LOW MEMORY LOCAL GEOMETRIC ROUTING WITH GUARANTEED DELIVERY IN MONOTONE SUBDIVISIONS

anthony mark d'angelo

A thesis submitted to the Faculty of Graduate and Post Doctoral Affairs
in partial fulfillment of the requirements for the degree of

Master of Computer Science

Carleton University
Ottawa, Ontario, Canada

ABSTRACT

Recently, deterministic geometric local routing algorithms were presented with guaranteed delivery for convex subdivisions using only one bit and for monotone subdivisions using only predecessor awareness (in "Local Routing in Convex Subdivisions", SOFSEM 2015: 140-151). We present a simple deterministic geometric local routing algorithm that guarantees delivery in monotone planar subdivisions (which include convex subdivisions) without a general position assumption. Our algorithm uses $\lceil \log_2 (d+1) \rceil$ bits of extra memory per message, where the d possible edge directions are fixed to $d/2$ different slopes.

ACKNOWLEDGMENTS

Thanks to my co-supervisors Prosenjit Bose and Stephane Durocher as well as to Jean-Lou De Carufel for helping me get this all together, for their patience, guidance, support, insight, helpful discussions, and many reviews of and improvements to the many drafts. Thanks to the examination committee for their helpful comments.

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

Thanks to my family and friends for their love and support.

Thanks to classical music for helping me concentrate and stay awake during the long hours and late nights writing my thesis.

Thanks to Sander Verdonschot for the LaTeX thesis template.

Thanks to you for reading this. =)

CONTENTS

1 INTRODUCTION

In this thesis we concentrate on a subclass of point-to-point routing algorithms for wireless networks. The chapter begins by motivating the study of low memory routing algorithms in Section 1.1. Following the motivation, in Section 1.2 we provide a brief review of local geometric routing results relevant to our algorithm presented in Chapter 2. In Section 1.3 we provide the problem statement, followed by the description of the local routing model used in this work in Section 1.4. Finally, the contribution of this thesis is summarized in Section 1.5.

In Chapter 2 we present our low memory algorithm for routing on monotone subdivisions as well as prove that the algorithm guarantees delivery. Chapter 3 summarizes the results, provides a discussion, and mentions directions for future work.

1.1 MOTIVATION

With the prevalence of mobile devices, significant research continues to improve the ability to route messages efficiently between nodes in wireless networks while minimizing consumption of resources such as power and memory. Various models represent wireless networks as a graph, representing nodes as points in the plane and direct wireless communication links as edges connecting the corresponding pairs of nodes. The inherent geometry of these graphs enables efficient routing algorithms to be developed (e.g., [2, 4, 5, 6, 8, 10]) using fewer resources (memory, computational, etc.) than is possible without geometry in general graphs.

1.2 RELATED WORK

Since various models represent wireless networks as graphs, we use the following terms interchangeably: *network* and *graph*; *node* and *vertex*; *communication link* and *edge*; *packet* and *message*.

Stated briefly, the goal of a point-to-point routing algorithm is to deliver a message from some source node to some destination node [11]. In order to successfully find a path in point-to-point routing, some algorithms store state information at visited nodes to be used and updated if the message ever returns to that node. A *memoryless* algorithm is one that does not persistently store any state information at visited nodes [4]; once the message leaves the node it is as if the node had never seen the message.

An *online* algorithm is one that does not have access to the whole network map [11]. Online algorithms that want to reason about the network map need to store in the message header a portion of the network visited by the routing algorithm. How much of the map can be stored depends on the number of extra bits allocated to the message header.

For a routing algorithm to guarantee delivery, the message header must contain the identity of the destination node. In some models, the algorithm also has access to some limited number of extra storage bits in the message header that can be used to help with routing. These extra bits can be modified by the algorithm at will, though sometimes it may not be necessary. For example, in some algorithms that remember the origin node in the message header (such as the one by Bonichon et al. [1] or the ones by Bose et al. [3]), the routing algorithm never needs to modify any bits stored in the message header. When there are bits in that extra memory that represent the state of the algorithm, those bits are referred to as *state bits*. A *stateless* routing algorithm is one that passes no state bits along in the message header for the purpose of routing [4]. We will only discuss algorithms in which the extra storage bits are either state bits or bits used to represent nodes (e.g., storing the coordinates of the predecessor or origin of the message).

A k-*local* routing algorithm is one in which a node knows which other nodes can be reached by forwarding the message k times [3]. A model is *predecessor-aware* when the node holding the message knows from which of its neighbours the message came.[1] For general abstract and topological graphs, Bose et al. [3] showed that delivery can be guaranteed with a deterministic memoryless stateless online algorithm with a sufficiently large k parameterized in terms of n, the number of nodes in the graph. If we remember where the message originated and have predecessor-awareness, delivery can be guaranteed with $k \geqslant n/4$. If we cannot remember where the message originated but have predecessor-awareness, delivery can be guaranteed with $k \geqslant n/3$. Whether or not we remember where the message originated, if we do not have predecessor-awareness, delivery can be guaranteed with $k \geqslant n/2$. Bose et al. also show that using their algorithms the path taken is $\Theta(1)$ times the length of the shortest path.

Using the geometry of graphs and the positions of nodes to find a path to the destination is termed *geometric routing*. A *local geometric* routing algorithm is one that makes forwarding decisions based on knowledge of the position of the node currently handling the message as well as the positions of its direct neighbours, i.e. the neighbours it is directly connected to by a link [4]. Nodes in this model know

1 In some models knowledge of the predecessor comes at no cost, while in other models the cost of representing predecessor information is counted as extra bits in the message header.

that the neighbours exist as well as their positions, and the message header contains the coordinates of the destination.

The goal of the algorithms discussed in this work is to use as little supplementary information as possible to guarantee delivery of the message. To that end we will consider *low memory* routing algorithms to be deterministic memoryless online $O(1)$-local geometric algorithms that use $O(\log n)$ extra bits in the message header (where n is the number of nodes in the graph).[2] Once again, note that in these algorithms the coordinates of the destination are included and accessible in the message header, but we do not consider these coordinates as using extra memory in the message header because this information is required.

This section begins by reviewing the ideas of greedy routing and face routing. Afterwards we provide the key results for low memory routing algorithms.

1.2.1 *The Greedy Idea*

One popular approach to geometric routing is *greedy routing* where the current node forwards the message to a neighbour that is "closer" to the destination than it is. The measure used to determine closeness varies in different algorithms. Two commonly used measures are: Euclidean distance to the destination; deviation in angle to the destination (i.e., the angle formed between the current node, the destination, and the candidate neighbour).

In general, greedy algorithms require extra bits to guarantee delivery. There are two common ways in which following the *greedy* rules alone can fail to deliver the message to the destination. The first is if, by following the same greedy rule at each node, a cycle or loop is formed. Without a different input, when a message revisits a node it will make the same decision and loop forever between a finite subset of the graph's vertices. The second common type of failure occurs at something called a *void* or *local minimum*. A void is encountered when the message gets to the vertex in question and there are no neighbours "closer" to the destination than the current vertex. At this point greedy routing cannot proceed on its own.

An early geometric routing algorithm was a greedy algorithm by Takagi and Kleinrock [12] attempting to maximize "progress". Their algorithm, called *Most Forward within radius* R (MFR), basically works as follows. First one considers the line segment connecting the current node and the destination, then one considers the perpendicular projection of neighbours onto that line segment that lie within radius R from the current node, and the message is sent to the neighbour that makes the most progress, i.e. whose perpendicular projection onto the line segment is closest to the destination.

2 The base for all logarithms in this thesis is 2.

Two other popular optimization approaches for greedy routing are decreasing Euclidean distance to the destination (based on Finn's *Cartesian Routing* [8]) and decreasing the difference in direction toward the destination. This last approach is based on the *Compass routing* algorithm proposed by Kranakis et al. [10]. The basic idea behind Compass routing is to move to a neighbour that is as close in direction to the destination as possible, minimizing the deviation in angle between the current node, the destination, and the candidate neighbour.

1.2.2 *The Face Routing Concept*

When we talk about planar graphs in the rest of this work, we mean a graph whose vertices are embedded in the 2-dimensional Euclidean plane and whose edges, represented by line segments, intersect only at common endpoints [11]. The edges of a planar graph partition the plane into faces (including an unbounded face). We call this a *planar subdivision* (which we will use interchangeably with *planar graph*).

Face routing is a routing strategy that only functions on planar graphs. The idea behind face routing is to visit the edges of a face in either a clockwise or counter-clockwise order, as if one were to walk around a room while keeping one's hand on the wall to explore the perimeter of the room. The face routing strategy works by first face routing a given face until a face is found that is closer to the destination, at which point face routing is begun on the closer face [6, 9, 10]. Generally, the process is repeated until the destination is reached.

To be able to walk around the face, a node needs to know which of its neighbours sent it the message, i.e. it needs to have predecessor-awareness. As far as the author knows, generally $\Omega(\log n)$ bits are needed to represent this information. When there are n nodes in the network, if each node gets a unique identifier (ID), the predecessor's ID can be represented by $\Theta(\log n)$ bits. When the points are placed on a grid whose size is polynomial in n, coordinates can be represented by $\Theta(\log n)$ bits.[3]

In the general application of face routing, more information is required than just the identity of the predecessor because we need to remember the point where we entered the face and started face routing (i.e., the entrance point) in order to avoid looping [6, 10]. Remembering the coordinates or IDs needed to identify the entrance point also takes $\Theta(\log n)$ bits because we need to remember $\Theta(1)$ coordinates or identifiers, each of which is represented by $\Theta(\log n)$ bits.

