SIAM J. COMPUT.
Vol. 44, No. 6, pp. 1626–1649

OPTIMAL LOCAL ROUTING ON DELAUNAY TRIANGULATIONS DEFINED BY EMPTY EQUILATERAL TRIANGLES*

PROSENJIT BOSE†, ROLF FAGERBERG‡, ANDRÉ VAN RENSSEN§¶, AND SANDER VERDONSCHOT†

Abstract. We present a deterministic local routing algorithm that is guaranteed to find a path between any pair of vertices in a half-θ_6-graph (the half-θ_6-graph is equivalent to the Delaunay triangulation where the empty region is an equilateral triangle). The length of the path is at most $5/\sqrt{3} \approx 2.887$ times the Euclidean distance between the pair of vertices. Moreover, we show that no local routing algorithm can achieve a better routing ratio, thereby proving that our routing algorithm is optimal. This is somewhat surprising because the spanning ratio of the half-θ_6-graph is 2, meaning that even though there always exists a path whose length is at most twice the Euclidean distance, we cannot always find such a path when routing locally. Since every triangulation can be embedded in the plane as a half-θ_6-graph using $O(\log n)$ bits per vertex coordinate via Schnyder's embedding scheme [W. Schnyder, *Embedding planar graphs on the grid*, in Proceedings of the 1st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 1990), ACM, New York, SIAM, Philadelphia, 1990, pp. 138–148], our result provides a competitive local routing algorithm for every such embedded triangulation. Finally, we show how our routing algorithm can be adapted to provide a routing ratio of $15/\sqrt{3} \approx 8.660$ on two bounded degree subgraphs of the half-θ_6-graph.

Key words. online routing, local routing, competitive routing, geometric spanner, theta-graph

AMS subject classifications. 52C99, 68M10, 68R10, 68W27, 90B18

DOI. 10.1137/140988103

1. Introduction.

A fundamental problem in networking is the routing of a message from one vertex to another in a graph. What makes routing more challenging is that often in a network the routing strategy must be *local*. Informally, a routing strategy is *local* when the routing algorithm must choose the next vertex to forward a message to based solely on knowledge of the source and destination vertex, the current vertex, and all vertices directly connected to the current vertex. Routing algorithms are considered *geometric* when the graph is embedded in the plane, with edges being straight line segments connecting pairs of points and weighted by the Euclidean distance between their endpoints. Geometric routing algorithms are important in wireless sensor networks (see [18] and [20] for surveys of the area) since they offer routing strategies that use the coordinates of the vertices to help guide the search as opposed to using the more traditional routing tables.

Papadimitriou and Ratajczak [19] posed a tantalizing question in this area that led to a flurry of activity: Does every 3-connected planar graph have a straight-line

*Received by the editors September 23, 2014; accepted for publication (in revised form) August 14, 2015; published electronically November 3, 2015. Extended abstracts containing the results in this paper appeared in SODA 2012 and CCCG 2012.

http://www.siam.org/journals/sicomp/44-6/98810.html

†School of Computer Science, Carleton University, Ottawa, ON K1S 5B6, Canada (jit@scs.carleton.ca, sander@cg.scs.carleton.ca). The research of these authors was supported in part by NSERC.

‡Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark (rolf@imada.sdu.dk). This author's research was partially supported by the Danish Council for Independent Research, Natural Sciences.

§National Institute of Informatics (NII), Tokyo, Japan (andre@nii.ac.jp). This author's research was partially supported by Carleton University's President's 2010 Doctoral Fellowship.

¶JST, ERATO, Kawarabayashi Large Graph Project.

embedding in the plane that admits a local routing strategy such as greedy routing?[1] They provided a partial answer by showing that 3-connected planar graphs can always be embedded in $\mathbb{R}^3$ such that they admit a greedy routing strategy. They also showed that the class of complete bipartite graphs, $K_{k,6k+1}$ for all $k \geq 1$, cannot be embedded such that greedy routing always succeeds since every embedding has at least one vertex that is not connected to its nearest neighbor. Bose and Morin [8] showed that greedy routing always succeeds on Delaunay triangulations. In fact, a slightly restricted greedy routing strategy known as *greedy-compass* is the first local routing strategy shown to succeed on all triangulations [6]. Dhandapani [11] proved the existence of an embedding that admits greedy routing for every triangulation, and Angelini, Frati, and Grilli [1] provided a constructive proof. Leighton and Moitra [17] settled Papadimitriou and Ratajczak's question by showing that every 3-connected planar graph can be embedded in the plane such that greedy routing succeeds. One drawback of these embedding algorithms is that the coordinates require $\Omega(n \log n)$ bits per vertex. To address this, He and Zhang [14] and Goodrich and Strash [13] gave succinct embeddings using only $O(\log n)$ bits per vertex. Recently, He and Zhang [15] showed that every 3-connected plane graph admits a succinct embedding with convex faces on which a slightly modified greedy routing strategy always succeeds.

In light of these recent successes, it is surprising to note that the above routing strategies solely concentrate on finding an embedding that guarantees that a local routing strategy will succeed but pay little attention to the quality of the resulting path. For example, none of the above routing strategies have been shown to be *competitive*. A geometric routing strategy is said to be competitive if the length of the path found by the routing strategy is not more than a constant times the Euclidean distance between its endpoints. This constant is called the *routing ratio*. Bose and Morin [8] show that many local routing strategies are not competitive but show how to route competitively on the Delaunay triangulation. However, Dillencourt [12] showed that not all triangulations can be embedded in the plane as Delaunay triangulations. This raises the following question: can *every* triangulation be embedded in the plane such that it admits a competitive local routing strategy? We answer this question in the affirmative.

