$ times the weight of the minimum spanning tree of $\mathcal{MBDG}(V)$, we conclude that $\mathcal{LMBDG}(V)$ has a weight of at most $2r + 1$ times the weight of the minimum spanning tree of $\mathcal{MBDG}(V)$. □

COROLLARY 5. *$\mathcal{LMBDG}(V)$ has a weight of at most $1.998(2r + 1)\max(\pi/2, \pi\sin(\theta/2) + 1)$ times that of the minimum spanning tree of $V$.*

PROOF. This follows since the weight of $MST(G)$ is at most $t$ times that of the weight of $MST(V)$, whenever $G$ is a $t$-spanner on $V$.

We can see this by considering, for every edge $uv$ in $MST(V)$, a spanning path in $G$ from $u$ to $v$. This has weight at most $t$ times $|uv|$, and thus the union of all these spanning paths for each edge in $MST(V)$ is a connected graph with weight at most $t$ times the weight of $MST(V)$. The weight of $MST(G)$ must then be at most the weight of this union of paths.

$\mathcal{MBDG}(V)$ is indeed a $1.998\max(\pi/2, \pi\sin(\theta/2) + 1)$-spanner on $V$ by Corollary 1. $\mathcal{LMBDG}(V)$ therefore has a weight of at most $1.998(2r + 1)\max(\pi/2, \pi\sin(\theta/2) + 1)$ times that of the minimum spanning tree of $V$. □

THEOREM 18. *$\mathcal{LMBDG}(V)$ has a degree of at most $5\lceil 2\pi/\theta\rceil$, a weight of at most $1.998(2r + 1)\max(\pi/2, \pi\sin(\theta/2) + 1)$ times that of a minimum spanning tree of $V$, and is a planar $1.998(1 + 1/r)\max(\pi/2, \pi\sin(\theta/2) + 1)$-spanner for the adjustable parameters $0 < \theta < \pi/2$ and $r > 0$.*

PROOF. The degree bound and planarity follow from Theorem 8, since $\mathcal{LMBDG}(V) \subseteq \mathcal{MBDG}(V)$. The weight bound follows from Corollary 5. Finally, the bound on the stretch factor follows from Corollary 4. □

At this point, we take a short digression to remark on a curious transition phenomenon of sorts, which concerns what bound we are able to prove on the weight. While we can say that the weight of $\mathcal{LMBDG}(V)$ is at most $1.998(2r + 1)\max(\pi/2, \pi\sin(\theta/2) + 1)$ times that of the minimum spanning tree of $V$, we can say something even stronger if $\theta$ is small. There is a threshold on $\theta$, precisely $\pi/3$, where the weight

of $\mathcal{LMBDG}(V)$ can be bounded to be no more than $(2r + 1)$ that of a minimum spanning tree on $V$, removing the $t$ factor that arises from the bound on the spanning ratio of $\mathcal{MBDG}(V)$. The reason for this is that if $\theta < \pi/3$, a minimum spanning tree of $V$ will be contained in $\mathcal{MBDG}(V)$. Let us work towards showing this.

LEMMA 8. *Let $uv_1$ and $uv_2$ be edges in a minimum spanning tree of $V$. Then, $\angle(v_1, u, v_2) \geq \pi/3$.*

PROOF. Refer to Figure 6.5. Let $uv_1$ and $uv_2$ be edges in a minimum spanning tree of $V$. Suppose for a contradiction that $\angle(v_1, u, v_2) < \pi/3$. Then, without loss of generality, we can say that $\angle(u, v_1, v_2) > \pi/3$. Since $\angle(v_1, u, v_2) < \pi/3$ and $\angle(u, v_1, v_2) > \pi/3$, we deduce that $|v_1v_2| < |uv_2|$. We can therefore replace $uv_2$ with $v_1v_2$ to get a lighter spanning tree, contradicting the minimality of the tree. Therefore, it must be that $\angle(v_1, u, v_2) \geq \pi/3$.

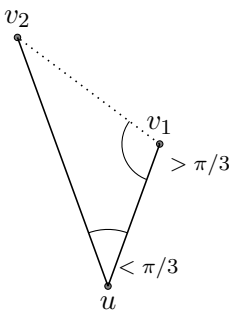



FIGURE 6.5: We can replace $uv_2$ with $v_1v_2$ to get a lighter tree.