Therefore, since general face routing techniques need to remember a constant number of coordinates or IDs, it takes $\Theta(\log n)$ extra

3 We assume all geometric graphs are embedded on a grid whose size is polynomial in n.

bits in the message header to face route in addition to the cost of predecessor-awareness. This means that a general face routing algorithm needs $\Omega(\log n)$ extra bits in the message header.

1.2.3 *Low Memory Results Overview*

A *convex subdivision* is a planar subdivision in which the outer face is the complement of a convex polygon and each internal face is a convex polygon [2]. A convex subdivision is a *regular subdivision* if and only if the vertices and edges are orthogonal projections of the vertices and edges of the lower convex hull of some 3-dimensional polytope onto the Euclidean plane [11]. A *triangulation* is a planar subdivision in which the outer face is the complement of a convex polygon and each interior face is a triangle [2]. A triangulation is a special kind of convex subdivision. A *Delaunay triangulation* is a triangulation where for each triangle, the circumscribing circle does not contain any vertices in its interior [11].

Kranakis et al. first showed that Compass routing guarantees delivery in Delaunay triangulations [10]. Bose and Morin [5] then showed that Compass routing actually guarantees delivery in the more general class of *regular triangulations*. In that same paper, Bose and Morin showed that Cartesian routing also guarantees delivery in Delaunay triangulations.

A deterministic stateless memoryless online local routing strategy that is not predecessor-aware is said to be *oblivious*. It was shown by Bose et al. [2] that oblivious routing algorithms can guarantee delivery on all triangulations. They were able to achieve successful oblivious routing on general triangulations with their *Greedy-Compass* algorithm, a combination of Cartesian routing and Compass routing. Basically, the line segment connecting the current node and the destination is used to find the first neighbours of the current node in the clockwise and counter-clockwise directions. The packet is then forwarded to whichever of the two is closest to the destination. Figure 1.1 shows an example of how the clockwise and counter-clockwise neighbours are chosen in Greedy-Compass and helps illustrate how Greedy-Compass differs from Compass routing. The Greedy-Compass algorithm would forward the packet to the clockwise neighbour w of the current node u (instead of the counter-clockwise neighbour v) because w is the closer of the two options to the destination with respect to Euclidean distance (which is easy to see because v lies outside of the circle centred at the destination with radius equal to the Euclidean distance to w). In Compass routing, the packet would be forwarded to v because δ is smaller than ϕ.

It was shown by Bose and Morin [5] that no oblivious routing algorithm can work for all planar graphs; specifically, Bose et al. [2] showed that every deterministic oblivious routing algorithm is de-

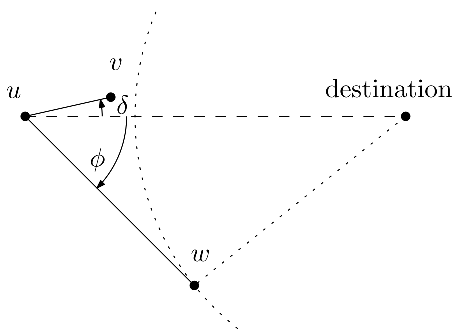

Figure 1.1: In the Greedy-Compass algorithm, the clockwise neighbour w and the counter-clockwise neighbour v of the current node u are chosen with respect to the line through u and the destination. Also shown is part of the circle (dotted) centred at the destination whose radius (also dotted) is the Euclidean distance to w. When the packet is at u, the Greedy-Compass algorithm would forward the packet to w because it has a smaller Euclidean distance to the destination than v does (since v is outside the circle), while the Compass routing algorithm would forward the packet to v because it minimizes the deviation in angle to the destination (because $\delta < \phi$).

feated by some convex subdivision. Consequently, to guarantee delivery on classes of graphs beyond triangulations requires additional information provided, for example, by predecessor-awareness or state bits modified dynamically during routing.

Using a single state bit (and without predecessor-awareness), Bose et al. [4] presented a deterministic local geometric routing algorithm called *OneBit* that guarantees delivery on convex subdivisions (assuming general position of vertex coordinates). The key observation is that if we draw a vertical line through the destination, due to the geometry of convex faces, we can route along the boundary of an ever-decreasing closed region that contains the destination. The idea is to use something similar to Compass routing to find the next node along the path, using a bit to determine whether the clockwise or counter-clockwise neighbour should be chosen. Every time the bit is flipped, the bounded region containing the destination shrinks.

That result, combined with the aforementioned result on the limit of oblivious routing algorithms, implies that one bit is both necessary and sufficient to guarantee delivery in convex subdivisions. Though no lower-bound has been proven for graphs more general than convex subdivisions, all existing algorithms that guarantee delivery on these classes of graphs require at least predecessor-awareness.

Bose et al. [4] presented a stateless predecessor-aware local routing algorithm called *PredAware* that guarantees delivery on monotone subdivisions. Their algorithm requires a general position assumption to define a spanning tree locally, which can then be traversed using a local depth-first search. Assuming that predecessor-awareness comes at the cost of $\Theta(\log n)$ extra bits in the message header, this algorithm requires $\Theta(\log n)$ bits of extra memory to succeed (where n is the number of vertices in the graph).

Kranakis et al. [10] came up with a predecessor-aware algorithm that guarantees delivery on arbitrary connected geometric plane graphs, but it needs to remember the position of the source node and an arbitrary point (not necessarily a node) in the graph. Alternatively, the position of this arbitrary point can be recovered by remembering the positions of four vertices. Essentially, the line segment connecting the source and destination is drawn and the algorithm face routes around the intersected faces, changing onto adjacent intersected faces as intersected edges are encountered. Rather than changing faces as soon as an intersected edge is encountered, the point of intersection is remembered, compared, and updated to be the intersected point on the face that is farthest from the source. This requires the algorithm to visit all of the edges of the face once to find this farthest point, and then a second time as it re-routes back to this farthest point to hop onto the intersected face closer to the destination. Bose et al. [6] improved upon this algorithm with FACE-2. Rather than remembering the source node, they store the coordinates of *some* point g that is used to draw the line segment between g and the destination. Initially g stores the location of the source. The algorithm then proceeds in the same way as before, except as soon as an intersection point is found on the current face that is different than the stored point g (that lies between g and the destination), g is updated to be this new intersection point (and the line segment used in determining intersection points is shortened), and face routing immediately starts on the new intersected adjacent face (rather than looking for a closer intersection point on the old face). These algorithms use predecessor-awareness in addition to the $\Theta(\log n)$ extra bits used to remember arbitrary points, therefore with $\Theta(\log n)$ extra bits in the message header face routing guarantees delivery on planar graphs.

More generally, *universal exploration sequences*[4] can be used to traverse any graph (without geometric information) and, therefore, to guarantee delivery using $\Theta(\log n)$ extra bits in the message header [7]. Universal exploration sequences require $\Theta(\log n)$ state bits to keep track of their position in the polynomial-sized sequence, as well as predecessor-awareness.

In summary: with *no* extra memory and without predecessor information we can guarantee delivery in triangulations; with *one state bit* of extra memory and without predecessor information we can guarantee delivery in convex subdivisions; with no extra memory but with *predecessor-awareness* we can guarantee delivery in mono-

4 Briefly stated, a universal exploration sequence is a set of forwarding instructions that will completely traverse any edge-labelled graph of a given size starting from any vertex in that graph with the ability to back-track along the path taken so far.

Table 1.1: Summary of previous low memory local geometric routing results.

Subdivision Type	Extra Bits of Memory	Predecessor Awareness	Paper
Triangulations	0	No	[2]
Convex [a]	1	No	[4]
Monotone [a]	0	Yes	[4]
General Planar	$\Theta(\log n)$	Yes	[10]
All Graphs [b]	$\Theta(\log n)$	Yes	[7]

a Assuming general positioning
b This result is the only one in the table that is not a geometric result.

tone subdivisions; with $\Theta(\log n)$ *extra bits and predecessor-awareness* we can guarantee delivery in general plane graphs; and with $\Theta(\log n)$ *state bits and predecessor-awareness* we can guarantee delivery in all graphs using a universal exploration sequence (this is the only result in this summary that is not a geometric routing algorithm). This summary is expressed in Table 1.1. The first column indicates the type of subdivision for which a result is being mentioned; the second column indicates how many extra bits of memory are used in the packet header by the best known result that guarantees delivery, excluding the cost of predecessor-awareness; the third column indicates whether the best known result that guarantees delivery requires predecessor-awareness; the last column contains a reference to the paper in which the best result for the subdivision type is presented.

1.3 PROBLEM STATEMENT

Using 1 state bit Bose et al. [4] showed how to route on convex subdivisions with guaranteed delivery. Assuming predecessor-awareness costs extra bits of memory in the message header, for any class of graphs more general than convex subdivisions the author is only aware of algorithms that use $\Omega(\log n)$ extra bits in the worst case, a significantly larger amount of information. In this thesis we seek to find an algorithm that can guarantee delivery on a class of graphs more general than convex subdivisions using few extra bits. Specifically, we seek algorithms that use $o(\log n)$ extra bits on a broad set of graphs, and as few as $O(1)$ extra bits on some graphs in the class. We would like the number of extra bits to be a function of a parameter of the graph instance, thereby helping bridge the gap between $\Theta(\log n)$ extra bits for general planar subdivisions and 1 state bit for convex subdivisions.