The half-θ_6-graph was introduced by Bonichon et al. [4], who showed that it is identical to the Delaunay triangulation where the empty region is an equilateral triangle. Although both graphs are identical, the local definition of the half-θ_6-graph makes it more useful in the context of routing. We formally define the half-θ_6-graph in the next section. Our main result is a deterministic local routing algorithm that is guaranteed to find a path between any pair of vertices in a half-θ_6-graph whose length is at most $5/\sqrt{3}$ times the Euclidean distance between the pair of vertices. On the way to proving our main result, we uncover some local properties of spanning paths in the half-θ_6-graph. Since Schnyder [21] showed that every triangulation can be embedded in the plane as a half-θ_6-graph using $O(\log n)$ bits per vertex coordinate, our main result implies that every triangulation has an embedding that admits a competitive local routing algorithm. Moreover, we show that no local routing algorithm can achieve a better routing ratio on a half-θ_6-graph, implying that our routing algorithm is optimal. This is somewhat surprising because Chew [10] showed that the spanning ratio of the half-θ_6-graph is 2. Thus, our lower bound provides a separation between

[1] A routing strategy is greedy when a message is always forwarded to the vertex whose distance to the destination is the smallest among all vertices in the neighborhood of the current vertex, including the current vertex.

the spanning ratio of the half-θ_6-graph and the best achievable routing ratio on the half-θ_6-graph. We believe that this is the first separation between the spanning ratio and routing ratio of any graph. It also makes the half-θ_6-graph one of the few graphs for which tight spanning and routing ratios are known. Finally, we show how our routing algorithm can be adapted to provide a routing ratio of $15/\sqrt{3}$ on two bounded degree subgraphs of the half-θ_6-graph introduced by Bonichon et al. [5]. To the best of our knowledge, this is the first competitive routing algorithm on a bounded-degree plane graph.

2. Preliminaries. In order to find a competitive path between any two vertices of a graph, such a path must first exist. Graphs that meet this criterion are called *spanners*. Formally, given a weighted graph G, we define the distance $d_G(u, v)$ between two vertices u and v to be the sum of the weights of the edges in the shortest path between u and v in G. A subgraph H of G is a *t-spanner* of G if for all pairs of vertices u and v, $d_H(u, v) \leq t \cdot d_G(u, v)$ for $t \geq 1$. We say that H is a *spanner* if it is a t-spanner for some constant t. The *spanning ratio* of H is the smallest t for which it is a t-spanner. The graph G is referred to as the *underlying graph*.

Unless otherwise noted, we assume that the underlying graph G is a straight-line embedding of the complete graph on a set of n points in the plane, with the weight of an edge (u, v) being the Euclidean distance $|uv|$ between u and v. A spanner of such a graph is called a *geometric spanner*. We focus on one specific class of geometric spanners: the half-θ_6-graph. In a slight abuse of notation, we often speak about the spanning ratio of the half-θ_6-graph. By this, we mean the maximum spanning ratio of any half-θ_6-graph on any set of n points in the plane. In the remainder of this section, we describe the construction of the half-θ_6-graph and introduce some notation.

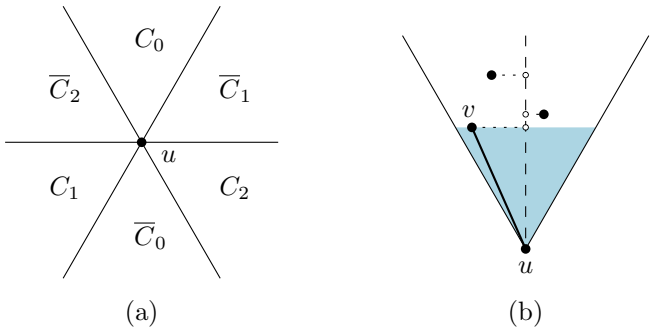

FIG. 1. (a) *The cones around a vertex u.* (b) *The construction of the half-θ_6-graph. In each positive cone, u connects to the vertex with the closest projection on the bisector of that cone.*

Given a set P of points in the plane, we consider each point $u \in P$ and partition the plane into six cones (regions in the plane between two rays originating from the same point) with apex u, each defined by two rays at consecutive multiples of $\pi/3$ radians from the positive x-axis. We label the cones $\overline{C}_1$, C_0, $\overline{C}_2$, C_1, $\overline{C}_0$, and C_2 in counterclockwise order around u, starting from the positive x-axis (see Figure 1a). The cones C_0, C_1, and C_2 are called *positive*, while the others are called *negative*. When the apex is not clear from the context, we use C_i^u to denote cone C_i with apex u.

To build the half-θ_6-graph, we consider each vertex u and add an edge to the "closest" vertex in each of its positive cones. However, instead of using the Euclidean distance, we measure distance by projecting each vertex in the cone onto the bisector

of the cone. We call the vertex in this cone whose projection is closest to u the *closest vertex* and connect it to u with an edge (see Figure 1b). For simplicity, we assume that no two points lie on a line parallel to a cone boundary, guaranteeing that each vertex connects to exactly one vertex in each positive cone. Hence the graph has at most $3n$ edges in total.

Given two vertices u and v such that v lies in a positive cone of u, we define their *canonical triangle* T_{uv} to be the triangle bounded by the cone of u that contains v and the line through v perpendicular to the bisector of that cone. For example, the shaded region in Figure 1b is the canonical triangle of u and v. Note that for any pair of vertices u and v, either v lies in a positive cone of u, or u lies in a positive cone of v, so there is exactly one canonical triangle (either T_{uv} or T_{vu}) for the pair. The construction of the half-θ_6-graph can alternatively be described as adding an edge between two vertices if and only if their canonical triangle is empty. This property will play an important role in our proofs.

3. Spanning ratio of the half-θ_6-graph. Bonichon et al. [4] showed that the half-θ_6-graph is a geometric spanner with spanning ratio 2 by showing that it is equivalent to the Delaunay triangulation based on empty equilateral triangles, which is known to have spanning ratio 2 [10]. One direction of this equivalence is easy to see, since, by construction, every edge of the half-θ_6-graph has an empty equilateral triangle. For the other direction, recall that every triangle in the Delaunay triangulation has its vertices on the boundary of an empty equilateral triangle in a fixed orientation. Bonichon et al. showed that for each edge of the Delaunay triangulation, one can shrink the empty equilateral triangle such that one of the endpoints of the edge lies on a corner and the other lies on the boundary, thereby proving it is an edge of the half-θ_6-graph. This correspondence also shows that the half-θ_6-graph is internally triangulated: every face except for the outer face is a triangle (this follows from the duality with the Voronoi diagram, along with the fact that all vertices in the Voronoi diagram have degree 3, provided that no four points lie on the same equilateral triangle).