□

LEMMA 9. *Fix a minimum spanning tree on $V$. Let $C$ be a cone with apex $u$ and angle measure less than $\pi/3$. If $uv$ is a minimum spanning tree edge contained in $C$, and if there is a $w \in V \cap C$ such that $|uw| \leq |uv|$, then we can replace $uv$ with $uw$ to get another minimum spanning tree.*

PROOF. Fix a minimum spanning tree on $V$. Let $C$ be a cone with apex $u$ and angle measure less than $\pi/3$, and let $uv$ be a minimum spanning tree edge contained in $C$. Suppose there is a $w \in V \cap C$ such that $|uw| \leq |uv|$. We consider two cases separately. In the first case, when the path in the minimum

spanning tree from $v$ to $w$ does not go through $u$ (see Figure 6.6), we can replace $uv$ with $uw$ to get a spanning tree no heavier.

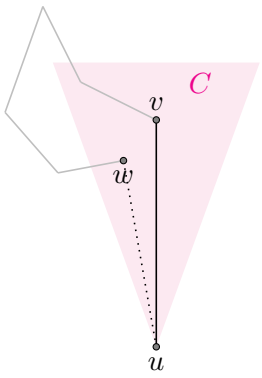



FIGURE 6.6: We can replace $uv$ with $uw$ to get a tree no heavier.

In the second case, when the path in the minimum spanning tree from $v$ to $w$ goes through $u$ (see Figure 6.7), we can replace $uv$ with $vw$ to get a lighter spanning tree. This is a contradiction to the minimality of the spanning tree and is thus an impossible case.

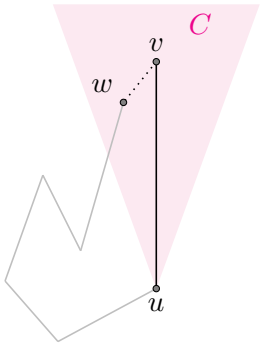



FIGURE 6.7: We can replace $uv$ with $vw$ to get a lighter tree.

Since we can fix up the first case, and the second is impossible, we have shown how that we can replace $uv$ with $uw$ to get another minimum spanning tree. □

THEOREM 19. *If $\theta < \pi/3$, a minimum spanning tree of $V$ is contained in $\mathcal{MBDG}(V)$.*

PROOF. It is a known fact that the Delaunay Triangulation $\mathcal{DT}(V)$ contains a minimum spanning tree of $V$ (see [**1**]). Fix a minimum spanning tree of $\mathcal{DT}(V)$. Suppose $uv$ is a minimum spanning tree

edge that is not in $\mathcal{MBDG}(V)$. It is therefore not fully protected. Without loss of generality, say it is not protected at $u$. Look at the cone $C \in \mathcal{C}_{u,\kappa}$ that contains $uv$. Let $um$ be the middle edge in $C$. Since the angle measure of $C$ is less than $\pi/3$, and by the definition of the middle edge which says $|um| \leq |uv|$, we can replace $uv$ with $um$ by Lemma 9 to get another minimum spanning tree. Since the angle measure of $C$ is less than $\pi/3$, there can only be one such edge $uv$ in $C$ that needs replacement, by Lemma 8. This says that we are replacing at most one minimum spanning tree edge with $um$, to get another minimum spanning tree. Repeating this process for all minimum spanning tree edges that are not fully protected, we will trade a set of $k \geq 0$ distinct minimum spanning tree edges that are not fully protected with $k$ distinct middle edges to get another minimum spanning tree; one that is contained in $\mathcal{MBDG}(V)$. □

COROLLARY 6. *If $\theta < \pi/3$, $\mathcal{LMBDG}(V)$ has a weight no more than $2r + 1$ times that of a minimum spanning tree of $V$.*

PROOF. This follows immediately from Theorem 19. □

Okay! Let us get back to the task at hand and discuss the efficiency of construction.

The following is derived from Lemma 3.3 in [**9**], along with final remark (A) in the same paper: "The method presented in Section 3 is quite general. It can be applied to any planar graphs $H, F$ embedded in the plane, where $H$ is a connected subgraph of $F$ and $F$ is triangulated outside $H$, in order to construct a subgraph $G$ of $F$ that approximates $F$ and has length $O(|H|)$." If use the Delaunay Triangulation of $V$ to triangulate $\mathcal{MBDG}(V)$ outside of $MST(\mathcal{MBDG}(V))$, we can then say the following:

THEOREM 20. *Given $\mathcal{DT}(V)$ and $\mathcal{MBDG}(V)$, the graph $\mathcal{LMBDG}(V)$ can be constructed in $O(n)$ time.*

Without $\mathcal{DT}(V)$, we can also construct $\mathcal{LMBDG}(V)$ in $O(n)$ time by using Chazelle's Linear Time Triangulation Algorithm [**7**] on all the faces of $\mathcal{MBDG}(V)$.

COROLLARY 7. *$\mathcal{LMBDG}(V)$ can be constructed in $O(n \lg n)$ time, and, if given the Delaunay Triangulation $\mathcal{DT}(V)$, $O(n)$ time.*

PROOF. This follows from Theorems 9 and 20. □

We have finally shown an efficient construction for a Bounded Degree and Light Planar Spanner. Let us now consider how to route on it, but first, how to augment the vertices with more information to facilitate this routing.

## 6.2 $weight$ Refers to Face Paths

So what exactly are the paths from $u$ to $v$ whose lengths are referred to by $weight(uv)$ if not the shortest paths? By induction, one derives that these are face paths in $\mathcal{LMBDG}(V)$ from $u$ to $v$ with a stretch factor of $1 + 1/r$. Let us state this more formally.

THEOREM 21. *Let $uv$ be an excluded settled edge. There is a face path in $\mathcal{LMBDG}(V)$ from $u$ to $v$ of length $weight(uv) \leq (1 + 1/r)\,|uv|$.*

PROOF. If $uv$ is the first excluded settled edge, then all edges of $\partial P(u, v)$ are included. By planarity, no edge will be added into the interior of the cycle consisting of $uv$ and $\partial P(u, v)$ once $uv$ is settled, and thus $uv$ will be a chord on the face in $\mathcal{LMBDG}(V)$ that coincides with $\partial P(u, v)$. Thus, $\partial P(u, v)$ is a face path in $\mathcal{LMBDG}(V)$ from $u$ to $v$ with a length of $weight(uv) \leq (1 + 1/r)\,|uv|$.

Consider now if $uv$ were an arbitrary excluded edge. Some edges of $\partial P(u, v)$ may thus be excluded. If none are excluded, then analogously to the first excluded settled edge, $\partial P(u, v)$ is a face path in $\mathcal{LMBDG}(V)$ from $u$ to $v$ with a length of $weight(uv)$. However, if some edges are excluded, then, by induction, for each excluded edge $pq$ along $\partial P(u, v)$, there is a face path in $\mathcal{LMBDG}(V)$ from $p$ to $q$ with a length of $weight(pq) \leq (1 + 1/r)\,|pq|$ (see Figure 6.8).

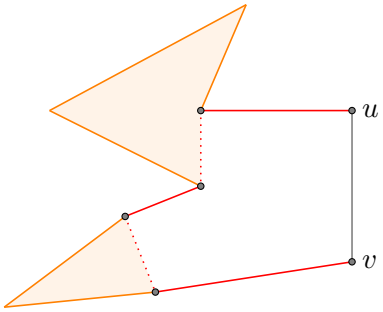



FIGURE 6.8: Excluded edges along $\partial P(u, v)$ have face paths inductively.

Replacing all $pq$ in $\partial P(u, v)$ by their face paths of weight at most $weight(pq)$, and since no edge will be added into the interior of the cycle consisting of $uv$ and $\partial P(u, v)$ once $uv$ is settled, $\partial P(u, v)$ with its excluded edges replaced by their face paths is a face path in $\mathcal{LMBDG}(V)$ from $u$ to $v$ with a length of $weight(uv) \leq (1 + 1/r)\,|uv|$.

Thus, if $uv$ is an excluded settled edge, there is a face path in $\mathcal{LMBDG}(V)$ from $u$ to $v$ of length $weight(uv) \leq (1 + 1/r)\,|uv|$. □

## 6.3 Signposts II

We can parlay the existence of these $(1 + 1/r)$-paths in the following way: whenever an edge is first determined to be excluded from $\mathcal{LMBDG}(V)$, we can store it as a *wraith-edge* at each of its endpoints, just like how edges that are not fully protected are stored as ghost-edges at the endpoint that protects them, in the construction of $\mathcal{MBDG}(V)$.

Let $uv$ be some excluded edge. Then, at $u$, we store $uv$ as a wraith-edge, along with one bit to indicate whether the starting edge of the $(1 + 1/r)$-path is the edge that is clockwise to $uv$ at $u$, or counterclockwise. By convention, we will use 1 to indicate clockwise and 0 otherwise. At $v$ we store the wraith-edge $uv$ with the opposite bit.

THEOREM 22. *$\mathcal{LMBDG}(V)$ stores $O(1)$ information at each vertex.*

PROOF. At each vertex, and for some adjustable parameter $\kappa$, there are at most $5\kappa$ edges, each with a constant number of bits to identify if they are extreme, middle, or penultimate, and $5\kappa$ ghost-edges and wraith-edges (those that are not fully protected, and those that are, but are pruned from $\mathcal{MBDG}(V)$ to give $\mathcal{LMBDG}(V)$), each with a constant number of bits to differentiate between them and to give a direction or orientation for a routing algorithm. The result follows that $\mathcal{LMBDG}(V)$ stores $O(1)$ information at each vertex. □

## 6.4 Routing on $\mathcal{LMBDG}(V)$

And finally, we are ready to route on $\mathcal{LMBDG}(V)$. When running Compromised Chew's Routing Algorithm on $\mathcal{MBDG}(V)$, at $v_i$, we take direct edges, Unguided Face Paths, and Guided Face Paths to search for some Delaunay Triangle $A_i$ and to route to one of its vertices. The direct edge in $\mathcal{MBDG}(V)$

may be a wraith-edge in $\mathcal{LMBDG}(V)$ instead, and there may be wraith-edges in the Unguided Face Paths and Guided Face Paths, if these edges have been pruned. When such a wraith-edge $uv$ is encountered in $\mathcal{LMBDG}(V)$ at $u$ and would have been routed along in $\mathcal{MBDG}(V)$, store $v$ and the orientation (1 or 0) of the $(1+1/r)$-path from $uv$ at $u$ in memory. Then, until $v$ has been reached, take the edge that is clockwise or counterclockwise to the edge arrived from, in accordance with the orientation stored (we consider $vu$ to be the edge arrived from at $u$). Figures 6.9 and 6.10 show examples of how Unguided and Guided Face Paths are routed respectively.

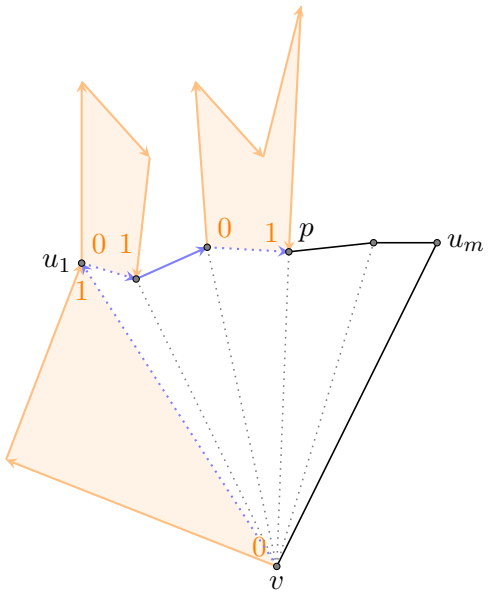



FIGURE 6.9: Routing an Unguided Face Path in $\mathcal{LMBDG}(V)$. The orange paths are $(1+1/r)$-paths of their corresponding edges.

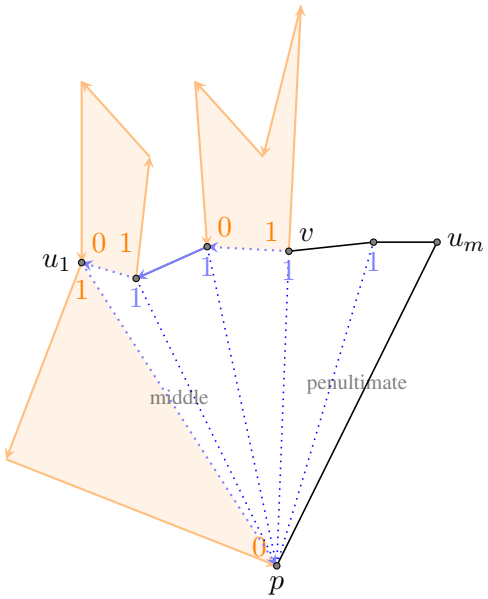



FIGURE 6.10: Routing a Guided Face Path in $\mathcal{LMBDG}(V)$. The orange paths are $(1 + 1/r)$-paths of their corresponding edges.

LEMMA 10. *Compromised Chew's Routing Algorithm on $\mathcal{LMBDG}(V)$ uses $O(1)$ memory.*

PROOF. While routing along a $(1 + 1/r)$-path, it is clear that we will not encounter any $(1 + 1/r)$-subpaths for any edge in the former $(1 + 1/r)$-path. This then follows from Theorem 15, since the only additional memory needed at any point in time is a constant amount to navigate a $(1 + 1/r)$-path. Compromised Chew's Routing Algorithm on $\mathcal{LMBDG}(V)$ thus uses $O(1)$ memory. □

THEOREM 23. *Compromised Chew's Routing Algorithm on $\mathcal{LMBDG}(V)$ is an $O(1)$-memory deterministic 1-local routing algorithm with a routing ratio no more than* $5.90(1+1/r)\max(\pi/2, \pi\sin(\theta/2)+1)$.

PROOF. The memory requirement follows from Lemma 10, while determinism and 1-locality follow from the workings of the algorithm. Lastly, the routing ratio follows from Theorem 14 since taking the $(1 + 1/r)$-paths from $u$ to $v$ instead of edges $uv$ multiplies the routing ratio by $(1 + 1/r)$. Compromised Chew's Routing Algorithm is thus an $O(1)$-memory deterministic 1-local routing algorithm on $\mathcal{LMBDG}(V)$ with a routing ratio no more than $5.90(1 + 1/r)\max(\pi/2, \pi\sin(\theta/2) + 1)$. □

CHAPTER 7

# Conclusion

---

We began with asking the question: Are there good planar networks that are cheap to build, and that can also be efficiently navigated without a map?

We then showed in Chapter 3 how to obtain in $O(n \lg n)$ time $\mathcal{BDG}(V)$, a bounded degree subgraph of the Delaunay Triangulation $\mathcal{DT}(V)$. In Chapter 4, we showed a scheme to place a constant number of signposts on the vertices of $\mathcal{BDG}(V)$, leaving the navigator clues to reconstruct the Delaunay Triangulation $\mathcal{DT}(V)$ on the marked and bounded degree graph $\mathcal{MBDG}(V)$. Using these clues, we showed in Chapter 5 how a navigator with a limited memory could simulate an efficient routing on the Delaunay Triangulation $\mathcal{DT}(V)$, which is known, on our construction $\mathcal{MBDG}(V)$ instead, and not only that, the efficiency of the simulation only experiences a constant blowup of the original efficiency. Finally, in Chapter 6, we showed how to obtain in $O(n)$ extra time $\mathcal{LMBDG}(V)$, a bounded weight subgraph of $\mathcal{MBDG}(V)$. We showed how the navigator could adjust the beforementioned routing strategy slightly to navigate this new terrain, again at only a constant blowup of the original efficiency.

Thus, $\mathcal{LMBDG}(V)$ is the answer to our question; it is a good planar network that is cheap to build, and that can also be efficiently navigated without a map. This result is stated more formally in Theorems 18 and 23. Efficiency of the construction is mentioned in Corollary 7, and a tighter bound is given on the weight in Corollary 6.

## 7.1 Further Work

Having answered this single question, we close with asking a few unanswered ones.

### 7.1.1 Tighter Bounds on the Routing Ratio of $\mathcal{LMBDG}(V)$

[**3**] gives a $5.72$ lower bound on the routing ratio of Chew's Routing Algorithm on the Delaunay Triangulation. Can we give a greater lower bound on the routing ratio of Compromised Chew's Routing Algorithm on $\mathcal{LMBDG}(V)$? Moreover, [**3**] also gives a $1.70$ lower bound on the routing ratio of any deterministic $k$-local routing algorithm on the Delaunay Triangulation. Can we give a lower bound that applies to the routing ratio of any such algorithm on $\mathcal{LMBDG}(V)$?

### 7.1.2 Navigating Unmarked Terrain

We have stated in Chapter 2 that it is more attractive when an online routing algorithm does not make use of auxiliary information attached to the vertices of the graph routed on. However, the success of Compromised Chew's Routing Algorithm on $\mathcal{LMBDG}(V)$ heavily relies on this kind of information placed at each vertex. More precisely, $O(1)$ words are placed at each vertex. Can one design an online routing algorithm on $\mathcal{LMBDG}(V)$, if not any other bounded degree and light planar spanner, that does not use this kind of auxiliary information?

### 7.1.3 Pushing the Degree Down

While [**4**] gives a degree bound of $27$, which $\mathcal{LMBDG}(V)$ bests, the second construction of a bounded degree planar spanner, seen in [**10**], gives a least upper bound on the degree of $24$, one less than that of $\mathcal{LMBDG}(V)$. Moreover, the degree bound in [**10**] grows fives times slower than that of $\mathcal{LMBDG}(V)$ as the adjustable parameter $\theta$ is tuned. Can we modify our construction, while preserving efficient navigability, to have a lower and slower growing degree bound?

### 7.1.4 Memoryless Navigators

Compromised Chew's Routing Algorithm on $\mathcal{LMBDG}(V)$ is an $O(1)$-memory routing algorithm, since, for starters, the last Delaunay Triangle used to make the last routing decision needs to be in memory. Can we design a routing algorithm on $\mathcal{LMBDG}(V)$ that is memoryless?

### 7.1.5 Tighter Weight Bound

Corollary 6 shows that if $\theta < \pi/3$, the weight of $\mathcal{LMBDG}(V)$ is at most $2r+1$ times that of a minimum spanning tree of $V$. Constrast this to the more general bound of $1.998(2r+1)\max(\pi/2, \pi\sin(\theta/2)+1)$ times that of a minimum spanning tree of $V$. There are a few questions to ask here. Is this really a threshold phenomenon, or do we simply not see a simple proof to unify the bounds for all choices of $\theta$? Is there an example, where $\theta \geq \pi/3$, and the $1.998(2r+1)\max(\pi/2, \pi\sin(\theta/2)+1)$ bound is tight? If not, can we prove the $2r+1$ bound for $\theta \geq \pi/3$?

### 7.1.6 Simulation of Alternative Algorithms

We currently simulate Compromised Chew's Routing Algorithm on $\mathcal{LMBDG}(V)$. This has a routing ratio of not more than $5.90$ on the Delaunay Triangulation, and this term multiplies into the bound of the routing ratio that applies to $\mathcal{LMBDG}(V)$. The Mixed Chord Arc Algorithm in [**2**] works very similarly to Chew's Routing Algorithm, which our approach is based of. However, it has a routing ratio of no more than $3.56$, which is much better than $5.90$. If we can simulate this algorithm on $\mathcal{LMBDG}(V)$, we can get a significantly better upper bound on the routing ratio on it.

# References


[1] BERG, M. D., CHEONG, O., KREVELD, M. V., AND OVERMARS, M. *Computational geometry: algorithms and applications*. Springer-Verlag TELOS, 2008.

[2] BONICHON, N., BOSE, P., CARUFEL, J., DESPRÉ, V., HILL, D., AND SMID, M. Improved routing on the delaunay triangulation. ESA.

[3] BONICHON, N., BOSE, P., DE CARUFEL, J.-L., PERKOVIĆ, L., AND VAN RENSSEN, A. Upper and lower bounds for online routing on delaunay triangulations. *Discrete & Computational Geometry 58*, 2 (2017), 482–504.

[4] BOSE, P., GUDMUNDSSON, J., AND SMID, M. Constructing plane spanners of bounded degree and low weight. *Algorithmica 42*, 3-4 (2005), 249–264.

[5] BOSE, P., AND MORIN, P. Online routing in triangulations. *SIAM journal on computing 33*, 4 (2004), 937–951.

[6] BOSE, P., AND SMID, M. On plane geometric spanners: A survey and open problems. *Computational Geometry 46*, 7 (2013), 818–830.

[7] CHAZELLE, B. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry 6*, 3 (1991), 485–524.

[8] GOTTLIEB, L.-A., AND RODITTY, L. Improved algorithms for fully dynamic geometric spanners and geometric routing. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms* (2008), Society for Industrial and Applied Mathematics, pp. 591–600.

[9] LEVCOPOULOS, C., AND LINGAS, A. There are planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees. *Algorithmica 8*, 1-6 (1992), 251–256.

[10] LI, X.-Y., AND WANG, Y. Efficient construction of low weight bounded degree planar spanner. In *International Computing and Combinatorics Conference* (2003), Springer, pp. 374–384.

[11] NARASIMHAN, G., AND SMID, M. *Geometric spanner networks*. Cambridge University Press, 2007.

[12] XIA, G. The stretch factor of the delaunay triangulation is less than 1.998. *SIAM Journal on Computing 42*, 4 (2013), 1620–1659.