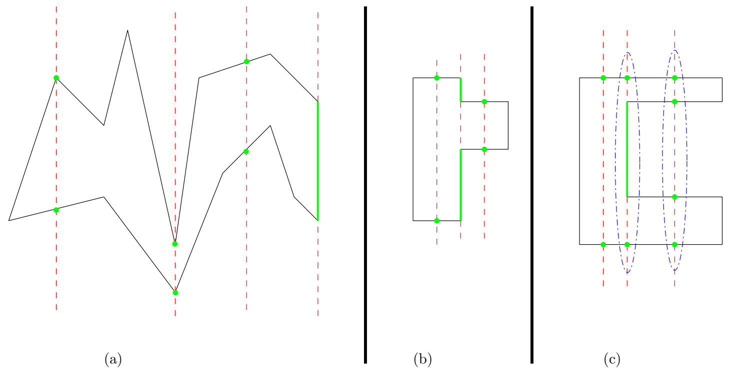

Figure 1.2: Both (a) and (b) are examples of x-monotone faces. The red, dashed, vertical lines intersect the boundaries of these faces in at most two connected sets (the intersected connected sets are highlighted in green). (c) This face is not x-monotone. The vertical line in the middle and on the right intersect the boundary of this face in three and four connected sets respectively (circled in blue).

1.4 MODEL

In this work we consider 2-dimensional undirected connected graphs embedded in the plane with straight line edges and no weights on the edges. We assume nodes know their coordinates and the coordinates of their neighbours, and we assume that the coordinates of the destination are available in the message header.

We assume the graphs do not change while routing is in progress. Furthermore, we only work with embedded planar graphs that are x-*monotone* subdivisions. A polygon is x-monotone if its boundary intersects every vertical line in at most two connected sets. The definition of x-monotonicity used in this work considers vertical line segments as x-monotone. A planar subdivision is x-monotone if each of its interior faces is an x-monotone polygon and its outer face is the complement of an x-monotone polygon. Monotone subdivisions are more general than convex subdivisions because monotone faces allow reflex vertices (i.e., vertices that have angles larger than 180°). Figure 1.2 gives some examples of faces and the connected sets formed by the intersections of their boundaries with vertical lines. The faces in Figure 1.2 (a) and Figure 1.2 (b) are x-monotone and the one in Figure 1.2 (c) is an example of a non-x-monotone face. Figure 1.3 illustrates what it means for the outer face to be the complement of an x-monotone polygon.

We assume that predecessor information is not available unless it is stored in the message header. We further assume that nodes and links do not fail, that there is perfect lossless 2-way transmission, that nodes do not run out of power, that transmission ranges do not change, that coordinates are accurate, and that coordinates are uniquely expressed with $\Theta(\log n)$ bits.

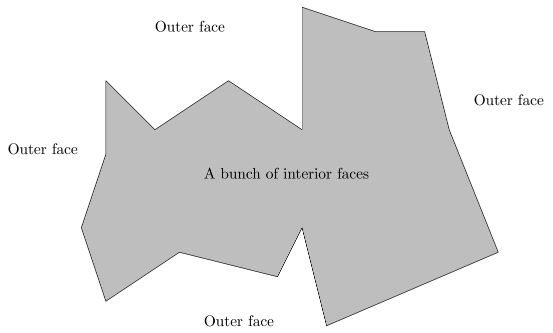

Figure 1.3: Imagine joining together all interior faces of a graph, erasing edges adjacent to other interior faces. The result is a polygon whose boundary borders the outer face. In this image, the polygon is x-monotone so we say that the outer face is the complement of an x-monotone polygon.

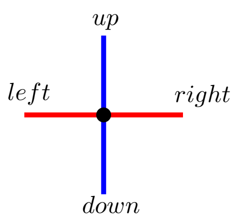

Figure 1.4: An orthogonal subdivision has two distinct slopes: vertical and horizontal. In this example, the four unique symmetric directions are broken up into two pairs: *up* and *down*, and *left* and *right*.

Our model is further constrained to d fixed edge directions[5] that are consistently known to the nodes in the graph. We parameterize the memory requirements of the routing algorithm in Chapter 2 in terms of d. Since the graphs are undirected, the d directions are symmetric (i.e., for every up there is a down). In other words, there are $\frac{d}{2}$ unique potential slopes for an edge which means there are d directions in an undirected graph. Figure 1.4 gives an example of the slopes in an orthogonal subdivision (in which d is four, the minimum value for d). There are two slopes (horizontal and vertical), and two pairs of symmetric directions (*up* and *down*, and *left* and *right*).

1.5 CONTRIBUTION

This work helps reduce the gap between local routing algorithms that require one state bit and those that use $\Omega(\log n)$ extra bits, where n is the number of nodes in the network represented on a polynomial-sized grid. We present a novel low memory local geometric routing algorithm that guarantees delivery on non-general-position x-monotone subdivisions using at most $\lceil \log(d+1) \rceil$ extra bits where each graph edge is restricted to one of d fixed placements about a

5 implying a maximum vertex degree of d

node defined by $\frac{d}{2}$ unique slopes. This parameterization helps bridge the gap for all values of d up to more general x-monotone subdivisions in which $d \in \Theta(n)$. For example, in orthogonal x-monotone subdivisions (in which d is four, the minimum possible value) our algorithm guarantees delivery using only three extra bits.

Unlike the PredAware algorithm of Bose et al. [4], our algorithm does not require general position, a necessary assumption for their algorithm. Their model assumes predecessor awareness at no cost; in this work we include the cost of predecessor awareness in the number of extra bits used. Since their algorithm performs a cyclic traversal with a period of at least $2n$, the average route length for every graph is at least $\frac{n}{2}$. In fact, for every monotone graph G there exist n pairs $\{s, t\}$ in G such that the traversal algorithm visits every node in G before reaching t. Our algorithm is more direct: it follows a monotone path in its first phase, before traversing a monotone sequence of faces in its second phase. For example, on a $\sqrt{n} \times \sqrt{n}$ grid, our algorithm follows a route of length $O(n)$ in the worst case, whereas the traversal algorithm of Bose et al. [4] follows a route of length $\Theta(n)$, not only in the worst case, but also in the average case. This is to be expected since the traversal is always identical regardless of the destination; the PredAware algorithm makes no attempt to navigate toward the destination node, but is essentially designed to traverse the whole graph.

2 LOW MEMORY LOCAL ROUTING IN MONOTONE SUBDIVISIONS

In this chapter we present a local geometric routing algorithm and prove that it guarantees delivery on x-monotone subdivisions. The algorithm consists of two phases, combining the techniques of greedy routing (refer to Section 1.2.1 and Section 2.1) and face routing (refer to Section 1.2.2 and Section 2.2). The first phase ensures that the packet arrives at an endpoint of an edge that is intersected by the vertical line through the destination; the second phase forwards the packet along a sequence of faces intersected by the vertical line through the destination until the packet reaches the destination. The location of the destination is passed with the message, but it is not considered as extra memory since knowledge of it is necessary and it is not modified by the algorithm. The first phase requires one flag bit. The second phase uses face routing which requires the ability to identify the predecessor at each step.

Let $G = (V, E)$ be the graph corresponding to the x-monotone subdivision under consideration. Without loss of generality, we denote the destination node by $t \in V$. Let l_t be the vertical line through t. Recall that we assume that all nodes have consistent knowledge of the directions in the graph. Since G has d fixed directions, predecessor information can be encoded using $\lceil \log d \rceil$ extra bits in the message header, represented by $\texttt{pred} \in \{0, \dots, \lceil \log d \rceil - 1\}$, which indicate the direction in which the predecessor vertex lies relative to the current vertex. Since predecessor information is not used in Phase 1, one of the $\lceil \log d \rceil$ predecessor bits from Phase 2 can be used to store the flag bit. A separate phase bit is used to keep track of which phase the algorithm is in, therefore the algorithm needs $\lceil \log (d+1) \rceil$ bits of extra memory. For example, if $d = 8$ we have $\lceil \log (8+1) \rceil = 4$ bits, but only the first three are needed to encode the predecessor. The fourth bit is used to mark the phase. Since predecessor information is only used in Phase 2, in Phase 1 we can use any of the first three bits as the flag bit. If $d = 6$, however, we have $\lceil \log (6+1) \rceil = 3$ bits and the combinations "110" and "111" are unused for encoding predecessor information (assuming we use the binary representations of the numbers $0 - 5$ to encode the six directions), so either of those values of $\texttt{pred}$ indicate that the algorithm is in Phase 1, and each of them represents a different state of the flag bit.

2.1 PHASE 1

We begin this section with a description of the first phase and prove that it works as advertised. The section ends with a note about why a flag bit is used in Phase 1.

We refer to an edge that is intersected by l_t as an *intersected edge* and a point that is intersected by l_t as an *intersected point*.

The first time Algorithm 1 is invoked, all that is known about the current vertex (i.e., the packet's source) is that it is some vertex in an x-monotone subdivision. The aim of Phase 1 is to deliver the packet to an endpoint of an intersected edge, after which it is known that the two faces incident to that edge are intersected by l_t. At that point the packet can follow the faces intersected by l_t to arrive at its destination.

As seen in Lemma 2.1, in an x-monotone subdivision there is an x-monotone path from every vertex to l_t. Since this is true, a greedy approach can be used to arrive at l_t: the packet is forwarded to a neighbour that is strictly closer to l_t if one exists, otherwise exploring neighbours vertically will lead to a neighbour closer to l_t. As seen in Corollary 2.2, after a finite number of forwarding decisions Phase 1 delivers the packet to an endpoint of an intersected edge.

Regardless of which phase the algorithm is in, the vertex with the packet knows its coordinates, the coordinates of its neighbours, and the coordinates of the destination t when invoking Algorithm 1 as well as the value of the phase bit which indicates whether the algorithm is in Phase 1 or Phase 2.

When a vertex receives the packet and invokes Algorithm 1, if it is in Phase 1, it also gets as input a flag bit called "up" that is used to keep track of the direction in which vertical edges are being explored. The first node that invokes Algorithm 1 (i.e., the packet's source) is responsible for initializing the phase bit such that the algorithm is in Phase 1, and it is responsible for initializing the flag bit to 0.

After an execution of Phase 1, the phase bit indicates whether the algorithm is still in Phase 1. If after an execution of the algorithm it is still in the first phase, the vertex currently holding the packet knows to which of its neighbours it must send the packet as well as the updated values of the phase bit and flag bit.

For a vertex v, let v_x and v_y denote the x and y coordinates of v, respectively. A *vertical neighbour* of the current vertex u is a vertex w neighbouring u such that $w_x = u_x$ and $w_y \neq u_y$ (refer to Figure 2.1 (a)).

Let $d_h(x, l)$ be the minimum Euclidean distance from point x to line l. A *forward neighbour* of the current vertex u is a neighbour v such that $d_h(v, l_t) < d_h(u, l_t)$ (refer to Figure 2.1 (b)).

LEMMA 2.1. *Let $G = (V, E)$ be an x-monotone subdivision. There exists an x-monotone path from every vertex $u \in V$ to l_t.*

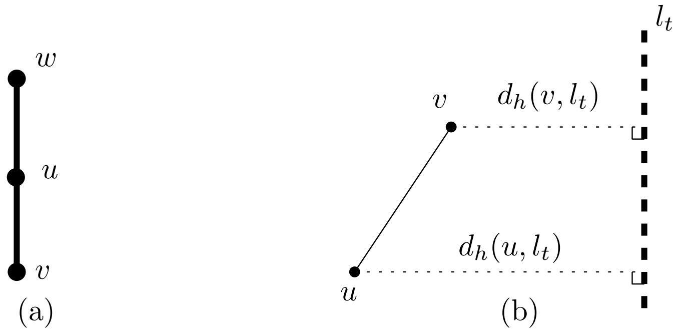

Figure 2.1: (a) A vertex u and two vertical neighbours w and v; (b) A vertex u and a forward neighbour v. The vertex v is a forward neighbour of u because $d_h(v, l_t) < d_h(u, l_t)$.

Proof. Let $u \in V$ be any vertex. If u already lies along l_t, there is nothing to prove. Assume for the rest of the proof that u does not lie along l_t. Next, we argue that from u, we can always follow a finite number of vertical edges (possibly none) and then move to a forward neighbour. Since this argument works for any $u \in V$ and since G is finite, this implies the statement of the lemma.

Let l_u be the vertical line through u. Suppose, without loss of generality, that l_t lies to the right of u. Since G is 2-connected, there are at least 2 edges incident to u. If all edges incident to u lie strictly to the left of l_u, since l_t lies to the right of u, then u is part of a face that is neither x-monotone nor the complement of an x-monotone polygon. Thus, u has at least one vertical neighbour or one forward neighbour.

Suppose that u has no forward neighbour. Let $u' \in l_u$ be the lowest vertex that can be reached from u while staying on l_u. Let $u'' \in l_u$ be the highest vertex that can be reached from u while staying on l_u. Notice that it is possible to have $u' = u$ or $u'' = u$, but not both since u has at least one vertical neighbour. If, on the path $\pi_{u',u''}$ from u' to u'' along l_u no vertex has a forward neighbour, then $\pi_{u',u''}$ is part of a face that is neither x-monotone nor the complement of an x-monotone polygon. This is a contradiction, therefore there is a vertex on the path from u' to u'' that has a forward neighbour. □

COROLLARY 2.2. *Given as input a current vertex u of an x-monotone subdivision $G = (V, E)$ with $u \in V$, the neighbourhood $N(u)$ of u, the phase bit, the "up" flag bit, and the destination t, after a finite number of forwarding decisions made by Phase 1, the packet is delivered to an endpoint of an intersected edge.*

Proof. By Lemma 2.1, there exists an x-monotone path from u to l_t. Let l_u be the vertical line through u, $u' \in l_u$ be the lowest vertex that can be reached from u while staying on l_u, and $u'' \in l_u$ be the highest vertex that can be reached from u while staying on l_u. Suppose, without loss of generality, that l_t lies to the right of u. Assume that u is not already an endpoint of an intersected edge (otherwise there is nothing to prove). Phase 1 either sends the message to a forward neighbour if it exists, or to a vertical neighbour. If u has no forward neighbour, the bit up used in Phase 1 makes the message travel from u to u' and then from u' to u'', or from u to u'' and then from u'' to u' (depending on the topology of the graph). By monotonicity, at least one of the vertices visited by the message on the path from u' to u'' along l_u has a forward neighbour. Let u^* be the first such vertex visited by the message. When the message reaches u^*, the message has to be forwarded to the forward neighbour of u^*. Indeed Phase 1 tests this case before forwarding the message to a vertical neighbour. Since the graph is finite, the Corollary follows. □

Algorithm 1

```
Phase 1: Find the vertical line $l_t$
Let $next$ denote the neighbour of the current vertex $u$ to which the message will be forwarded.
Let $v_{above}$ be the vertical neighbour of $u$ above $u$ (if one exists).
Let $v_{below}$ be the vertical neighbour of $u$ below $u$ (if one exists).
Let $up$ be a boolean flag with $0$ denoting down and $1$ denoting up.
1:  if $u = t$ then                                  ▷ the destination is reached
2:      Algorithm terminates
3:  if $phase$ = first phase then
4:      if t is a neighbour of $u$ then
5:          $next \leftarrow t$
6:      else if $u$ has a neighbour $q$ such that $\overline{uq}$ intersects $l_t$ then
7:          $phase \leftarrow$ second phase
8:          Go to Phase 2 with the appropriate input
9:      else if $u$ has any neighbour $q$ such that $d_h(u, l_t) > d_h(q, l_t)$
          then
10:         $next \leftarrow q$
11:     else if $up = 0$ and $v_{below}$ exists then
12:         $next \leftarrow v_{below}$
13:     else if $up = 0$ and $v_{below}$ does not exist then
14:         $up \leftarrow 1$
15:         $next \leftarrow v_{above}$              ▷ $v_{above}$ exists by monotonicity.
16:     else if $up = 1$ and $v_{above}$ exists then
17:         $next \leftarrow v_{above}$
18:     else                                         ▷ $up = 1$ and $v_{above}$ does not exist
19:         $up \leftarrow 0$
20:         $next \leftarrow v_{below}$              ▷ $v_{below}$ exists by monotonicity
21:     Forward the message to $next$
```

Phase 2: Find the destination t

Let n be the clockwise (CW) neighbour of u with respect to the predecessor p represented by the extra bits $pred$.

```
22: if $phase$ = second phase then
23:     if $t$ is a neighbour of $u$ then
24:         $next \leftarrow t$
25:     else if $u$ is a close vertex then    ▷ Recall that a close vertex is an endpoint of an intersected edge.
26:         if $u$ is on $l_t$ then
27:             if $u$ has a vertical neighbour $q \neq p$ leading toward $t$ then
28:                 $next \leftarrow q$
29:             else if $u$ has a vertical neighbour $q = p$ leading toward $t$ then
30:                 $next \leftarrow n$
31:             else
32:                 Find the closest intersected edge $\overline{uq}$.
33:                 $next \leftarrow q$
34:         else
35:             Find the closest intersected edge $\overline{uq}$ such that the intersection of $\overline{uq}$ and $l_t$ is below $t$.
36:             if $q \neq p$ then
37:                 if $q$ lies on $l_t$ then
38:                     $next \leftarrow q$
39:                 else
40:                     $next \leftarrow$ CW neighbour of $u$ with respect to $q$
41:             else
42:                 $next \leftarrow n$
43:     else
44:         $next \leftarrow n$
45:     Forward the message to $next$ and set $pred$ to represent $u$
```

2.1.1 *A Note About the Use of a Flag Bit*

With regards to our algorithm, it should be pointed out that simply rotating the coordinate system in Phase 1 by some arbitrarily small amount to avoid the case where there are no neighbours leading toward l_t (to force a vertical neighbour to become a neighbour leading toward l_t) will not work since it may make faces non-x-monotone. Since the algorithm is deterministic, any such rule would need to make the same decision in the same scenario with the same information. If the rule says to always rotate the up-direction to the right, it is easy to come up with an example where the destination is on the left and the packet gets stuck. An example can also be found for when the destination is on the right. If the rule is to rotate the up-direction in the direction that t lies, a similar example can be constructed by mirroring the first. Note that though the faces in question would have become non-x-monotone due to the rotation, since the graph started x-monotone, one of the ends of the piece that was initially vertical will certainly have something leading towards l_t. To attempt to find this piece, however, requires keeping track of which of the two ends of the vertical piece has been explored; implementing that would be equivalent to the part of the algorithm it was trying to remove.

2.2 PHASE 2

Phase 2 delivers the packet to t by following faces intersected by l_t toward t. This is done by face routing an intersected face until an intersected edge of a closer intersected face is found, at which point the packet starts face routing on the closer face. Since t lies on l_t, it suffices to face route on one side of l_t as long as vertices that lie on l_t are explored when encountered.

We show in Lemma 2.4 that the first time Phase 2 is executed after Phase 1 the packet is at an endpoint of an intersected edge, and that the endpoint in question lies on l_t or to one side of it. Lemma 2.6 shows that the packet never crosses from one side of l_t to another while in Phase 2, and after each forwarding decision in Phase 2, progress is made toward t. Corollary 2.7 then shows that t is found after a finite number of forwarding decisions.

Regardless of which phase the algorithm is in, the vertex with the packet knows its coordinates, the coordinates of its neighbours, and the coordinates of the destination t when invoking Algorithm 1 as well as the value of the phase bit.

When a vertex receives the packet and invokes Algorithm 1, if it is in Phase 2, it also gets some bits $pred$ that indicate the neighbour p that should be considered as the predecessor.

We consider two kinds of intersected edges: vertical and non-vertical. If the vertex u that currently holds the packet is part of a vertical in-

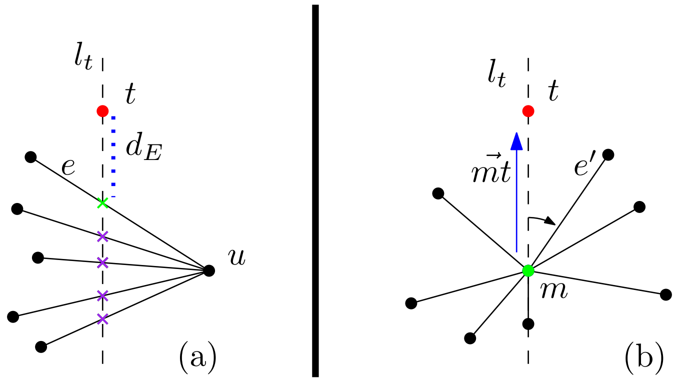

Figure 2.2: (a) The intersected edge e is closer to t than the other intersected edges with the common endpoint u because the distance d_E (illustrated by the blue dashed line) from the intersection point of e (the green cross) to t is shortest (compared to the distance to t from the purple crosses); (b) The distance d_E to the intersection points is the same for multiple edges. The closest point of all of these edges to t is the point m (green vertex). The closer intersected edge in this case is the first edge e' encountered by sweeping the ray $\vec{mt}$ (blue arrow) in the clockwise direction.

tersected edge $\overline{uv}$, it is said that $\overline{uv}$ *leads toward* t if $|(v_y - t_y)| < |(u_y - t_y)|$.

There are two kinds of non-vertical intersected edges: those for which the endpoints lie on opposite sides of l_t; and those for which one endpoint lies *on* l_t and the other lies on either side of l_t. Let a *close vertex* be an endpoint of an intersected edge.

DEFINITION 2.3. *Consider an intersected edge* e. *Let* d_E *denote the Euclidean distance from the closest intersected point of* e *to* t. *An intersected edge is* closer to t *than another intersected edge with a common endpoint if it has a smaller* d_E *(because it is closer to* t*). This case is illustrated in Figure 2.2 (a). It could happen that* d_E *is the same for many incident intersected edges leading to a tie. In that case, let the closest intersected point on* e *be* m. *Consider the first edge* e' *incident to the point* m *reached by sweeping a ray* $\vec{mt}$ *about* m *in a clockwise manner. In this case, we consider* e' *to be the closest intersected edge. This case is illustrated in Figure 2.2 (b).*

At the end of Phase 1, when the packet arrives at a close vertex k, the algorithm changes the phase bit from the first to the second phase. However, it must ensure that when Phase 2 is executed for the first time, what will be considered the current vertex will lie on l_t if possible, otherwise it will lie to a specific side of l_t. This selection is done as follows: first, the intersected edge $\overline{kq}$ closest to t is found; then an intersected point m on $\overline{kq}$ is chosen; the ray $\vec{mt}$ is considered to be the "up" direction so that t lies above $\overline{kq}$; the vertices k and q are then input into Phase 2 as either the predecessor or the current vertex such that at the start of Phase 2 the current vertex lies on l_t if possible (and is the closer of the two to t if they both lie on l_t), otherwise such that the current vertex is on the right side of $\vec{mt}$ with t above. This adjustment may involve forwarding the packet once. Lemma 2.6 shows that in Phase 2 the packet never crosses l_t, so we do not need a bit to remember which side of l_t the packet is on. The only time an interior face has an intersected edge both above and below t is when t is on that face, but one of those intersected edges must be incident to a sequence of vertical intersected edges that lead to t (by monotonicity). If the packet encounters an intersected edge on its current face that lies on the side of t opposite where it started, then if the packet goes to this edge it will follow vertical intersected edges to t (refer to line 27).

Therefore, without loss of generality, we can consider the packet as being on l_t or to the right of l_t while in Phase 2, we can consider the packet as encountering non-vertical intersected edges that are below t (instead of both above and below), and we can consider t as being above the current vertex.

After an execution of Phase 2, the phase bit indicates to the successor vertex that the algorithm is still in Phase 2. If after an execution of Phase 2 the packet has not yet arrived at t, the vertex currently holding the packet knows to which of its neighbours it must send the packet as well as the updated value of $\mathtt{pred}$.

LEMMA 2.4. *The first time Phase 2 is executed the packet starts on a close vertex that lies on l_t or that lies on the right side of l_t and t can be considered as being above.*

Proof. The algorithm only switches to Phase 2 when the packet gets to a close vertex (see line 6). By definition, the close vertex k at which this happens is part of at least one edge intersected by l_t. At this point, of all edges incident to k, the intersected edge $\overline{kq}$ that is closest to t is chosen (refer to Definition 2.3). The coordinates are then temporarily adjusted using a point m on $\overline{kq}$ that is intersected by l_t such that $\vec{mt}$ is the "up" direction and t lies above $\overline{kq}$. Finally, k and q are assigned to be the predecessor or the current vertex such that at the start of Phase 2, either the current vertex lies on l_t if possible (and is the closer of the two to t if both vertices considered lie on l_t), or it lies to the right of l_t with $\vec{mt}$ as the up direction (one of these two scenarios is guaranteed by monotonicity). This is done by executing Phase 2 for the first time either at k (with *phase* = second phase and $\mathtt{pred} = q$) or at q (by forwarding the packet to q with *phase* = second phase and $\mathtt{pred} = k$). □

The *clockwise (CW) neighbour* v of the current vertex u is defined with respect to the predecessor p of u. Since the graph is 2-connected, every edge is adjacent to two faces. Consider the common face F adjacent to both edges $\overline{pu}$ and $\overline{uv}$ defined by the CW angle $\angle puv$ starting at edge $\overline{pu}$. As illustrated in Figure 2.3 (a), we say F is the current face and the packet is *on* F.

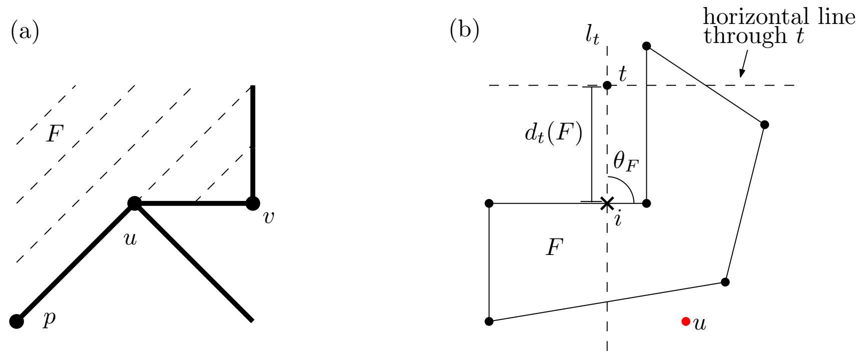

Figure 2.3: (a) We say the packet is on the current face F that is the common face adjacent to both edges $\overline{pu}$ and $\overline{uv}$ where p is the predecessor, u is the current vertex, and v is the clockwise neighbour of u; (b) The face F is an interior face intersected by l_t. The vertex u is the vertex currently holding the packet. The point i is the closest point to t on F such that i is intersected by l_t and on the same side of the horizontal line through t as the packet. The value $d_t(F)$ is the Euclidean distance between t and i. Also shown is θ_F, the minimum clockwise angle $\angle tij$ for all $j \in F$ where j is adjacent to i.

Let F be a face intersected by l_t. Let i be the closest point to t on F such that i is intersected by l_t and such that i lies on the same side of the horizontal line through t as the packet, as illustrated in Figure 2.3 (b). Let $d_t(F)$ be the Euclidean distance between t and i. Let θ_F be the minimum CW angle $\angle tij$ for all $j \in F$ where j is a vertex adjacent to the point i. Let $D_t(F) = (d_t(F), \theta_F)$ and let $D_t(A) < D_t(B)$ if $d_t(A) < d_t(B)$, or $d_t(A) = d_t(B)$ and $\theta_A < \theta_B$. Consider two adjacent faces F_1 and F_2 which are both intersected by l_t. F_1 *is closer than* F_2 to t if $D_t(F_1) < D_t(F_2)$.

LEMMA 2.5. *Let* e *be a non-vertical intersected edge incident to the outer face. Suppose that* t *is not incident to* e*. By 2-connectivity,* e *is incident to the outer face and some inner face. If* t *lies to the same side of* e *as the part of the outer face that is incident to* e*, then* t *lies along a sequence of vertical intersected edges of the outer face, and this sequence of edges is incident to an intersected endpoint of* e*.*

Proof. An example of the situation described in the lemma is shown in Figure 2.4.

If t and the outer face lie on the same side of e, then either both t and the outer face lie above e or they both lie below e. Without loss of generality, assume they both lie above e. Since e is intersected, this means that some point of e has the same x-coordinate as t, and since t is not incident to e every point in e has a smaller y-coordinate than t. If t lies anywhere other than above one of the endpoints of e, part of the outer face would lie between the part of the graph that contains t and the part of the graph that lies below e, meaning

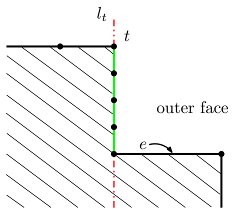

Figure 2.4: An example of the situation described in Lemma 2.5. The edge e is a non-vertical intersected edge of the outer face, the destination t is not incident to e, and the stripes denote interior faces. The outer face and t both lie above e and t lies along a sequence of vertical intersected edges of the outer face incident to e.

that the outer face would not be the complement of an x-monotone polygon. That would be a contradiction to the model, so t must have the same x-coordinate as one of e's endpoints. Since interior faces are x-monotone and the outer face is the complement of an x-monotone polygon, and since the graph is 2-connected with straight edges, the only way t could have the same x-coordinate as an endpoint of e is if it lies along a sequence of vertical intersected edges of the outer face incident to the endpoint of e with which it shares an x-coordinate.

Therefore, if the destination t is not incident to e, if t and the outer face lie on the same side of e, then t must lie along a sequence of vertical intersected edges of the outer face incident to e. □

This next lemma has to consider a lot of subcases to make its argument. To sum it up, it basically shows that if after a forwarding decision the packet has not arrived at its destination, since the packet started either on l_t or to its right and t is above the packet, any forwarding decision moves the packet closer to the destination either along l_t or along an intersected face on the right side of l_t.

LEMMA 2.6. *Let the inputs for Phase 2 be the destination* t, *the current vertex* u *that lies on or to the right of* l_t *on a face intersected by* l_t, *the bits* $\mathtt{pred}$ *representing the predecessor* p, *the phase bit, and let the current face be* F. *After each forwarding decision made in Phase 2 either the packet is sent to* t *and the algorithm terminates, the packet moves along* l_t *strictly closer to* t, *the packet moves onto an adjacent face* F' *such that* $D_t(F') < D_t(F)$, *or it makes progress on* F. *The packet is never forwarded onto the left side of* l_t.

Proof. Recall that a close vertex is an endpoint of an intersected edge. Recall that when considering a close vertex, the closer intersected edge of which it is incident is the edge whose intersection with l_t is closest, or in the case of a tie it is the edge forming the smallest CW angle with t (see Definition 2.3).

Let the CW neighbour of u with respect to p be n.

Let the Euclidean distance of an intersected edge e_i to t be the shortest Euclidean distance between t and a point of e_i that is intersected by l_t.

By Lemma 2.4, at the start of Phase 2 the packet is either on l_t or on the right of l_t, and t can be considered as being above. Recall that the only time an interior face has an intersected edge both above and below t is when t is on that face, but one of those intersected edges must be incident to a sequence of vertical intersected edges that lead to t (by monotonicity). Therefore if the packet encounters an intersected edge on its face that lies on the side of the horizontal line through t opposite where it started, then if the packet goes to this edge it will follow vertical intersected edges to t (refer to line 27) and will not have crossed onto the left side of l_t. If the intersected vertical edges leading to t lie on the same side of the horizontal line through t as where the packet started on the current face, then the cases below show that the packet arrives at t.

If $u = t$ or t is a neighbour of u then the packet is sent to t, the algorithm terminates (refer to lines 1 and 23), and the packet is on l_t. Otherwise, there are two cases to consider: the case where u is a close vertex, and the case where it is not.

Case 1: u **is a close vertex**

There are two sub-cases to consider: the case where u lies on l_t and the case where it lies to the right of l_t.

Case 1.1: u **lies on** l_t

When u lies on l_t there are three cases: u has a vertical neighbour $q \neq p$ leading toward t; p is the vertical neighbour leading toward t; and the case where u does not have a vertical neighbour leading toward t.

Case 1.1.1: u **has a vertical neighbour** $q \neq p$ **leading toward** t

Figure 2.5 and its subfigures show some examples of this case. In this case, q is strictly closer to t than u. Therefore, when the packet is sent to q (refer to line 27), the packet moves along l_t to a vertex that is strictly closer to t than the vertex at which it started and remains on l_t. If this progress along l_t is ever reversed, it is because the packet encountered Case 1.1.3, but then the reversal is part of face routing the current face (i.e., the face that lies between t and the vertex on this sequence of vertical intersected edges with the smallest Euclidean distance to t), which must lead to either an intersected edge that has a smaller Euclidean distance to t than any vertex on this sequence of vertical intersected edges does, or it will lead to t itself. The progress made while face routing will not be reversed, so ultimately progress is made.

Case 1.1.2: p **is the vertical neighbour leading toward** t

Figure 2.6 shows an example of this case. In this case, the algorithm sends the packet to the CW neighbour n (and therefore it stays on F) which is the closest intersected edge $\overline{uq}$ where $q \neq p$ (refer to line 29).

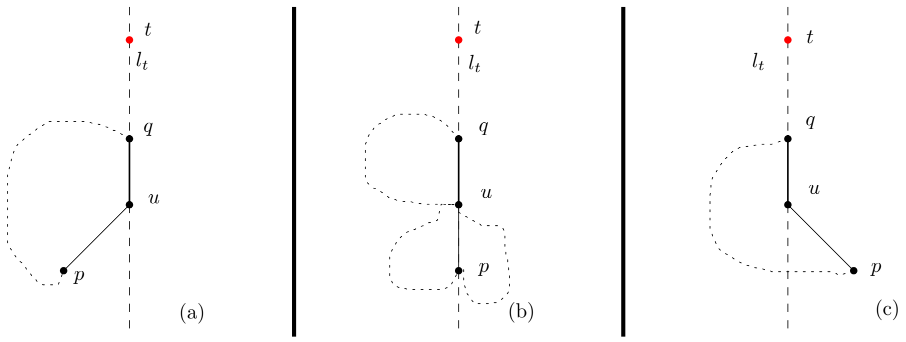

Figure 2.5: In this case, u lies on l_t and has a vertical neighbour $q \neq p$ closer to t. The subfigures (a), (b), and (c) show some example scenarios and potential positions of p.

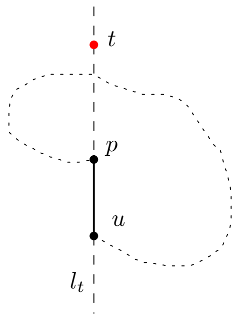

Figure 2.6: In this case, u lies on l_t and the vertical neighbour closer to t is the predecessor p.

The vertex n must lie on l_t below the current vertex, or to the right of l_t. This follows by monotonicity and by the fact that n is chosen in the CW direction. By Lemma 2.5, F is not the outer face (otherwise there would be vertical intersected edges incident to u that could be followed to t). By monotonicity, there must be an intersected edge on F that has a smaller Euclidean distance to t than any vertex on this sequence of vertical intersected edges does. By moving to the CW neighbour n the packet makes progress on F toward an intersected edge with a smaller Euclidean distance to t and the packet does not cross onto the left side of l_t. This action makes the next execution potentially go through Case 1.1.2 again, otherwise it leads to Case 1.2.2 and then Case 2. In neither of those cases does this progress around F get reversed.

Case 1.1.3: u does not have a vertical neighbour leading toward t

Figure 2.7 shows some examples of this case. In this case, the algorithm begins by finding the closest intersected edge $\overline{uq}$ and sends the packet to q (refer to line 31). The vertex q must lie on l_t or to the right of l_t by reasoning similar to Case 1.1.2. Therefore the packet does not cross onto the left side of l_t. Let the face defined by u, q, and the CW neighbour of q with respect to u be F'. Note that if q

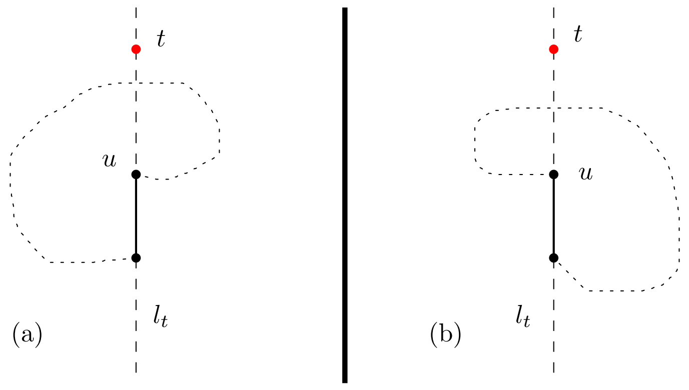

Figure 2.7: In this case, u lies on l_t and has no vertical neighbour closer to t. The subfigures (a) and (b) show two example scenarios.

was the predecessor, then sending the packet to q puts it on the face adjacent to F defined by $\overline{up}$. By Lemma 2.5, F' is not the outer face. Since $\overline{uq}$ is the closest intersected edge, if $F \neq F'$ then F' must have an intersected edge with a smaller Euclidean distance to t than any intersected edge on F does, if $F = F'$ there must be an intersected edge that has a smaller Euclidean distance to t than $\overline{uq}$ does, or in either case it has an intersected edge that is above t (in which case t is on F' by monotonicity). Therefore, by sending the packet to q either it is now on a face F' such that $D_t(F') < D_t(F)$, or it makes progress on F toward an intersected edge that has a smaller Euclidean distance to t than the intersected edges visited so far, or it makes progress toward an intersected edge on F' that lies above t (which has been seen to lead to the packet being delivered to t).

<u>Case 1.2:</u> u lies to the right of l_t

In this case, the algorithm starts by finding the closest intersected edge $\overline{uq}$ such that the intersection of $\overline{uq}$ and l_t is below t (refer to line 34). There are two sub-cases to consider: the case where $q \neq p$; and the case where $q = p$.

<u>Case 1.2.1:</u> $q \neq p$

There are two cases: q lies on l_t; q lies to the left of l_t.

<u>Case 1.2.1.1:</u> q lies on l_t

Figure 2.8 shows an example of this case. In this case, the packet is sent to q (refer to line 37), so the packet does not cross onto the left side of l_t. In this case, either q was the CW neighbour of u or it was not. If q was the CW neighbour of u, then the packet makes progress on F. If q was not the CW neighbour of u, then $\overline{uq}$ must have a shorter Euclidean distance to t than any intersected edge on F does since $\overline{uq}$ is the closest intersected edge of u such that the intersection of $\overline{uq}$ and l_t is below t. In this case by moving to q the packet is now on a face F' with $D_t(F') < D_t(F)$.

<u>Case 1.2.1.2:</u> q lies to the left of l_t

Figure 2.9 shows an example of this case. In this case, the packet is

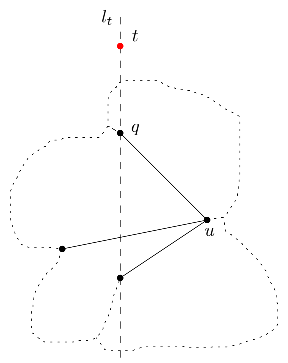

Figure 2.8: In this case, the closest intersected edge of u that lies below t has the other endpoint q that lies on l_t, $q \neq p$. The algorithm continues face routing by sending the packet to q.

sent to the CW neighbour v of u with respect to q on face F' (refer to line 39). Since $\overline{uq}$ was the closest intersected edge such that the intersection of $\overline{uq}$ and l_t is below t, v lies to the right of l_t or on l_t. It cannot be the case that v lies to the left of l_t, because that would either mean that $\overline{uv}$ was the intersected edge that should have been chosen, or that t lies between the two edges $\overline{uq}$ and $\overline{uv}$ and therefore F' is not x-monotone. Therefore, the packet does not cross onto the left side of l_t.

If v lies on l_t then it must be above t (otherwise $\overline{uv}$ should have been chosen instead of $\overline{uq}$), it must be on the same face as t, and it must be incident to vertical intersected edges that lead to t, otherwise F' is not x-monotone. In this case, once at v the packet will be continually forwarded along the vertical intersected edges until it gets to t (refer to line 27).

Consider the case where v lies to the right of l_t. Note that v cannot be the CW neighbour of u with respect to p because then $q = p$. In this case, either q was the CW neighbour of u, or $\overline{uq}$ has a shorter Euclidean distance to t than any intersected edge on F does. If q was the CW neighbour of u, then by moving to v the packet is now on a face F'' with $D_t(F'') < D_t(F)$ because F'' must have an intersected edge with a shorter Euclidean distance to t than any intersected edge on F does, or F'' has an intersected edge above t (which has been seen to lead to the packet being delivered to t).

If q was not the CW neighbour of u, then $\overline{uq}$ must have a shorter Euclidean distance to t than any intersected edge on F does since $\overline{uq}$ is the closest intersected edge of u such that the intersection of $\overline{uq}$ and l_t is below t. In this case by moving to v the packet is now on a face F''' with $D_t(F''') < D_t(F)$ because F''' must have an intersected edge that has a shorter Euclidean distance to t than $\overline{uq}$ does, or an intersected edge above t (which has been seen to lead to the packet being delivered to t).

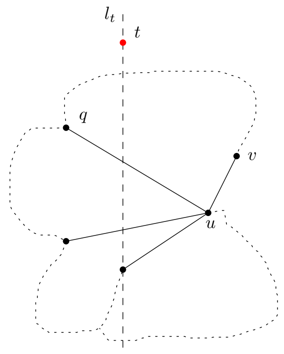

Figure 2.9: In this case, the closest intersected edge of u that lies below t has the other endpoint q that lies to the left of l_t, $q \neq p$. The algorithm continues face routing on a closer face by sending the packet to v, the CW neighbour of u with respect to q.

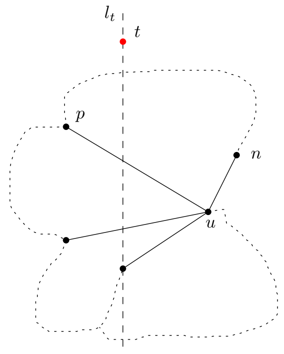

Figure 2.10: In this case, the closest intersected edge of u that lies below t has p as the other endpoint. The algorithm continues face routing by sending the packet to n.

Case 1.2.2: $q = p$

Figure 2.10 shows an example of this case. In this case, the packet is sent to the CW neighbour n (refer to line 41). It cannot be the case that n lies to the left of l_t, because that would either mean that $\overline{un}$ should have been chosen instead of $\overline{uq}$, or that t lies between the two edges $\overline{uq}$ and $\overline{un}$ and therefore the current face is not x-monotone. Note that if n lies on l_t then it must be above t (because $\overline{uq}$ was chosen instead of $\overline{un}$), the current face must be the one that t is on, and n must be incident to vertical intersected edges that lead to t, otherwise the face is not x-monotone. In that case, once at n the packet will continually be forwarded along vertical intersected edges until it gets to t (refer to line 27). If n lies to the right of l_t, then the packet makes progress on F toward another intersected edge that has a shorter Euclidean distance to t than any intersected edge seen so far does, or it makes progress toward an intersected edge that lies above t which has been seen to lead to t.

Case 2: u **is not a close vertex**

Figure 2.11 shows an example of this case. In this case the packet is

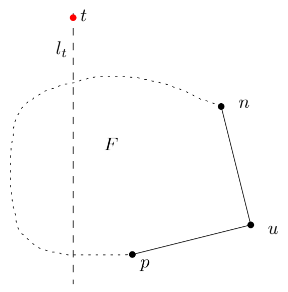

Figure 2.11: In this case the current vertex u is not a close vertex, lies to the right of l_t, and lies below t. The algorithm continues face routing the face F by sending the packet to the CW neighbour n of u with respect to the predecessor p.

sent to n (refer to line 43). By Lemma 2.4, during the first execution of Phase 2 the packet either lies on l_t or to its right. Since a packet can only be forwarded to a direct neighbour, since u is not a close vertex, and since in Case 1 the packet never crosses onto the left side of l_t, u and n must both lie to the right of l_t. Therefore, after the packet is sent to n either it makes progress on F toward an intersected edge that has a shorter Euclidean distance to t than any intersected edge seen so far does, or it makes progress toward an intersected edge that lies above t (which has been seen to lead to t), and in either case the packet does not cross onto the left side of l_t.

Therefore, after each forwarding decision made in Phase 2 either the packet is sent to t and the algorithm terminates, the packet moves along l_t strictly closer to t, the packet moves onto an adjacent face F' such that $D_t(F') < D_t(F)$, or it makes progress on F, and the packet is never forwarded to the left of l_t. □

COROLLARY 2.7. *Given as input the destination* t*, a current vertex* u *that is part of a face intersected by* l_t *in an* x*-monotone subdivision, the neighbourhood* $N(u)$ *of* u*, the predecessor bits* pred*, and the phase bit, after a finite number of forwarding decisions made by Phase* 2*, the packet is delivered to* t*.*

Proof. By Lemma 2.4, the first time Phase 2 is executed the packet starts on a close vertex that lies on l_t or to the right of l_t with t above. By Lemma 2.6, the packet never crosses l_t onto the left side and after each forwarding decision made in Phase 2, either the packet is sent to t and the algorithm terminates, the packet moves along l_t strictly closer to t, the packet moves onto an adjacent face F' such that $D_t(F') < D_t(F)$, or it makes progress on F. Since the graph is finite, the packet cannot make progress forever, therefore the corollary follows. □

THEOREM 2.8. *Given as input a starting vertex* s *of an* x*-monotone subdivision, the neighbourhood* $N(s)$ *of* s*, a phase bit, a flag bit, predecessor bits, and the destination* t*, after a finite number of forwarding decisions Algorithm 1 delivers the packet to* t*.*

Proof. If at any point t is a neighbour of the current vertex, the packet is forwarded to t (refer to line 4 and line 23). If at any point the packet is at t, the algorithm terminates (refer to line 1). The rest of the theorem follows from Corollary 2.2 and Corollary 2.7. □

In this section we have proven that with its two phases combining Cartesian greedy routing and face routing, our algorithm guarantees delivery on monotone subdivisions using $\lceil \log(d+1) \rceil$ bits of extra memory.

3 CONCLUSION

We begin the concluding chapter with a summary of the results of the thesis, followed by a discussion of results, and ultimately by mentioning directions for future research.

3.1 SUMMARY OF RESULTS

In this thesis we presented a local geometric routing algorithm that guarantees delivery in plane monotone subdivisions by passing a small number of extra bits with the message header.

The algorithm works in two phases, combining Cartesian greedy routing (i.e. greedy routing that minimizes Euclidean distance to the destination) with face routing. There are two important assumptions, the first being that the graph is x-monotone. We also assume that edges can only lie along $\frac{d}{2}$ fixed slopes, meaning there are only d possible edge directions, and that all nodes have consistent knowledge of these d directions in which their neighbours may lie.

The first phase navigates the packet toward the vertical line through the destination using Cartesian greedy routing when possible, otherwise exploring vertical edges until such time that Cartesian greedy routing is once again possible. Only one bit is required to perform Phase 1, and then another bit is required to keep track of the current phase of the algorithm.

Once at the vertical line through the destination, Phase 2 follows the faces intersected by that line until the packet arrives at the destination. This is accomplished using a slightly modified version of face routing that keeps the packet to one side of the vertical line. Face routing requires predecessor-awareness. Assuming that a node is not able to tell which neighbour sent it a message, nodes need to rely on extra bits containing the unique ID (i.e. coordinates) of the sender. Generally this is done with $\Theta(\log n)$ extra bits, but we are able to accomplish it with fewer. We do this by exploiting the fact that when d is less than the number of nodes n in the graph, because everyone has consistent knowledge of the directions, predecessor information can be known for cheaper by making note of the *direction* of the predecessor rather than the *coordinates* of the predecessor.

It is in this way that our algorithm guarantees delivery in x-monotone subdivisions using $\lceil \log (d+1) \rceil$ extra bits in the message header.

3.2 DISCUSSION

Our deterministic local geometric routing algorithm improves on the *PredAware* routing algorithm [4] in two ways: by removing the general position requirement; and by reducing the number of extra storage bits used to guarantee delivery in constrained-slope planar monotone subdivisions from $\Omega(\log n)$ extra bits[1] to $\lceil \log(d+1) \rceil$ while the number of edge directions d is less than the number of vertices n. This parameterization of d helps bridge the gap between algorithms that require $\Theta(1)$ extra bits and those that use $\Theta(\log n)$ extra bits for all values of d up to general monotone subdivisions in which $d \in \Theta(n)$.

Seeing as how the same approach can be used to know predecessors in PredAware (i.e. parameterizing in terms of the number of slopes), the most significant result is that we can remove the general position assumption of PredAware using $\Theta(1)$ more bits of information (once again, as long as the number of edge directions d is less than the number of vertices n and all nodes know the possible directions in which neighbours can lie).

The monotonicity property is leveraged in our algorithm. The reason we can do Phase 1 with one bit using the basic strategy from *OneBit* [4] is that monotonicity guarantees that before we flip our flag bit twice we will find an edge leading forward. The reason we can do face routing in Phase 2 without needing to remember any boundary points is because monotonicity guarantees that when we encounter an intersected edge, we know which adjacent face is closer to the destination.

3.3 FUTURE WORK

A formal analysis and comparison of the lengths of the paths produced by our algorithm from Chapter 2 and the lengths produced by PredAware is left as future work.

PredAware is a traversal algorithm for x-monotone subdivisions; it can be used to route indirectly since a message eventually visits every vertex. The algorithm presented in Chapter 2 on the other hand is more direct and just does point-to-point routing. This leads to the question: what are the capabilities and limits of predecessor information and state bits in a geometric setting?

A related area for future research is trying to tighten the bounds on the memory requirements for the classes of graphs between geometric plane convex subdivisions and general geometric plane graphs. For example, since a back-tracking-capable traversal is possible in any kind of undirected graph using predecessor-awareness and $\Theta(\log n)$ state bits [7], are predecessor-awareness and $\Omega(\log n)$ extra bits really

1 assuming general predecessor-awareness costs $\Omega(\log n)$ extra bits in the packet header

required for point-to-point routing in general undirected geometric plane graphs? What classes of geometric graphs can be routed using $O(1)$ extra bits or $o(\log n)$ (or $o(\log d)$) extra bits (predecessor-aware or predecessor-oblivious)? Is there some non-trivial combination of state bits, predecessor information, and geometric information that can be used to locally overcome problematic reflex vertices independent of the class of graphs?

Our algorithm is ultimately the result of an investigation seeking to discover necessary and sufficient information requirements to guarantee delivery in local geometric routing on a class of graphs more generic than convex subdivisions. Specifically, we were trying to find a parameterization of extra bits depending on the maximum number of reflex vertices in a face. We started with orthogonal subdivisions and then generalized the result, which is how the d direction constraint came about. We realized during the course of this investigation that we could actually have as many reflex vertices in a face as we wanted and be able to successfully route using information independent of the number of reflex vertices and not have to remember extra points during the face routing in Phase 2. We realized that the property we were depending on was monotonicity. Is there a way to parameterize the information required in terms of the maximum number of reflex vertices allowed in a face? Is predecessor-awareness required for successful local geometric routing in classes of graphs beyond convex subdivisions?

It may be worth investigating whether or not a variation of the algorithm from this work can be used for routing in a class of graphs beyond monotone subdivisions. Assume that we have some general subdivision divided into two half-planes by a vertical line through the destination. Assume that we only need to deliver the packet to an edge intersected by the vertical line, at which point it is someone else's problem (face routing would still work but probably requires storing additional points to avoid looping). Is there some property of closed polygons that can be exploited and parameterized such that we can locally find our way to the vertical line defining the half-plane? For example, if we consider bounding a face by two vertical lines at the left-most and right-most points, perhaps there is a way to keep track of the net sum of angles seen in a face in such a way as to know when the right-most vertical line is seen, at which point we want to move "forward" at the next possible opportunity. Of course, such a strategy is only useful if it can be successful with less information than it takes to face route.

On the other hand, Phase 2 of our algorithm is the phase that demands the most information for monotone subdivisions. Is there a different approach for Phase 2 that reduces the information required? Once we know the packet is at an edge intersected by a vertical line through the destination, can we follow this line without face routing?

An area for future research is an investigation into whether a slight modification of the algorithm of Chapter 2 can guarantee delivery in a graph that is not 2-connected (without simply performing face routing to the destination). Perhaps there is an interesting result to be had if there is some kind of constraint on the 1-connected subgraphs or if we allow more supplementary information (e.g., more state bits).

Our algorithm requires the nodes to have a consistent knowledge of the $\frac{d}{2}$ slopes in the graph, a somewhat unnatural constraint. Can we achieve a similar result without needing knowledge of slopes? What if that consistent slope constraint is replaced by the constraint of a bounded degree?

BIBLIOGRAPHY

[1] N. Bonichon, P. Bose, J.-L. De Carufel, L. Perković, and A. van Renssen. Upper and lower bounds for online routing on delaunay triangulations. In *Proc. ESA*, Lecture Notes in Computer Science, pages 203–214. Springer, 2015.

[2] P. Bose, A. Brodnik, S. Carlsson, E. D. Demaine, R. Fleischer, A. López-Ortiz, P. Morin, and J. I. Munro. Online routing in convex subdivisions. *International Journal of Computational Geometry and Applications*, 12(4):283–296, 2002. Special issue of selected papers from the *11th Annual International Symposium on Algorithms and Computation (ISAAC 2000)*.

[3] P. Bose, P. Carmi, and S. Durocher. Bounding the locality of distributed routing algorithms. *Distributed computing*, 26(1):39–58, 2013.

[4] P. Bose, S. Durocher, D. Mondal, M. Peabody, M. Skala, and M. A. Wahid. Local routing in convex subdivisions. In *Proc. SOFSEM*, volume 8939 of *Lecture Notes in Computer Science*, pages 140–151. Springer, 2015.

[5] P. Bose and P. Morin. Online routing in triangulations. *SIAM Journal on Computing*, 33(4):937–951, 2004. Preliminary version appears in *Proceedings of the Tenth International Symposium on Algorithms and Computation (ISAAC'99)*, pages 113–122, LNCS 1741, Springer-Verlag, 1999.

[6] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia. Routing with guaranteed delivery in *ad hoc* wireless networks. *Wireless Networks*, 7(6):609–616, 2001. Special issue of selected papers from the *3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM'99)*.

[7] M. Braverman. On ad hoc routing with guaranteed delivery. In *Proc. ACM PODC*, pages 418–418. ACM, 2008.

[8] G. G. Finn. Routing and addressing problems in large metropolitan-scale internetworks. Technical Report ISI/RR-87-180, Information Sciences Institute, 1987.

[9] H. Frey, S. Rührup, and I. Stojmenović. Routing in wireless sensor networks. In *Guide to Wireless Sensor Networks*, pages 81–111. Springer, 2009.

[10] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proc. CCCG*, pages 51–54, 1999.

[11] P. Morin. *Online Routing in Geometric Graphs*. PhD thesis, School of Computer Science, Carleton University, January 2001.

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

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of September 16, 2016 (`classicthesis` version 4.1).