In this section, we provide an alternative proof of the spanning ratio of the half-θ_6-graph. Our proof shows that between any pair of points, there always exists a path with spanning ratio 2 that lies in the canonical triangle. This property plays an important role in our routing algorithm, which we describe in section 5. For a pair of vertices u and w, our bound is expressed in terms of the angle α between the line from u to w and the bisector of their canonical triangle (see Figure 2a).

THEOREM 1. *Let u and w be vertices with w in a positive cone of u. Let m be the midpoint of the side of T_{uw} opposing u, and let α be the unsigned angle between uw and um. There exists a path between u and w in the half-θ_6-graph, of length at most*

$$(\sqrt{3} \cdot \cos\alpha + \sin\alpha) \cdot |uw|,$$

where all vertices on this path lie in T_{uw}.

The expression $\sqrt{3} \cdot \cos\alpha + \sin\alpha$ is increasing for $\alpha \in [0, \pi/6]$. Inserting the extreme value $\pi/6$ for α, we arrive at the following.

COROLLARY 2. *The spanning ratio of the half-θ_6-graph is* 2.

We note that the bounds of Theorem 1 and Corollary 2 are tight: for all values of $\alpha \in [0, \pi/6]$ there exists a point set for which the shortest path in the half-θ_6-graph for some pair of vertices u and w has length arbitrarily close to $(\sqrt{3} \cdot \cos\alpha + \sin\alpha) \cdot |uw|$. A simple example appears later in the proof of Theorem 4.

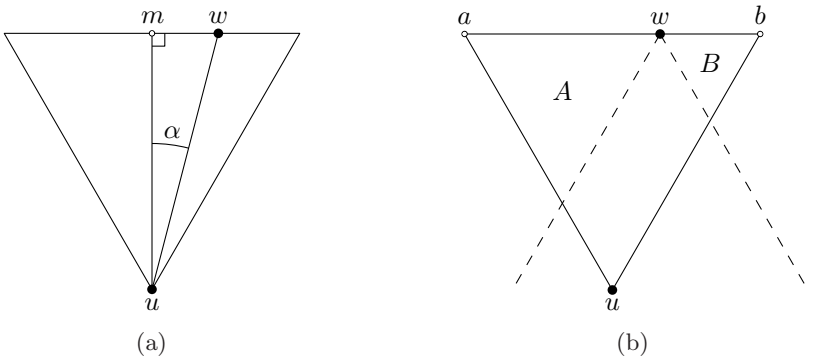

FIG. 2. (a) *Two vertices u and w with their canonical triangle T_{uw}. The angle α is the unsigned angle between the line uw and the bisector of the cone containing w.* (b) *The corners a and b, and the regions A and B.*

Proof of Theorem 1. Given two vertices u and w, we assume without loss of generality that w lies in C_0^u. We prove the theorem by induction on the rank, when ordered by area, of the triangles T_{xy} for all pairs of points x and y where y lies in a positive cone of x. Let a and b be the upper left and right corners of T_{uw}, and let $A = T_{uw} \cap C_1^w$ and $B = T_{uw} \cap C_2^w$, as illustrated in Figure 2b.

Our inductive hypothesis is the following, where $\delta(u, w)$ denotes the length of the shortest path from u to w in the part of the half-θ_6-graph induced by the vertices in T_{uw}.

1. If A is empty, then $\delta(u, w) \leq |ub| + |bw|$.
2. If B is empty, then $\delta(u, w) \leq |ua| + |aw|$.
3. If neither A nor B is empty, then $\delta(u, w) \leq \max\{|ua| + |aw|, |ub| + |bw|\}$.

We first note that this induction hypothesis implies Theorem 1: using the side of T_{uw} as the unit of length, we have from Figure 2a that $|wm| = |uw| \cdot \sin\alpha$ and $\sqrt{3}/2 = |um| = |uw| \cdot \cos\alpha$. Hence the induction hypothesis gives us that $\delta(u, w)$ is at most $1 + 1/2 + |wm| = \sqrt{3} \cdot (\sqrt{3}/2) + |wm| = (\sqrt{3} \cdot \cos\alpha + \sin\alpha) \cdot |uw|$, as required.

Base case. T_{uw} has rank 1. Since there are no smaller canonical triangles, w must be the closest vertex to u. Hence the edge (u, w) is in the half-θ_6-graph, and $\delta(u, w) = |uw|$. Using the triangle inequality, we have $|uw| \leq \min\{|ua| + |aw|, |ub| + |bw|\}$, so the induction hypothesis holds.

Induction step. We assume that the induction hypothesis holds for all pairs of points with canonical triangles of rank up to i. Let T_{uw} be a canonical triangle of rank $i + 1$.

If (u, w) is an edge in the half-θ_6-graph, the induction hypothesis follows by the same argument as in the base case. If there is no edge between u and w, let v be the vertex closest to u in the positive cone C_0^u, and let a' and b' be the upper left and right corners of T_{uv}. By definition, $\delta(u, w) \leq |uv| + \delta(v, w)$, and by the triangle inequality, $|uv| \leq \min\{|ua'| + |a'v|, |ub'| + |b'v|\}$.

We perform a case distinction on the location of v: a. v lies neither in A nor in B. b. v lies inside A. c. v lies inside B. The case where v lies inside B is analogous to the case where v lies inside A, so we only discuss the first two cases, which are illustrated in Figure 3.

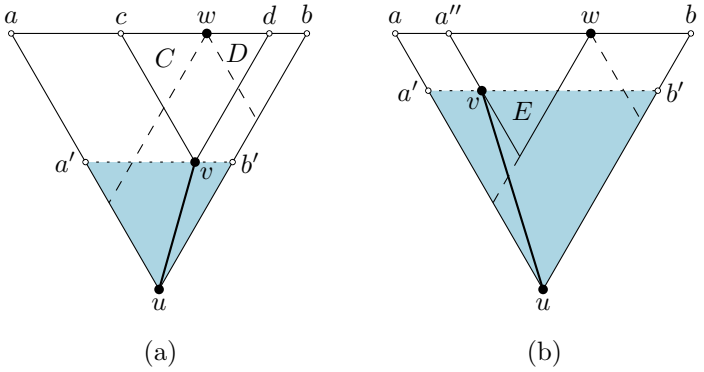

FIG. 3. *The two cases:* (a) *v lies in neither A nor B;* (b) *v lies in A.*

Case a. Let c and d be the upper left and right corners of T_{vw}, and let $C = T_{vw} \cap C_1^w$ and $D = T_{vw} \cap C_2^w$ (see Figure 3a). Since T_{vw} has smaller area than T_{uw}, we apply the inductive hypothesis on T_{vw}. Our task is to prove all three statements of the inductive hypothesis for T_{uw}.

1. If A is empty, then C is also empty, so by induction $\delta(v, w) \leq |vd| + |dw|$. Since v, d, b, and b' form a parallelogram, we have

$$\begin{aligned}\delta(u, w) &\leq |uv| + \delta(v, w)\\ &\leq |ub'| + |b'v| + |vd| + |dw|\\ &= |ub| + |bw|,\end{aligned}$$

which proves the first statement of the induction hypothesis. This argument is illustrated in Figure 4a.

2. If B is empty, an analogous argument proves the second statement of the induction hypothesis.

3. If neither A nor B is empty, by induction we have $\delta(v, w) \leq \max\{|vc| + |cw|, |vd| + |dw|\}$. Assume, without loss of generality, that the maximum of the right-hand side is attained by its second argument $|vd| + |dw|$ (the other case is analogous). Since vertices v, d, b, and b' form a parallelogram, we have that

$$\begin{aligned}\delta(u, w) &\leq |uv| + \delta(v, w)\\ &\leq |ub'| + |b'v| + |vd| + |dw|\\ &\leq |ub| + |bw|\\ &\leq \max\{|ua| + |aw|, |ub| + |bw|\},\end{aligned}$$

which proves the third statement of the induction hypothesis. This argument is illustrated in Figure 4b.

Case b. Let $E = T_{uv} \cap T_{wv}$, and let a'' be the upper left corner of T_{wv} (see Figure 3b). Since v is the closest vertex to u in one of its positive cones, T_{uv} is empty, and hence E is also empty. Since T_{wv} is smaller than T_{uw}, we can apply induction on it. As E is empty, the first statement of the induction hypothesis for T_{wv} applies, giving us that $\delta(v, w) \leq |va''| + |a''w|$. Since $|uv| \leq |ua'| + |a'v|$ and v, a'', a, and a' form a parallelogram, we have that $\delta(u, w) \leq |ua| + |aw|$, proving the second and third statements in the induction hypothesis for T_{uw}. This argument is illustrated in Figure 4c. Since v lies in A, the first statement in the induction hypothesis for T_{uw} is vacuously true. ∎

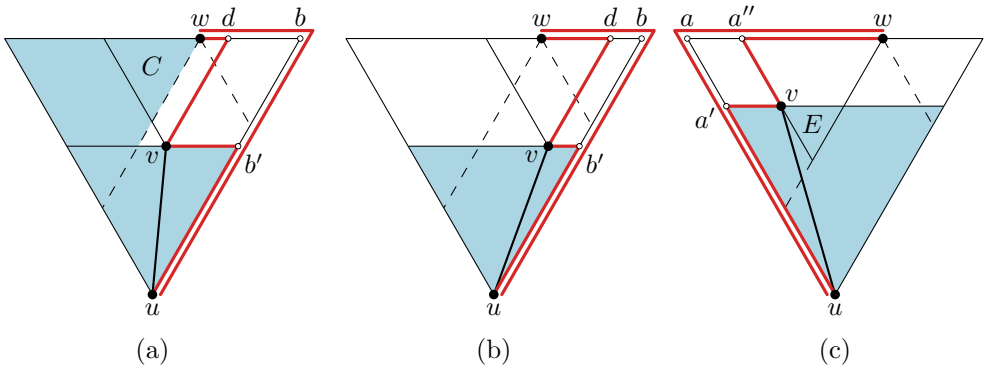

FIG. 4. *Visualization of the path inequalities in three cases:* (a) *v lies in neither A nor B and one of A or B is empty (cases a.1 and a.2 in our proof);* (b) *v lies in neither A nor B and neither is empty (case a.3);* (c) *v lies in A or B (case b). The paths occurring in the equations are drawn with thick lines, and shaded areas indicate empty regions.*

4. Remarks on the spanning ratio. The full-θ_6-graph, introduced by Keil and Gutwin [16], is similar to the half-θ_6-graph except that all six cones are positive cones. Thus, the full-θ_6-graph is the union of two copies of the half-θ_6-graph, where one half-θ_6-graph is rotated by $\pi/3$ radians. The half-θ_6-graph and the full-θ_6-graph both have a spanning ratio of 2, with lower bound examples showing that it is tight for both graphs. This is surprising since the full-θ_6-graph can have twice the number of edges of the half-θ_6-graph.

Note that since the full-θ_6-graph consists of two rotated copies of the half-θ_6-graph, one question that comes to mind is "What is the best spanning ratio if one is to construct a graph consisting of two rotated copies of the half-θ_6-graph?" Can one do better than a spanning ratio of 2? Consider the following construction. Build two half-θ_6-graphs as described in section 2, but rotate each cone of the second graph by $\pi/6$ radians. For each pair of vertices, there is a path of length at most $\sqrt{3}\cos\alpha+\sin\alpha$ times the Euclidean distance between them, where α is the angle between the line connecting the vertices in question, and the closest bisector. Since this function is increasing, the spanning ratio is defined by the maximum possible angle to the closest bisector, which is $\pi/12$ radians, giving a spanning ratio of roughly 1.932.

By using k copies, we improve the spanning ratio even further: if each is rotated by $\pi/(3k)$ radians, we get a spanning ratio of $\sqrt{3}\cos\frac{\pi}{6k}+\sin\frac{\pi}{6k}$. This is better than the known upper bounds for the full-θ_{3k}-graph [9] for $k \leq 3$ and for the Yao$_{3k}$-graph [3] for $k \leq 4$.

COROLLARY 3. *The union of k copies of the half-θ_6-graph, each rotated by $\pi/(3k)$ radians, is a geometric spanner with up to $3k$ edges and spanning ratio at most $\sqrt{3}\cos\frac{\pi}{6k}+\sin\frac{\pi}{6k}$.*

5. Routing in the half-θ_6-graph. In this section, we give matching upper and lower bounds for the routing ratio on the half-θ_6-graph. We begin by defining our model. Formally, a routing algorithm A is a deterministic k-local, m-memory routing algorithm if the vertex to which a message is forwarded from the current vertex s is a function of s, t, $N_k(s)$, and M, where t is the destination vertex, $N_k(s)$ is the k-neighborhood of s, and M is a memory of size m, stored with the message. The k-neighborhood of a vertex s is the set of vertices in the graph that can be reached from s by following at most k edges. For our purposes, we consider a unit of memory

to consist of a $\log_2 n$ bit integer or a point in $\mathbb{R}^2$. Our model also assumes that the only information stored at each vertex of the graph is $N_k(s)$. Since our graphs are geometric, we identify each vertex by its coordinates in the plane. A routing algorithm is *d-competitive* provided that the total distance traveled by the message is never more than d times the Euclidean distance between source and destination. Analogous to the spanning ratio, the *routing ratio* of an algorithm is the smallest d for which it is d-competitive.

We present a deterministic 1-local 0-memory algorithm that achieves the upper bounds, but our lower bounds hold for any deterministic k-local 0-memory algorithm, provided k is a constant. Our bounds are expressed in terms of the angle α between the line from the source to the destination and the bisector of their canonical triangle (see Figure 2a).

THEOREM 4. *Let u and w be two vertices, with w in a positive cone of u. Let m be the midpoint of the side of T_{uw} opposing u, and let α be the unsigned angle between uw and um. There is a deterministic 1-local 0-memory routing algorithm on the half-θ_6-graph for which every path followed has length at most*

(i) $(\sqrt{3} \cdot \cos\alpha + \sin\alpha) \cdot |uw|$ *when routing from u to w,*

(ii) $(5/\sqrt{3} \cdot \cos\alpha - \sin\alpha) \cdot |uw|$ *when routing from w to u,*

and this is the best possible for deterministic k-local, 0-memory routing algorithms, where k is constant.

The first expression is increasing for $\alpha \in [0, \pi/6]$, while the second expression is decreasing. Inserting the extreme values $\pi/6$ and 0 for α, we get the following worst-case version of Theorem 4.

COROLLARY 5. *Let u and w be two vertices, with w in a positive cone of u. There is a deterministic 1-local 0-memory routing algorithm on the half-θ_6-graph with routing ratio*

(i) 2 *when routing from u to w,*

(ii) $5/\sqrt{3} = 2.886\ldots$ *when routing from w to u,*

and this is the best possible for deterministic k-local, 0-memory routing algorithms, where k is constant.

Since the spanning ratio of the half-θ_6-graph is 2, the second lower bound shows a separation between the spanning ratio and the best possible routing ratio in the half-θ_6-graph.

Since every triangulation can be embedded in the plane as a half-θ_6-graph using $O(\log n)$ bits per vertex via Schnyder's embedding scheme [21], an important implication of Theorem 4 is the following.

COROLLARY 6. *Every n-vertex triangulation can be embedded in the plane using $O(\log n)$ bits per coordinate such that the embedded triangulation admits a deterministic 1-local 0-memory routing algorithm with routing ratio at most $5/\sqrt{3}$.*

In the remainder of this section, we prove Theorem 4. We split the proof into two cases, depending on whether the destination lies in a positive (section 5.1) or negative (section 5.2) cone of the source. In each case, we present first a proof of the lower bound, then a description of the routing algorithm, and finally a proof of the upper bound.

5.1. Positive routing.

LEMMA 7 (lower bound for positive routing). *Let u and w be two vertices, with w in a positive cone of u. Let m be the midpoint of the side of T_{uw} opposing u, and let α be the unsigned angle between uw and um. For any routing algorithm, there are*

instances for which the path followed has length at least $(\sqrt{3}\cdot\cos\alpha+\sin\alpha)\cdot|uw|$ *when routing from* u *to* w.

Proof. Let the side of T_{uw} be the unit of length. From Figure 2a, we have $|wm| = |uw|\cdot\sin\alpha$ and $\sqrt{3}/2 = |um| = |uw|\cdot\cos\alpha$. From Figure 5, the spanning ratio of the half-θ_6-graph is at least $1+1/2+|wm| = \sqrt{3}\cdot(\sqrt{3}/2)+|wm| = (\sqrt{3}\cdot\cos\alpha+\sin\alpha)\cdot|uw|$, since the point in the upper left corner of T_{uw} can be moved arbitrarily close to the corner. As there is no shorter path between u and w, this is a lower bound for *any* routing algorithm. □

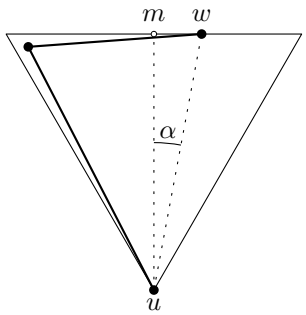

FIG. 5. *The lower bound example when routing to a vertex in a positive cone.*

Routing algorithm. While routing, let s denote the current vertex, and let t denote the fixed destination (i.e., t corresponds to w in Theorem 4). To be deterministic, 1-local, and 0-memory, the routing algorithm needs to determine which edge (s, v) to follow next based only on s, t, and the neighbors of s. We say we are *routing positively* when t is in a positive cone of s and *routing negatively* when t is in a negative cone. (Note the distinction between "positive routing" and "routing positively": the first describes the conditions *at the start* of the routing process, while the second does so *during* the routing process. In other words, positive routing describes a routing process that starts by routing positively. It is very common for positive routing to include situations where we are routing negatively; see, e.g., Figure 8b.)

For ease of description, we assume without loss of generality that t is in cone C_0^s when routing positively and in cone $\overline{C}_0^s$ when routing negatively. When routing positively, T_{st} intersects only C_0^s among the cones of s. When routing negatively, T_{ts} intersects $\overline{C}_0^s$, as well as the two positive cones C_1^s and C_2^s. Let $X_0 = \overline{C}_0^s \cap T_{ts}$, $X_1 = C_1^s \cap T_{ts}$, and $X_2 = C_2^s \cap T_{ts}$. Let a be the corner of T_{ts} contained in X_1 and b the corner of T_{ts} contained in X_2. These definitions are illustrated in Figure 6.

The routing algorithm will only follow edges (s, v) where v lies in the canonical triangle of s and t. Routing positively is straightforward since there is exactly one edge (s, v) with $v \in T_{st}$, by the construction of the half-θ_6-graph. The challenge is to route negatively. When routing negatively, at least one edge (s, v) with $v \in T_{ts}$ exists, since, by Theorem 1, s and t are connected by a path inside T_{ts}. The core of our routing algorithm is how to choose which edge to follow when there is more than one. Intuitively, when routing negatively, our algorithm tries to select an edge that makes measurable progress toward the destination. When no such edge exists, we are forced to take an edge that does not make measurable progress; however, we are able to then deduce that certain regions within the canonical triangle are empty. This allows us to bound the total distance traveled while not making measurable progress. We provide a formal description of our routing algorithm below.

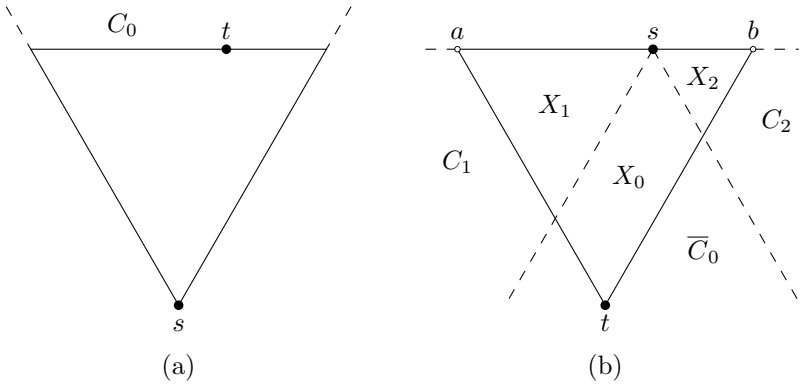

FIG. 6. *Routing terminology when* (a) *routing positively and* (b) *routing negatively.*

Our routing algorithm can be in one of four cases. We call the situation when routing positively case A, and divide the situation when routing negatively into three further cases: both X_1 and X_2 are empty (case B), either X_1 or X_2 is empty (case C), or neither is empty (case D). Since X_1 and X_2 correspond to positive cones of s, each contains the endpoint of at most one edge (s, v). These edges contain a lot of information about the regions X_1 and X_2. In particular, if there is no edge in the corresponding cone, then the entire cone must be empty. And if there is an edge, but its endpoint lies outside of the region, the region is guaranteed to be empty. This allows our algorithm to *locally* determine if X_1 and X_2 are empty, and therefore which case we are in.

Since we are routing to a destination in a positive cone of the source, our routing algorithm starts in case A. Routing in this case is straightforward, as there is only one edge (s, v) with v in T_{ts} that we can follow. We now turn our attention to routing in cases B and C (it turns out that case D never occurs when routing to a destination in a positive cone of the source; we come back to it when describing negative routing in section 5.2).

In case B, both X_1 and X_2 are empty, so there must be edges (s, v) with $v \in X_0$, as s and t are connected by a path in T_{ts} by Theorem 1. If $|as| \geq |sb|$, the routing algorithm follows the last edge in clockwise order around s; if $|as| < |sb|$, it follows the first edge. In short, when both sides of T_{ts} are empty, the routing algorithm favors staying close to the largest empty side of T_{ts}. Note that $|as|$ and $|sb|$ can be computed locally from the coordinates of s and t.

In case C, exactly one of X_1 or X_2 is empty. If there exist edges (s, v) with $v \in X_0$, the routing algorithm will follow one of these, choosing among them in the following way: If X_1 is empty, it chooses the last edge in clockwise order around s. Else X_2 is empty, and it chooses the first edge in clockwise order around s. In short, the routing algorithm favors staying close to the empty side of T_{ts}. If no edges (s, v) with $v \in X_0$ exist, the routing algorithm follows the single edge (s, v) with v in X_1 or X_2.

Upper bound. The proof of the upper bound uses a potential function ϕ, defined as follows for each of the cases A, B, and C. For the potential in case C, $x \in \{a, b\}$ is the corner contained in the nonempty one of the two areas X_1 and X_2.

Case A: $\phi = |sa| + \max(|at|, |tb|)$
Case B: $\phi = |ta| + \min(|as|, |sb|)$
Case C: $\phi = |ta| + |sx|$

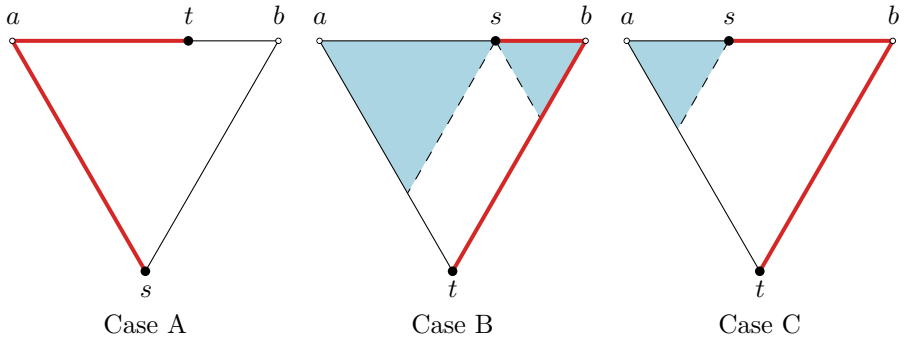

FIG. 7. *The potential ϕ in each case. Thick lines designate potential, and shading designates empty areas.*

This definition is illustrated in Figure 7. We will refer to the first term of ϕ (i.e., $|sa|$ in case A and $|ta|$ in cases B and C) as the *vertical part* of ϕ and to the rest as the *horizontal part*. Note that since all sides of the canonical triangle have equal length, a and b are interchangeable in the vertical part. The proof makes extensive use of the following observation about equilateral triangles.

OBSERVATION 1. *In an equilateral triangle, the diameter (the longest distance defined by any two points in the triangle) is equal to the side length.*

Our aim is to prove the following claim: for any routing step, the reduction in ϕ is at least as large as the length of the edge followed. This allows us to "pay" for each edge with the difference in potential, thereby bounding the total length of the path by the initial potential. We do this by case analysis of the possible routing steps.

Case A. For a routing step starting in case A, v can be in a negative or a positive cone of t. The first situation leads to case A again. The second leads to case B or C, since the area of T_{st} between s and v must be empty by construction of the half-θ_6-graph. These situations are illustrated in Figure 8.

If we remain in case A after following edge (s, v), the reduction of the vertical part of ϕ (dashed in Figure 8a) is at least as large as $|sv|$ by Observation 1. Therefore we can use it to pay for this step. Since T_{vt} is contained in T_{st}, both $|at|$ and $|bt|$ decrease. Thus the horizontal part of ϕ decreases, too, as it is the maximum of the two. Hence the claim holds for this situation.

For the situation ending in case C (the second illustration after the arrow in Figure 8b), we again use the reduction of the vertical part of ϕ to pay for the step. The rest of the vertical part precisely covers the new horizontal part. Since T_{tv} is contained in T_{st}, the new vertical part is a portion of either ta or tb. This can be covered by the current horizontal part, as it is the maximum of $|ta|$ and $|tb|$. Thus the claim holds for this situation as well. Finally, for the situation ending in case B, the final value of ϕ is at most that of the situation ending in case C, so again the claim holds.

Case B. A routing step starting in case B (illustrated in Figure 9) cannot lead to case A, as the step stays within T_{ts}. We first show that it always results in Case B or C, meaning that at least one of X_1 or X_2 is empty again. The algorithm follows an edge (s, v) with $v \in X_0$. If s is to the left of t, it follows the first edge in clockwise

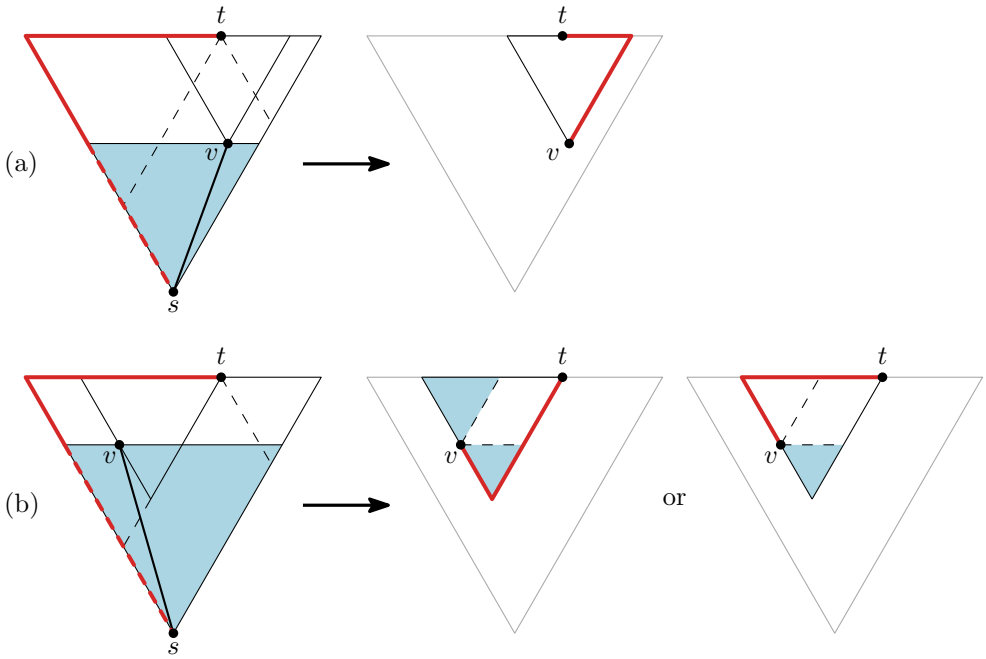

FIG. 8. *Routing in case A.* (a) v *lies in a negative cone of* t*;* (b) v *lies in a positive cone of* t*. Thick dashed lines indicate which parts of the potential are used to pay for the edge.*

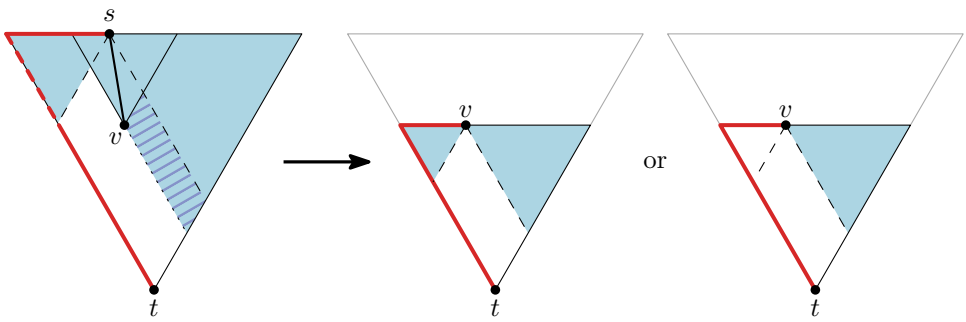

FIG. 9. *Routing in case B.*

order around s; otherwise it follows the last one. We consider only the case where s is to the left of t; the other case is symmetric. By the construction of the half-θ_6-graph, the existence of the edge (s, v) implies that T_{vs} is empty. It follows that the hatched area in Figure 9 is also empty: if not, the topmost point in it would have an edge to s, while coming before v in the clockwise order around s, contradicting the choice of v by the routing algorithm. Therefore X_2 will again be empty, resulting in case B or C.

By Observation 1, the reduction in the vertical part of ϕ is at least as large as $|sv|$. In addition, the horizontal part of ϕ can only decrease. If it remains on the same side of the triangle, this follows from the fact that v lies in X_0 and T_{tv} is contained in T_{ts}. And the only case where the potential switches sides is when we end up in case B again but the other side is shorter than the current one, reducing the potential even further. Hence the claim holds.

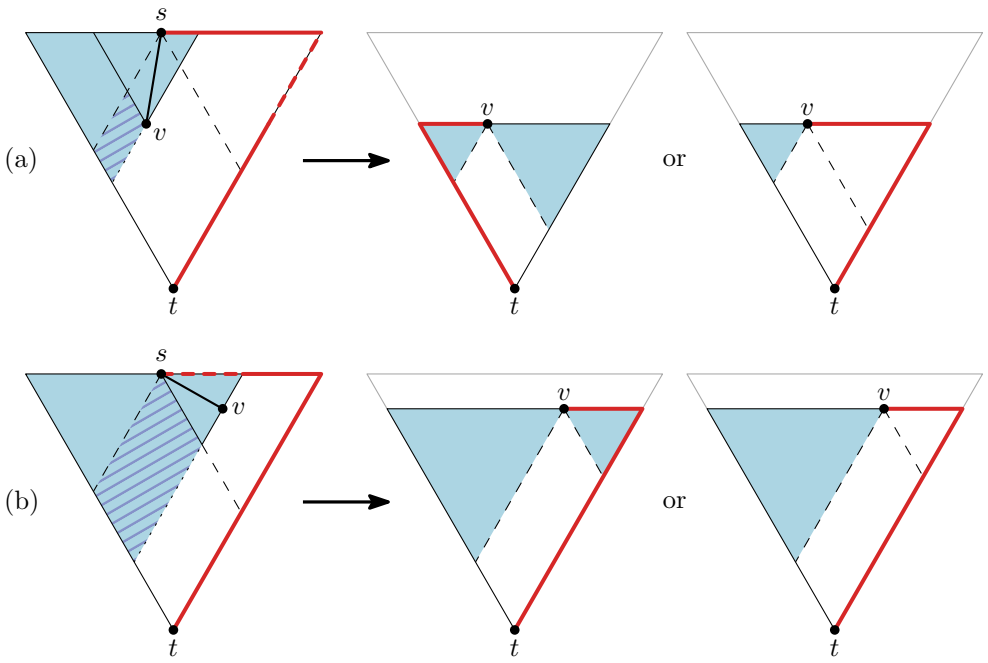

FIG. 10. *Routing in case C.*

Case C. As in the previous case, a routing step starting in case C cannot lead to case A, and we show that it cannot lead to case D, either. There are two situations, depending on whether edges (s, v) with $v \in X_0$ exist. For the situation where such edges do exist (illustrated in Figure 10a), the analysis is exactly the same as for a routing step starting in case B.

For the situation where edges (s, v) with $v \in X_0$ do not exist, the start of the step is illustrated on the left of the arrow in Figure 10b. Again, T_{sv} must be empty by the construction of the half-θ_6-graph, which implies that the hatched area must also be empty: if not, the topmost point in it would have an edge to s, contradicting that edges (s, v) with $v \in X_0$ do not exist. Thus, the routing step can only lead to case B or C. Looking at the potential, the vertical part can only decrease, and by Observation 1, the reduction of the horizontal part of ϕ is at least as large as $|sv|$. Thus we can pay for this step as well, and the claim holds in both situations.

LEMMA 8 (upper bound for positive routing). *Let u and w be two vertices, with w in a positive cone of u. Let m be the midpoint of the side of T_{uw} opposing u, and let α be the unsigned angle between uw and um. There is a deterministic 1-local 0-memory routing algorithm on the half-θ_6-graph for which every path followed has length at most $(\sqrt{3} \cdot \cos\alpha + \sin\alpha) \cdot |uw|$ when routing from u to w.*

Proof. That the algorithm is deterministic, 1-local, and 0-memory follows from the description of the algorithm, so we only need to prove the bound on the distance. We showed that for any routing step, the reduction in ϕ is at least as large as the length of the edge followed. Since ϕ is always nonnegative, this implies that no path followed can be longer than the initial value of ϕ. As all edges have strictly positive length, the routing algorithm must terminate. Since we are routing to a vertex in a positive cone, we start in case A, with an initial potential of $|ua| + \max(|aw|, |wb|)$. Taking the side of T_{uw} as the unit of length reduces this to $1 + 1/2 + |wm|$, and using the same

analysis as in Lemma 7, we obtain the desired bound of $(\sqrt{3}\cdot\cos\alpha+\sin\alpha)\cdot|uw|$. ∎

5.2. Negative routing. Next we turn our attention to the case when we are routing to a destination in a negative cone of the source. We start by deriving a lower bound, and then we present the required extensions to our routing algorithm and finish with the matching upper bound.

LEMMA 9 (lower bound for negative routing). *Let u and w be two vertices, with w in a positive cone of u. Let m be the midpoint of the side of T_{uw} opposing u, let α be the unsigned angle between uw and um, and let k be a constant. For any deterministic k-local 0-memory routing algorithm, there are instances for which the path followed has length at least $(5/\sqrt{3}\cdot\cos\alpha-\sin\alpha)\cdot|uw|$ when routing from w to u.*

Proof. Consider the two instances in Figure 11. