Online Routing in Geometric Graphs

By

Patrick Ryan Morin

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Doctor of Philosophy

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

January 2001

© Copyright
2001, Patrick Ryan Morin

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis,

Online Routing in Geometric Graphs

submitted by

Patrick Ryan Morin

Dr. Frank Dehne
(Director, School of Computer Science)

Dr. Prosenjit Bose
(Thesis Supervisor)

Dr. Jörg-Rüdiger Sack
(Thesis Supervisor)

Dr. Binay Bhattacharya
(External Examiner)

Carleton University
January 2001

Abstract

This thesis considers the problem of finding a path from a source to a destination in a graph in which only local information is available. This type of routing problem occurs regularly in robotics, parallel and distributed computing, mobile networks, and everyday life. In particular, the research focuses on the case where the graph is geometric (nodes of the graph have locations in space) and planar (edges of the graph do not cross).

The results in this thesis fall into four categories:

1. natural and intuitive algorithms that work on some well known and structured geometric graphs,

2. algorithms for special classes of graphs that find paths approximating shortest paths,

3. algorithms for arbitrary planar graphs,

4. algorithms for embedding graphs nicely so that simple algorithms can be used to find paths between vertices, and

5. simulation results that help to determine which routing algorithms work best in different settings.

In studying these problems we draw on a wide range of techniques from computer science and mathematics, improve some previous results, and report a number of open problems and directions for continuing research.

Acknowledgements

I would like to thank Binay Bhattacharya, Jit Bose, David Bremner, Jason Gao, Silvia Götz, Danny Krizanc, Anil Maheshwari, NSERC, Jörg Sack, Ivan Stojmenović, Jorge Urrutia, and Tanya Whitehead for having contributed, in one way or another, to the completion of this thesis.

Contents

List of Figures

Guide to Notation

Here are some of the notations that are used frequently in this thesis, in no particular order.

t	the vertex being routed to (the target vertex)
s	the vertex at which routing begins (the source vertex)
$dist(a, b)$	the Euclidean distance between points a and b
$dist(e, a)$	the radius of the smallest circle centered at point a that intersects segment e.
$\overset{\text{ccw}}{\angle}\, a, b, c$	the counterclockwise angle formed by points a, b, and c
$\overset{\text{cw}}{\angle}\, a, b, c$	the clockwise angle formed by points a, b, and c
$\angle a, b, c$	$\min\{\overset{\text{cw}}{\angle}\, a, b, c, \overset{\text{ccw}}{\angle}\, a, b, c\}$
$\mathrm{E}[X]$	the expected value of the random variable X
G	a geometric graph
T	a triangulation
V	a set of vertices (points)
E	a set of edges (segments)
F	a set of faces (polygons)
$\mathrm{MST}(S)$	the minimum spanning tree of the point set S
$\mathrm{GG}(S)$	the Gabriel graph of the point set S
$\mathrm{VD}(S)$	the Voronoi diagram of the point set S
$\mathrm{DT}(S)$	the Delaunay triangulation of the point set S
$\|P\|$	the number of edges in the path P
$\mathrm{length}(P)$	the sum of the lengths of edges in the path P
$\preceq_p$	defined in Section 5.2.1 beginning on page 57

$\mathrm{FT}(G,p)$	the face tree of G with respect to the point p
H_x	the xth harmonic number $\sum_{i=1}^{x} 1/i$
$\lhd(u,v)$	the triangle to the left of the edge (u,v)
$\circ(u,v)$	the face to the left of edge (u,v)
$\perp(a,b)$	the perpendicular bisector of a and b
∂X	the boundary of the point set X
$P(G)$	the set of all pairs of vertices (u,v) in G such that u is in the same conntected component of G as v
$S(\mathcal{A},G)$	the set of all pairs of vertices (s,t) in G such that routing algorithm $\mathcal{A}$ succeeds in routing from s to t
$W(\mathcal{A},G,s,t)$	the walk taken by routing algorithm $\mathcal{A}$ when routing from s to t in G
$\mathrm{SEP}(G,s,t)$	the shortest Euclidean path from s to t in G
$\mathrm{SLP}(G,s,t)$	the shortest link path from s to t in G
$\mathrm{SR}(\mathcal{A},G)$	the success rate of $\mathcal{A}$ in G
$\mathrm{AED}(\mathcal{A},G)$	the average Euclidean dilation of $\mathcal{A}$ in G
$\mathrm{ALD}(\mathcal{A},G)$	the average link dilation of $\mathcal{A}$ in G

Chapter 1

Introduction

Path-finding or routing is a problem that is central to a number of fields, including geographic information systems, robotics, and communication networks. In many cases, knowledge about the environment in which routing takes place is incomplete, and the vehicle/robot/packet must find its way by learning the environment. Algorithms for routing in these types of environments are referred to as *online* algorithms.

In many applications involving routing, the networks in which routing takes place also contain geographic or geometric information. For instance, in road networks the locations of intersections can be described by longitude and latitude. In robotics, robots can sometimes determine their position by triangulating using three broadcast beacons in known locations. In wireless networks, hosts can be equipped with global positioning system devices so that they know their location in space.

In this thesis we study online routing problems in geometric networks, with a focus on the particular case in which the underlying network is planar. In the remainder of this chapter, we give a formal model for the study of these problems, give examples which motivate the study of these problems and provide an outline of the rest of the thesis.

1.1 The Model and Terminology

A *geometric graph* is a graph $G = (V, E)$ in which the vertex set V is a set of n points in $\mathbb{R}^d$, and the edge set E consists of m pairs from V. A *planar geometric graph* G is a geometric graph in which the vertex set is taken from $\mathbb{R}^2$, and the line segments defined by the edges of G intersect only at their endpoints (i.e., at the vertices of G). The edges of G partition the plane into a set of faces F, including an external face. In many instances, it is convenient to abuse notation slightly and say that a vertex v is in G, denoted $v \in G$, when in fact we mean $v \in V$. Similarly, for an edge e we use the notation $e \in G$ to mean $e \in E$. Along the same lines, for two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ we set $G_1 \cap G_2 = (V, E_1 \cap E_2)$.

The *neighbourhood* $\mathrm{N}(v)$ of a vertex $v \in V$ is the set of all vertices adjacent to v, i.e., all u such $(u, v) \in E$. A *walk* from s to t in G is a sequence of vertices $s = v_1, \ldots, v_k = t$ of G such that $(v_i, v_{i+1}) \in E$ for all $i \leq i < k$. A *path* in G from s to t is a walk from s to t such that $v_i \neq v_j$ for all $i \neq j$. The graph G is *connected* if for every pair of vertices $s, t \in E$, there exists a path in G from s to t. In the remainder of this thesis we will assume that all graphs are connected, unless specified otherwise.

The simplest routing problem we study is the *(online) point-to-point routing problem*, in which the input consists of two vertices $s, t \in V$. A solution to the problem consists of a walk from s to t. Initially, the algorithm for solving the point-to-point routing problem knows only s, t, and $\mathrm{N}(s)$. The algorithm learns $\mathrm{N}(v)$ only after examining (visiting) v.

We say that a routing algorithm $\mathcal{A}$ *succeeds* in routing from s to t if the algorithm terminates in a finite number of steps, otherwise $\mathcal{A}$ *fails* to route from s to t. We say that a routing algorithm $\mathcal{A}$ is *defeated* by a graph G if there exists a pair of vertices $s, t \in V$ such that $\mathcal{A}$ fails to route from s to t.

In this thesis we are primarily concerned with algorithms for routing in planar geometric graphs. As we will see in the following section, even by restricting our attention to planar graphs this work has a number of applications.

1.2 Motivation

There are a number of reasons for studying online routing problems in planar geometric graphs. First and foremost (for the author) is that they are interesting theoretical problems. In studying these problems we will make use of elements of computer science, computational geometry, planar graph theory, and even classical polytope theory.

However, for the more practically oriented reader, we provide three examples of applications of this work. The first example is an everyday occurence in which online routing is performed by human beings. The second example is from the field of fault-tolerant parallel computing. The third is from the field of mobile computing.

1.2.1 Walking in Strange Cities

Consider a tourist who is walking for their first time in the city of Toronto and trying to find their way to the CN Tower t (see Figure 1.1).[1] The CN Tower is tall enough that it can be seen from most anywhere in the city, and the streets of Toronto are straight enough that when standing at one intersection it is possible to see the next intersection in each direction. Since the person is on foot, highways and the resulting over and underpasses can be ignored.

Thus, Toronto can be represented by a planar geometric graph G in which the vertices represent intersections and the edges represent portions of streets joining two intersections. The problem of finding a path to the CN tower then becomes a problem of online routing in a planar geometric graph.

We can hardly expect our pedestrian to execute an elaborate algorithm to find their way to the CN tower. Rather, we expect that our pedestrian will use some greedy heuristic algorithm $\mathcal{A}$ such as always moving to the next intersection that takes her closest to t. Thus, studying and understanding $\mathcal{A}$ can help in planning the layouts of city streets and of placing appropriate signage in cases when $\mathcal{A}$ fails.

The above example may seem overly-contrived, in particular since it is one of only

[1] The map of Toronto in Figure 1.1 is taken from excite Maps, online at `http://city.net/maps/`
.

Figure 1.1: A map of Toronto.

a few examples in which the destination is always visible and hence known to the pedestrian. However, the technology of global positioning system (GPS) devices are becoming increasingly common. These are units that can determine their location on the surface of the earth through communication with satelites. For a modest price, handheld units are available for hikers, climbers and even pedestrians. Furthermore, it also expected that in a few years most new cars will come equipped with an onboard GPS device [46]. Thus, it may very well soon be the case that anyone driving a car or walking in a city can apply online routing techniques to reach a destination whose geographic location is known.

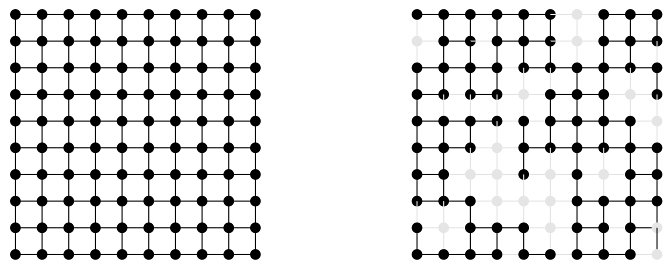

(a) (b)

Figure 1.2: A 10×10 mesh (a) without faults and (b) with vertex and edge faults.

1.2.2 Fault-Prone Meshes

An $n \times n$ *mesh* $M = (V, E)$ is a geometric graph in which the vertex set

$$V = \left\{ \begin{array}{ccc} (1,1), & \ldots, & (n,1), \\ \vdots & \ddots & \vdots \\ (1,n), & \ldots, & (n,n) \end{array} \right\} ,$$

and an edge is (v_1, v_2) is present in E if and only if $dist(v_1, v_2) = 1$, where $dist(x, y)$ denotes the Euclidean distance between x and y (see Figure 1.2 (a)). Meshes are an interconnection network studied extensively in the field of parallel algorithms (c.f., Leighton [53]). The folklore algorithm for routing between two vertices of a mesh moves the packet first to the correct column (x coordinate) and then to the correct row (y coordinate).

This simple algorithm works well provided that all the processing elements (vertices) and communication elements (edges) of the mesh are working properly. However, sometimes these elements may have faults (Figure 1.2 (b)), in which case this routing algorithm might fail. Furthermore, these failures are unpredictable, and vertices may not have information about which vertices and/or edges of the mesh have failed. In this case, the problem of routing between two vertices of M becomes an (online) point-to-point routing problem in a planar geometric graph.

1.2.3 Mobile *ad hoc* Wireless Networks

Mobile *ad hoc* networks (MANETs) consist of wireless hosts that communicate with each other in the absence of fixed infrastructure. Two nodes in a MANET can communicate if the distance between them is less than the minimum of their two broadcast ranges [5]. In many cases, MANETs are pieced together in an uncontrolled manner, changes in topology are frequent and unstructured, and hosts may not know the topology of the entire network. Thus, routing between the nodes of a MANET is often an online routing problem.

The *unit disk graph* $U = (V, E)$ is a geometric graph in which the edge (u, v) is present if and only if $dist(u, v) \leq 1$. Unit disk graphs are a generally accepted model of MANETs in which all nodes have the same broadcast range.

Note that, unlike the mesh, the positions of the vertices of U are arbitrary, and therefore U may not be planar. However, we will show that if U is connected, then a planar subgraph U' of U can be extracted using only local information. Indeed, the subset of edges U incident on a vertex u of U' can be computed given only $\mathrm{N}(v)$. Therefore, the problem of routing on U can be reduced to a problem of routing on a planar geometric graph.

For two vertices $u, v \in U$, let $\mathrm{disk}(u, v)$ be the disk with diameter (u, v). Then, the *Gabriel graph* [26] $\mathrm{GG}(S)$ is a geometric graph in which the edge (u, v) is present if and only if $\mathrm{disk}(u, v)$ contains no other points of S. Let $U' = \mathrm{GG}(S) \cap U$. The following lemma shows that the Gabriel graph is useful for extracting a connected planar subgraph from U.

Lemma 1. *If U is connected then U' is connected.*

Proof. It is well known that a minimum spanning tree $\mathrm{MST}(V)$ is a subset of $\mathrm{GG}(V)$ [70]. Thus, we need only prove that $\mathrm{MST}(V) \subseteq U$ if U is connected. Assume for the sake of contradiction that $\mathrm{MST}(V)$ contains an edge (u, v) whose length is greater than 1. Removing this edge from $\mathrm{MST}(V)$ produces a graph with two connected components, $C_u(V)$ and $C_v(V)$. Since U is connected it contains an edge (w, x) of length not greater than 1 such that $w \in C_u(V)$ and $x \in C_v(V)$. By replacing the edge (u, v) with (w, x) in $\mathrm{MST}(V)$ we obtain a connected graph on S with weight less than MST(V), a contradiction. □

Let (u, v) be an edge of U such that $(u, v) \notin \mathrm{GG}(V)$. Then, by the definition of $\mathrm{GG}(V)$ there exists a point w that is contained in the disk with u and v as diameter, and this point acts as a *witness* that $(u, v) \notin \mathrm{GG}(V)$. The following lemma shows that every such edge can be identified and eliminated by u and v using only local information.

Lemma 2. *Let u and v be vertices of U such that $(u, v) \notin \mathrm{GG}(V)$ and let w be a witness to this. Then $(u, w) \in U$ and $(v, w) \in U$.*

Proof. Let m be the midpoint of (u, v). Then $dist(u, m) \leq 1/2$, $dist(v, m) \leq 1/2$ and $dist(w, m) \leq 1/2$. Therefore, by the triangle inequality, $dist(u, w) \leq 1$, $dist(v, w) \leq 1$ and (u, w) and (v, w) are in U. □

Thus, upon reaching a vertex $v \in S$, a packet can eliminate the edges incident on v that are not in U' by simply eliminating any edge that is not in $\mathrm{GG}(\mathrm{N}(v) \cup \{v\})$. Lemma 1 guarantees that if we apply this algorithm to each vertex of U then the resulting graph is connected. Since $\mathrm{GG}(V)$ is planar [67, 65, 26], U' is also planar. Thus, we have reduced the problem of routing in a unit graph to one of routing in a planar geometric graph. This reduction can be used as the basis of routing algorithm for MANETs [12, 47].

1.3 Bibliographic Notes

The example used in Section 1.2.1 of a pedestrian walking to the CN tower appears in the paper by Kranakis *et al.* [52]. The technique used to extract a connected planar subgraph from a unit graph described in Section 1.2.3 is described by Bose *et al.* [12]. The same reduction is also discussed by Karp and Kung [47].

Chapter 2

Summary of the Thesis

In this chapter we summarize, chapter by chapter, the new results obtained in this thesis. This discussion is included mainly for evaluation purposes, and most of the contents in this chapter are available in the sections entitled "Summary and Open Problems" and "Bibliographic Notes" that are available at the end of each chapter.

In order to avoid too much repetition, we assume the reader knows some basic definitions that are provided in the subsequent chapters, but which are common in the field of computational geometry. Most of these terms can be found in the index at the end of this thesis.

2.1 Chapter 1

In Chapter 1, we introduce the online routing problem studied in this thesis and attempt to demonstrate its importance by way of 3 examples. The example of online routing in *ad hoc* wireless network, i.e., unit disk graphs, shows how a problem of routing on a non-planar graph that is common in wireless networks can be reduced to a problem of routing on a planar graph.

The paper by Bose *et al.* [12] describes the process of reducing the problem of routing on a unit disk graph to problem of routing on a planar graph and presents experimental results for several different routing algorithms. The reduction introduced by Bose *et al.* is also further studied by Karp and Kung [47].

2.2 Chapter 3

In Chapter 3, we study very simple routing algorithms. These are intuitive algorithms based on notions like distance and direction, and resemble techniques we would use in day to day life when getting from one place to another. Some of the algorithms we study have been studied previously and some are new.

The main focus of this chapter is to determine the limits of various classes of algorithms. We say that an algorithm is *memoryless* if the decision about which vertex to visit when situated at vertex v and destined for t is only a function of v, t, and $N(v)$. An algorithm is *randomized* if the next vertex visited is taken from a random distribution on $N(v)$, and is *deterministic* otherwise.

The main new results in this chapter are:

1. There is a deterministic memoryless routing algorithm (called GREEDY-COMPASS) that works for all triangulations.
2. There is no deterministic memoryless routing algorithm that works for all convex subdivisions.
3. There are algorithms that use randomization or memory that work for all convex subdivisions.

The results in Chapter 3 appear in the papers by Bose *et al.* [8, 10].

2.3 Chapter 4

In Chapter 4, we continue our study of routing algorithms with an emphasis on efficiency. We measure efficiency by means of the *competitive ratio*, i.e., the ratio between the length of the path found by the routing algorithm and the length of the shortest path. We study the problem under both the Euclidean distance measure and the link distance measure.

Our main results for Euclidean distance are:

1. There are online routing algorithms that achieve a constant competitive ratio for Delaunay, greedy, and minimum-weight triangulations. These algorithms are obtained by combining results of Papadmitriou and Yannakakis [69] for online routing in layered digraphs with results of Dobkin *et al.* [21] and Das and Joseph [16] for planar spanners.

2. There is no algorithm that achieves a competitive ratio of $o(\sqrt{n})$ for all triangulations with n vertices.

For the link distance measure we obtain the following result:

1. Unlike the Euclidean case, there is no algorithm that achieves a competitive ratio of $o(\sqrt{n})$ for Delaunay, greedy, or minimum-weight triangulations.

The contents of Chapter 4 are reported in the papers by Bose *et al.* [8, 10].

2.4 Chapter 5

In Chapter 5, we stop considering special classes of input graphs and instead consider general planar graphs. We also consider two other routing problem, *broacasting* (visiting every vertex of the graph), and *geocasting* (visiting all vertices of the graph contained in a specific geographic region).

The work in this chapter is an extension and refinement of the pioneering work by de Berg *et al.* [19] on traversing a planar subdivision using only a constant amount of additional storage. The results we obtain are the following:

1. There is an $O(1)$ memory routing algorithm for routing between any two vertices of any connected planar geometric graph. (This result is not new; a different algorithm achieving the same result was previously reported by Kranakis *et al.* [52].)

2. There is an $O(1)$ memory broadcasting algorithm that works for any connected planar graph and uses only $O(n \log n)$ steps.

3. There is an $O(1)$ memory geocasting algorithm that works for any connected planar graph and requires only $O(k \log k)$ steps, where k is the complexity of all faces of the graph that intersect the geocasting region.

In addition to this, our results have the effect of reducing the running time of the original algorithm of de Berg *et al.* from $O(n^2)$ to $O(n \log n)$.

The results in Chapter 5 appear in the paper by Bose and Morin [11] in the context of traversing a planar subdivision stored in a pointer based representation.

2.5 Chapter 6

In Chapter 6, we consider the problem of embedding a non-geometric planar graph so that routing algorithms can be used to route between its vertices. We achieve the following results:

1. If we allow arbitrarily precise coordinates, it is possible to obtain a planar embedding of any planar graph so that a (rather complicated) memoryless routing algorithm can perform shortest path routing on the graph.

2. Any graph that supports a convex embedding (one in which all faces are convex) can be embedded so that a simple memoryless routing algorithm can be used to route between its vertices.

3. There is a deterministic memoryless routing algorithm that works for all regular subdivisions.

This second result makes use of two classical results, one from graph theory (Tutte's convex-embedding theorem) and one from polytope theory (Steinitz' theorem) and provides a link between the two. A byproduct of our proof is that the set of graphs with convex embeddings (so called subdivided circuit graphs) is exactly the set of subdivisions of lower convex hulls of 3-polytopes.

2.6 Chapter 7

In Chapter 7, we describe experimental results for some of the routing algorithms described in this thesis. These results are for randomly generated graphs of different types, including Delaunay triangulations, Graham triangulations, meshes with faults, and unit disk graphs.

Some of the results in Chapter 7 appear in the papers by Bose *et al.* [10, 12].

2.7 Summary

This thesis represents an in-depth study of online routing problems in planar geometric graphs. Table 2.1 shows the results obtained in this thesis for routing on different types of graphs, along with a reference to the chapter containing the result. A "Yes" indicates that a routing algorithm is known that achieves this result while a "No" indicates a negative result showing that no such routing algorithm is possible. An arrow in a reference indicates that the result is implied by the more general result pointed to by the arrow. An F indicates that the result is trivial, and/or folklore.

Overall, the primary contribution of this thesis is a better understanding of which routing algorithms can be used in different contexts, and what sorts of guarantees we can expect from these algorithms, both in terms of efficiency and success rates. In addition to this, we provide some high-level experimental results that give evidence about how these algorithms might perform in an actual implementation. We hope that all our results, both theoretical and experimental, can help serve as a guide to practitioners about which algorithms make suitable candidates for implementation in a particular context.

Class of graphs	Deterministic oblivious		Randomized oblivious[a]		Constant memory	
DT	Yes	$\downarrow$	Yes	$\leftarrow$	Yes	$\downarrow$
GT/MWT	Yes	$\downarrow$	Yes	$\leftarrow$	Yes	$\downarrow$
Triangulations	Yes	Ch. 3	Yes	$\leftarrow$	Yes	$\downarrow$
Conv. Subdv.	No	Ch. 3	Yes	Ch. 3	Yes	$\downarrow$
Planar graphs	No	$\uparrow$	No	F	Yes	Ch. 5

Class of graphs	Euclidean competitive		Link competitive		Const. Mem. broadcasting	
DT	Yes	Ch. 4	No	Ch. 4	Yes	$\downarrow$
GT/MWT	Yes	Ch. 4	No	Ch. 4	Yes	$\downarrow$
Triangulations	No	Ch. 4	No	$\uparrow$	Yes	$\downarrow$
Conv. Subdv.	No	$\uparrow$	No	$\uparrow$	Yes	$\downarrow$
Planar graphs	No	$\uparrow$	No	$\uparrow$	Yes	Ch. 5

[a] In this column, we consider only algorithms that use a constant number of random bits per step. Otherwise, it is well known that a random walk on any graph G will eventually visit all vertices of G.

Table 2.1: Summary of result for online routing in planar geometric graphs.

Chapter 3

Simple Routing Algorithms

In this chapter we consider simple routing algorithms much like those we use every day in finding our way from place to place. Two notions occur frequently in our study, namely *distance* and *direction*. We will find that most natural routing algorithms are based on one or both of these concepts.

We begin our discussion with some definitions related to various specializations of geometric graphs and give a classification of online routing algorithms based on their use of memory and randomization. We then discuss some simple routing algorithms and describe some of the inherent limitations of simple algorithms.

3.1 Preliminaries

A *convex subdivision* is a planar geometric graph in which each face is a convex polygon, except the outer face, which is the complement of a convex polygon. A *triangulation* is a convex subdivision in which each interior face is a triangle.[1] Stated another (equivalent) way, a triangulation is a planar geometric graph with a maximal number of edges.

We say that a triangulation T is a *Delaunay triangulation* if it has the property that for every internal face (triangle) t of T, the circle that circumscribes t does not

[1] Note that this is different than the graph theoretic notion of a triangulation, in which every face is a triangle. In graph theory terms, what we are describing is a near-triangulation.

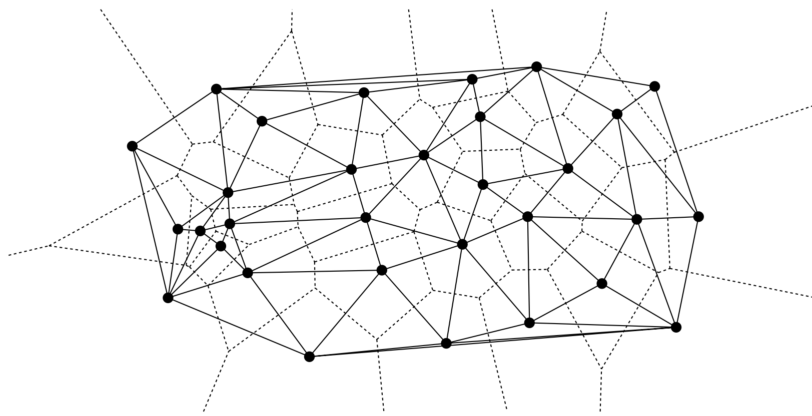


Figure 3.1: A Voronoi diagram and its dual Delaunay triangulation.

contain any point of V in its interior. Note that under this definition, the Delaunay triangulation of a point set may not be unique if four or more vertices of T are cocircular.

For a point set $S = \{p_1, \ldots, p_n\}$, the *Voronoi diagram* is a partitioning of the plane in to regions $R_1, \ldots, R_n$ where R_i is the locus of points that are closer to p_i than to any other point of S. The Voronoi diagram is a planar geometric graph in which some of the vertices are "points at infinity," and the edges of the Voronoi diagram are segments of the perpendicular bisectors of pairs of points in S. When no four points of S are cocircular, the Delaunay triangulation with vertex set S is the straight-line face dual of the Voronoi diagram of S and *vice versa* [6]. A Voronoi diagram and its dual Delaunay triangulation are shown in Figure 3.1.

A convex subdivision $S = (V, E)$ is a *regular subdivision* if and only if V and E are the orthogonal projection of the vertices and edges, respectively, of the lower convex hull of some 3-dimensional polytope P onto the x, y-plane. If S is a triangulation, then it is a *regular triangulation*. Delaunay triangulations are a special case of regular triangulations in which all the vertices of P are on a paraboloid.

The *greedy triangulation* $T = (V, E)$ of a point set $V = \{v_1, \ldots, v_n\}$ is defined by the following greedy (hence the name) incremental algorithm for its construction. Initially, the edge set E is empty. At each step, find the closest pair (v_i, v_j) of vertices in V such that the line segment (v_i, v_j) does not properly intersect any edge already in E and add the edge (v_i, v_j) to E. The algorithm terminates when no more edges can be added.

A triangulation $T = (V, E)$ is a *minimum-weight triangulation* if it minimizes the sum $\sum_{(u,v)\in E} dist(u, v)$. Minimum-weight triangulations are an especially elusive construct, and very little is known about their properties. In particular, the existence of a polynomial time algorithm for constructing the minimum-weight triangulation of a point set is still an open question.

Okabe *et al.* [67] provide an encyclopedic treatment of Delaunay triangulations, Voronoi diagrams, and their variants and applications. A discussion of regular triangulations (and more generally regular subdivisions) can be found in the book by Ziegler [81]. Greedy and minimum-weight triangulations are covered in Preparata and Shamos [70].

3.2 Classification of Routing Algorithms

In this section we give formal definitions of routing algorithms and classify routing algorithms based on their use of randomization and/or memory. For the most part, we will not work with these formal definitions as they are too cumbersome. However, we include them for the sake of mathematical rigour.

3.2.1 Deterministic Memoryless Algorithms

We say that an online routing algorithm $\mathcal{A}$ is *deterministic and memoryless* if the next node visited from a vertex v depends only on v and $\mathrm{N}(v)$, i.e., we can define $\mathcal{A}$ by a transition function δ of the form

$$\delta : [\mathbb{R}^2, (\mathbb{R}^2)^*, \mathbb{R}^2] \to \mathbb{R}^2 \tag{3.1}$$

The first argument of δ corresponds to the vertex v currently being visited, the second argument corresponds to $\mathrm{N}(v)$, the third corresponds to the destination t, and the output of δ is a point in $\mathrm{N}(v)$ that will be visited next.

If we define δ^i as

$$\delta^i(s,t) = \begin{cases} s & \text{if } i = 0 \\ \delta(\delta^{i-1}(s,t), \mathrm{N}(\delta^{i-1}(s,t)), t) & \text{otherwise} \end{cases} \tag{3.2}$$

then we see that algorithm $\mathcal{A}$ succeeds in routing from s to t if and only if there exists an $i \geq 0$ such that $\delta^i(s,t) = t$.

The distinguishing characteristic of memoryless algorithms is that if they visit the same vertex twice, then they have failed, since they are stuck in a loop. More formally, $\mathcal{A}$ fails to route from s to t if and only if $\delta^i(s,t) = \delta^j(s,t) \neq t$, for some $i \neq j$.

3.2.2 k-Bit Randomized Memoryless Algorithms

A routing algorithm $\mathcal{A}$ is a *k-bit randomized memoryless* routing algorithm if the next vertex visited from a vertex v is a function only of v, $\mathrm{N}(v)$, and k random bits r. I.e., if the transition function δ of $\mathcal{A}$ is of the form

$$\delta : [\mathbb{R}^2, (\mathbb{R}^2)^*, \mathbb{R}^2, \{0,1\}^k] \to \mathbb{R}^2 \quad . \tag{3.3}$$

The first three arguments and the output of δ are as before, while the fourth argument represents a random k bit string.

We say that a randomized memoryless routing algorithm fails if the probability that the algorithm reaches t when beginning at s is 0. More formally, we can define

$$\delta^i(s,t,R) = \begin{cases} s & \text{if } i = 0 \\ \delta(\delta^{i-1}(s,t,R), \mathrm{N}(\delta^{i-1}(s,t,R)), t, r_i) & \text{otherwise} \end{cases} \quad , \tag{3.4}$$

where $R = r_1, r_2, \ldots$ is an infinite sequence of k bit strings. Then $\mathcal{A}$ fails to route from s to t if and only if

$$\delta^i(s,t,R) \neq t \tag{3.5}$$

for all i and all R. At first it may seem that this definition of failure is overly weak, since it may be the case that only one of infinitely many R causes i to succeed. However, it is not difficult to verify that if $\mathcal{A}$ is not defeated by a graph G then for any pair of vertices s and t of G the probability that $\mathcal{A}$ succeeds is 1. (This follows from the fact that $\mathcal{A}$ is memoryless.)

3.2.3 k Memory Algorithms

An algorithm $\mathcal{A}$ is a k memory algorithm if the next vertex visited after a vertex v is a function only of v, t, $\mathrm{N}(v)$, and M, where M is a memory of size k that contains information about previously visited vertices. More formally, the transition function δ of $\mathcal{A}$ must be of the form

$$\delta : [\mathbb{R}^2, (\mathbb{R}^2)^*, \mathbb{R}^2, \Sigma^k] \rightarrow \mathbb{R}^2 \quad . \tag{3.6}$$

The first two arguments as well as the output are the same as for other routing algorithms and the third argument represents the memory used by the algorithm. The set Σ is the alphabet of items representable by one unit of memory. For our purposes, Σ consists of the set of points in $\mathbb{R}^2$ as well as all $\log n$ bit integers.

Additionally, $\mathcal{A}$ defines a *carry-forward* function γ of the form

$$\gamma : [\mathbb{R}^2, (\mathbb{R}^2)^*, \mathbb{R}^2, \Sigma^k] \rightarrow \Sigma^k \tag{3.7}$$

that defines how $\mathcal{A}$ uses the memory M. We can then define

$$\gamma^i(s,t) = \begin{cases} \lambda & \text{if } i = 0 \\ \gamma(\delta^{i-1}(s,t), \mathrm{N}(\delta^{i-1}(s,t)), t, \gamma^{i-1}(s,t)) & \text{otherwise} \end{cases} \quad , \tag{3.8}$$

where λ denotes the empty string, and

$$\delta^i(s,t) = \begin{cases} s & \text{if } i = 0 \\ \delta(\delta^{i-1}(s,t), \mathrm{N}(\delta^{i-1}(s,t)), t, \gamma^{i-1}(s,t)) & \text{otherwise} \end{cases} \tag{3.9}$$

As with memoryless algorithms, $\mathcal{A}$ succeeds in routing from s to t if and only if there exists an $i \geq 0$ such that $\delta^i(s,t) = t$.

3.3 Routing on Triangulations

Finally, we are ready to study some routing algorithms. In an intuitive sense at least, triangulations are the most structured form of planar graph. It is therefore natural to expect that they represent easier inputs for online routing algorithms. With this in mind, we begin by studying the simplest routing algorithms for triangulations.

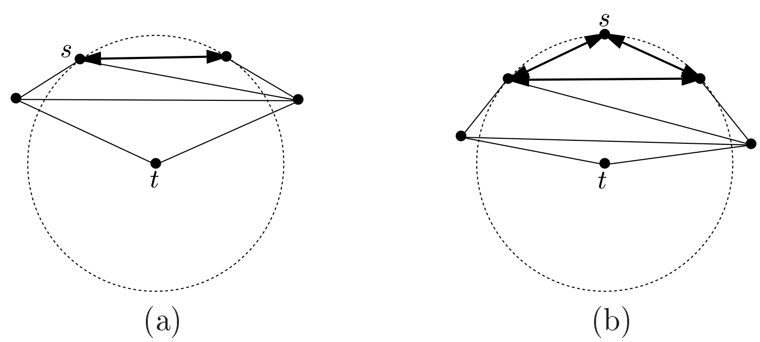


Figure 3.2: Triangulations that defeat the greedy routing algorithm.

Throughout the remainder of this thesis we will use terminology taken mostly from packet-switched data networks. Thus, we will describe algorithms as wanting to route a *packet* from s to t and *forwarding* or *moving* the packet between *vertices* or *nodes* in order to achieve this goal. We do this only because of the need to fix some terminology, and the reader should keep in mind that the algorithms we describe can be applied in more context than packet-switched data networks.

3.3.1 The greedy Algorithm

The GREEDY routing algorithm always moves a packet situated at the vertex v and destined for vertex t to the neighbor u of v that minimizes $dist(u,t)$. In the case of ties, one of the vertices is chosen arbitrarily. Formally, GREEDY is defined by the transition function

$$\mathrm{gdy}(v, \mathrm{N}(v), t) = u \in \mathrm{N}(v) : dist(u,t) \leq dist(w,t) \text{ for all } w \in \mathrm{N}(v) \ . \tag{3.10}$$

The GREEDY algorithm can be defeated by a triangulation T in two ways: (1) the packet can get trapped moving back and forth on an edge of the triangulation (Figure 3.2 (a)), or (2) the packet can get trapped on a cycle of three or more vertices (Figure 3.2 (b)).

Although GREEDY fails on some triangulations, it always works for Delaunay triangulations.

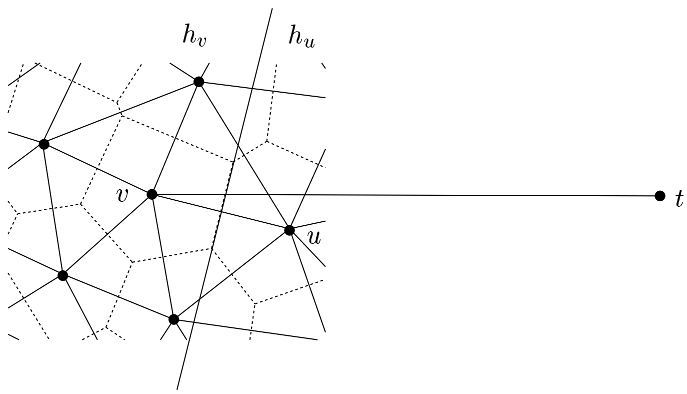


Figure 3.3: The proof of Theorem 1.

Theorem 1. *The* GREEDY *routing algorithm is a memoryless algorithm that is not defeated by any Delaunay triangulation.*

Proof. That GREEDY is a memoryless algorithm follows from (3.10).

Let $T = (V, E)$ be any Delaunay triangulation. We proceed by showing that every vertex v of T has a neighbor that is strictly closer to t than v is. Thus, at each routing step, the packet gets closer to t and therefore, after at most n steps, reaches t. Refer to Figure 3.3.

Let VD(V) be the Voronoi diagram of V and let e be the first edge of VD(V) intersected by the directed line segment (v, t). The edge e must exist, because v and t are contained in two different Voronoi cells. Furthermore, e is on the boundary of two Voronoi cells, one for v and one for some other vertex u, and the supporting line of e partitions the plane into two open half planes $h_v = \{p : dist(p, v) < dist(p, u)\}$ and $h_u = \{p : dist(p, u) < dist(p, v)\}$. Since the Voronoi diagram is the straight line face dual of the Delaunay triangulation, the edge $(u, v) \in T$. Also, by the choice of e, $t \in h_u$, i.e., $dist(u, t) < dist(v, t)$. $\square$

3.3.2 The compass Algorithm

The COMPASS routing algorithm always moves a packet situated at the vertex v to the neighbour u of v that minimizes the angle $\angle u, v, t$. In the case of ties, one of the

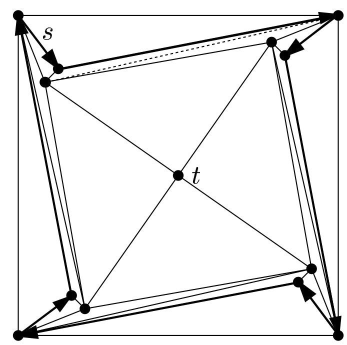


Figure 3.4: A triangulation that defeats the COMPASS routing algorithm.

(at most 2) vertices is chosen using some arbitrary deterministic rule. Formally, the algorithm is defined by the transition function

$$\mathrm{cmp}(v, \mathrm{N}(v), t) = u \in \mathrm{N}(v) : \angle u, v, t \leq \angle w, v, t \text{ for all } w \in \mathrm{N}(v) \ . \tag{3.11}$$

One might initially believe (as we did) that the COMPASS algorithm can always be used to find a path between any two vertices in a triangulation. However, the triangulation in Figure 3.4 defeats COMPASS. When starting from one of the vertices on the outer face of T, and routing to t, COMPASS gets trapped on the cycle shown in bold.

The following lemma shows that any triangulation that defeats COMPASS causes the packet to get trapped in a cycle. In the following, and in the remainder of this thesis, we use the notation $\mathrm{cmp}(v)$ as a shorthand for $\mathrm{cmp}(v, \mathrm{N}(v), t)$.

Lemma 3. *Let T be a triangulation that defeats* COMPASS*, and let t be a vertex such that* COMPASS *fails to route a packet to t when given some other vertex as the source. Then there exists a cycle $C = (v_0, \ldots, v_{k-1})$ ($k \geq 3$) in T such that $\mathrm{cmp}(v_i) = v_{i+1}$ for all $0 \leq i < k$.*[2]

Proof. Since T defeats COMPASS, and COMPASS is a memoryless algorithm, then either there is an edge (u, v) such that $\mathrm{cmp}(u) = v$ and $\mathrm{cmp}(v) = u$, or there is the situation described in the lemma. We prove that there can be no such edge (u, v). Suppose

[2] *Here, and in the remainder of this section, subscripts are taken* $\bmod k$.

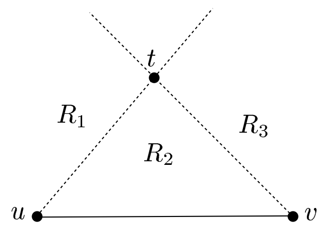


Figure 3.5: The proof of Lemma 3.

such an edge (u, v) does exist. Then there is a triangle (u, v, w) in T such that w is in the same closed half-plane bounded by the line through u and v as t. Referring to Figure 3.5, the vertex w must be in the one of the regions R_1, R_2, or R_3. Now, if w is in R_1 or R_2, then $\angle w, u, t < \angle v, u, t$ and if w is in R_3 then $\angle w, v, t < \angle u, v, t$. But this is a contradiction, since it violates the assumption that $\mathrm{cmp}(u) = v$ and $\mathrm{cmp}(v) = u$. $\square$

We call such a cycle, C, a *trapping cycle* in T for t. Next we characterize trapping cycles in terms of a visibility property of triangulations. Let t_1 and t_2 be two triangles in T. Then we say that t_1 *obscures* t_2 if there exists a ray originating at t that strikes t_1 first and then t_2. Let u and v be any two vertices of T such that $\mathrm{cmp}(u) = v$. Then define $\triangleleft(u, v)$ as the triangle of T that is contained in the closed half-plane bounded by the line through uv and that contains t. We obtain the following useful characterization of trapping cycles.

Lemma 4. *Let T be a triangulation that defeats* COMPASS *and let $C = v_0, \ldots, v_{k-1}$ be a trapping cycle in T for vertex t. Then $\triangleleft(v_i, v_{i+1})$ is either identical to, or obscures $\triangleleft(v_{i-1}, v_i)$, for all $0 \le i < k$.*

Proof. Refer to Figure 3.6. Assume that $\triangleleft(v_i, v_{i+1})$ and $\triangleleft(v_{i-1}, v_i)$ are not identical, otherwise the lemma is trivially true. Let w be the third vertex of $\triangleleft(v_i, v_{i+1})$. Then w cannot lie in the cone defined by t, v_i and v_{i+1} and having apex at v_i, otherwise we would not have $\mathrm{cmp}(v_i) = v_{i+1}$. But then the line segment joining w and v_{i+1} obscures v_i and hence $\triangleleft(v_i, v_{i+1})$ obscures $\triangleleft(v_{i-1}, v_i)$. $\square$

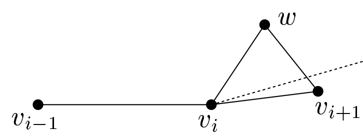


Figure 3.6: The proof of Lemma 4.

Edelsbrunner [22] shows that if T is a regular triangulation, then T has no set of triangles that obscure each other cyclically from *any* viewpoint. This result, combined with Lemma 4, proves our main result on the COMPASS routing algorithm.

Theorem 2. *The* COMPASS *routing algorithm is a memoryless algorithm that is not defeated by any regular triangulation.*

3.3.3 The greedy-compass Algorithm

So far we have seen GREEDY and COMPASS, routing algorithms based on distance and direction, respectively. Both of these algorithms are defeated by some triangulations, but in different ways. Next we show that, by combining elements of GREEDY and COMPASS, we obtain a memoryless routing algorithm that is not defeated by any triangulation.

Let $\mathrm{cw}(v)$ be the vertex $u \in \mathrm{N}(v)$ that minimizes the clockwise angle $\overset{\mathrm{cw}}{\angle}\, t, v, u$ and let $\mathrm{ccw}(v)$ be the vertex $u \in \mathrm{N}(v)$ that minimizes the counterclockwise angle $\overset{\mathrm{ccw}}{\angle}\, t, v, u$ (See Figure 3.7). We call $\mathrm{cw}(v)$ and $\mathrm{ccw}(v)$ the *compass neighbours* of v and denote by $\mathrm{CN}(v)$ the set $\{\mathrm{cw}(v), \mathrm{ccw}(v)\}$.

The GREEDY-COMPASS routing algorithm moves a packet situated at vertex v to $u \in \mathrm{CN}(v)$ such that $dist(u,t)$ is minimized, with ties being broken arbitrarily. Formally, GREEDY-COMPASS is defined by the transition function

$$\mathrm{gc}(v, \mathrm{N}(v), t) = u \in \mathrm{CN}(v) : dist(u,t) \leq dist(w,t) \text{ for all } w \in \mathrm{CN}(v)\ . \tag{3.12}$$

Theorem 3. *The* GREEDY-COMPASS *algorithm is a memoryless algorithm that is not defeated by any triangulation.*

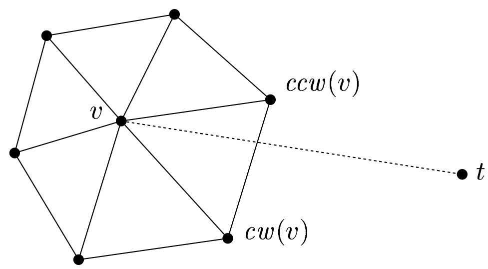


Figure 3.7: Definition of cw(v) and ccw(v).

Proof. Suppose, by way of contradiction that a triangulation T and a pair of vertices s and t exist such that GREEDY-COMPASS does not find a path from s to t.

The same argument used in the proof of Lemma 3 shows that there must be a cycle of vertices $C = v_0, \ldots, v_{k-1}$ of T such that GREEDY-COMPASS moves from v_i to v_{i+1} for all $0 \leq i \leq k$, i.e., GREEDY-COMPASS gets trapped cycling through the vertices of C.[3] Furthermore, exactly the same argument used in the proof of Lemma 4 shows that the destination t is contained in the interior of C.

Claim 1. *All vertices of C must lie on the boundary of a disk D centered at t.*

Proof (of claim). Suppose, by way of contradiction, that there is no such disk D. Then let D be the disk centered at t and having the furthest vertex of C from t on its boundary. Consider a vertex v_i in the interior of D such that v_{i+1} is on the boundary of D. (Refer to Figure 3.8.) Assume, wlog, that $v_{i+1} = \mathrm{ccw}(v_i)$. Then it must be that $\mathrm{cw}(v_i)$ is not in the interior of D, otherwise GREEDY-COMPASS would not have moved to v_{i+1}. But then the edge $(\mathrm{cw}(v_i), \mathrm{ccw}(v_i))$ cuts D into two regions, R_1 containing v_i and R_2 containing t. Since C passes through both R_1 and R_2 and is contained in D then it must be that C enters region R_1 at $\mathrm{cw}(v_i)$ and leaves R_1 at $v_{i+1} = \mathrm{ccw}(v_i)$. However, this cannot happen because both $\mathrm{cw}(\mathrm{cw}(v_i))$ and $\mathrm{ccw}(\mathrm{cw}(v_i))$ are contained in the halfspace bounded by the supporting line of $(\mathrm{cw}(v_i), \mathrm{ccw}(v_i))$ and containing t, and are therefore not contained in R_1. □

[3] Here, and in the remainder of this proof, all subscripts are taken mod$k$.

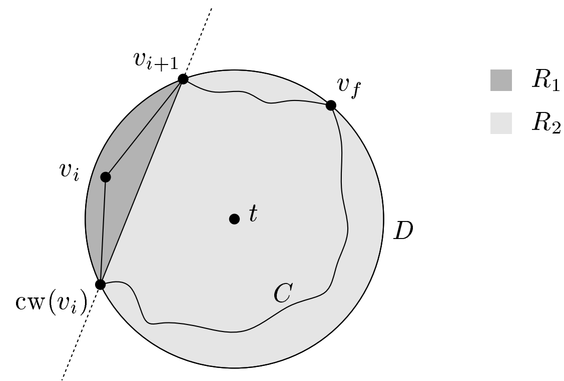


Figure 3.8: The proof of Theorem 3.

Thus, we have established that all vertices of C are on the boundary of D. However, since C contains t in its interior and the triangulation T is connected, it must be that for some vertex v_j of C, $\mathrm{cw}(v_j)$ or $\mathrm{ccw}(v_j)$ is in the interior of D. Suppose that it is $\mathrm{cw}(v_j)$. But then we have a contradiction, since the GREEDY-COMPASS algorithm would have gone to $\mathrm{cw}(v_j)$ rather than v_{j+1}. □

3.4 Routing on Convex Subdivisions

Our study of routing on triangulations has been quite successful, and we managed to find a deterministic memoryless routing algorithm that works on all triangulations. This success might make us optimistic that we can generalize this work to convex subdivisions. Unfortunately, this is not the case.

3.4.1 An Impossibility Result

Next we show that there is no generalization of GREEDY-COMPASS that will work for all convex subdivisions.

Theorem 4. *Every deterministic oblivious routing algorithm is defeated by some convex subdivision.*

Proof. We exhibit a finite collection of convex subdivisions such that any deterministic oblivious routing algorithm is defeated by at least one of them.

There are 17 vertices that are common to all of our subdivisions. The destination vertex t is located at the origin. The other 16 vertices $V = \{v_0, \ldots, v_{15}\}$ are the vertices of a regular 16-gon centered at the origin and listed in counterclockwise order.[4] In all our subdivisions, the even-numbered vertices $v_0, v_2, \ldots, v_{14}$ have degree 2. The degrees of the other vertices vary. All of our subdivisions contain the edges of the regular 16-gon.

Assume, by way of contradiction, that there exists a routing algorithm $\mathcal{A}$ that works for any convex subdivision. Since the even-numbered vertices in our subdivisions always have the same two neighbours in all subdivisions, $\mathcal{A}$ always makes the same decision at a particular even-numbered vertex. Thus, it makes sense to ask what $\mathcal{A}$ does when it visits an even-numbered vertex, without knowing anything else about the particular subdivision that $\mathcal{A}$ is routing on.

For each vertex $v_i \in V$, we color v_i black or white depending on the action of $\mathcal{A}$ upon visiting v_i, specifically, black for moving counterclockwise and white for moving clockwise around the regular 16-gon. We claim that all even-numbered vertices in V must have the same color. If not, then there exists two vertices v_i and v_{i+2} such that v_i is black and v_{i+2} is white. Then, if we take $s = v_i$ in the convex subdivision shown in Figure 3.9, the algorithm becomes trapped on one of the edges (v_i, v_{i+1}) or (v_{i+1}, v_{i+2}) and never reaches the destination t, contradicting the assumption that $\mathcal{A}$ works for any convex subdivision.

Therefore, assume wlog that all even-numbered vertices of V are black, and consider the convex subdivision shown in Figure 3.10. From this figure it is clear that, if we take $s = v_1$, $\mathcal{A}$ cannot visit x after v_1, since then it gets trapped among the vertices $\{v_{12}, v_{13}, v_{14}, v_{15}, v_0, v_1, x\}$ and never reaches t.

Note that we can rotate Figure 3.10 by integral multiples of $\pi/4$ while leaving the vertex labels in place and make similar arguments for v_3, v_5, v_7, v_9, v_{11}, v_{13} and v_{15}. However, this implies that $\mathcal{A}$ is defeated by the convex subdivision shown in Figure 3.11 since if it begins at any vertex of the regular 16-gon, it never enters the

[4] In the remainder of this proof, all subscripts are implicitly taken $\bmod 16$.

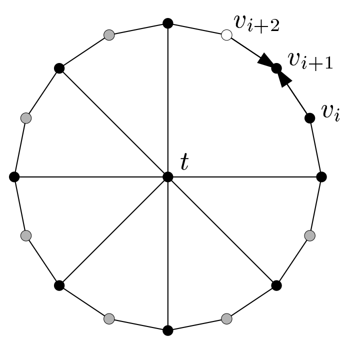


Figure 3.9: All vertices on the convex hull must have the same color.

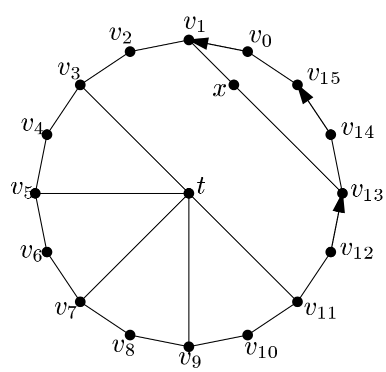


Figure 3.10: $\mathcal{A}$ cannot visit x after v_1.

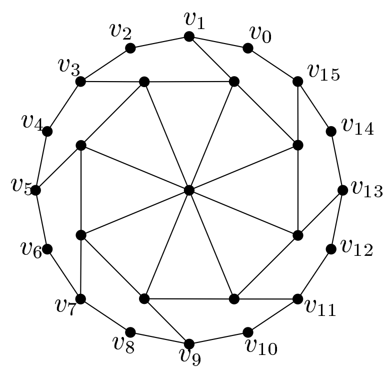


Figure 3.11: $\mathcal{A}$ is defeated by this subdivision.

interior of the 16-gon. We conclude that no oblivious online routing algorithm works for all convex subdivisions. □

The reader may notice that the graphs used in the proof of Theorem 4 are not strictly convex, i.e., they contain faces whose vertices have interior angle π. With a little more work and the addition of more vertices the graphs used in the proof can be made strictly convex, thus the theorem also holds for strictly convex subdivisions [8].

Intuitively, this proof shows that deterministic oblivious routing algorithms tend to be either left-biased or right-biased (black or white). In either case, an algorithm is defeated by a convex subdivision whose faces overlap cyclically, with the direction of the overlap (counterclockwise or clockwise) depending on whether the algorithm is left-biased or right-biased.

There are two avenues we can now pursue. We can allow more powerful algorithms, or we can restrict the class of convex subdivisions. In the next two sections we show that if we allow our algorithms to use randomization or memory, we can obtain an algorithm that is neither left-biased nor right-biased. Much later, in Section 6.3.3 we will see that if we restrict ourselves to regular convex subdivisions then a memoryless deterministic algorithm suffices.

3.4.2 The random-compass Algorithm

Next, we show that if we allow randomization, we can overcome the negative result of the previous section. We consider a 1-bit randomized memoryless routing algorithm.

The RANDOM-COMPASS algorithm moves a packet situated at vertex v to one of $\{\mathrm{cw}(v), \mathrm{ccw}(v)\}$ with equal probability. Formally, RANDOM-COMPASS is defined by the transition function

$$rcmp(v, \mathrm{N}(v), r) = \begin{cases} \mathrm{cw}(v) & \text{if } r = 0 \\ \mathrm{ccw}(v) & \text{if } r = 1 \end{cases} . \tag{3.13}$$

The following theorem illustrates the power gained by the introduction of randomness.

Theorem 5. *The* RANDOM-COMPASS *algorithm is a 1-bit randomized memoryless algorithm that is not defeated by any convex subdivision.*

Proof. That RANDOM-COMPASS is memoryless follows from its definition. That it is a 1-bit randomized algorithm follows from the fact that it selects from at most 2 choices at each step.

Assume, by way of contradiction, that there is a convex subdivision G with two vertices s and t such that the probability of reaching s from t using RANDOM-COMPASS is 0. Then there is a subgraph H of G containing s, but not containing t, such that for all vertices $v \in H$, $\mathrm{cw}(v) \in H$ and $\mathrm{ccw}(v) \in H$.

The vertex t is contained in some face f of H. We claim that this face must be convex. For the sake of contradiction, assume otherwise. Then there is a reflex vertex v on the boundary of f such that the line segment (t, v) does not intersect any edge of H. However, this cannot happen, since $\mathrm{ccw}(v)$ and $\mathrm{cw}(v)$ are in H, and hence v would not be reflex.

Since G is connected, it must be that for some vertex u on the boundary of f, $\mathrm{cw}(u)$ or $\mathrm{ccw}(u)$ is contained in the interior of f. But this vertex in the interior of f is also in H, contradicting the fact that f is a convex face of H. We conclude that there is no convex subdivision that defeats RANDOM-COMPASS. □

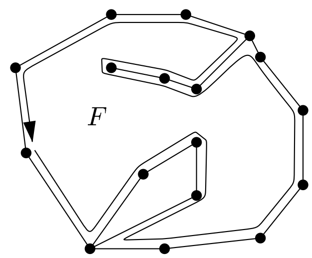


Figure 3.12: Traversal of the face f using the "right-hand rule."

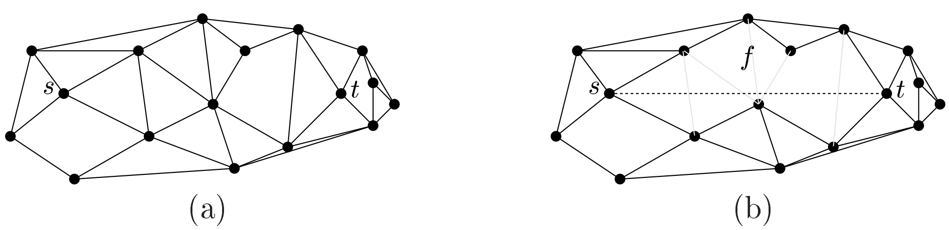


Figure 3.13: The planar geometric graphs (a) T and (b) T'.

3.4.3 The right-hand Algorithm

The folklore "right-hand rule" for exploring a maze states that if a player in a maze walks around never lifting her right-hand from the wall, then she will eventually visit every wall in the maze. More specifically, if the maze is the face of a connected planar geometric graph, the player will visit every edge and vertex of the face [6] (see Figure 3.12).

Consider the planar geometric graph T' obtained by deleting from T all edges that properly intersect the line segment joining s and t (see Figure 3.13). The vertices s and t are on the boundary of the same face f of T'. The RIGHT-HAND routing algorithm uses the right-hand rule to traverse the face f beginning at s and continuing until reaching t.

Theorem 6. *The* RIGHT-HAND *routing algorithm is an $O(1)$ memory routing algorithm that is not defeated by any convex subdivision.*

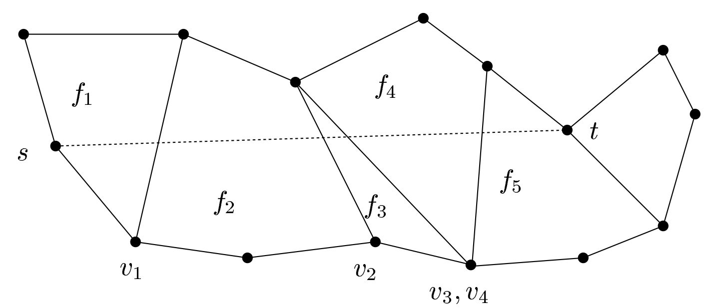


Figure 3.14: The proof of Theorem 6.

Proof. To see that RIGHT-HAND is an $O(1)$ memory algorithm we note that the traversal of the face f can be accomplished by remembering the segment (s,t) (in order to identify edges of T that intersect it), and the last vertex visited (in order to choose the next vertex visited by the right-hand rule).

To prove that RIGHT-HAND is not defeated by any convex subdivision we need only show that s and t are in the same connected components of T', since s and t are clearly on the same face of T'. Refer to Figure 3.14 for what follows. Let $f_1, \ldots, f_k$ be the faces of T intersected by (s,t) in order from s to t. Note that, for each $1 \leq i < k$, f_i and f_{i+1} have an edge in common, namely the edge intersected by (s,t). Let v_i be the vertex of f_i shared with f_{i+1} so that s, t, v_i is a right turn. Then, since each f_i is convex, there is a path P_i from v_{i-1} to v_i on the boundary of f_i that stays on the right side of the line l through s and t. Also because of convexity, there is a path P_0 from s to v_i that intersects l only at s and a path P_{k-1} from v_{k-1} to t that intersects l only at t. The path $P_0, P_1, \ldots, P_{k-1}$ is a path from s to t in T'. $\square$

3.5 Summary and Open Problems

In this chapter we have discussed memoryless, randomized memoryless and $O(1)$ memory algorithms for routing in triangulations and convex subdivisions. We have shown that the use of memory or randomization allows for routing algorithms that work on a larger class of planar graphs, and that the separation occurs somewhere

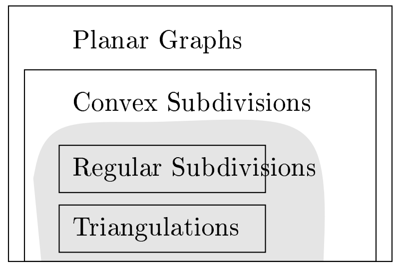


Figure 3.15: The limits of memoryless deterministic algorithms (shaded)

Class of graphs	Deterministic memoryless	Randomized memoryless	Deterministic w. memory
Delaunay triangulations	GREEDY	RANDOM-COMPASS	RIGHT-HAND
Regular triangulations	COMPASS	RANDOM-COMPASS	RIGHT-HAND
Triangulations	GREEDY-COMPASS	RANDOM-COMPASS	RIGHT-HAND
Convex subdivisions	Not possible	RANDOM-COMPASS	RIGHT-HAND

Table 3.1: Summary of results so far.

between triangulations and convex subdivisions (see Figure 3.15).[5]

Our results so far are summarized in Table 3.1.

We have shown that GREEDY, COMPASS, GREEDY-COMPASS, and RIGHT-HAND reach the destination vertex t after at most $O(n)$ steps. However, our analysis of RANDOM-COMPASS only shows that a message reaches t with probability 1. It does not give any upper bound on the expected number of steps when the random bits are taken uniformly at random from $\{0, 1\}$.

The example of a regular convex n-gon and some knowledge of probability can be used to show that, for some convex subdivisions, the expected number of steps

[5] In fact, in Chapter 6 we will see that it occurs somewhere between regular subdivisions and convex subdivisions.

required by the RANDOM-COMPASS can be in $\Omega(n^2)$. To see see this, note that routing between two diametrically opposed vertices in a convex n-gon using RANDOM-COMPASS is equivalent to repeatedly flipping a coin until the number of heads exceeds the number of tails by $n/2$, or vice-versa. It is a straightforward exercise in probability to prove that the expected number of coin tosses required is $\Theta(n^2)$.

Furthermore, the execution of RANDOM-COMPASS can be thought of as a random walk on a directed graph in which each vertex has out-degree 2 and that has a single sink, t. Using this view of RANDOM-COMPASS, combined with results on hitting times for random walks (c.f. Raghavan and Motwani [71]) it should be possible to prove that the expected number of steps is also $O(n^2)$ for any convex subdivision. These types of questions lead to the following open problem.

Open Problem 1. *Study the expected number of steps taken by* RANDOM-COMPASS *(or some other randomized routing algorithm) under various assumptions about the graph G.*

Along similar lines, Bose and Devroye [9] study the expected number of intersection between a line and the Delaunay triangulation of a random point set. This result is closely related to the number of steps required by RIGHT-HAND to route between the vertices of a Delaunay triangulation, since the number of steps performed is proportional to the number of edges of the triangulation intersecting the line segment (s, t).

Open Problem 2. *Study the expected behaviour of routing algorithms under various distributions of input graphs.*

3.6 Bibliographic Notes

The results described in this chapter can be found in the two papers by Bose *et al.* [8, 10]. An informal classification of algorithms based on randomization and memory is also included there.

Many authors have studied algorithms similar in nature to GREEDY and COMPASS, usually in the context of wireless networks modeled as unit disk graphs. For the most part, this work is experimental and little, if anything, is proven about these algorithms.

Lin and Stojmenović study an $O(1)$ memory variant of the GREEDY algorithm called GEDIR and show that it does not have trapping cycles (though it can still fail). They also give empirical results that show that GEDIR is competitive with COMPASS and MFR (defined below) in terms of success rate and lengths of paths found.

Kranakis *et al.* [52] introduce the COMPASS algorithm and study it in a theoretical setting. They prove (among other things) that COMPASS is not defeated by any Delaunay triangulation. They also give an $O(1)$ memory algorithm that is not defeated by any connected planar geometric graph, thus proving a stronger result than Theorem 6.

A number of routing algorithms have been defined in terms of the notion of progress. Let v be the vertex at which the packet is currently stored, and let $proj(x)$ be the orthogonal projection of the point x onto the line containing v and t. The *progress* of a neighbour $u \in \mathrm{N}(v)$ is defined as $p(u, v, t) = dist(v, t) - dist(proj(u, t))$.

Let $\mathrm{N}'(v)$ be the set of all $u \in N(v)$ such that u has positive progress. Nelson and Kleinrock [66] define an algorithm that forwards a packet to a randomly chosen element of $\mathrm{N}'(v)$, the idea being that such randomized paths help prevent congestion when many routing tasks are executing simultaneously.

Hou and Li [40] define an algorithm that always sends a packet to the nearest-neighbour of v in $N'(v)$. The motivation for this choice is to be able to minimize the transmission ranges of nodes. This is particularly important in radio networks since two nodes whose transmission regions overlap cannot transmit simultaneously on the same frequency.

Takagi and Kleinrock [78] describe the MFR (most-forward within region) algorithm. The MFR algorithm always forwards the packet to the neighbour u of v whose progress is maximized. The motivation for this choice is to minimize the number of hops the packet has to take in reaching its destination. Lin and Stojmenović [54] use a linear algebra argument to show that MFR does not have trapping cycles, though it can still fail.

A completely different class of geometric routing algorithms are those based on flooding. Although these do not satisfy our definition of a routing algorithm, we discuss them here for the sake of completeness. Note that many of these algorithms attempt to solve a more complicated routing problem than those described by our ideal model. In particular, the algorithms attempt to deal with moving nodes in the network, resulting in invalid or inaccurate location information.

Although flooding algorithms have their differences, they do share some common structure. During the execution of these algorithms, nodes forward packets to several of their neighbours at once. All packets are given serial numbers and these are memorized by nodes of the network. In this way, if a node receives a packet more than once it only forwards it the first time. The attribute that distinguishes different flooding algorithms is how a node v decides which of its neighbours to forward packets to.

In the LAR (location-aided routing) algorithms [51, 50] the packet is forwarded only to those neighbours of v that lie within a certain geographic region defined by the source, destination, and current location of the packet. The definition of this region is a parameter and the authors suggest certain possibilities such as the minimum bounding rectangle or convex hull of the source and a disk around the destination.

The DREAM (Distance Routing Effect Algorithm for Mobility) algorithm [5] attempts to take moving nodes into account. The DREAM algorithm forwards packets only to those nodes contained in the convex hull of the current node v and a disk of radius d about the node t. Here $d = s_t \cdot dist(v, t)$ is a function of t's speed and the distance to t. The idea behind this choice being that t is moving and the further it is from v, the further it will be from its current location by the time the packet arrives.

Chapter 4

Competitive Algorithms for Triangulations

Thus far we have considered only the question of whether routing algorithms can find *a* path between any two vertices in T. An obvious direction for research is to consider the length of the path found by a routing algorithm.

The *walk* taken by a routing algorithm $\mathcal{A}$ with transition function δ while routing from s to t is simply the sequence of vertices $(\delta^0(s), \delta^1(s), \ldots, \delta^k(s))$, where k is the least value such that $\delta^k(s) = t$. The length of a walk $W = (v_0, \ldots, v_k)$ can be measured in two ways; the *Euclidean length* of W, denoted length(W) is the sum of edge lengths of W, i.e.,

$$\text{length}(W) = \sum_{i=1}^{k-1} dist(v_i, v_{i+1}) \ . \tag{4.1}$$

The *link length* of W, denoted $|W|$, is simply the number of edges used in W, i.e.,

$$|W| = k \ . \tag{4.2}$$

The choice of which metric to use depends on the application, and sometimes even on the various operating parameters in the application. In some robotics applications, the limiting factor is the ground speed of the robot, in which case the relevant measure is Euclidean. In other applications, because of timing issues the limiting factor may be the number of stops and turns that the robot has to make, in which case the link metric would be the correct choice. For a discussion of applications in which the link metric is appropriate, see the survey article by Maheshwari *et al.*[64].

When analyzing the performance of online algorithms we use the paradigm of *competitive analysis* [7]. We say that a routing algorithm $\mathcal{A}$ is c-competitive for a graph $G = (V, E)$, if for every pair of vertices $s, t \in V$, the length (sum of the edge lengths) of the walk between s and t found by $\mathcal{A}$ is at most c times the length of the shortest path between s and t in G. We say that $\mathcal{A}$ has a *competitive ratio* of c if it is c-competitive.

In the case of randomized algorithms, we use the expected length of the walk found by $\mathcal{A}$, where the random bits given as input to $\mathcal{A}$ are uniformly distributed over all bit strings. More precisely, if l is the length of the walk taken by $\mathcal{A}$ when routing from s to t and and l' is the length of the shortest path from s to t then the *expected stretch* of $\mathcal{A}$ is $\mathrm{E}[l]/l'$. The competitive ratio of $\mathcal{A}$ on G is the maximum expected stretch taken over all pairs of vertices in G.

For a class C of graphs, we say that routing algorithm $\mathcal{A}$ is c-competitive for C if for every graph $G \in C$, $\mathcal{A}$ is c-competitive for G. If $\mathcal{A}$ is c-competitive for C, where c is a constant, then we say simply that $\mathcal{A}$ is *competitive* for C.

This chapter addresses questions about the competitive ratio in both the Euclidean and link length metrics. For the former we obtain competitive algorithms for a class of triangulations that includes Delaunay, greedy, and minimum-weight triangulations. For the latter, we give lower bounds that show these results cannot be duplicated for the link length metric.

4.1 Euclidean Length

In this section we consider the competitiveness of routing algorithms under the Euclidean length metric. Unless otherwise specified, all references to the length of a path P, including references to competitive ratios, refer to length(P).

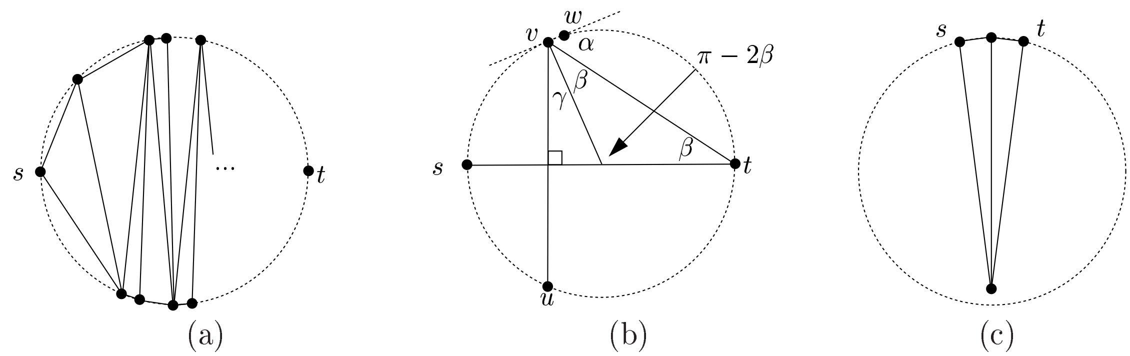


Figure 4.1: The proof of Theorem 7.

4.1.1 Negative Results

It is not difficult to contrive triangulations for which none of the algorithms described in Chapter 3 are c-competitive for any constant c. Thus it is natural to restrict our attention to a well behaved class of triangulations. Unfortunately, even for the class of Delaunay triangulations none of the algorithms seen so far are c-competitive.

Theorem 7. *For any constant c, there exist Delaunay triangulations for which none of the* GREEDY, COMPASS, GREEDY-COMPASS, RANDOM-COMPASS, *and* RIGHT-HAND *algorithms are c-competitive.*

Proof. We begin with GREEDY and GREEDY-COMPASS. Consider the set of points placed on a circle and then triangulated to obtain the zig-zag triangulation T shown in Figure 4.1.a. The points are placed so that each vertex v has a neighbor on the opposite side of the line through s and t that is closer to t than v's two neighbors on the same side of the line. Since the points are cocircular, this is a valid Delaunay triangulation.

Note that there exists a path from s to t that traverses the outer face of the triangulation and has length at most $(\pi/2) \cdot dist(s,t)$. Thus, $(\pi/2) \cdot dist(s,t)$ is an upper bound on the length of the shortest path between s and t. The length of the "zig-zag" path that uses the diagonals of T between s and t is $\Theta(n) \cdot dist(s,t)$, and this is the path taken by the GREEDY and GREEDY-COMPASS algorithms. Thus, GREEDY and GREEDY-COMPASS are not competitive for this triangulation.

To show that the COMPASS algorithm is not competitive, we again consider a set of cocircular points and make a zig-zag triangulation. Let v be any point on the circle with diameter s, t. Consider the clockwise angle α between the tangent line passing through v and the line through v and t. Compare this with the counterclockwise angle between the line perpendicular to s and t that passes through v and the line through v and t. Referring to Figure 4.1.b, we have

$$\alpha = \pi/2 - \beta \tag{4.3}$$

$$\gamma = \pi/2 - 2\beta \ , \tag{4.4}$$

and therefore $\gamma + \beta = \pi/2 - \beta = \alpha$, i.e., the two angles are equal. Thus if compass routing were to choose between the tangent line and the line crossing the circle it would be a tie. Now, by placing a point w on the circle close to v we can make $\angle w, v, t = \alpha - \epsilon$ for arbitrarily small $\epsilon > 0$. Similarly, by placing a point u on the opposite side of the circle we can make $\angle u, v, t = \alpha - \epsilon - \delta$ for arbitrarily small $\delta > 0$, so that $\text{cmp}(v) = u$. Since ϵ and δ can be arbitrarily small, we can repeat this construction as often as we like, thereby making the COMPASS path arbitrarily long.

To see that the RANDOM-COMPASS and RIGHT-HAND algorithms are not competitive consider a configuration of points like that in Figure 4.1.c. By making s and t almost collinear with a third point, it is possible to produce arbitrarily long thin triangles that make the length of the path taken by RIGHT-HAND arbitrarily long. Furthermore, in this configuration the probability that the path found by RANDOM-COMPASS is the same as that found by RIGHT-HAND is 1/2, and thus the expected length of the path taken by RANDOM-COMPASS can be arbitrarily large. □

4.1.2 A Competitive Algorithm for Delaunay Triangulations

Since none of the algorithms described in Chapter 3 is competitive, even for Delaunay triangulations, an obvious question is whether there exists any algorithm that is competitive for Delaunay triangulations. In this section we answer this question in the affirmative. In fact, we prove an even stronger result by giving an algorithm that finds a walk in a Delaunay triangulation $T = (V, E)$ from s to t whose cost is at most

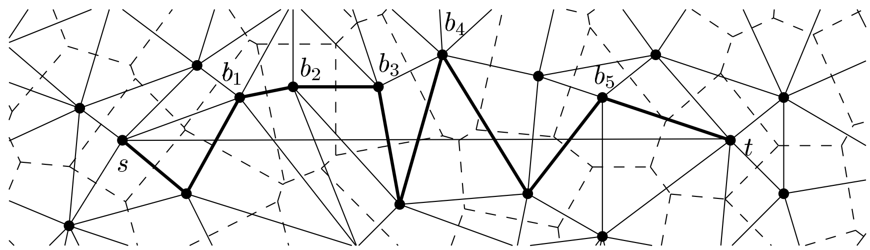


Figure 4.2: A path obtained by the VORONOI algorithm.

a constant times $dist(s, t)$, as opposed to the length of the shortest path between s and t.

Our algorithm is based on the proof of Dobkin *et al.* [21] that the Delaunay triangulation approximates the complete Euclidean graph to within a constant factor in terms of shortest path length. In the following we will use the notation $\mathrm{x}(p)$ (resp. $\mathrm{y}(p)$) to denote the x-coordinate (resp. y-coordinate) of the point p.

Consider the directed line segment from s to t. This segment intersects regions of the Voronoi diagram of V in some order, say $R_1, \ldots, R_m$, where R_1 is the Voronoi region of s and R_m is the Voronoi region of t. The VORONOI algorithm for routing in Delaunay triangulations moves the packet from s to t along the path $P_V = v_1, \ldots, v_m$ where v_i is the vertex defining R_i. An example of a walk obtained by the VORONOI algorithm is shown in Figure 4.2. Since the Voronoi region of a vertex can be computed given only its neighbours in the Delaunay triangulation, it follows that VORONOI is an $O(1)$ memory routing algorithm.

The VORONOI algorithm on its own is not competitive for all Delaunay triangulations (this would be too easy), as can be seen from Figure 4.3. However, it does have some properties that allow us to derive a competitive algorithm. Dobkin *et al.* [21] prove the following.

Lemma 5 (Dobkin *et al.*). *P_V is x-monotone, i.e., $\mathrm{x}(v_i) < \mathrm{x}(v_{i+1})$ for all $1 \le i < m$.*

This property alone is not enough, and the reader should prepare themselves for some definitions. Let f denote the boundary of the union of all triangles of T

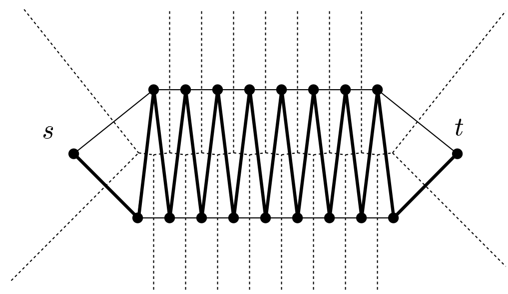


Figure 4.3: The VORONOI algorithm is not c-competitive for all Delaunay triangulations.

intersected by the segment (s, t). Let $b_1, \ldots, b_l$ be the subsequence of vertices of P_V that are above or on the x axis, (refer to Figure 4.2). Let $P_{f,i}$ denote the boundary of f between b_i and b_{i+1}, in clockwise order, and let $c_{i,1}, \ldots, c_{i,n_i}$ denote the lower convex hull[1] of $P_{f,i}$. Finally, let $P_{V,i,j}$ denote the VORONOI path from $c_{i,j}$ to $c_{i,j+1}$ in T.

With these definitions, the graph G_{dfs} is defined as follows. G_{dfs} contains the vertices and edges of all triangles of T that intersect the segment (s, t) and the vertices and edges of $P_{V,i,j}$ for all $1 \leq i < l$ and $1 \leq j \leq n_i$. Dobkin *et al.* [21] prove the following:

Lemma 6 (Dobkin *et al.*). *Let $c_{\mathrm{dfs}} = (1+\sqrt{5})\frac{\pi}{2}$.[2] Then, G_{dfs} contains a path from s to t whose length is at most $c_{\mathrm{dfs}} \cdot dist(s, t)$.*

Let G_{m} be the graph containing only the vertices and edges of triangles of T that intersect the segment (s, t). Then G_{m} is the graph of a triangulated polygon having two ears, i.e, it is a Hamiltonian polygon. The following lemma shows that, in terms of the shortest path from s to t, G_{m} is as good at G_{dfs}.

Lemma 7. *G_{m} contains a path from s to t whose length is at most $c_{\mathrm{dfs}} \cdot dist(s, t)$.*

[1] The lower convex hull of a point set S is the part of the convex hull of S beneath the line joining the leftmost and rightmost endpoints of S.

[2] *We call c_{dfs} the Dobkin–Friedman–Supowit constant [21].*

Proof. Note that G_{m} is a subgraph of G_{dfs} and that the only edges of G_{dfs} not in G_{m} are those of $P_{V,i,j}$, for all applicable i and j. Define $P_{f,i,j}$ as the boundary of P between $c_{i,j}$ and $c_{i,j+1}$, in the clockwise direction. We claim that all the vertices of $P_{f,i,j}$ appear in $P_{V,i,j}$. Therefore, by triangle inequality

$$\mathrm{length}(P_{f,i,j}) \leq \mathrm{length}(P_{V,i,j}) \ , \tag{4.5}$$

and the shortest path from s to t in G_{m} is no more than the shortest path from s to t in G_{dfs}, as required.

Refer to Figure 4.4 for what follows. Assume for the sake of contradiction that there is a vertex q in $P_{f,i,j}$ that is not in $P_{V,i,j}$. As part of their proof, Dobkin *et al.* show that $P_{V,i,j}$ remains entirely above the segment $(c_{i,j}, c_{i,j+1})$. Therefore, let Q be the polygon bounded by $P_{V,i,j}$ and the segment $(c_{i,j}, c_{i,j+1})$. Since q is on $P_{f,i,j}$ and, by planarity, $P_{f,i,j}$ is contained in Q, it must be that q is contained in Q.

Since, by Lemma 5, Q is monotone in the direction from $c_{i,j}$ to $c_{i,j+1}$, it can be partitioned into trapezoids whose top sides are edges of $P_{V,i,j}$, whose bottom sides are on the line segment $(c_{i,j}, c_{i,j+1})$ and whose left and right sides are perpendicular to $(c_{i,j}, c_{i,j+1})$. Refer to Figure 4.4.

Let a and b be the two vertices of $P_{V,i,j}$ that define the trapezoid containing q. We claim that a and b cannot be consecutive on $P_{V,i,j}$ because their Voronoi regions do not share an edge that intersects $(c_{i,j}, c_{i,j+1})$. We will prove this by showing that in the Voronoi diagram of $\{q, a, b\}$ the bisector of a and b does not intersect the segment $(c_{i,j}, c_{i,j+1})$. This is sufficient, since this bisector contains the bisector of a and b in the original Voronoi diagram.

Let C be the circle with center on $(c_{i,j}, c_{i,j+1})$ and with a and b on its boundary. If the bisector of a and b in the Voronoi diagram of q, a and b intersects the segment $(c_{i,j}, c_{i,j+1})$ then C must not contain q. However, C does contain the top, left, right, and bottom sides of the trapezoid containing q. But this can't be, since then C contains the entire trapezoid and contains q. We conclude that there is no point q on $P_{f,i,j}$ that is not on $P_{V,i,j}$. $\square$

The way in which Dobkin *et al.* prove Lemma 6 is to prove that there is a path P_{dfs} that contains $b_0, \ldots, b_l$, in that order, and has length at most $c_{\mathrm{dfs}} \cdot \mathit{dist}(s,t)$.

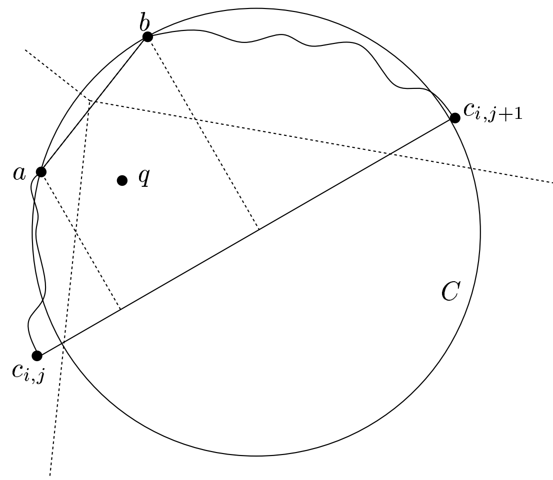


Figure 4.4: The proof of Lemma 7.

Furthermore, P_{dfs} has the property that the portion between b_i and b_{i+1} is either $P_{V,i,1}, \ldots, P_{V,i,n_i}$ or the portion $P_{V,i}$ of P_V whose endpoints are b_i and b_{i+1}.

Therefore, to find a competitive routing algorithm for Delaunay triangulations, it is sufficient to find a competitive algorithm for routing from b_i to b_{i+1}. Furthermore, the above discussion, in conjunction with Lemma 7, shows that this algorithm needs only use edges of $P_{f,i}$ and P_{V_i}. The following algorithm, which we call PARALLEL-SEARCH shows how we can search any two paths $P_1 = v_1, \ldots, n_v$ and $P_2 = u_1, \ldots, u_{n_u}$, where $v_1 = u_1 = a$ and $v_{n_v} = u_{n_u} = b$ are common start and and vertices.

Informally, the algorithm works by taking one step along P_1, then taking two steps along P_2, then four steps along P_1, and so on until the destination vertex b is reached. More precisely:

```
let d = min{dist(a, v_2), dist(a, u_2)}
repeat
    travel along P_1 until reaching b or a vertex v_i such that length(v_i, ..., v_{i+1}) > d
    if b was reached then quit
    return to a
    d ← 2d
    travel along P_2 until reaching b or a vertex u_i such that length(u_i, ..., u_{i+1}) > d
```

```
8:     if b was reached then quit
9:     return to a
10:    d ← 2d
11: until reaching b
```

Lemma 8. PARALLEL-SEARCH *reaches* b *after travelling a total distance of at most* $9 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\}$

Proof. Let $d_f = 2^k c$ be the value of d during the final exploration step (Line 3 or 7) of the algorithm. Therefore, the total distance travelled by the algorithm is equal to

$$
\begin{aligned}
D &= 2 \cdot \sum_{i=1}^{k-1} 2^i c + x && (4.6)\\
&\leq 2^{k+1} c + x \ , && (4.7)
\end{aligned}
$$

where x is the distance travelled during the last exploration step. There are now two cases to consider:

Case 1: The algorithm terminated while exploring the shorter of the two paths P_1 or P_2. Then $d_f \leq 4 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\}$, otherwise the algorithm would have reached b in the previous iteration of the algorithm. Therefore

$$
\begin{aligned}
D &\leq 8 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\} + x && (4.8)\\
&= 9 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\} \ . && (4.9)
\end{aligned}
$$

Case 2: The algorithm terminated while exploring the longer of the two paths P_1 or P_2. Then $x \leq d_f \leq 2 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\}$, otherwise the algorithm would have reached b in the previous exploration step. Then

$$
\begin{aligned}
D &\leq 4 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\} + x && (4.10)\\
&\leq 6 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\} \ . && (4.11)
\end{aligned}
$$

In either case, the conditions of the lemma are satisified. □

In summary, a competitive algorithm for travelling between s and t in a Delaunay triangulation proceeds by using PARALLEL-SEARCH to find 9-competitive paths from

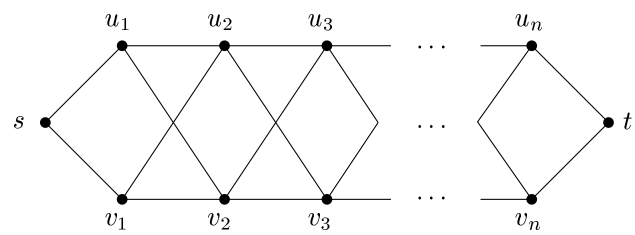


Figure 4.5: A layered graph

b_i to b_{i+1} for all $1 \leq i \leq l$. From the above discussion, the overall result is that PARALLEL-VORONOI travels an overall distance of at most $9 \cdot c_{\text{dfs}} \cdot dist(s,t)$. All the steps of the PARALLEL-VORONOI can be implemented without much difficulty using $O(1)$ memory. This yields the following result.

Theorem 8. *Algorithm* PARALLEL-VORONOI *is an $O(1)$ memory routing algorithm that is $(9 \cdot c_{\text{dfs}})$-competitive for all Delaunay triangulations.*

4.1.3 Layered Digraphs and Hamiltonian Polygons

In this section we review results of Papadimitriou and Yannakakis [69] for routing on layered digraphs. We then show that these results can be used to build competitive routing algorithms for special types of triangulated polygons.

A *layered graph G* with n layers is a (non-geometric) graph on vertex set

$$\{s, t, v_1, \dots, v_n, u_1, \dots, u_n\} \ .$$

We call $\{u_i, v_i\}$ the ith layer of G. For all $1 \leq i < n$, the ith layer is connected to the $i+1$st layer with four edges, (u_i, u_{i+1}), (u_i, v_{i+1}), (v_i, u_{i+1}), and (v_i, v_{i+1}). In addition to this, s is adjacent to u_1 and v_1 and t is adjacent to u_n and v_n. The edge weights of G are positive and obey the triangle inequality. See Figure 4.5 for an example.

Layered graphs are studied by Papadimitriou and Yannakakis [69] who prove the following.

Theorem 9 (Papadimitriou and Yannakakis). *There exists an $O(1)$ memory online routing algorithm for routing from s to t in G that is 9-competitive.*

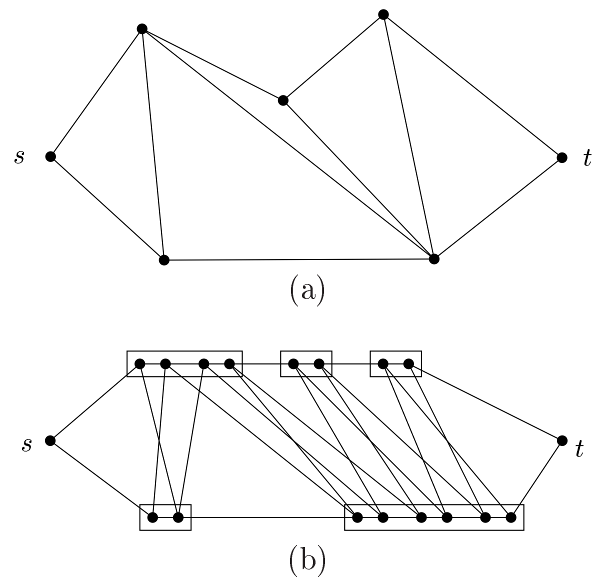


Figure 4.6: Examples of (a) a triangulated simple polygon and (b) the corresponding layered graph.

Consider a simple polygon P whose interior is partitioned into triangles whose vertices are the vertices of P. The vertices and edges of this partition induce a geometric graph $G(P)$. We call a vertex v of $G(P)$ an *ear* if it has degree 2. An example is shown in Figure 4.6.a.

If $G(P)$ has exactly two ears s and t, then we call $G(P)$ a *Hamiltonian polygon.* If $G(P)$ is Hamiltonian, then it naturally defines a layered graph G in the following way (refer to Figure 4.6). The vertices s and t of $G(P)$ correspond to the vertices s and t in G. Let e_i be the ith edge of $G(P)$ intersected by the directed segment (s,t). For each such edge e_i we add four vertices, u_{2i-1}, u_{2i}, v_{2i-1}, and v_{2i}. The vertices are then connected with edges in the manner described above for layered graphs. The edges of G are weighted with the length of the corresponding edge in $G(P)$.

From the above discussion, it becomes clear that any routing algorithm for layered graphs can be made into a routing algorithm for routing between the vertices s and t of $G(P)$. Given the result of Papadimitriou and Yannakakis (Theorem 9), we obtain the following.

Corollary 1. *There exists an $O(1)$ memory online routing algorithm for routing from s to t in $G(P)$ that is 9-competitive.*

4.1.4 A Competitive Algorithm for Triangulations with the Diamond Property

Das and Joseph [16] generalize the proof of Dobkin *et al.* to show that triangulations with a certain special property also approximate the complete Euclidean graph in terms of shortest path length. In this section we give a competitive routing algorithm for triangulations that satisfy this property.

Let α be any angle $0 < \alpha \leq \pi/2$. For an edge e of a triangulation $T = (V, E)$, consider the two isosceles triangles t_1 and t_2 whose base is e and with base angle α. Then we say that e satisfies the *diamond property* with parameter α if one of t_1 or t_2 does not contain any point of V in its interior. We say that a triangulation T satisfies the *diamond property* if every edge of T satisfies the diamond property.

Let T be a triangulation satifying the diamond property for some $0 < \alpha < 90$ and define G_{m} as in the previous section. Then, Das and Joseph prove the following.

Lemma 9. *Let $c_{\mathrm{dj}}(\alpha)$ be a constant that depends only on α. Then, G_{m} contains a path from s to t whose length is at most $c_{\mathrm{dj}}(\alpha) \cdot dist(s,t)$.*

Since G_{m} is the graph of a triangulated polygon, Corollary 1 implies a 9-competitive routing algorithm for routing from s to t in G_{m}. Let us call this algorithm PARALLEL-DIAMOND.

Theorem 10. *Algorithm* PARALLEL-DIAMOND *is an $O(1)$ memory routing algorithm that is $(9 \cdot c_{\mathrm{dj}}(\alpha))$-competitive for any triangulation satisfying the diamond property*

More concretely, since any Delaunay triangulation satisfies the diamond property with $\alpha = \pi/2$, any greedy triangulation satisfies the diamond property with $\alpha = \pi/4$, and any minimum-weight triangulation satisfies the diamond property with $\alpha = \pi/8$ [16], we obtain the following result.

Corollary 2. *There exists an $O(1)$ memory routing algorithm that is competitive for the class of Delaunay triangulations, greedy triangulations and minimum-weight triangulations.*

4.1.5 A Lower Bound for Arbitrary Triangulations

We have given competitive algorithms for Delaunay, greedy, and minimum-weight triangulations. A natural question, therefore, is whether there is a competitive algorithm for arbitrary triangulations. In this section we prove that there is no such algorithm.

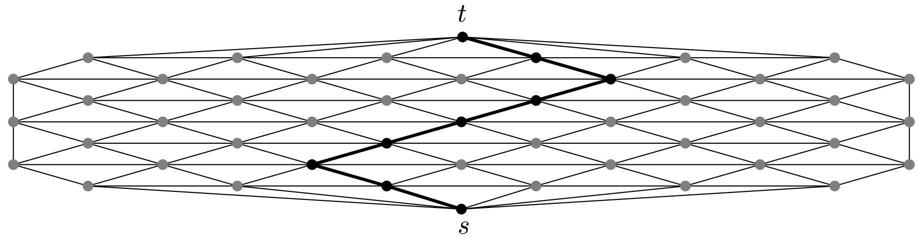
Our proof is a modification of that used by Papadimitriou and Yannakakis [69] to show that no online algorithm for finding a destination point among n axis-oriented rectangular obstacles in the plane is $o(\sqrt{n})$-competitive.

Theorem 11. *Under the Euclidean distance metric, no deterministic routing algorithm is $o(\sqrt{n})$ competitive for all triangulations.*

Proof. Consider a portion of the hexagonal having height and width $\Theta(n)$ so that it contains $\Theta(n \times n)$ vertices. Now modify this portion of the lattice in the following way. Scale the x-coordinates of all vertices so that each edge is of length $\Theta(n)$. Also add two additional vertices, s and t, centered horizontally, at one unit below the bottom row and one unit above the top row, respectively. Finally, add edges to the lattice, in order to complete it to a triangulation T. See Figure 4.7.a for an illustration.

Let $\mathcal{A}$ be any deterministic routing algorithm and observe the actions of $\mathcal{A}$ as it routes from s to t. In particular, consider the first $n+1$ steps taken by $\mathcal{A}$ as it routes from s to t. Then $\mathcal{A}$ visits at most $n+1$ vertices of T, and these vertices induce a subgraph T_{vis} consisting of all vertices visited by $\mathcal{A}$ and all edges adjacent to these vertices.

For any vertex v of T not equal to s or t, define the *x-span* of v as the interval between the rightmost and leftmost x-coordinate of $\mathrm{N}(v)$. The length of any x-span is $\Theta(n)$, and the width of the original triangulation T is $\Theta(n^2)$. By the pigeonhole principle, this implies that there is some vertex v_b on the bottom row of T whose



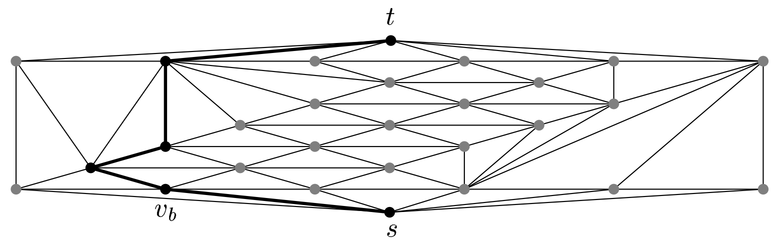


Figure 4.7: (a) The triangulation T with the path found by $\mathcal{A}$ indicated. (b) The resulting triangulation T' with the "almost-vertical" path shown in bold.

x-coordinate is at most $n\sqrt{n}$ from the x-coordinate of s and is contained in $O(\sqrt{n})$ x-spans of the vertices visited in the first $n+1$ steps of $\mathcal{A}$.

We now create the triangulation T' that contains all vertices and edges of T_{vis}. Additionally, T' contains the set of edges forming an "almost vertical" path from v_b to the top row of T'. This almost vertical path is a path that is vertical wherever possible, but uses minimal detours to avoid edges of T_{vis}. Since only $O(\sqrt{n})$ detours are required, the length of this path is $O(n\sqrt{n})$. Finally, we complete T' to a triangulation in some arbitrary way that does not increase the degrees of vertices on the first $n+1$ steps of $\mathcal{A}$. See Figure 4.7.b for an example.

Now, since $\mathcal{A}$ is deterministic, the first $n+1$ steps taken by $\mathcal{A}$ on T' will be the same as the first $n+1$ steps taken by $\mathcal{A}$ on T, and will therefore travel a distance of $\Theta(n^2)$. However, there is a path in T' from s to t that first visits v_b (at a cost of $O(n\sqrt{n})$), then uses the "almost-vertical" path to the top row of T' (at a cost of $O(n\sqrt{n})$) and then travels directly to t (at a cost of $O(n\sqrt{n})$). Thus, the total cost of this path, and hence the shortest path, from s to t is $O(n\sqrt{n})$.

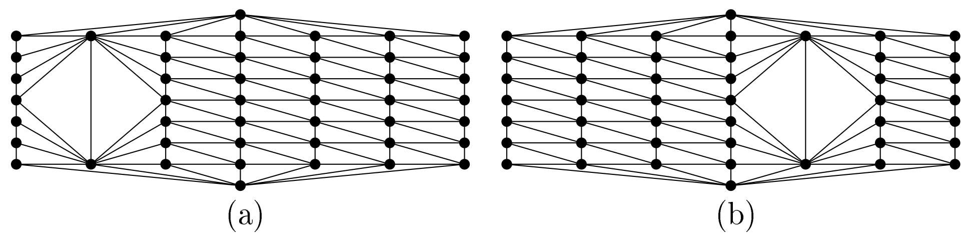


Figure 4.8: The point sets (a) $S_{49,2}$ and (b) $S_{49,5}$ along with their Delaunay triangulations.

We conclude that $\mathcal{A}$ is not $o(\sqrt{n})$-competitive for T'. Since the choice of $\mathcal{A}$ is arbitary, and T' contains $O(n)$ vertices, this implies that no deterministic routing algorithm is $o(\sqrt{n})$ competitive for all triangulations with n vertices. □

4.2 Link Length

In this section we switch gears and consider the problem of finding competitive algorithms for the link length metric. Unless otherwise specified, all references to the length of a path P refer to the link length, $|P|$ of P. Our previous success with Delaunay triangulations might make us hopeful that we can also find a competitive algorithm for the link length metric. Unfortunately, our investigations yield only negative results, and the main contributions of this section are impossibility results.

We obtain our results by constructing a "bad" family of point sets as follows. Let C_i be the set of $\sqrt{n}$ points $\{(i\sqrt{n}, 1), (i\sqrt{n}, 2), \ldots, (i\sqrt{n}, \sqrt{n})\}$. We call C_i the *ith column.* Let $D_i = \{(i\sqrt{n}, 1), (i\sqrt{n}, \sqrt{n})\}$, and define a family of point sets $S = \bigcup_{j=1}^{\infty}\{S_{j^2}\}$ where $S_n = \{S_{n,1}, \ldots, S_{n,\sqrt{n}}\}$ and

$$S_{n,i} = \bigcup_{j=1}^{i-1} C_j \cup D_i \cup \bigcup_{j=i+1}^{\sqrt{n}} C_j \cup \{(\sqrt{n}/2, 0), (\sqrt{n}/2, \sqrt{n}+1)\} \qquad (4.12)$$

Two Delaunay triangulations of members of the set S_{49} are shown in Figure 4.8.

Theorem 12. *Under the link length metric, no algorithm is $o(\sqrt{n})$-competitive for all Delaunay triangulations.*

Proof. We use the notation $\mathrm{DT}(S_{n,i})$ to denote the Delaunay triangulation of $S_{n,i}$. Although the Delaunay triangulation of $S_{n,i}$ is not unique, we will assume $\mathrm{DT}(S_{n,i})$ is triangulated as in Figure 4.8. Note that, in $\mathrm{DT}(S_{n,i})$, the shortest path between the topmost vertex s and bottom-most vertex t is of length 3, independent of n and i. Furthermore, any path from s to t whose length is less than $\sqrt{n}$ must visit vertices from one of the columns C_{i-1}, C_i, or C_{i+1}.

The remainder of the proof is based on the following observation: If we choose an element i uniformly at random from $\{1, \ldots, \sqrt{n}\}$, then the probability that a routing algorithm $\mathcal{A}$ has visited a vertex of C_{i-1}, C_i, or C_{i+1} after k steps is at most $3k/\sqrt{n}$. Letting $k = \sqrt{n}/6$, we see that the probability that $\mathcal{A}$ visits a vertex of C_{i-1}, C_i, or C_{i+1} after $\sqrt{n}/6$ steps is at most $1/2$.

Letting d_i denote the (expected, in the case of randomized algorithms) delivery time when routing from s to t in $S_{n,i}$ using $\mathcal{A}$, we have

$$\frac{1}{\sqrt{n}} \cdot \sum_{i=1}^{\sqrt{n}} d_i \geq \sqrt{n}/12 \ . \tag{4.13}$$

Since, for any $S_{n,i}$, the shortest path from s to t is 3 there must be some i for which the competitive ratio of $\mathcal{A}$ for $S_{n,i}$ is at least $\sqrt{n}/36 \in \Omega(\sqrt{n})$. □

Theorem 13. *Under the link length metric, no algorithm is $o(\sqrt{n})$-competitive for all greedy triangulations.*

Proof. This follows immediately from the observation that for any $S_{n,i}$, a Delaunay triangulation of $S_{n,i}$ is also a greedy triangulation of $S_{n,i}$. □

Theorem 14. *Under the link length metric, no algorithm is $o(\sqrt{n})$-competitive for all minimum-weight triangulations.*

Proof. We claim that for members of S, any greedy triangulation is also a minimum-weight triangulation. To prove this, we use a result on minimum-weight triangulations due to Aichholzer *et al.* [1]. Let $K_{n,i}$ be the complete graph on $S_{n,i}$. Then an edge e of $K_{n,i}$ is said to be a *light edge* if every edge of $K_{n,i}$ that crosses e is not shorter than e. Aichholzer *et al.* prove that if the set of light edges contains the edges of a triangulation then that triangulation is a minimum-weight triangulation.

Algorithm	Class of Graphs	Memory	Comp. ratio
PARALLEL-VORONOI	Delaunay triangulations	$O(1)$	$9 \cdot c_{\mathrm{dfs}}$
PARALLEL-DIAMOND	Greedy/M-W triangulations	$O(1)$	$9 \cdot c_{\mathrm{dj}}$

Table 4.1: Competitiveness Results for the Euclidean length metric.

There are only 5 different types of edges in the greedy triangulation of $S_{n,i}$. (1) vertical edges within a column, (2) horizonal edges between adjacent columns, (3) diagonal edges between adjacent columns, (4) edges used to triangulate column i, and (5) edges used to join s and t to the rest of the graph. It is straightforward to verify that all of these types of edges are indeed light edges. □

4.3 Summary and Open Problems

In this chapter we have studied the competitive (or not) behaviour of online routing algorithms in triangulations. Our results for the Euclidean length metric are summarized in Table 4.1.

Since we have only considered three special types of triangulations, the following problem is an obvious direction for ongoing research.

Open Problem 3. *For what other classes of geometric graphs do competitive routing algorithms exist?*

We have given an $\Omega(\sqrt{n})$ lower bound on the competitive ratio of deterministic algorithms for routing on arbitrary triangulations. This raises two questions.

Open Problem 4. *Is there an algorithm for routing in arbitrary triangulations whose competitive ratio can be bounded as a function of n?*

Open Problem 5. *Is there an $o(\sqrt{n})$-competitive randomized algorithm for routing on arbitrary triangulations?*

We have also shown that algorithms for Delaunay, greedy and minimum-weight triangulations that are competitive with respect to link length do not exist. However, our lower bound construction does not match the best known upper bound. This suggests the following open problem.

Open Problem 6. *Close the gap between the $\Omega(\sqrt{n})$ lower bound provided by Theorem 12 and the $O(n)$ upper bound provided by the* RIGHT-HAND *algorithm on the competitive ratio of algorithms for routing in triangulations.*

The difficulty in obtaining competitive algorithms for the link length metric seems to come from the fact that Euclidean distances have very little effect on link length, at least for the types of proximity relations used to define Delaunay, greedy and minimum-weight triangulations. This suggests the following open problem.

Open Problem 7. *Is there a naturally arising class of geometric graphs that admits a competitive algorithm in the link length metric (meshes don't count)?*

4.4 Bibliographic Notes

Prior to the work described in this chapter, a significant amount of time had been spent by other researchers on finding competitive routing algorithms for routing in general and restricted classes of polygons [13, 17, 18, 28, 36, 37, 38, 34, 41, 42, 43, 48, 61, 59, 60, 57, 58, 55, 56, 75, 76]. Although these algorithms are interesting in their own right, most of the techniques they use are not immediately applicable to routing on geometric networks. In particular, these algorithms make implicit use of the fact that the dual graph of a triangulated polygon is a tree or, in some restricted cases, a path.

Previous work on competitive algorithms for routing on geometric graphs has focused on three cases: the line, sets of concurrent rays, trees. In these settings, the location of t is unknown, but a lower bound d_l on its distance from s is given, and it can be recognized once it is reached.

Randomized algorithms for the line and (more generally) sets of m concurrent rays have been studied by Hipke [33] and Kao *et al.* [45, 44]. The optimal randomized strategy for searching a set of m concurrent rays has a competitive ratio of $1 + 2a^m/((a-1)\ln a)$, for $a > 1$, and this matches the lower bound.

For the line problem, Baeza-Yates *et al.* [3] obtain a strategy with an optimal deterministic competitive ratio of 9 that works by repeatedly doubling the length of the line that the algorithm explores, while alternating between left and right exploration steps. For a set of m concurrent rays, they obtain a strategy with a competitive ratio of $1 + 2m^m/(m-1)^{m-1} \leq 1 + 2em$. The line searching algorithm is also the basis of the work by Papadimitriou and Yannakakis [69] on layered graphs. Klein [49] and Schuirer [75] study the problem of searching in geometric trees (trees embedded in Euclidean d-space). Schuirer [75] obtains the better competitive ratio of $1 + 2m^m/(m-1)^{m-1}$ for a tree with m leaves.

Icking *et al.* [34, 43] further study the line and ray problems under the additional restriction that an upper bound d_u on the distance to the source is known. López-Ortiz and Schuirer [62] refine these results and obtain deterministic competitive ratios of $9m - O(1/\log^2 d)$ and $1 + 2m^m/(m-1)^{m-1} - O(1/\log^2 d)$ for searching on a line and a set of m concurrent rays, respectively. Here d is the distance from s to t, and need not be known in advance. They also give matching lower bounds, thus closing the problem with possibly the exception of tightening the constants in the big-O notation.

The PARALLEL-VORONOI algorithm described in this chapter is the first competitive online algorithm for routing in Delaunay triangulations. It is described by Bose and Morin [10].

The PARALLEL-DIAMOND algorithm is the first online algorithm for approximating shortest paths in greedy and minimum-weight triangulations and appears here for the first time.

The $\Omega(\sqrt{n})$ lower bounds on competitive ratios appear in the paper by Bose *et al.* [8].

Chapter 5

Routing in Planar Geometric Graphs

Thus far we have considered the special case where the graph on which routing takes place is a triangulation or a convex subdivision. In this chapter we relax this restriction and consider arbitrary planar geometric graphs.

We have seen in Chapter 4 that we cannot expect to get competitive algorithms for arbitary geometric graphs, either in the link length or in the Euclidean distance metrics. Rather, we would like to know whether there exists algorithms that will always be able to route a packet to its destination in any planar geometric graph. At the end of Chapter 3 we saw that any such algorithm will require either memory or randomization in order to succeed.

In this chapter we give an $O(1)$ memory algorithm for solving the point-to-point routing problem in an arbitary connected planar geometric graph. Additionally, we consider other types of routing problems. In particular, we also study algorithms for broadcasting (point-to-plane routing) and geocasting (point-to-region routing).

The discussion begins by defining the broadcasting and geocasting problems. We then define a structure on the faces of a planar geometric graph that is helpful in solving these problems. Using this structure, we then show how to solve the problems of routing, broadcasting, and geocasting in arbitrary planar geometric graphs.

5.1 Routing, Broadcasting and Geocasting

The *point-to-point* routing problem, as defined in Chapter 1 is that of finding a path between two vertices in a geometric graph. In this section we present two other types of routing problems.

The *broadcasting* or *point-to-plane* routing problem is that of routing a message from one vertex s to all other vertices. Recall the notation of Section 3.2, in which $\mathcal{A}$ is a routing algorithm and $\delta^i(s,t)$ is the vertex visited by $\mathcal{A}$ at the ith step of the algorithm. For a broadcasting algorithm, we can define $\delta^i(s)$ in a similar manner, except without an explicit destination vertex t. Then we say that $\mathcal{A}$ solves the broadcasting problem for a geometric graph $G = (V, E)$ if for all $t \in V$, there exists an i such that $\delta^i(s) = t$.

The *geocasting* or *point-to-region* routing problem is that of routing a packet from a vertex s to all vertices contained in some convex geometric region r_t. Here we assume that $s \in r_t$. We say that a routing algorithm $\mathcal{A}$ solves the geocasting problem for a graph $G = (V, E)$, if for every vertex $t \in V \cap r_t$, there exists an i such that $\delta^i(s, r_t) = t$. Note that broadcasting is a special case of geocasting in which r_t is the entire Euclidean plane.

The *delivery time* of a geocasting (and hence also broadcasting) algorithm is defined as

$$\max_{v \in V \cap r_t} \{\min_i \{\delta^i(s, r_t) = v\}\} \ , \tag{5.1}$$

i.e., it is the number of steps taken by the packet before all vertices in the destination region have seen the packet.

Note that by using the same model as Section 3.2, we are restricting ourselves to algorithms in which there is always exactly one copy of each packet at any given time during the execution of the algorithm. The reason we do this is that it is the least restrictive assumption in terms of applications. In contexts such as robotics, vehicle routing, or pedestrian path finding it is simply not possible to have duplicate copies of a robot/vehicle/pedestrian. Although we analyze algorithms under this model, we note that our algorithms do allow for the duplication of packets, in which case the number of messages sent by the algorithms remains unchanged, but in some cases the execution time is decreased.

Note that there is a trivial randomized memoryless algorithm for the broadcasting problem. At each stage, the packet simply chooses a random neighbour of the current vertex to visit. The result is a random walk in the graph G, and if this walk is allowed to continue indefinitely, all vertices of G will eventually be visited.

If we allow our algorithms to use $O(n)$ memory for solving the broadcasting and geocasting problems, then we can easily achieve an algorithm with optimal delivery time by simulating depth-first search [39]. The depth-first search algorithm makes use of a stack and some mark bits at each vertex of G, which can be simulated with a memory of size $O(n)$. This results in a broadcasting algorithm with delivery time $O(n)$. Thus, the challenge is in finding an algorithm with small memory requirements and low delivery time.

5.2 The Face Tree

In this section we describe the *face tree*, a tree defined on the faces of a planar geometric graph G. The face tree is defined with respect to a point p. We begin by using p to define a total order $\preceq_p$ on the edges of G.

5.2.1 The $\preceq_p$ Order and Entry Edges

To help define $\preceq_p$ we introduce some notation. Let $e = (u, v)$ be an edge such that $dist(u, p) \leq dist(v, p)$. Define $dist(e, p)$ be the radius of the smallest circle C centered at p that intersects e, and let $\mathrm{c}(e)$ be the point at which C intersect e. For two points a and b, let $\overrightarrow{ab}$ be the direction from a to b measured in radians, counterclockwise from the positive x-direction.

We define the *key* of an edge $e = (u, v)$ as the 4-tuple

$$\mathrm{key}_p(e) = (dist(e,p), \quad \overrightarrow{p\mathrm{c}(e)}, \quad \angle p, u, v, \quad \overrightarrow{uv}) \ . \tag{5.2}$$

Lemma 10. *For any two edges $e_1 \neq e_2$ of a planar geometric graph* $\mathrm{key}(e_1) \neq \mathrm{key}(e_2)$.

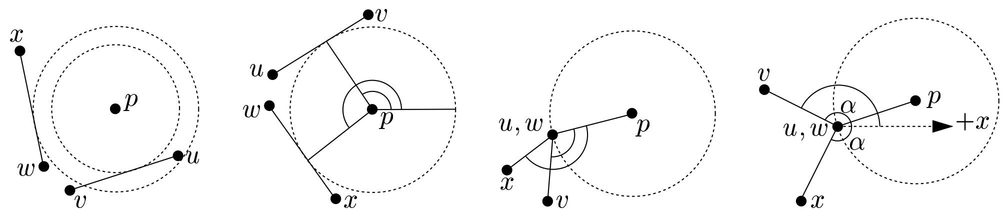


Figure 5.1: How the 4 key values are used to show that $(u, v) \preceq_p (w, x)$.

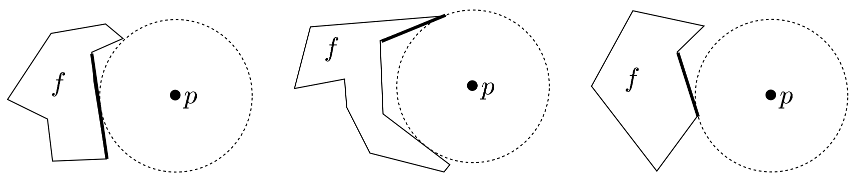


Figure 5.2: Examples of $\mathrm{entry}(f, p)$, with $\mathrm{entry}(f, p)$ shown in bold.

Proof. Let $e_1 = (u, v)$ and $e_2 = (w, x)$, and assume, wlog, that $dist(u, p) \leq dist(v, p)$ and $dist(w, p) \leq dist(x, p)$. Assume for the sake of contradiction that $\mathrm{key}(e_1) = \mathrm{key}(e_2)$. Then $dist(e_1, p) = dist(e_2, p)$ and $\mathrm{c}(e_1) = \mathrm{c}(e_2)$. Since e_1 and e_2 can only intersect at their endpoints, it must be the case that $u = w$. But then the interiors of e_1 and e_2 must intersect, since $\overrightarrow{uv} = \overrightarrow{wx}$, a contradiction. □

The relation $\preceq_p$ on the edges of e is defined by lexicographic comparison of key values using the $\leq$ operator. By Lemma 10, it follows that $\preceq_p$ defines a total order on the edges of any geometric graph G. Figure 5.1 shows how the four key values can be used to evaluate the relation $\preceq_p$.

For a face f with a set S of edges on its boundary, we define $\mathrm{entry}(f, p)$ as

$$\mathrm{entry}(f, p) = x \in S : x \preceq_p y \text{ for all } y \neq x \in S \tag{5.3}$$

(see Figure 5.2). We call $\mathrm{entry}(f, p)$ the *entry edge* of f. Since $\preceq_p$ is a total order, $\mathrm{entry}(f, p)$ is well-defined.

Lemma 11. *For any face f of G that does not contain p in its closure,* $\mathrm{entry}(f, p)$ *is on the boundary of two faces of G.*

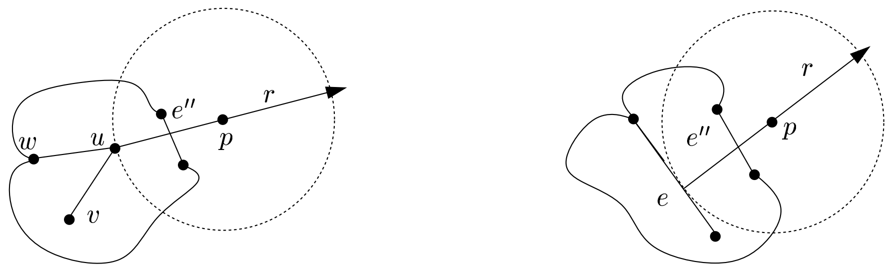


Figure 5.3: The proof of Lemma 11.

Proof. Let $\text{entry}(f,p) = e = (u,v)$ and suppose, for the sake of contradiction that e is only on the boundary of f. We distinguish between two cases. See Figure 5.3.

Case 1: $\text{c}(e)$ is a vertex. Without loss of generality assume that $\text{c}(e) = u$, and that the clockwise angle $\overset{\text{cw}}{\angle}(p,u,v) \leq \pi$ (otherwise reverse the roles of clockwise and counterclockwise). Let r be the ray originating at u and containing p. Consider the edge $e' = (u,w)$ that is the next edge in the counterclockwise adjacency list of u. Since $\text{entry}(f,p) \neq e'$, it must be that $\angle p,u,v \leq \angle p,u,w$. and therefore r intersects the interior of f. Since p is not contained in f, r must intersect some edge e'' on the boundary of f and between u and p on r. But then $\mathit{dist}(e'',p) < \mathit{dist}(e,p)$, contradicting the fact that $\text{entry}(f,p) = e$.

Case 2: $\text{c}(e)$ is in the interior of e. Then we can define r as the ray originating at $\text{c}(e)$ and containing p and obtain the same contradiction as above. □

5.2.2 Defining the Face Tree

The entry edges of G define a relationship on the faces of G. For a face f of G that does not contain p, we define $\text{parent}(f,p)$ as the face $f' \neq f$ of G that has $\text{entry}(f,p)$ on its boundary. By Lemma 11, this value is well-defined. The values of $\text{parent}(f,p)$ define a structure on the faces of G.

The *face tree* $\text{FT}(G,p)$ of G with respect to p is the graph whose vertex set consists of the faces of G. For each face f of G that does not contain p, $\text{FT}(G,p)$ includes the edge $(f, \text{parent}(f,p))$. An example of a face tree is shown in Figure 5.4.

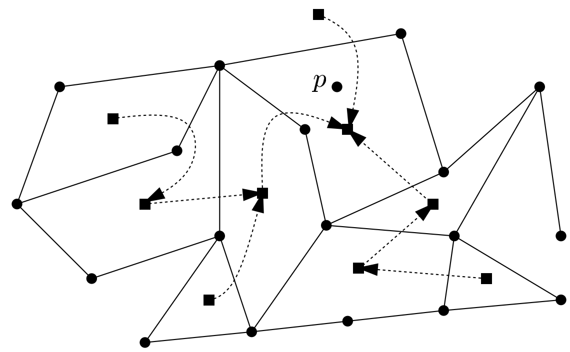


Figure 5.4: A planar geometric graph and its corresponding face tree. Vertices of $\mathrm{FT}(G,p)$ are shown as boxes. Edges of $\mathrm{FT}(G,p)$ pass through their defining entry edge and are directed from child to parent.

Lemma 12. $\mathrm{FT}(G,p)$ *is a tree.*

Proof. The number of vertices in $\mathrm{FT}(G,p)$ is equal to the number of faces in G, and the number of edges in $\mathrm{FT}(G,p)$ is one less than this number. Thus, all we need to show is that $\mathrm{FT}(G,p)$ is connected.

We will show that for every vertex f such that $\mathrm{parent}(f,p) = f'$ and $\mathrm{entry}(f',p)$ is defined, $\mathrm{entry}(f',p) \neq \mathrm{entry}(f,p)$ and $\mathrm{entry}(f',p) \preceq_p \mathrm{entry}(f,p)$. Since there are only a finite number of vertices in $\mathrm{FT}(G,p)$, this implies that following the parent pointers leads to a vertex f_r such that $\mathrm{entry}(f_r,p)$ is undefined. Since there is only 1 such vertex (the one containing p), there can be at most 1 connected component in $\mathrm{FT}(G,p)$.

Let f' be the parent of f, and assume that $\mathrm{entry}(f',p)$ is defined, otherwise there is nothing to prove. Let $e = (u,v) = \mathrm{entry}(f,p)$. Then we distinguish between two cases:

Case 1: $\mathrm{c}(e,p)$ is a vertex u. Consider the open ray r originating at u that contains p. From the proof of Lemma 11, the first face of G whose closure intersects r is then f'. Furthermore, since p is not contained in f', r intersects some edge e' on the boundary of f' at some point between u and v on r. Since $dist(e',p) < dist(e,p)$, it follows that $\mathrm{entry}(f',p) \neq \mathrm{entry}(f,p)$ and $\mathrm{entry}(f',p) \preceq_p \mathrm{entry}(f,p)$.

Case 2: $\mathrm{c}(e,p)$ is in the interior of e. In this case define r and the open ray originating at $\mathrm{c}(e,p)$ and containing p and proceed as in Case 1. □

5.3 Algorithms Using the Face Tree

In this section we show how to use the face tree to solve the routing, broadcasting and geocasting problems. The following lemma, which is trivial given that $\preceq_p$ defines a total order on the edges of G, shows that $\mathrm{FT}(G,p)$ is available to a routing algorithm.

Lemma 13. *Given p, the value of* $\mathrm{entry}(f,p)$ *can be computed using $O(1)$ memory by traversing the face f once.*

Proof. The traversal of f can be done using the right-hand rule, and since $\preceq_p$ is a total order $\mathrm{entry}(f,p)$ is computed by keeping track of the minimum edge visited with respect to $\preceq_p$. Both can easily be done with $O(1)$ memory. □

5.3.1 Point-to-point Routing

The FACE-ROUTE algorithm (Algorithm 1) solves the problem of routing from s to t by finding a path from s to the root of the face tree $FT(G,t)$. In the pseudocode, $\mathrm{opposite}(e,f)$ denotes the face $f' \neq f$ with e on its boundary and $\mathrm{face\_of}(v)$ is any face with v on its boundary.

Algorithm 1 The FACE-ROUTE algorithm.

```
v ← s
f ← face_of(v)
while v ≠ t do
    traverse f, until returning to v or reaching t, while computing entry(f, t).
    if v was reached then
        return to an enpoint w of entry(f, t).
        v ← w.
        f ← opposite(entry(f, t), f).
    else
        v ← t.
    end if
end while
```

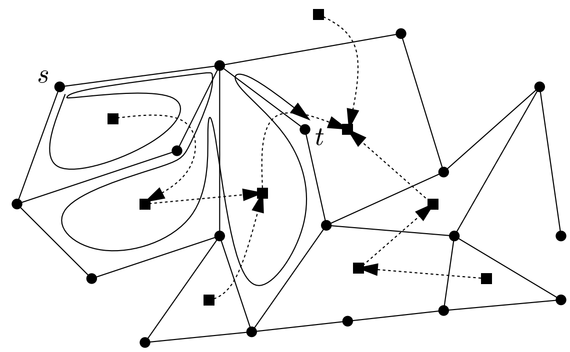


Figure 5.5: The path taken by a packet in the FACE-ROUTE algorithm.

Some steps of the algorithm need to be expanded on. As explained above, computing entry(f, t) in line 4 involves traversing f using the right hand rule. While doing this, the algorithm also keeps track of two additional values, one is a count of the number of steps taken (edges traversed) during the entire traversal, the other is the number of steps taken at the point where entry(f, t) was found.

When returning to entry(f, t) in the lines 8 and 11, the algorithm then uses these values to return via the shorter of the two possible paths. In particular, if $|f|$ is the number of steps taken while traversing f using the right hand rule, then the algorithm can use these values to compute entry(f, t) and return to entry(f, t) in at most $\lfloor 3|f|/2 \rfloor$ steps.

The path of a packet travelling from s to t using FACE-ROUTE is shown in Figure 5.5.

Theorem 15. *Algorithm* FACE-ROUTE *is an* $O(1)$ *memory routing algorithm with delivery time at most* $3|E|$.

Proof. That FACE-ROUTE is an $O(1)$ memory algorithm follows from the fact that for a face f, entry(f, t) can be computed using $O(1)$ memory with a traversal of f.

To prove that the delivery time of the algorithm is at most $3|E|$ we note that the algorithm considers each face of G at most once. The cost of visiting a face f is at most $\lfloor 3|f|/2 \rfloor$. Therefore, the delivery time of the algorithm is at most $\sum_{f \in F} \lfloor 3|f|/2 \rfloor$. Since each edge of E contributes exactly 2 to this sum, the delivery time is at most

$3|E|$. □

5.3.2 Broadcasting

Next we describe the FACE-BROADCAST algorithm (Algorithm 2) for solving the broadcasting problem. The algorithm works by performing a depth-first traversal of the face tree $\mathrm{FT}(G, s')$, where s' is a point arbitrarily close to s that is not on the boundary of any face.

Algorithm 2 The FACE-BROADCAST algorithm.

1: $f \leftarrow \mathrm{face\_of}(s')$
2: $e_{start} \leftarrow e \leftarrow \mathrm{edge\_of}(f)$
3: **repeat**
4:     **if** $e = \mathrm{entry}(f, s')$ **then**
5:         {* return to parent of f *}
6:         $f \leftarrow \mathrm{opposite}(e, f)$
7:     **else if** $e = \mathrm{entry}(\mathrm{opposite}(e, f), s')$ **then**
8:         {* visit child of f *}
9:         $f \leftarrow \mathrm{opposite}(e, f)$
10:     **end if**
11:     $e \leftarrow \mathrm{next}(e, f)$
12: **until** $e = e_{start}$

The notation $\mathrm{face\_of}(s')$ denotes the face containing s'. The notation $\mathrm{next}(e, f)$ denotes the edge that follows e while traversing f using the right hand rule. If f is an inner face, then $\mathrm{next}(e, f)$ is the next edge on the boundary of f in counterclockwise order, otherwise (f is the external face) it is the next edge on the boundary of f in clockwise order.

Although our model of online routing algorithms does not allow packets to move to edges, and has no concept of faces, it is convenient to describe algorithms using these notions. It is straightforward to see that the operation in FACE-BROADCAST can be simulated in our model. In particular, visiting any edge e requires only that we visit one of the endpoints of e.

If we assume for the time being that the algorithm has access to an oracle that allows it to make the tests in lines 4 and 7, then the algorithm implements a classic

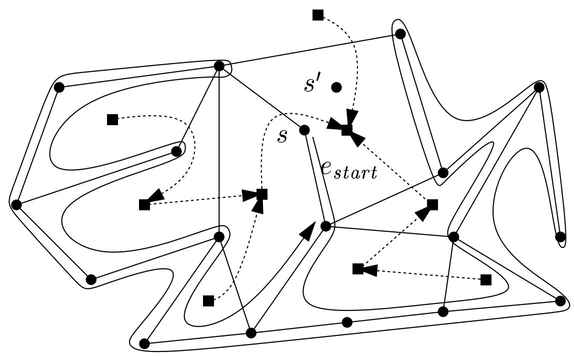


Figure 5.6: The path taken by a packet in the FACE-BROADCAST algorithm.

algorithm for traversing a rooted ordered tree without a stack (c.f., Cormen *et al.* [15, Exercise 11.4-5]). Each face of G can be viewed as a node in the tree, and each entry edge can be viewed as a pair of pointers between a parent and child in the tree (edges that are not entry edges for any face can be ignored).

An example of the execution of the algorithm is given in Figure 5.6. The correctness of the algorithm then follows from the correctness of the tree traversal algorithm. Furthermore, with our oracle assumption, the delivery time is easily seen to be $O(m)$ (recall that m is the number of edges in G).

Next we show that the tests in lines 4 and 7 can be performed in such a way that the amortized cost is $O(\log m)$ per test. Consider a face f with m edges $e_0, \ldots, e_{m-1}$. We say that an edge e_i is a *k-minimum* in f if $e_i \preceq_{s'} e_j$ for all $i-k \leq j \leq i+k$.[1] We define $\mathrm{minval}(e_i)$ as the maximum value for which e_i is a k-minimum. The following lemma provides an efficient means of testing if $e_i = \mathrm{entry}(f, s')$.

Lemma 14. *$\sum_{i=0}^{m-1} \mathrm{minval}(e_i) \leq m \cdot (H_m - 1)$, where H_x is the x^{th} harmonic number defined as $H_x = \sum_{i=1}^{x} 1/i$.*

Proof. If e_i is a k-minimum, then none of $e_{i-k}, \ldots e_{i-1}, e_{i+1}, \ldots, e_{i+k}$ is a k-minimum. Therefore, at most $\lfloor m/(k+1) \rfloor$ edges of f are k-minima. Thus,

$$\sum_{i=0}^{m-1} \mathrm{minval}(e_i) \quad = \quad \sum_{k=1}^{m} |\{e_i : e_i \text{ is a } k\text{-minimum}\}| \tag{5.4}$$

[1] In the remainder of this section, subscripts are taken mod$m$.

$$
\begin{aligned}
&\leq \sum_{k=1}^{m} \lfloor m/(k+1) \rfloor && (5.5)\\
&\leq m \cdot (H_m - 1) \ . && (5.6)
\end{aligned}
$$

□

Harmonic numbers have been studied extensively, and are known to satisfy the inequalities $\ln x \leq H_x \leq \ln x + 1$ [32, Section 6.3]. This suggests the TEST-ENTRY algorithm (Algorithm 3) for testing whether $\mathrm{entry}(f, s') = e_i$.

Algorithm 3 The TEST-ENTRY algorithm

```
k ← 1
j ← i
repeat
    while j ≠ i + k do
        j ← j + 1
        if e_i ⋠_{s'} e_j then
            output false
        end if
    end while
    k ← 2k
    while j ≠ i − k do
        j ← j − 1
        if e_i ⋠_{s'} e_j then
            output false
        end if
    end while
    k ← 2k
until k ≥ ⌈2n/3⌉
output true
```

Lemma 15. *The* TEST-ENTRY *algorithm correctly determines if* $\mathrm{entry}(f, s') = e_i$ *after at most* $9 \cdot \mathrm{minval}(e_i)$ *steps.*

Proof. The algorithm is clearly correct, since it only return false if an element e_j is found such that $e_i \not\preceq_{s'} e_j$, and only returns true after comparing e_i to all other edges of f.

To bound the running time, we note that if the algorithm terminates with $k = 2^i$, then $\text{minval}(e_i) > 2^{i-2}$. The total number of steps taken during the algorithm is

$$\begin{aligned} S(e_i) &= \sum_{j=0}^{i-1} 2 \cdot 2^j + \text{minval}(e_i) && (5.7) \\ &< 2^{i+1} + \text{minval}(e_i) && (5.8) \\ &\leq 8\text{minval}(e_i) + \text{minval}(e_i) && (5.9) \\ &= 9\text{minval}(e_i) \ , && (5.10) \end{aligned}$$

as required. □

When the tests in the FACE-BROADCAST algorithm are implemented using the TEST-ENTRY algorithm, we obtain the following result.

Theorem 16. *Algorithm* FACE-BROADCAST *is an $O(1)$ memory broadcasting algorithm with delivery time at most $40 \cdot m \cdot (H_m - 1)$.*

Proof. That FACE-BROADCAST is an $O(1)$ memory algorithm follows from the pseudocode in Algorithm 2 and Algorithm 3.

That FACE-BROADCAST delivers the message to each vertex in G follows from the fact that each vertex is incident to at least one edge, each edge is on the boundary of at least one face, and each face is visited by FACE-BROADCAST.

To prove the bound on the delivery time we note that each edge $e = (u, v)$ is tested at most 4 times (twice when it is traversed in direction $\overrightarrow{uv}$ and twice when it is traversed in direction $\overrightarrow{vu}$). By Lemma 15 the amortized cost of each test plus the cost of returning to e is at most $10 \cdot H_m$, for a total delivery time of $40 \cdot m \cdot (H_m - 1)$. □

5.3.3 Geocasting

To solve the geocasting problem we use a modification of FACE-BROADCAST. In particular, we redefine $\text{entry}(f, p)$ as the minimum edge (with respect to $\preceq_p$) that intersects r_t. I.e., for a face f, with a set S of edges on its boundary, we define $\text{entry}(f, p)$ as

$$\text{entry}(f, p) = x \in S : x \text{ intersects } r_t \text{ and } x \preceq_p y \text{ for all } y \neq x \in S \ . \qquad (5.11)$$

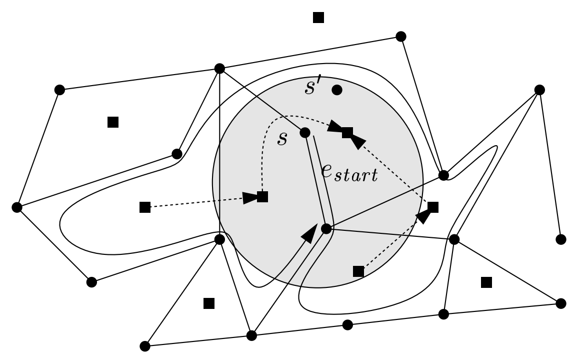


Figure 5.7: The path taken by a packet in the FACE-GEOCAST algorithm. The geocasting region r_t is shown as a shaded disk.

We refer to this algorithm as FACE-GEOCAST (Algorithm 4). An example of a path taken by a packet in the FACE-GEOCAST algorithm is shown in Figure 5.7.

Algorithm 4 The FACE-GEOCAST algorithm.

$f \leftarrow$ face_of(s')
$e_{start} \leftarrow e \leftarrow$ edge_of(f)
repeat
  if e intersects r_t **then**
    if $e =$ entry(f, s') **then**
      {* return to parent of f *}
      $f \leftarrow$ opposite(e, f)
    else if $e =$ entry(opposite$(e, f), s')$ **then**
      {* visit child of f *}
      $f \leftarrow$ opposite(e, f)
    end if
  end if
  $e \leftarrow$ next(e, f)
until $e = e_{start}$

Theorem 17. *Algorithm* FACE-GEOCAST *is an* $O(1)$ *memory geocasting algorithm with delivery time at most* $40 \cdot m' \cdot (H_{m'} - 1)$, *where* m' *is the total complexity of all faces of* G *that intersect* r_t.

Proof. That FACE-GEOCAST is an $O(1)$ memory algorithm follows from the pseudocode.

To show the correctness of the algorithm we observe that the set of vertices and edges that are on the boundaries of faces that intersect r_t form a connected subgraph H of G. Every face f of H, except the face containing s' and possibly the outer face have entry(f, s') defined. Furthermore, since r_t is convex, the arguments used in the proof of Lemma 12 still hold with respect to the new definition of entry edges. I.e., the entry edges define a tree T rooted at face_of(s') whose vertex set is the set of all edges that intersect r_t. Since the algorithm visits all the faces of T it must visit all the vertices in r_t.

The bound on the delivery time follows from the observation that the delivery time cannot be more than the delivery time of the FACE-BROADCAST algorithm restricted to the subgraph of G that contains only the vertices and edges on boundaries of the faces that intersect r_t. □

5.4 Summary and Open Problems

In this chapter we have given $O(1)$ memory algorithms for point-to-point routing, broadcasting and geocasting. Table 5.1 compares the results obtained in this chapter to previous work.

The algorithm for point-to-point routing is asymptotically optimal, while it is not clear whether the algorithms for broadcasting and geocasting are.

Open Problem 8. *Close the gap between the $O(n \log n)$ upper bound and the $\Omega(n)$ lower bound on the delivery time of an $O(1)$ memory broadcasting algorithm.*

We believe that a variation of the FACE-ROUTE algorithm could be used to solve the following open problem.

Open Problem 9. *Define and give routing algorithms for a model that takes into account dynamic networks in which vertices move and edges and vertices are inserted and deleted during the execution of the algorithm.*

Point-to-point routing				
Algorithm	References	Memory	Randomized	Delivery Time
GEOMETRIC-ROUTING	[52, 12]	$O(1)$	N	$3m$
FACE-ROUTE	here	$O(1)$	N	$3m$
Broadcasting				
Algorithm	References	Memory	Randomized	Delivery Time
RANDOM-WALK	trivial	none	Y	$O(n^2)$
DEPTH-FIRST-SEARCH	trivial	$O(n)$	N	$O(n)$
BKOO/BROADCAST	[19, 12]	$O(1)$	N	$O(n^2)$
FACE-BROADCAST	here	$O(1)$	N	$20 \cdot m \cdot H_m$
Geocasting				
Algorithm	References	Memory	Randomized	Delivery Time
BKOO/GEOCAST	[19, 12]	$O(1)$	N	$O((m')^2)$
FACE-BROADCAST	here	$O(1)$	N	$20 \cdot m' \cdot H_{m'}$

Table 5.1: Summary of Results in Chapter 5.

The difficulty here lies in defining a model that is both reasonably dynamic, yet does not allow for nasty executions that can defeat any algorithm.

5.5 Bibliographic Notes

The problem of visiting all the vertices of a planar geometric graph using $O(1)$ memory has received considerable attention. Gold *et al.* [30, 31, 29] gave an $O(n)$ time algorithm for traversing a triangulation given a point in the lower-left corner of the triangulation. The algorithm works by clasifying the edges of each triangle as either, “in”, “out” or “in-out” to obtain a walk that traverses each triangle in the triangulation.

Avis and Fukuda [2] give an algorithm for traversing arrangements of lines and convex polyhedra. The algorithm uses convexity in order to define, for each vertex v, a vertex parent(v) such that the parent pointers define a tree. This general technique, which we have made use of, is referred to as “reverse-search,” since it can be viewed as running the simplex algorithm for linear programming in reverse.

Edelsbrunner *et al.* [23] give an algorithm for traversing monotone subdivisions. The algorithm works by directing each edge in the direction of monotonicity (say the $+x$ direction). With the addition of two dummy vertices, the resulting directed graph is a directed acyclic graph D. Furthermore, for each vertex v of D, the set of incoming (outgoing) edges of v occur consecutively in clockwise order about v. This structure of D allows for a traversal algorithm very similar in nature to the standard tree traversal algorithm.

The most recent result in this area is that of de Berg *et al.* [19] who give an $O(n^2)$ time algorithm for traversing an arbitrary planar geometric graph. The face tree structure is based on the work of de Berg *et al.* [19]. In their work, they define $\text{entry}(f,p)$ using a rule very similar to our $\preceq_p$ order and show that, for an edge e on the boundary of f, testing if $e = \text{entry}(f,p)$ can be done in one traversal of f. Using this technique they derive an algorithm that enumerates a subdivision using $O(\sum_{f\in F}|f|^2)$ operations, which is $O(n^2)$ in the worst case. Thus, applying their work directly leads to a broadcast algorithm with $O(n^2)$ delivery time.

Kranakis *et al.* [52] describe a method for point-to-point routing in planar geometric graphs that they call *geometric routing*. Their algorithm is similar to the face tree method described in this chapter, but has the additional restriction that $\text{entry}(f,t)$ must intersect the line segment (s,t). Empirical results for their algorithm are reported by Bose *et al.* [12].

The face tree is described by Bose and Morin [11]. They describe algorithm FACE-BROADCAST as a means of traversing a subdivision without using mark bits, thus reducing the running time of the de Berg *et al.* algorithm from $O(n^2)$ to $O(n\log n)$.

The FACE-BROADCAST algorithm described in this chapter has some deep links with the distributed algorithm of Hirschberg and Sinclair [35] for leader election on a ring (see also Lynch [63, Chapter 3]). In fact, if one dualizes a face f, so that each edge becomes a vertex and each vertex becomes an edge, then the set of messages sent in testing whether an edge of f is an entry edge is almost exactly the same set of messages sent by the corresponding vertex in the Hirschberg–Sinclair leader election algorithm.

Chapter 6

Geometric Network Design

In Chapter 3 and Chapter 4 we studied the behaviour of routing algorithms on various classes of graphs. Our results suggest that if we are given a set of sites in the plane, then some ways of interconnecting the sites are better than others. For example, Delaunay triangulations tend to be especially amenable to online routing.

Another version of the problem, which we study in this chapter, is that we are given the interconnections between sites (an abstract graph G) and our job is to place the sites (assign positions to the vertices of G) in such a way as to facilitate online routing. This act of placing the sites is referred to as embedding the graph G. As usual, we are interested in embeddings of G that have the property that the resulting geometric graph is planar.

The remainder of this chapter is organized as follows: Our first result is to show how to embed any planar graph G so that a (rather complicated) geometric routing algorithm allows one to find the shortest path between any two vertices of G. We then study methods of embedding special kinds of planar graphs so that simple routing algorithms like COMPASS can be applied to the embeddings. We also show that not all planar graphs can be embedded so that GREEDY and COMPASS work on them. However, before we embark on the last leg of our journey, we review some definitions and results in graph theory.

For all the embedding algorithms described in this chapter we assume the real RAM model of computation in which algorithms can, in constant time, perform exact arithmetic operations on arbitrarily precise real numbers.

6.1 Graph Theory Review

An *(abstract) graph* $G = (V, E)$ is a graph whose vertex set is $V = \{1, \ldots, n\}$ and whose edges are pairs of integers in the range $[1, n]$. Recall that a *path* P in G is a sequence of vertices $(v_1, \ldots, v_k)$ such that $v_i \neq v_j$ for all $i \neq j$ and $(v_i, v_{i+1}) \in E$ for all $1 \leq i < k$. A *cycle* in G is a path $(v_1, \ldots, v_k)$ such that $(v_k, v_1) \in E$. Two paths (cycles) $(v_1, \ldots, v_k)$ and $(u_1, \ldots, u_l)$ are *edge-disjoint* if $(v_i, v_{i+1}) \neq (u_j, u_{j+1})$ for all $1 \leq i < k$ and $1 \leq j < l$.

We say that G is *connected* if for every pair of vertices $u, v \in V$, there exists a path in G from u to v. G is k-connected if $G \setminus X$ is connected for all $X \in V^{k-1}$.

A *planar drawing* of G is a drawing of G in the plane so that the vertices of G are represented by points, the edges of G by curves joining the vertices, and curves intersect only at common endpoints. A graph G is *planar* if it has a planar drawing. A theorem of Fáry [24] states that any planar graph has a planar drawing in which all edges are represented by straight line segments. Such drawings can be encoded by assigning positions to the vertices of G.

An *embedding* of G is a mapping $\Gamma : V \to \mathbb{R}^2$. We define $\Gamma(G) = (V', E')$ as the geometric graph where $V' = \{\Gamma(1), \ldots, \Gamma(n)\}$ and for each edge (u, v) in G, E' contains the edge $(\Gamma(u), \Gamma(v))$. We say that Γ is a *planar embedding* of G if $\Gamma(G)$ is a planar geometric graph.

6.2 The Ultimate Combination?

We begin by studying an embedding/routing combination that supports routing algorithms that always find shortest paths. In recent years, significant research has been spent on finding planar embeddings of planar graphs such that the vertices of the embedded graph lie on the vertices of a grid. This seems like a natural starting point for our algorithms. The tightest result in this area is due to Schnyder [74] (see also de Fraysseix *et al.*[20]).

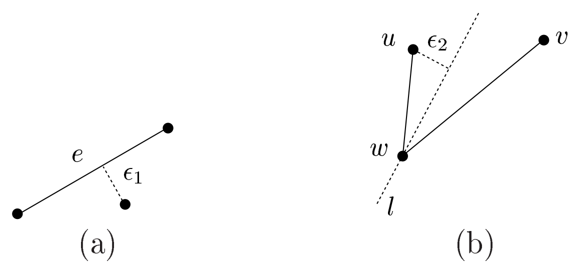


Figure 6.1: The definitions of (a) ϵ_1 and (b) ϵ_2.

Theorem 18 (Schnyder 1990). *For any planar graph G, there exists a planar embedding Γ of G such that $\Gamma : V \rightarrow \{1, \dots, n\}^2$. Furthermore, the embedding Γ can be computed in $O(n)$ time where n is the number of vertices in G.*

The approach used by our embedding algorithm is to take an embedding obtained from Theorem 18 and then tweak the locations of the vertices in order to encode shortest path routing information. The following lemma shows that this tweaking can always be done.

Lemma 16. *Let Γ be a planar embedding of G and let Γ' be any embedding of G such that $dist(\Gamma(i), \Gamma'(i)) < \epsilon$, for all $1 \leq i \leq n$. Then there always exists some $\epsilon > 0$ such that Γ' is planar.*

Proof. Let $\Gamma(G) = (V', E')$. We define ϵ as follows (see Figure 6.1):

$$\epsilon_1 = \min\{dist(v, e) : v \in V', e \in E' \setminus \mathrm{N}(v)\} \ . \tag{6.1}$$

For a vertex w with neighbours u and v, let $l(u, w, v)$ the line that bisects the angle $\angle u, w, v$. Then we define

$$\epsilon_2 = \min\{dist(l(u, w, v), u) : u, v, w \in V', u \neq v, (u, w) \in E', (v, w) \in E'\} \tag{6.2}$$

and set $\epsilon = \min\{\epsilon_1, \epsilon_2, 2\}/2$.

If $\Gamma'(G)$ is not planar then there are two edges e_1 and e_2 in $\Gamma'(G)$ that intersect at a point that is not an endpoint of one of the edges. There are two cases to consider:

Case 1: e_1 and e_2 are defined by 4 different vertices. Define tube(i, j) as the convex hull of the two circles of radius ϵ centered at $\Gamma(i)$ and $\Gamma(j)$. Then it must be that tube(e_1) and tube(e_2) intersect. But (6.1) ensures that this can never occur.

Case 2: e_1 and e_2 are defined by 3 different vertices i, j and k. Then it must be that $\Gamma'(i)$, $\Gamma'(j)$ and $\Gamma'(k)$ are collinear. But (6.2) ensures that this can never occur. □

Our next observation is that it is possible to encode an arbitrary amount of information in a real number in the interval $[0, \epsilon/2)$. In particular, it is possible to encode a shortest path routing table for any vertex. Thus, given an embedding Γ of G on the $n \times n$ grid, a shortest path routing table from a vertex v to all other vertices can be encoded by adding to the y-coordinate of each $\Gamma(i)$ a number in the interval $[0, \epsilon/2)$. We omit details of the encoding, since they are neither difficult nor particularly interesting.

Some care needs to be taken, since decoding these routing tables requires knowledge of ϵ, which depends on the initial embedding Γ and is therefore not available to a routing algorithm. However, this can easily be handled by adding the value of $\epsilon/2$ to the x-coordinate of each $\Gamma(i)$.

In summary, we obtain the HIGH-PRECISION-EMBED algorithm (Algorithm 5) for computing the embedding Γ' of a graph $G = (V, E)$.

Algorithm 5 The HIGH-PRECISION-EMBED algorithm.

```
1: compute the embedding Γ using Theorem 18
2: compute ε
3: for all v ∈ V do
4:    compute a shortest path routing table for v
5:    let μ be the encoding of v's routing table in [0, ε/2)
6:    Γ'(v) ← Γ(v) + (ε/2, μ)
7: end for
```

Lemma 17. *Algorithm* HIGH-PRECISION-EMBED *produces a planar embedding* Γ' *in* $O(n^2)$ *time.*

Proof. That the resulting embedding Γ' of G is planar follows from Lemma 16 and the fact that $dist(\Gamma'(i), \Gamma(i)) \leq \epsilon/\sqrt{2}$ for all $1 \leq i \leq n$. Step 1 of the algorithm can be accomplished in linear time using Theorem 18. Step 2 can be done trivially in $O(n^2)$ time. Each of the shortest path computations in Step 4 can be done in linear time using breadth-first search. Step 5 is easily implemented in linear time (remember, we're assuming a real RAM), and Step 6 takes constant time. Thus, the overall running time is $O(n^2)$. □

The HIGH-PRECISION-ROUTE routing algorithm (Algorithm 6) for routing on the geometric graph $\Gamma'(G)$ simply uses the encoded values to route along the shortest path from s to t. To do all this, the algorithm only needs to know v, t and $\mathrm{N}(v)$. Therefore, HIGH-PRECISION-ROUTE is a memoryless routing algorithm. Summarizing, we obtain the following result.

Algorithm 6 The HIGH-PRECISION-ROUTE routing algorithm.

```
v ← s
while v ≠ t do
    ε ← 2 · (x(t) − ⌊x(t)⌋)
    μ ← y(t) − ⌊y(t)⌋
    decode μ to find the next vertex v′ on the shortest path from v to t
    v ← v′
end while
```

Theorem 19. *For any planar graph G, a planar embedding Γ of G can be computed in $O(n^2)$ time such that the* HIGH-PRECISION-ROUTE *path between any two vertices s and t of $\Gamma(G)$ contains the minimum number of edges.*

Although it seems that Theorem 19 is the best result one could hope for, it is unsatisfactory because of the precision required in the location of vertices. In a bit model of computation, it would take $\Omega(n \log n)$ bits to represent each vertex identifier. Although our real RAM model allows us to do this using only the vertex identifiers, it is clear that in practice it would be simpler to just compute and store routing tables at each vertex.

6.3 Embeddings for Simple Routing Algorithms

Because of the drawbacks of the HIGH-PRECISION-EMBED/HIGH-PRECISION-ROUTE combination it is natural to want to study embeddings that support simple routing algorithms. Since we have already seen three very simple algorithms in Chapter 3 we study those. We begin with the (simpler) negative results and end with the (more complicated) constructive results.

6.3.1 Negative Results

Throughout this section $G = (V, E)$ will be a (abstract) graph. Let C be a cycle in G. We partition the edges of G into *pieces* as follows: Two edges e_1 and e_2 are in the same piece if there exists a path in G using e_1 and e_2 that does not contain any vertices of C. A *bridge* of C is the subgraph of G induced by the edges of a piece.

Let $B_1, B_2, \ldots, B_k$ be the bridges of C. Each B_i contains one or more vertices on C, and we call these the *attachments* of B_i. The following lemma, used in some proofs of Kuratowski's theorem, is intuitive but requires some topology for a rigorous proof (c.f., Bondy and Murty [6]).

Lemma 18. *If B_i and B_j have 3 attachments in common, then any planar embedding of G has all the vertices of B_i in the interior of the cycle defined by C and all the vertices of B_j in the exterior (or vice-versa).*

Theorem 20. *There exists a planar graph G such that no planar embedding of G supports* GREEDY, COMPASS, *or* GREEDY-COMPASS.

Proof. Consider the graph G shown in Figure 6.2 and let Γ be any embedding of G. Let C be the cycle (v_1, v_2, v_3, v_4). Then, by Lemma 18, one of the $\Gamma(s_1)$ or $\Gamma(s_2)$ must be contained in the interior of $\Gamma(C)$. Suppose without loss of generality that it is $\Gamma(s_1)$. All that needs to be shown is that GREEDY (respectively COMPASS, GREEDY-COMPASS) visits $\Gamma(v_6)$ when routing to some vertex $t \neq \Gamma(v_6)$.

For the GREEDY and GREEDY-COMPASS algorithms, the next vertex visited after s_1 depends on which side of the perpendicular bisector $\perp (\Gamma(v_5), \Gamma(v_6))$ the destination vertex t lies. For the COMPASS algorithm the next vertex visited after s_1 depends

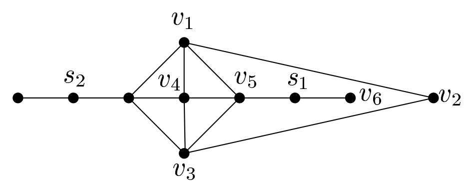


Figure 6.2: A graph that defeats COMPASS, GREEDY, and GREEDY-COMPASS.

on which side of the angular bisector of $\angle\Gamma(v_5), \Gamma(s_1), \Gamma(v_6)$ t lies. Since $\Gamma(s_1)$ and $\Gamma(s_1, v_6)$ are contained in $\Gamma(C)$, there must be some vertex $t \in \Gamma(C)$ on the wrong ($\Gamma(v_6)$) side of the appropriate bisector. Thus, any embedding of G defeats both GREEDY and COMPASS. $\square$

6.3.2 Embeddings for compass

Next we show that there is a large class of graphs that do support COMPASS. However, before beginning, we review some results from polytope theory.

A 3-*polytope* P is the convex hull of a set of 4 or more points in $\mathbb{R}^3$ not all contained in a common plane. The boundary of P, denoted ∂P is partitioned into *vertices* (points), *edges* (open line segments), and *faces* (open polygons). The *skeleton* of P, denoted skel(P) is an (abstract) graph defined by the edges and vertices of P. We call P a *realization* of skel(P).

A theorem of Steinitz is that a graph G has a realization as a 3-polytope if and only if G is 3-connected and planar [81, Chapter 4]. Steinitz' theorem has been generalized in several ways. A generalization of particular utility in the current context is due to Barnette [4].

Theorem 21 (Barnette 1970). *Let G be any 3-connected planar graph and let C be a cycle in G. There exists a realization of G as a polytope P such that the cycle C' in* skel(P) *corresponding to C satisfies $\partial proj(P) = proj(C')$. Such a realization can be computed in $O(n^2)$ time.*

This theorem says that we can choose any cycle in G and realize a polytope P so that the corresponding cycle in P is on the boundary of the shadow of P.

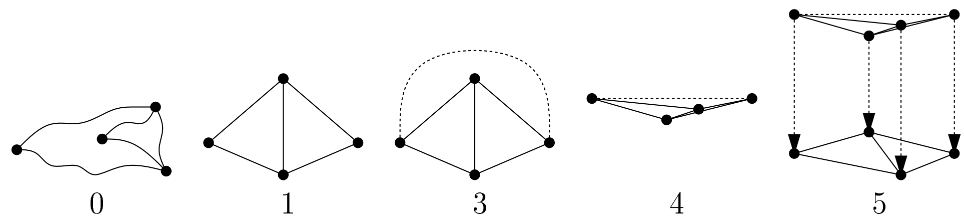


Figure 6.3: The operation of the COMPASS-EMBED algorithm.

A *near-triangulation* G is an abstract graph that has a planar embedding Γ such that $\Gamma(G)$ is a triangulation. The COMPASS-EMBED algorithm (Algorithm 7) can be used to embed any near-triangulation so that it supports compass. The operation of the algorithm is shown in Figure 6.3.

Algorithm 7 The COMPASS-EMBED algorithm.

1: compute an embedding Γ of G as a triangulation
2: let C be the cycle in $\Gamma(G)$ defining the outer face of G
3: add edges to G until it is maximal and planar to obtain a graph G'
4: use Theorem 21 to compute a 3-polytope P that is a realization of G such that $\partial proj(P) = proj(C')$, where C' is the cycle in skel(G) corresponding to C
5: project the vertices and edges of P corresponding to vertices and edges of G onto the x, y plane to obtain Γ

Lemma 19. *Algorithm* COMPASS-EMBED *produces a planar embedding* Γ *of* G *in* $O(n^2)$ *time.*

Proof. To prove the correctness of the algorithm, we need to show that G' is 3-connected, so that Theorem 21 can be applied in Step 4, and that the embedding Γ is planar. The former follows from the fact that every maximal planar graph is 3-connected [6]. The latter follows from the fact that the edges of P corresponding to edges of G are all on the lower (or upper) hull of P (since they are all on one side of C'), thus the embedding Γ is a regular triangulation (see Section 3.1).

To prove the running time on the algorithm we note simply that all steps are easily implemented in $O(n^2)$ time using known or cited algorithms. □

Theorem 22. *Let $\Gamma(G)$ be the embedding of a near-triangulation G computed by* COMPASS-EMBED*. Then $\Gamma(G)$ does not defeat the* COMPASS *algorithm.*

Proof. This follows immediately from Theorem 2 and the fact that $\Gamma(G)$ is, by definition, a regular triangulation. □

6.3.3 The left-compass Algorithm

Next we present a routing/embedding combination that allows us to perform routing on a more general class of graphs. The routing algorithm is a variant of COMPASS that is more capable of handling non-triangular faces. We begin with some definitions.

Let C_1 and C_2 be two edge-disjoint cycles in a graph $G = (V, E)$. The *connectivity* between C_1 and C_2 in G, denoted $\lambda(C_1, C_2)$ is defined as the least integer k such that there are complementary subsets E_1 and E_2 of E satisfying: (1) $C_1 \subseteq E_1$ and $C_2 \subseteq E_2$, and (2) the number of vertices incident with members of both E_1 and E_2 is k. Intuitively (though not exactly), k is the minimum number of vertices that need to be removed in order to separate C_1 from C_2 in G.

Let G be a planar graph and C be a cycle in G such that there exists some embedding Γ of G in which $\Gamma(C)$ is the boundary of a face of $\Gamma(G)$. Then (G, C) is a *subdivided circuit graph* [6] if the following conditions are met:

1. G is 2-connected.

2. For each cycle C' in G that is edge-disjoint from C, $\lambda(C', C) \geq 3$.

3. There is no path $P = (v_1, \ldots, v_k)$ in G such that $v_2, \ldots, v_{k-1}$ are all degree 2 vertices and (v_1, v_k) is an edge in C.

Subdivided circuit graphs were defined by Tutte [79] who showed that a planar graph G has an embedding as a convex subdivision with C on the outer face if and only if (G, C) is a subdivided circuit graph. This result is commonly referred to as *Tutte's theorem.*

We will give a routing and embedding algorithm for subdivided circuit graphs. As with the COMPASS-EMBED algorithm, the embedding algorithm will make use of

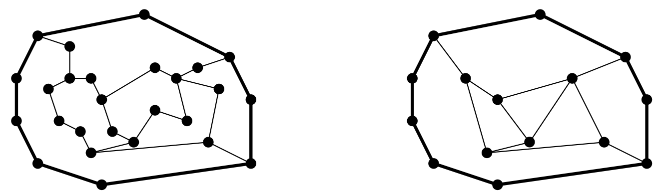


Figure 6.4: The graphs (a) G and (b) G^c. The cycle C is shown in bold.

Theorem 21 (Barnette's theorem). However, the algorithm does not add extra edges to make G a triangulation, since this would result in badly-shaped faces when these edges were removed. Rather, the algorithm applies some transformations to G to obtain a nicer 3-connected graph G' that still "looks a lot like" G.

Let $P = (v_1, \ldots, v_k)$ be any path in G that joins two vertices of degree 3 or more and which has no vertices of degree greater than 2 in its interior. A *path compression* operation involves removing $v_2, \ldots, v_{k-1}$ from G and inserting the edge (v_1, v_k). The first step in our embedding algorithm is to perform path compression on all paths of G that are not contained in C. We call the resulting graph G^c. The path compression step is illustrated in Figure 6.4.

Lemma 20. *G^c is planar, 2-connected, and does not contain parallel edges.*

Proof. That G^c is planar and 2-connected is clear, since each individual path compression operation preserves planarity (recall that G^c is an abstract graph) and 2-connectivity.

Next, suppose that the edge (u, v) appears twice in G^c. These two occurrences of (u, v) correspond to a cycle $C_{u,v}$ in G. By Property 3 above, $C_{u,v}$ and C must be edge-disjoint. But then $\lambda(C_{u,v}, C) \leq 2$, contradicting Property 2. □

The second, and final, step in computing the graph G' is to add a dummy vertex v that is adjacent to all vertices of C and call the resulting graph G'.

Lemma 21. *G' is planar and 3-connected.*

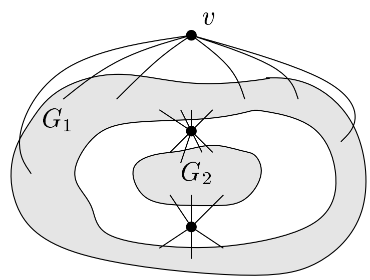


Figure 6.5: The proof of Lemma 21.

Proof. By the definition of subdivided circuit graphs, there exists an embedding of G' with all the vertices of C on a single face. Therefore it is clear that v and all its incident edges can be added while preserving planarity.

Next we prove that G' is 3-connected. Suppose, by way of contradiction, that G' has two vertices v_1 and v_2 whose removal disconnects G'. Neither v_1 nor v_2 can be the dummy vertex v since G^c is itself 2-connected (Lemma 20).

$G \setminus \{v_1, v_2\}$ contains two (or more) connected components G_1 and G_2. Furthermore, all vertices of $C \setminus \{v_1, v_2\}$ are contained in one component, say G_1 (because the dummy vertex v is adjacent to all vertices of $C \setminus \{v_1, v_2\}$) (Refer to Figure 6.5). Now, consider G_2. We claim that G_2 cannot contain any cycle C', since then $\lambda(C', C) \leq 2$, contradicting Property 2 in the definition of a subdivided circuit graph.

In G_2, all vertices except possibly those incident to v_1 or v_2 in G^c have degree greater than 2. Thus, G_2 must be a tree whose leaves are a subset of the vertices adjacent to v_1 or v_2 in G. We claim that G_2 must have exactly two leaves. Suppose that this were not the case, then the subgraph G_2^+ of G induced by the vertices of G_2 and $\{v_1, v_2\}$ contains a cycle C' such that $\lambda(C', C) \leq 2$.

Therefore, G_2 is a tree with exactly two leaves, i.e., a path. But this is a contradiction, since then G_2 would have been reduced to the edge (v_1, v_2) during the path compression phase of the embedding algorithm. □

At this point we can apply Theorem 21 to the graph G' and choose C^c as the special cycle. Projecting the lower (upper) hull of the resulting polytope P onto the x, y-plane gives a regular subdivision S' in which the vertices of S' correspond to

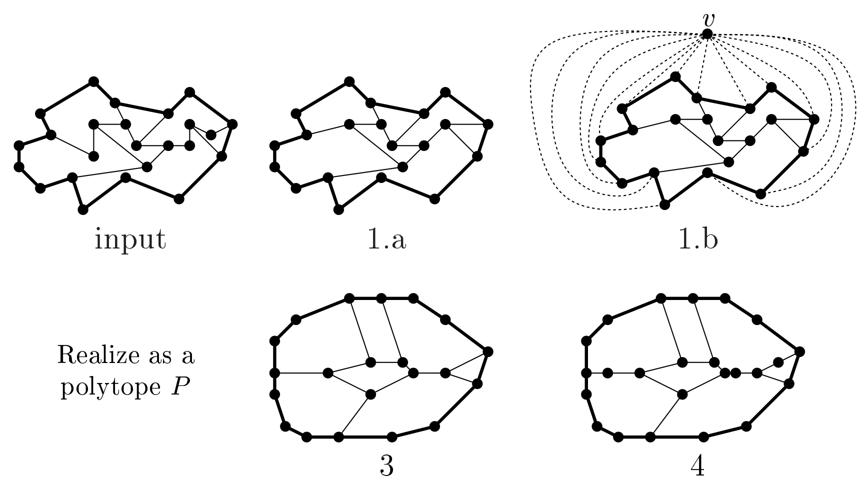


Figure 6.6: The operation of the LEFT-COMPASS-EMBED algorithm. The cycle C is shown in bold.

vertices of G'. The edges of S' correspond to paths in G whose interior vertices all have degree 2. Thus, we can subdivide the edges of S' to obtain an embedding of G. We refer to this embedding algorithm as the LEFT-COMPASS-EMBED algorithm (Algorithm 8). The operation of LEFT-COMPASS-EMBED is shown in Figure 6.6.

Algorithm 8 The LEFT-COMPASS-EMBED algorithm.

1: compute the graph G'
2: compute the polytope P using Theorem 21
3: project the lower (upper) hull of P to obtain the regular subdivision S'
4: subdivide the edges of S' to obtain an embedding Γ of G

Theorem 23. *Algorithm* LEFT-COMPASS-EMBED *computes a planar embedding of* G *in* $O(n^2)$ *time.*

Proof. All steps of the algorithm can easily be implemented in $O(n^2)$ time using known techniques. That the resulting embedding is planar follows from the fact that it is the orthogonal projection of the lower (upper) convex hull of a 3-polytope. That it is an embedding of G follows from the algorithm for its construction. □

We note that Theorem 23 provides an alternate (less direct) proof of Tutte's theorem, since S is a convex subdivision with C on its outer face.

Next we describe the routing algorithm LEFT-COMPASS that we use for routing on $\Gamma(G)$. The algorithm is memoryless and deterministic and always moves the neighbour u of the current vertex v which minimizes the counterclockwise angle $\overset{\text{ccw}}{\angle}\, t, u, v$. More formally, LEFT-COMPASS is defined by the transition function

$$\mathrm{lcmp}(v) = u \in \mathrm{N}(v) : \overset{\text{ccw}}{\angle}\, t, u, v \leq \overset{\text{ccw}}{\angle}\, t, w, v, \text{ for all } w \in \mathrm{N}(v) \ . \tag{6.3}$$

Intuitively, LEFT-COMPASS is a "left-biased" version of the COMPASS algorithm. This biasing saves LEFT-COMPASS from repeatedly traversing the same edge in a convex subdivision.

Lemma 22. *Let S be a convex subdivision that defeats* LEFT-COMPASS*, and let t be a vertex such that* LEFT-COMPASS *fails to route a packet to t when given some other vertex as the source. Then there exists a cycle $C = (v_0, \ldots, v_{k-1})$ $(k \geq 3)$ in S such that* $\mathrm{lcmp}(v_i) = v_{i+1}$ *for all $0 \leq i < k$.*[1]

Proof. Since S defeats LEFT-COMPASS, and LEFT-COMPASS is a memoryless algorithm, then either there is an edge (u, v) such that $\mathrm{lcmp}(u) = v$ and $\mathrm{lcmp}(v) = u$, or there is the situation described in the lemma. We prove that there can be no such edge (u, v).

Refer to Figure 6.7. Assume that such an edge (u, v) exists, then $\overset{\text{ccw}}{\angle}\, t, u, v \geq \pi$ or $\overset{\text{ccw}}{\angle}\, t, v, u, \geq \pi$. Suppose then that $\overset{\text{ccw}}{\angle}\, t, v, u \geq \pi$. Then for all $w \in N(v)$, $\overset{\text{ccw}}{\angle}\, w, v, u > \overset{\text{ccw}}{\angle}\, t, v, u$. But this implies that v is a reflex vertex on the boundary of some face f, contradicting the fact that S is a convex subdivision. □

Theorem 24. *Let $\Gamma(G)$ be any planar geometric graph output by the* LEFT-COMPASS-EMBED *algorithm. Then $\Gamma(G)$ does not defeat* LEFT-COMPASS.

Proof. Assume by way of contradiction that $\Gamma(G)$ defeats LEFT-COMPASS. Let $v_0, \ldots, v_{k-1}$ be the set of vertices such that $\mathrm{lcmp}(v_i) = v_{i+1}$ for $0 \leq i < k$. By Lemma 22, such a set of vertices exists. Let $\circ(v_i, v_{i+1})$ be the face on the left of the edge (v_i, v_{i+1}). Then,

[1] *Here, and in the remainder of this section, subscripts are taken* $\bmod k$.

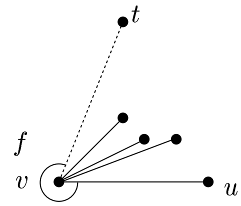


Figure 6.7: The proof of Lemma 22.

by the same argument used in the proof of Lemma 4, either $\circ(v_{i-1}, v_i) = \circ(v_i, v_{i+1})$ or $\circ(v_i, v_{i+1})$ overlaps $\circ(v_{i-1}, v_i)$ with respect to the viewpoint t. Thus, there is a set of faces that overlap cyclically from the viewpoint t.

The set of faces of $\Gamma(G)$ define the same polygons as the faces of S', and S' is a regular subdivision. However, Edelsbrunner's result [22] states that no regular subdivision has a set of faces that overlap cyclically from any viewpoint. Thus, we obtain the required contradiction. □

6.4 Summary and Open Problems

In this chapter we have studied embedding and routing combinations. We saw that using existing results on grid embeddings of planar graphs, a routing/embedding combination that supports shortest-path routing was not very difficult to achieve, even with a memoryless routing algorithm. Unfortunately, the algorithm is more of theoretical interest, since the vertex coordinates require $\Omega(n \log n)$ bits to represent.

On the more practical side, we studied how the simple algorithm COMPASS and its variant LEFT-COMPASS can be used in combination with embedding algorithms. Although these algorithms don't guarantee shortest paths, they are simple and practical to implement and have some interesting theoretical properties. Along the way, we were able to unify a classic result in planar graph theory (Tutte's theorem) with a classic result in polytope theory (Barnette's version of Steinitz' theorem).

As for open problems, there are still many of them. We have argued that the HIGH-PRECISION-EMBED/HIGH-PRECISION-ROUTE combination is unsatisfactory for several reasons. This raises the following problem.

Open Problem 10. *Is there an embedding/routing combination that embeds vertices on a polynomial sized grid and always does (approximate) shortest path routing?*

An affirmative answer to this question could improve on some existing results in the area of *compact routing*. Specifically, it would give a compact routing algorithm where vertices receive $O(\log n)$ bit identifiers and which achieves (approximate) shortest-path routing.

We have given embedding algorithms for COMPASS and its variant LEFT-COMPASS, but have not obtained corresponding results for the GREEDY algorithm.

Open Problem 11. *Study embeddings that support* GREEDY.

A number of problems related to embeddings for the COMPASS algorithm also remain open.

Open Problem 12. *Does every tree have an embedding on a polynomial sized grid that supports* COMPASS*?*

An *orthogonal embedding* is one in which all edges are parallel to the x or y axis. Singh [77] has shown that not all trees of maximum degree 4 have an orthogonal embedding that supports COMPASS. This raises the following open problem.

Open Problem 13. *Does every binary tree have an orthogonal embedding that supports the* COMPASS *algorithm?*

Our embeddings for the COMPASS and LEFT-COMPASS algorithms make use of a high precision arithmetic model of computation. An interesting (and fundamental) problem in polytope theory is that of realizing polytopes on small point sets.

Open Problem 14. *Let $f(n)$ be the minimum number such that any 3-connected planar graph can be realized as a 3-polytope on a grid of size $f(n) \times f(n) \times f(n)$. Prove that $f(n)$ is polynomial.*

The closely related problem of drawing planar graphs on small grids has received considerable attention [20, 74] and met with success. The problem of realizing 3-polytopes on small grids has been considered by Onn and Sturmfels [68] who show that $f(n) \leq 43^n$, by Richter-Gebert [73] who shows that $f(n) \leq 2^{13n^2}$ and by Chrobak *et al.* [14] who show that $f(n) \leq 2^{O(n \log n)}$ and give a linear time algorithm for achieving this bound.

6.5 Bibliographic Notes

Shortest path routing in abstract graphs has usually been accomplished by means of routing tables, which in some cases can be stored compactly. For a general overview of compact routing techniques, see the survey article by van Leeuwen and Tan [80].

For planar graphs Gavoille and Hanusse [27] have shown that shortest path routing can be implemented by storing $8n+o(n)$ bits at each vertex, in contrast to the obvious $O(n \log n)$ bits required to store routing tables. The algorithm also has the advantage that the node identifiers are $1, \ldots, n$ and that each routing step requires only $O(\log^2 n)$ bit operations.

Frederickson and Janardan [25] have shown that by storing $O(n^{4/3} \log n)$ bits over *all* vertices, it is possible to perform routing so that paths are at most 3 times the length of the shortest path. The same work also shows how to compute a path of length at most 7 times the shortest path by storing only $O((1/\epsilon)n^{1+\epsilon} \log n)$ bits over all vertices.

A careful analysis of the HIGH-PRECISION-EMBED/HIGH-PRECISION-ROUTE combination given in this chapter reveals that it requires $\Omega(n \log n)$ bits of storage at each vertex and is therefore not competitive with the results of Gavoille and Hanusse or Frederickson and Janardan. It may be possible to reduce the memory requirements by applying their techniques, but doing so does not appear to be a particularly enlightening or useful exercise.

Much of Singh's Master's thesis [77] (see also Kranakis *et al.* [52]) is devoted to embeddings that support the COMPASS algorithm. Singh shows how to embed any tree to support the COMPASS algorithm, but the ratio of minimum to maximum edge length can be exponential in n. Another variant includes orthogonal embeddings of trees of maximum degree 4 that support COMPASS when the source s is always the root of the tree (referred to as single-source routing). Singh also gives impossibility results showing that not all trees of maximum degree 4 have orthogonal embeddings that support COMPASS, and that not all outerplanar graphs have embeddings for which COMPASS always finds a shortest path.

All results on embedding and routing presented in this chapter appear here for the first time.

Chapter 7

Experimental Results

In this chapter we take a step back from theory and study the *actual* behaviour of online routing algorithms on different types of input graphs. More precisely, we study the empirical performance of online routing algorithms on randomly generated planar geometric graphs.

The purpose of this chapter is not to develop highly detailed implementations of routing algorithms specified down to the protocol level. Nor is it to develop sophisticated simulation models of real-life networks. Rather, we study idealized routing algorithms on randomly generated graphs. Our hope is that the information we gather can act as a guide to help determine which algorithms should work well in practice. However, the reader should be warned that, for any particular application, implementation and protocol details can be very important and should not be neglected.

The experimental results in this chapter are grouped by the type of input graph, with Delaunay triangulations, Graham triangulations (defined below), meshes, and unit disk graphs being the types of graphs considered. We choose to study these types of graphs because they represent several different applications of online routing.

Our experiments measure three quantities. Let G be a graph and denote by $P(G)$ all pairs of vertices u and v such that u and v are in the same connected component of G. For a routing algorithm $\mathcal{A}$ we denote by $S(\mathcal{A}, G)$ all pairs of vertices (s, t) in G

such that $\mathcal{A}$ succeeds in routing from s to t. More formally,

$$S(\mathcal{A}, G) = \{(s,t) \in P(G) : \mathcal{A} \text{ succeeds in routing from } s \text{ to } t\} \ .$$

We call $S(\mathcal{A}, G)$ the *successes* for $\mathcal{A}$ on G. The *success rate* of $\mathcal{A}$ on G is then defined as

$$\mathrm{SR}(\mathcal{A}, G) = \frac{|S(\mathcal{A}, G)|}{|P(G)|} \ .$$

The value $\mathrm{SR}(\mathcal{A}, G)$ measures how often $\mathcal{A}$ succeeds in routing tasks on G.

To measure the efficiency of $\mathcal{A}$ on G we measure the *average Euclidean dilation* $\mathrm{AED}(\mathcal{A}, G)$ and the *average link dilation* $\mathrm{ALD}(\mathcal{A}, G)$. Let $W(\mathcal{A}, G, s, t)$ denote the walk taken by $\mathcal{A}$ when routing from s to t. The we define

$$\mathrm{AED}(\mathcal{A}, G) = \frac{1}{|S(\mathcal{A}, G)|} \cdot \sum_{(s,t) \in S(\mathcal{A}, G)} \frac{\mathrm{length}(W(\mathcal{A}, G, s, t))}{\mathrm{length}(\mathrm{SEP}(G, s, t))} \ ,$$

where $\mathrm{SEP}(G, s, t)$ denotes a shortest path from s to t in G under the Euclidean distance measure.

Similarly, we define

$$\mathrm{ALD}(\mathcal{A}, G) = \frac{1}{|S(\mathcal{A}, G)|} \cdot \sum_{(s,t) \in S(\mathcal{A}, G)} \frac{|W(\mathcal{A}, G, s, t)|}{|\mathrm{SLP}(G, s, t)|} \ ,$$

where $\mathrm{SLP}(G, s, t)$ denotes a shortest path from s to t in G under the link distance measure.

7.1 Delaunay Triangulations

Figures 7.1 and 7.2 show results for routing in randomly generated Delaunay triangulations with up to 500 vertices. These triangulations were generated by selecting n points uniformly distributed in the unit square and computing their Delaunay triangulation.

Figure 7.1 shows the average Euclidean dilation for GREEDY, COMPASS, GREEDY-COMPASS, RANDOM-COMPASS, VORONOI, and PARALLEL-VORONOI, while Figure 7.2

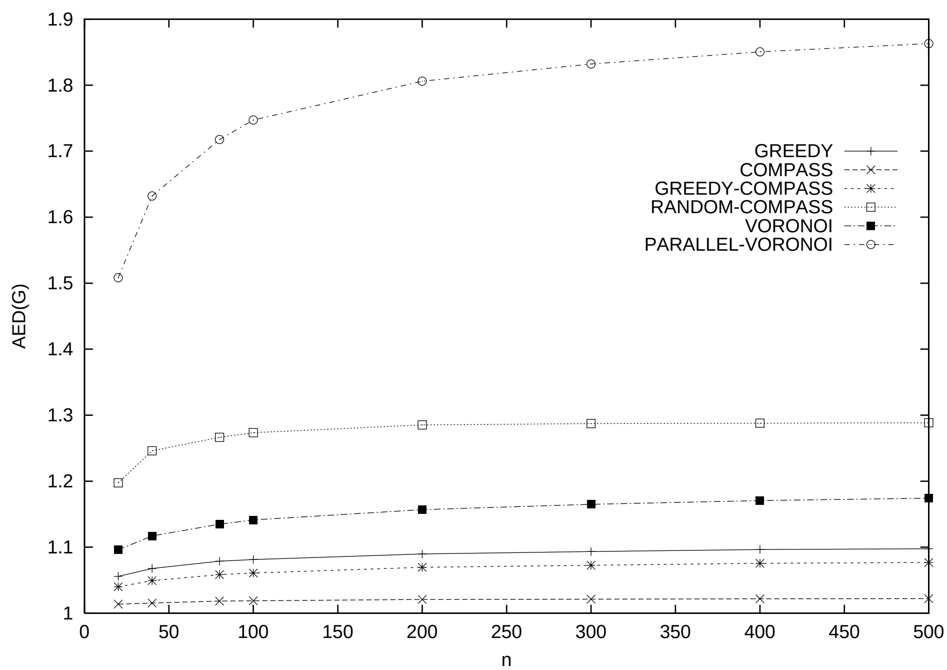


Figure 7.1: Average Euclidean dilation of routing algorithms on Delaunay triangulations.

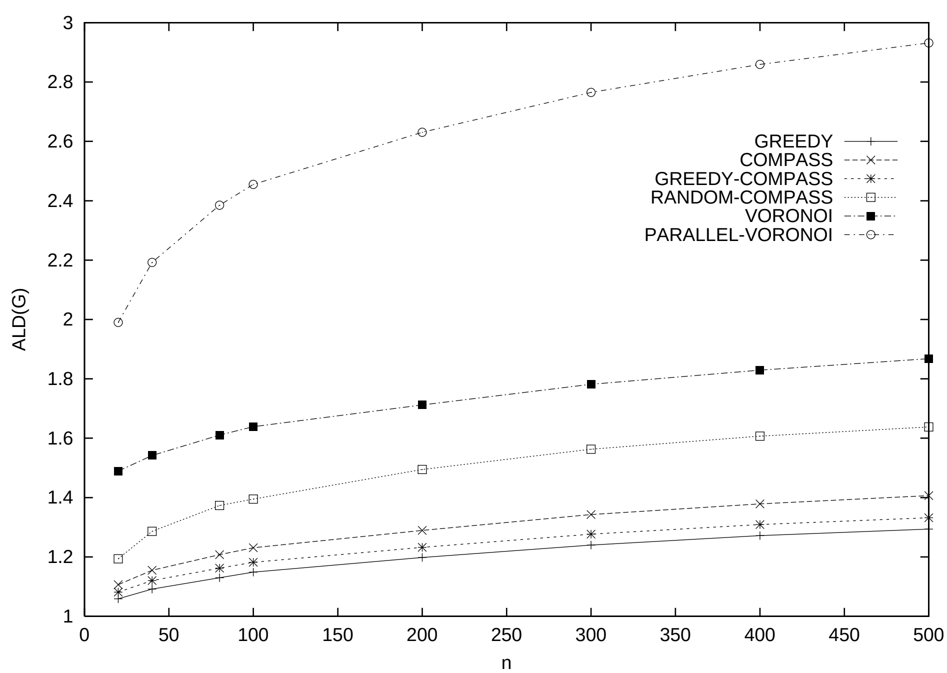


Figure 7.2: Average Link dilation of routing algorithms on Delaunay triangulations.

Rank	Euclidean distance	Link distance
1.	COMPASS	GREEDY
2.	GREEDY-COMPASS	GREEDY-COMPASS
3.	GREEDY	COMPASS
4.	VORONOI	RANDOM-COMPASS
5.	RANDOM-COMPASS	VORONOI
6.	PARALLEL-VORONOI	PARALLEL-VORONOI

Table 7.1: Ranking routing algorithms based on average dilation.

shows the average link dilation for the same algorithms. Because all of these algorithms always succeed on Delaunay triangulations, there are no statistics on success rate.

These graphs show that these algorithms can be ranked fairly clearly in terms of average dilation. This ranking is shown in Table 7.1 for both Euclidean and link distance. One interesting point about this ranking is that under Euclidean distance, COMPASS performs better than GREEDY, while the opposite is true for link distance. This can be explained by the following intuition: By traversing the edge whose direction is closest to the direction $\overrightarrow{st}$, the COMPASS algorithm attempts to maximize the ratio of progress (reduction in distance to t) to Euclidean distance travelled, and therefore results in efficient paths under the Euclidean distance measure. On the other hand, GREEDY attempts to make the maximum progress at each step, regardless of the Euclidean distance travelled. The result is that GREEDY generally requires fewer steps and therefore results in efficient paths under the link distance measure

Another interesting, and somewhat unfortunate, aspect of these results is that PARALLEL-VORONOI, the only algorithm that is competitive under the Euclidean distance measure, performs the worst under both the Euclidean and link distance measures. This is due to the fact that PARALLEL-VORONOI is quite conservative, and uses backtracking to ensure that it achieves the bound on its competitive ratio. However, the configurations that cause the other algorithms to perform poorly don't seem to occur frequently in random point sets, and this backtracking effort is wasted.

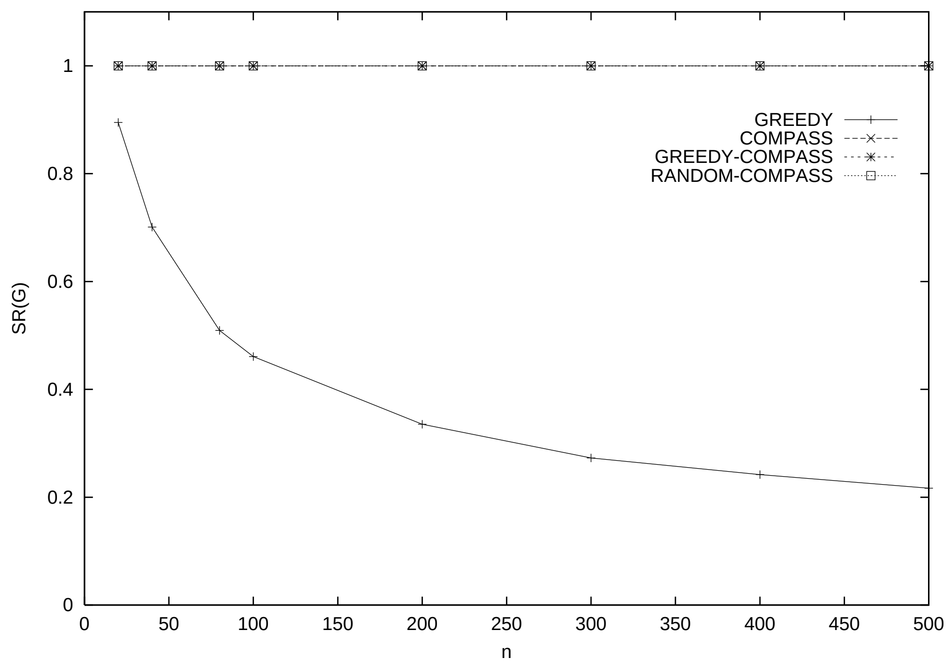


Figure 7.3: Success rates of routing algorithms on Graham triangulations.

7.2 Graham Triangulations

A *Graham triangulation* is a triangulation obtained by sorting a set of input points by x-coordinate and then using Graham's scan to triangulate the resulting x-monotone chain (c.f., [70]). In this section we discuss results for routing on randomly generated Graham triangulations. As before, these triangulations were generated by selecting uniformly distributed points in the unit square and then triangulating them. These results are shown in Figures 7.3–7.5.

Figure 7.3 shows success rates of various algorithms on Graham triangulations with up to 500 vertices. Interestingly, the only algorithm that does not have a success rate of 1 is GREEDY. In fact, the success rate of GREEDY is very bad and seems to decrease as the number of vertices increases, becoming as low as .2 for triangulations with 500 vertices. Because of this, the results on average dilation are not very meaningful for GREEDY, and we omit them from our discussion.

Figures 7.4 and 7.5 show results for average Euclidean dilation and average link

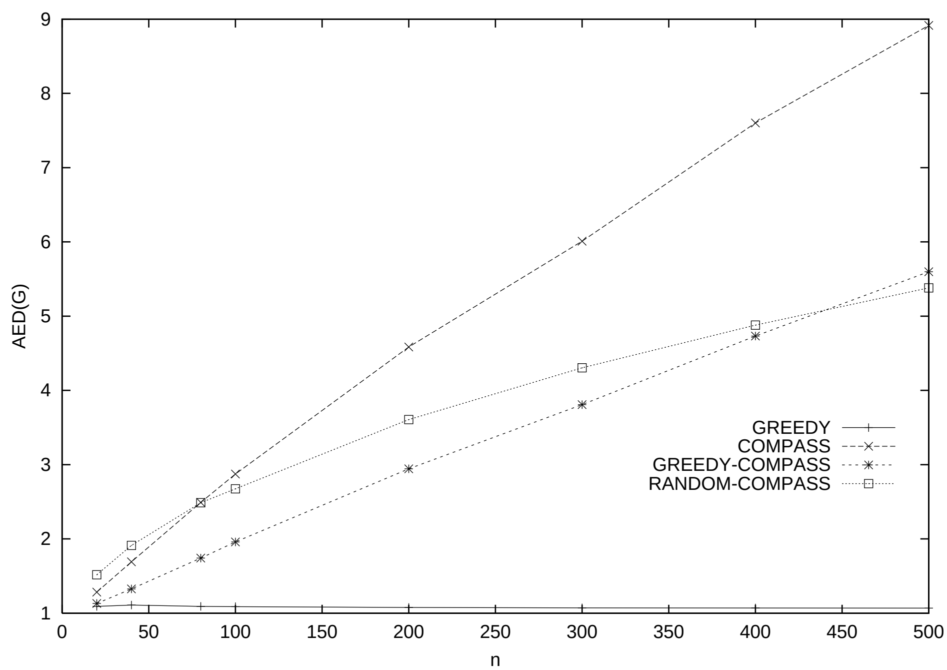


Figure 7.4: Average Euclidean dilation of routing algorithms on Graham triangulations.

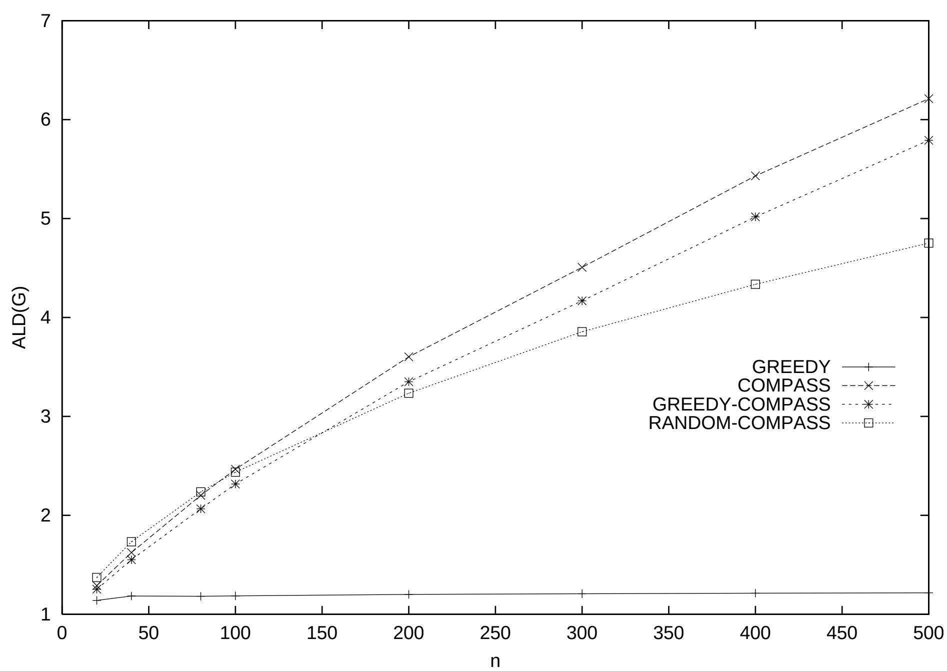


Figure 7.5: Average Link dilation of routing algorithms on Graham triangulations.

dilation, respectively. Unlike our results for Delaunay triangulations, the GREEDY-COMPASS and RANDOM-COMPASS algorithms outperform the COMPASS algorithm under both Euclidean and link distance. This is because, with Graham triangulations, the COMPASS algorithm has a tendency to "overshoot" its destination by traversing edges that take it beyond the destination vertex t. The GREEDY-COMPASS algorithm avoids this problem by taking the compass neighbour of the current vertex that is closest to t.

What is perhaps more surprising about these results is that as the number of vertices increases, RANDOM-COMPASS begins to perform better than both COMPASS and GREEDY-COMPASS under both distance measures. The reason for this is unclear, and requires further study.

7.3 Meshes with Faults

Next, we study the performance of routing algorithms on regular square meshes with randomly placed faults. An n vertex *faulty mesh* with *fault rate* α is obtained by taking a mesh with $n(1+\alpha)$ vertices and removing αn vertices chosen uniformly at random without replacement.[1]

Figure 7.6 shows the success rates for COMPASS, GREEDY, and GREEDY-COMPASS on meshes with number of vertices ranging from 25 to 215 and fault rates ranging from 5% to 50%. Basically, the success rates for the 3 algorithms are indistinguishable, and become as low as 40% for large meshes with high fault rates.

Figure 7.7 shows average dilation for COMPASS, GREEDY, and GREEDY-COMPASS compass on faulty meshes. Note that, because all edges have length 1, the average Euclidean and average link dilation are identical. Again, the three algorithms are indistinguishable, but all achieve average dilation close to 1.

The average dilation for all three algorithms tends to peak when the fault rate is in the range 20–30%. This is because in this range, the success rate is still relatively high, but there are enough faults that many routing tasks must route around the faults, and do so sub-optimally. When the fault rate is lower than this, most routing

[1] Here we assume that $n(1+\alpha)$ is a perfect square.

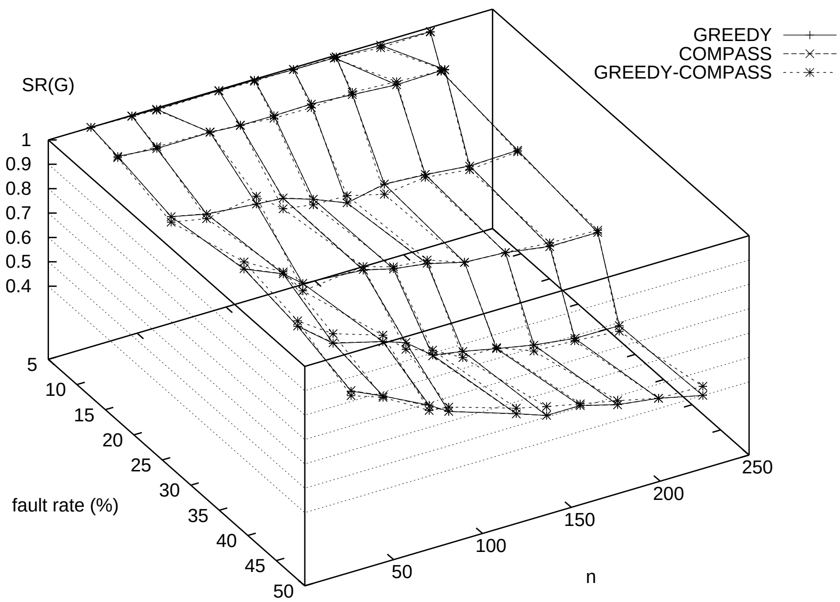


Figure 7.6: Success rates of routing algorithms on faulty meshes.

tasks do not encounter faults and hence perform shortest path routing. When the fault rate is higher, longer routing tasks nearly always fail and hence do not contribute to the average dilation. It is because of this effect that a low average dilation is only meaningful when the success rate is high.

In contrast, Figure 7.8 shows the average dilation for the FACE-ROUTE algorithm which has a success rate of 1. The cost of this guaranteed success is a much higher average dilation, and FACE-ROUTE has average dilations ranging between 4 and 10.

7.4 Unit Disk Graphs

Finally, we study the performance of routing algorithms on randomly generated unit disk graphs with varying numbers of vertices, n, and average degree Δ. These graphs were generated by selecting n uniformly distributed points in the unit square, sorting the $\binom{n}{2}$ interpoint distances and setting the value of a "unit" to achieve the desired average degree, Δ. These plots show results for $20 \leq n \leq 200$ and $3 \leq \Delta \leq 10$.

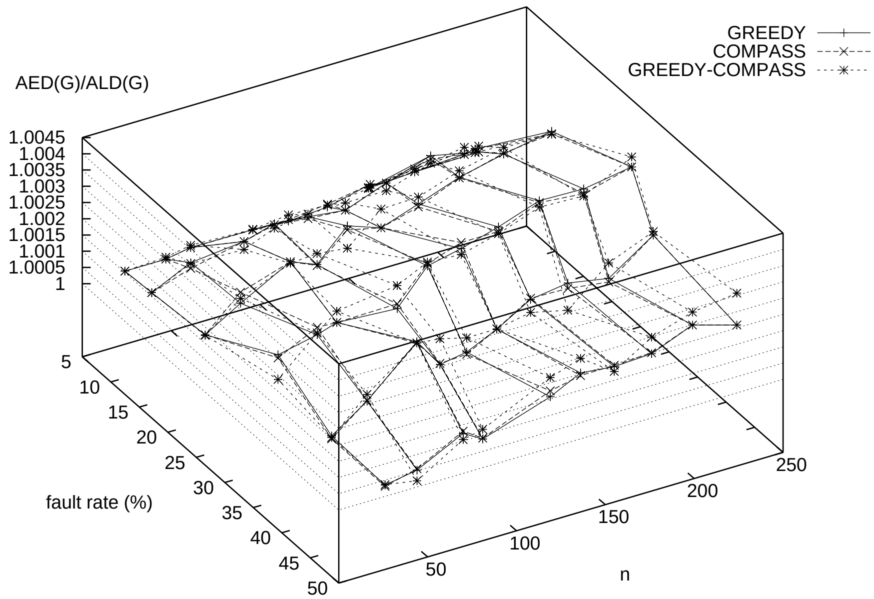


Figure 7.7: Average dilation of routing algorithms on faulty meshes.

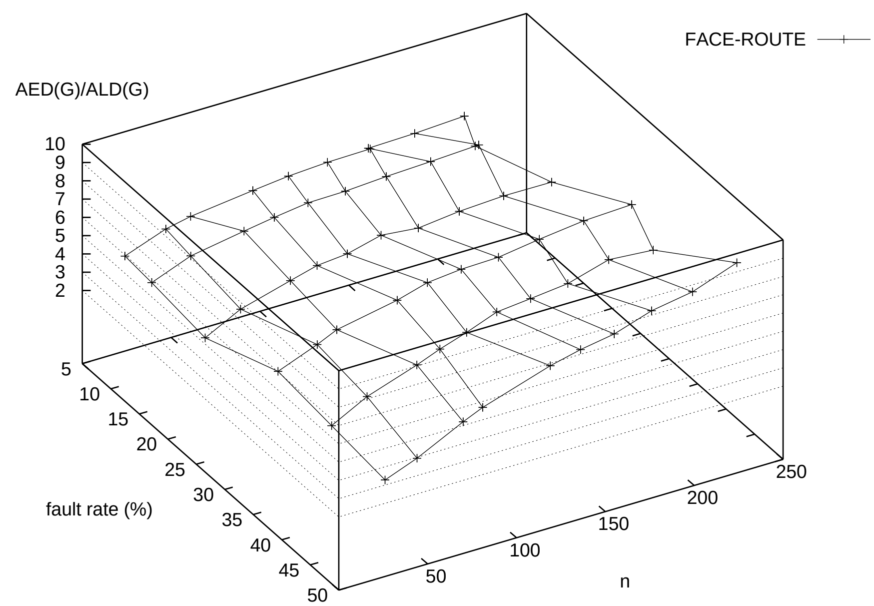


Figure 7.8: Average dilation of FACE-ROUTE on faulty meshes.

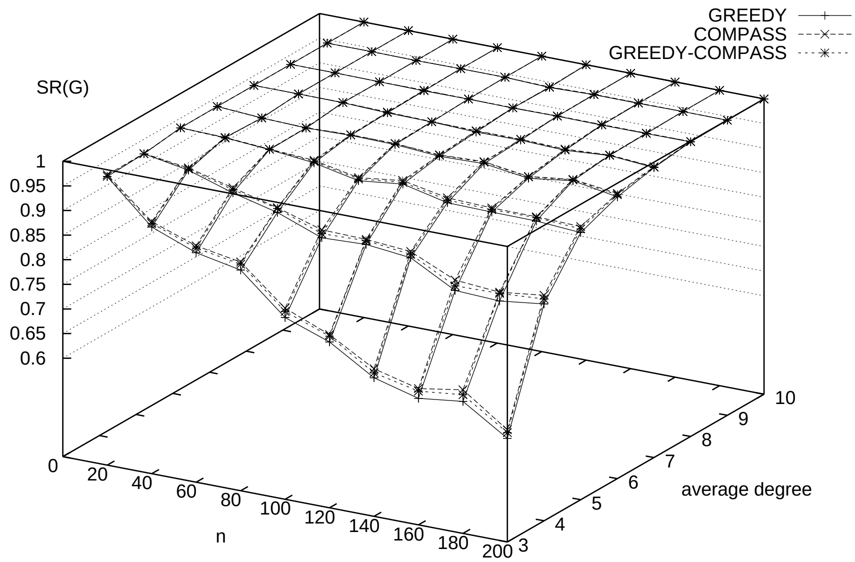


Figure 7.9: Success rates of routing algorithms on unit graphs.

Figure 7.9 shows success rates for GREEDY, COMPASS and GREEDY-COMPASS. Although the success rates for the three algorithms are very similar, there seems to be a consistent ranking with COMPASS achieving the highest success rate, closely followed by GREEDY-COMPASS and GREEDY. Success rates vary between 1 for graphs with high average degree and few nodes and .6 for graphs with low average degree and large numbers of nodes.

At this point we caution the reader that, in order to provide a more understandable visualization of the data, the viewpoint in the upcoming figures is different than that of Figure 7.9

Figures 7.10 and 7.11 show average Euclidean and link dilation, respectively, for the same three routing algorithms. The average dilation for all three algorithms is very close to 1. As was the case for Delaunay triangulations, COMPASS performs better under the Euclidean distance measure while GREEDY performs better under the link distance measure, with GREEDY-COMPASS falling somewhere in between the two.

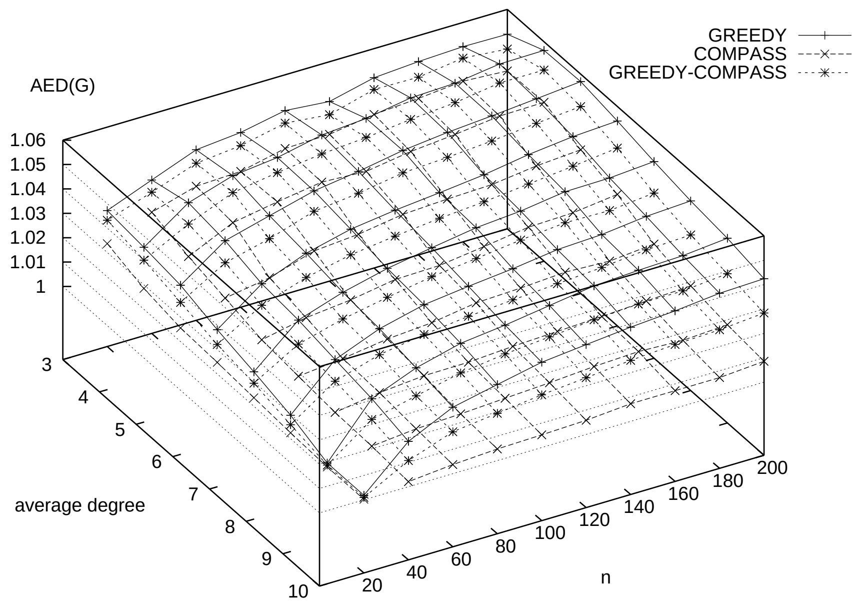


Figure 7.10: Average Euclidean dilation of routing algorithms on unit graphs.

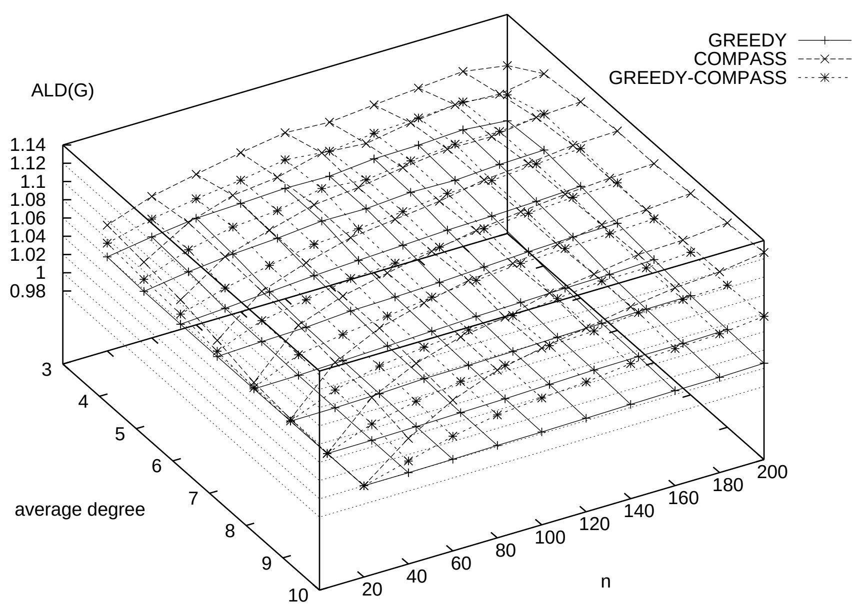


Figure 7.11: Average link dilation of routing algorithms on unit graphs.

Finally, we discuss results for FACE-ROUTE. Note that, unlike GREEDY, COMPASS and GREEDY-COMPASS, FACE-ROUTE requires that the input graph be planar, i.e., that its edges meet only at their endpoints. In general, unit disk graphs do not have this property. However, as discussed in Section 1.2.3, there is a simple local rule that can be applied by a routing algorithm to obtain a planar subgraph of a unit disk graph. The results we describe are therefore for this planar subgraph of the unit disk graph. One final caveat: Although we measure the performance of FACE-ROUTE on the planar subgraph, the values for the average dilation are computed using shortest paths in the original, possibly non-planar, graph. Thus, even if FACE-ROUTE could achieve perfect shortest path routing, the average dilation would be greater than 1.

Figures 7.12 and 7.13 show average dilation results for the FACE-ROUTE algorithm under the Euclidean and link distance measures, respectively. As expected, the algorithm exhibits large average dilation under both measures, with values in the range 4–10 under the Euclidean measure and 4–14 under the link measure. In defence FACE-ROUTE, it does always achieve a success rate of 1.

One final point to note is that the average link dilation of FACE-ROUTE increases as the average degree increases, but the average Euclidean dilation does not increase, or even decreases. This is because the reduction of the unit disk graph tends to preserve short edges over long edges. This property seems to have little effect on the Euclidean lengths of paths, but requires that more edges are crossed, which increases the link lengths of paths.

7.5 Summary and Open Problems

In this chapter we have experimentally studied most of the routing algorithms described in this thesis. Although the best routing algorithm to use in a particular application depends on specifics of the application and implementation environment, there are at least three guiding principles that arise from our study.

1. Heuristics based on direction, like COMPASS, tend to work better when the object is to minimize Euclidean distance travelled, while heuristics based on

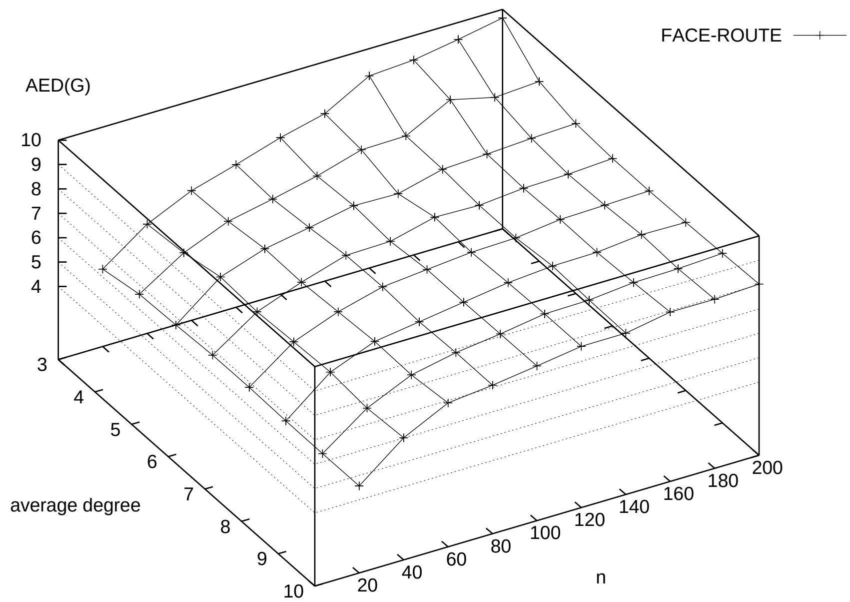


Figure 7.12: Average Euclidean dilation of FACE-ROUTE on unit graphs.

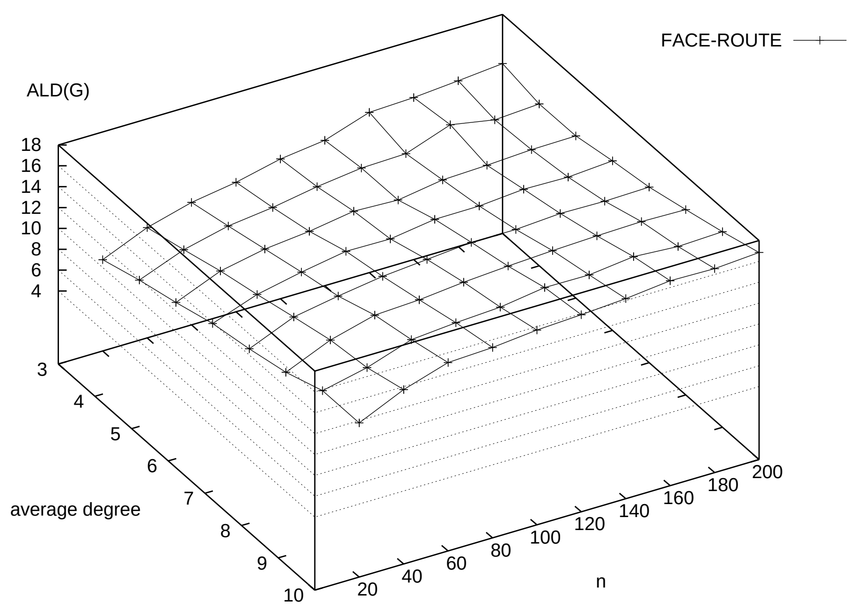


Figure 7.13: Average link dilation of FACE-ROUTE on unit graphs.

distance, like GREEDY tend to work better when the goal is to minimize the number of edges traversed.

2. Elaborate algorithms that offer a guarantee on their worst-case performance, such as PARALLEL-VORONOI, often don't perform as well as simple algorithms in the so-called average case and probably do not work as well in practice.

3. Elaborate algorithms that have a success rate of 1 usually do so at the cost of higher average dilation.

Although points 2 and 3 would appear to imply that much of the work in this thesis is impractical, we believe that this is not the case. The place for elaborate algorithms with guarantees on their competitive ratio or success rates is as a backup for simpler heuristic algorithms. As an example, one could try to use a simple heuristic algorithm like GREEDY until it fails, and then switch to a more sophisticated algorithm like FACE-ROUTE. Another option is to alternate between the two. Several such *hybrid* algorithms are discussed in the paper by Bose *et al.* [12]. Figure 7.14 (taken from [12]) shows results obtained by combining the FACE algorithm of Kranakis *et al.* [52] with GREEDY(labelled GEDIR).

As for open problems, the very good asymptotic behaviour of RANDOM-COMPASS on Graham triangulations remains a mystery.

Open Problem 15. *Provide an explanation for the behaviour of* RANDOM-COMPASS *on Graham triangulations. Why does it perform better than* GREEDY-COMPASS*?*

7.6 Bibliographic Notes

Experimental work on the competitiveness of routing algorithms in Delaunay triangulations and under the Euclidean distance measure appears in the paper by Bose and Morin [10]. Experimental results for routing on unit disk graphs appear in the paper by Bose *et al.* [12]. The latter paper also discusses combinations of GREEDY with algorithms that have a success rate of 1. Another paper by Bose *et al.* [8] contains

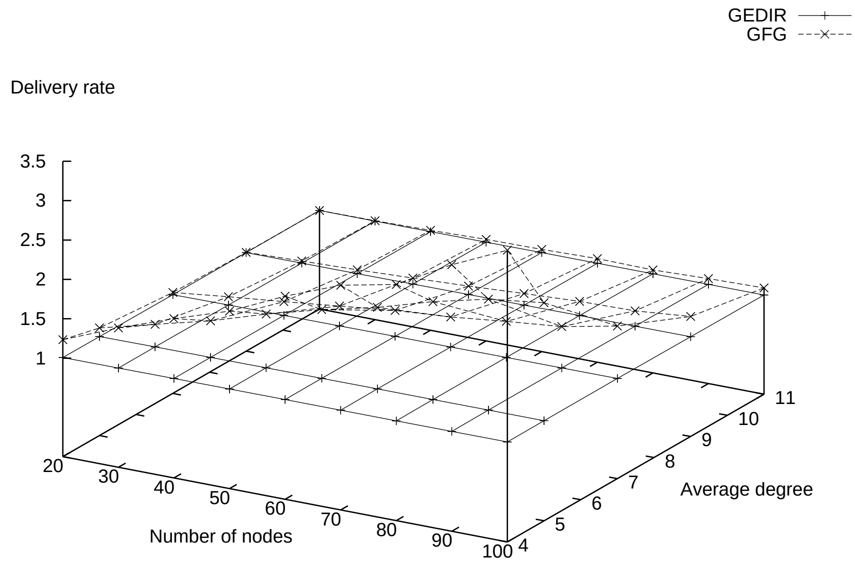


Figure 7.14: Average link dilation of GFG algorithm, proposed by Bose *et al.* [12].

a short discussion of experimental results for GREEDY-COMPASS under the Euclidean distance measure in its conclusions.

A very large source of experimental results for different types of online routing algorithms is the literature on mobile computing. Most of the references given in the bibliographic notes of Chapter 3 contain experimental results for various routing algorithms under various assumptions about the routing algorithm, input graph, networking environment, and level of knowledge the algorithm has about the input graph. A good starting point for this kind of information is the *MobiCom* proceedings from 1995 onwards.

Chapter 8

Summary and Conclusions

In this thesis we have studied online routing problems in planar geometric graphs. Aside from proving some theorems about online routing algorithms, this work has illustrated relationships between online routing problems and other areas of mathematics and computer science, including polytope theory, planar graph theory, and distributed computing.

We conclude with a summary of our contributions and some more open problems that did not fit into any of the previous chapters.

8.1 Summary of Contributions

Here we summarize the main points covered in the individual chapters of this thesis. For a more detailed description of the contributions the reader is referred to Chapter 2

The contributions of the individual chapters of this thesis are as follows.

Chapter 1 In this chapter we introduced the problem of routing on planar geometric graphs and motivated it with three potential applications. In showing how to reduce routing in mobile *ad hoc* networks to routing in planar geometric graphs we gave an extremely simple distributed algorithm for extracting a connected planar subgraph of a unit disk graph. From a distributed computing point of view this result is surprising since it shows that the message complexity of extracting a planar subgraph from a unit disk graph is $O(|V| + |E|)$ while the message complexity of leader election is $\Omega(|V| \log |V| + |E|)$ [63, 72].

Chapter 3 In this chapter, we considered the special case of routing in triangulations and convex subdivisions using very simple routing algorithms. We described new routing algorithms and found new properties of two previously known routing algorithms. Our characterization of trapping cycles for the COMPASS algorithm also allowed us to make use of some powerful techniques from polytope theory in Chapter 6. In this chapter we also showed a clear separation between deterministic memoryless algorithms and algorithms that make use of memory or randomization.

Chapter 4 In this chapter, we continued our study of routing in triangulations with an emphasis on the length of the paths found by our routing algorithms. In the Euclidean distance metric, our results included competitive routing algorithms for three fundamental geometric structures, namely Delaunay triangulations, greedy triangulations, and minimum-weight triangulations. This latter result is especially surprising, since very little is known about minimum-weight triangulations. We also showed that no competitive algorithm exists for all triangulations. In the link length metric, we showed that no competitive routing algorithms are possible for Delaunay, greedy, or minimum-weight triangulations.

Chapter 5 In this chapter we studied the problems of routing, broadcasting and geocasting in planar geometric graphs. Our algorithms combined geometry with techniques from distributed algorithms to obtain efficient solutions for these problems. In deriving our routing algorithms we made use of and improved the previous best known results on traversing planar subdivisions with constant additional storage. These subdivision traversal techniques have applications in computational geometry, geographic information systems and computer graphics.

Chapter 6 In this chapter we studied the problem of embedding planar graphs so that they admit geometric routing algorithms. In doing so, we found that by using a real model of computation it was not particularly difficult, or interesting, to embed any planar graph so that it could support geometric shortest path routing.

We also considered the simple routing algorithms GREEDY and COMPASS and studied embeddings of graphs that support them. We showed that there exist planar graphs having no embedding that supports GREEDY or COMPASS. We also showed that any near-triangulation has an embedding which supports COMPASS and that any subdivided circuit graph has an embedding that supports LEFT-COMPASS.

The results on embeddings for COMPASS and LEFT-COMPASS make use of some elegant results from planar graph theory and the theory of 3-polytopes. As a byproduct of our results, we obtained a proof that the set of graphs with convex embeddings (subdivided circuit graphs) is exactly the set of subdivisions of skeletons of lower convex hulls of 3-polytopes, thus unifying Tutte's theorem and Barnette's version of Steinitz' theorem.

Overall, the results obtained in our study of online routing have proven to be interesting and non-trivial. In working on these problems we have also generated a list of 18 open problems that continue to deserve attention.

8.2 Open Problems

In this section we describe some open problems that did not fit into any of the previous chapters. Any of these problems is rich enough to provide an entire chapter's worth of material on its own.

In Chapter 6, we considered methods of embedding graphs so that simple routing algorithms were able to work on them. However, the algorithms we described were centralized algorithms that made use of knowledge about the entire graph to be embedded. This observation suggests the following open problem.

Open Problem 16. *Give distributed algorithms for planarity testing, extraction of maximal planar subgraphs, and planar embedding.*

Rajsbaum and Urrutia [72] study the problem of finding the convex hull of a planar geometric graph G by a distributed algorithm whose underlying communication network is G. Since convex hull construction is only a small part of the field known as computational geometry, the following open problem suggests itself.

Open Problem 17. *Study distributed algorithms for computational geometry problems.*

Finally, in this thesis we have restricted our attention to planar geometric graphs with non-crossing edges. While there are many applications that are included in this model, there are some applications in which it makes sense to relax the planarity restriction.

Open Problem 18. *Study any of the problems in this thesis for the case of d-dimensional geometric graphs ($d \geq 3$) and/or 2-dimensional geometric graphs with possibly crossing edges.*

8.3 Final Note

We hope that the reader has enjoyed reading these results as much as the author has enjoyed relating them. If not, at least the knowledge might be useful the next time the reader is lost in a strange city.

Bibliography

[1] O. Aichholzer, F. Aurenhammer, S.-W. Cheng, N. Katoh, G. Rote, M. Taschwer, and Y.-F. Xu. Triangulations intersect nicely. *Discrete and Computational Geometry*, 16(4):339–359, 1996.

[2] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete and Computational Geometry*, 8(295–313), 1992.

[3] R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.

[4] D. W. Barnette. Projections of 3-polytopes. *Israel Journal of Mathematics*, 8:304–308, 1970.

[5] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward. A distance routing effect algorithm for mobility (DREAM). In *ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom'98)*, pages 76–84, 1998.

[6] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Elsevier/North-Holland, 1976.

[7] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[8] P. Bose, A. Brodnik, S. Carlsson, E. D. Demaine, R. Fleischer, L. Jacobsen, A. López-Ortiz, P. Morin, and J. I. Munro. Online routing in convex subdivisions. In *Proceedings of the 11th International Symposium on Algorithms and Computation (ISAAC 2000)*, 2000.

[9] P. Bose and L. Devroye. Intersections with random geometric objects. *Computational Geometry Theory and Applications*, 10(3):139–154, 1998.

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

[11] P. Bose and P. Morin. An improved algorithm for subdivision traversal without extra storage. In *Proceedings of the 11th International Symposium on Algorithms and Computation (ISAAC 2000)*, 2000.

[12] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia. Routing with guaranteed delivery in *ad hoc* wireless networks. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM'99)*, pages 48–55, 1999.

[13] C. Bröcker and S. Schuirer. Searching rectilinear streets completely. In *Proceedings of the 7th Workshop on Algorithms and Data Structures (WADS'99)*, 1999. To appear.

[14] M. Chrobak, M. T. Goodrich, and R. Tamassia. Convex drawings of graphs in two and three dimensions. In *Proceedings of the Twelfth Annual ACM Symposium On Computational Geometry (SoCG '96)*, pages 319–328, 1996.

[15] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, 1990.

[16] G. Das and D. Joseph. Which triangulations approximate the complete graph? In *Proceedings of the International Symposium on Optimal Algorithms*, pages 168–192, 1989.

[17] A. Datta, C. Hipke, and S. Schuirer. Competitive searching in polygons—beyond generalized streets. In *Proceedings of the 6th International Symposium on Algorithms and Computation (ISAAC'95)*, pages 32–41, 1995.

[18] A. Datta and C. Icking. Competitive searching in a generalized street. *Computational Geometry Theory and Applications*, 13(109–120), 1999.

[19] M. de Berg, M. van Kreveld, R. van Oostrum, and M. Overmars. Simple traversal of a subdivision without extra storage. *International Journal of Geographic Information Systems*, 11:359–373, 1997.

[20] H. de Fraysseix, J. Pach, and R. Pollack. Small sets supporting Fáry embeddings of planar graphs. In *Proceedings of the 20th ACM Symposium on the Theory of Computing (STOC'88)*, pages 426–433, 1988.

[21] D. Dobkin, S. J. Friedman, and K. J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete and Computational Geometry*, 5:399–407, 1990.

[22] H. Edelsbrunner. An acyclicity theorem for cell complexes in d dimension. *Combinatorica*, 10(3):251–260, 1988.

[23] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15:317–340, 1986.

[24] I. Fáry. On straight line representing of planar graphs. *Acta Universitatis Szegediensis. Acta Scientiarum Mathematicarum*, 11:229–233, 1948.

[25] G. N. Frederickson and R. Janardan. Efficient message routing in planar networks. *SIAM Journal on Computing*, 18(4):843–857, 1989.

[26] K. R. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18:259–278, 1969.

[27] C. Gavoille and N. Hanusse. Compact routing tables for graphs of bounded genus. In *26th International Colloquium on Automata, Languages and Programming (ICALP'99)*, pages 351–360, 1999.

[28] S. K. Ghosh and S. Saluja. Optimal on-line algorithms for walking with minimum number of turns in unknown streets. *Computational Geometry Theory and Applications*, 8(5):241–266, 1997.

[29] C. Gold and S. Cormack. Spatially ordered networks and topographic reconstructions. In *Proceedings of the 2nd International Symposium on Spatial Data Handling*, pages 74–85, 1986.

[30] C. M. Gold, T. D. Charters, and J. Ramsden. Automated contour mapping using triangular element data structures and an interpolant over each irregular triangular domain. *Computer Graphics*, 11(2):170–175, 1977.

[31] C. M. Gold and U. Maydell. Triangulation and spatial ordering in computer cartography. In *Proceedings of the Canadian Cartographic Association Annual Meeting*, pages 69–81, 1978.

[32] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, 2nd edition, 1994.

[33] C. Hipke. Online-algorithmen zur kompetitiven Suche in einfachen Polygonen. Master's thesis, Universität Freiburg, 1994.

[34] C. Hipke, C. Icking, R. Klein, and E. Langetepe. How to find a point on a line within a fixed distance. *Discrete Applied Mathematics*, 93:67–73, 1999.

[35] D. S. Hirschberg and J. B. Sinclair. Decentralized extrema-finding in circular configurations of processes. *Communications of the ACM*, 23(11):627–628, 1980.

[36] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. An efficient competitive strategy for learning a polygon. In *Abstracts of 12th European Workshop on Computational Geometry (EuroCG'96)*, pages 107–108, 1996.

[37] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. A competitive strategy for learning a polygon. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA'97)*, pages 166–174, 1997.

[38] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem: A new strategy and a new analysis technique. In *Proceedings of the 3rd Workshop on Algorithmic Foundations of Robotics*, pages 211–222, 1998.

[39] J. Hopcroft and R. Tarjan. ACM algorithm 447: Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[40] T. C. Hou and V. O. K. Li. Transmission range control in multihop packet radio networks. *IEEE Transactions on Communications*, 34(1):38–44, 1986.

[41] C. Icking and R. Klein. Competitive strategies for autonomous systems. Technical Report 175, Department of Computer Science, FernUniversität Hagen, 1995.

[42] C. Icking and R. Klein. Searching for the kernel of a polygon: A competitive strategy. In *Proceedings of the 11th ACM Symposium on Computational Geometry (SoCG'95)*, pages 258–266, 1995.

[43] C. Icking, R. Klein, and E. Langetepe. An optimal competitive strategy for walking in streets. In *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science (STACS'99)*, pages 110–120, 1999.

[44] M.-Y. Kao, Y. Ma, M. Sipser, and Y. Yin. Optimal constructions of hybrid algorithms. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms (SODA'94)*, pages 373–381, 1994.

[45] M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown evironment: An optimal randomized algorithm for the cow-path problem. In *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 441–447, 1993.

[46] E. D. Kaplan, editor. *Understanding GPS: Principles and Applications.* Artech House, 1996.

[47] B. Karp and H. T. Kung. GPSRP: Greedy perimeter stateless routing for wireless networks. In *ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, 2000.

[48] R. Klein. Walking an unknown street with bounded detour. *Computational Geometry Theory and Applications*, 1:325–351, 1992.

[49] Rolf Klein. *Algorithmische Geometrie*. Addison-Wesley, 1997.

[50] Y.-B. Ko and N. H. Vaidya. Geocasting in mobile ad hoc networks: Location-based multicast algorithms. Technical Report TR-98-018, Texas A&M University, September 1998.

[51] Y.-B. Ko and N. H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom'98)*, pages 66–75, 1998.

[52] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG'99)*, 1999. available online at `http://www.cs.ubc.ca/conferences/CCCG/elec_proc/c46.ps.gz`.

[53] F. T. Leighton. *Introduction to Parallel Algorithm and Architectures: Arrays, Trees and Hypercubes*. Morgan Kaufman, 1992.

[54] X. Lin and I. Stojmenović. GEDIR: Loop-free location based routing in wireless networks. In *Proceedings of the 11th International Conference on Parallel and Distributed Computing and Systems (PDCS'99)*, 1999. to appear.

[55] A. López-Ortiz and S. Schuirer. Going home through an unknown street. In *Proceedings of the 3th Workshop on Algorithms and Data Structures (WADS'95)*, pages 135–146, 1995.

[56] A. López-Ortiz and S. Schuirer. Simple, efficient and robust strategies to traverse streets. In *Proceedings of the 7th Canadian Conference on Computational Geometry (CCCG'95)*, pages 135–146, 1995. available online at `http://cgm.cs.mcgill.ca/cccg98/proceedings/cccg98-lopez-exact.ps.gz`.

[57] A. López-Ortiz and S. Schuirer. Generalized streets revisited. In *Proceedings of the 4th European Symposium on Algorithms (ESA'96)*, pages 546–558, 1996.

[58] A. López-Ortiz and S. Schuirer. Walking streets faster. In *Proceedings of the 5th Scandinavian Workshop on Algorithm Theory (ESA'96)*, pages 345–356, 1996.

[59] A. López-Ortiz and S. Schuirer. Position-independent near optimal searching and on-line recognition in star polygons. In *Proceedings of the 5th Workshop on Algorithms and Data Structures (WADS'97)*, pages 284–296, 1997.

[60] A. López-Ortiz and S. Schuirer. Position-independent near optimal searching and on-line recognition in star polygons. In *Proceedings of the 13th ACM Symposium on Computational Geometry (SoCG'97)*, pages 445–447, 1997.

[61] A. López-Ortiz and S. Schuirer. The exact cost of exploring streets with a CAB. In *Proceedings of the 10th Canadian Conference on Computational Geometry (CCCG'98)*, 1998.

[62] A. López-Ortiz and S. Schuirer. The ultimate strategy to search on m rays? In *Proceedings of the 4th International Conference on Computing and Combinatorics (COCOON'98)*, pages 75–84, 1998.

[63] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1997.

[64] A. Maheshwari, J.-R. Sack, and H. Djidjev. Link distance problems. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 519–558. Elsevier Science, 2000.

[65] D. W. Matula and R. R. Sokal. Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical Analysis*, 12:205–222, July 1980.

[66] R. Nelson and L. Kleinrock. The spatial capacity of a slotted ALOHA multihop packet radio network with capture. *IEEE Transactions on Communications*, 32(6):684–694, 1984.

[67] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tesselations: Concepts and Applications of Voronoi Diagrams.* John Wiley and Sons, 1992.

[68] S. Onn and B. Sturmfels. A quantitative Steinitz' theorem. *Beiträge zur Algebra und Geometrie/Contributions to Algebra and Geometry*, 35:125–129, 1994.

[69] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.

[70] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry*. Springer-Verlag, New York, 1985.

[71] P. Raghavan and R. Motwani. *Randomized Algorithms*. Cambridge University Press, 1995.

[72] S. Rajsbaum and J. Urrutia. Some problems in distributed computational geometry. In *Proceedings of the 6th International Colloquium on Structural Information & Communication Complexity (SIROCCO-99)*, pages 233–248, 1999.

[73] J. Richter-Gebert. *Realization Spaces of Polytopes*, volume 1643 of *Lecture Notes in Mathematics*. Springer–Verlag, 1996.

[74] W. Schnyder. Embedding planar graphs on the grid. In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, pages 138–148, 1990.

[75] S. Schuirer. Online searching in geometric trees. In *Proceedings of the 9th Canadian Conference on Computational Geometry (CCCG'97)*, pages 135–140, 1997.

[76] S. Schuirer and I. Semrau. An optimal strategy for searching in unknown streets. In *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science (STACS'99)*, pages 121–131, 1999.

[77] H. Singh. Compass routing in geometric graphs. Master's thesis, University of Ottawa, 1999.

[78] H. Takagi and L. Kleinrock. Optimal transmission ranges for randomly distributed packet radio terminals. *IEEE Transactions on Communications*, 32(3):246–257, 1984.

[79] W. T. Tutte. Convex representations of graphs. *Proceedings of the London Mathematical Society*, 3(10):304–320, 1960.

[80] J. van Leeuwen and R. Tan. Computer networks with compact routing tables. In G. Rozenberg and A. Salomaa, editors, *The Book of L*, pages 259–273. Springer–Verlag, New York, 1986.

[81] Günter M. Ziegler. *Lectures on Polytopes.* Number 154 in Graduate Texts in Mathematics. Springer-Verlag, New York, 1994.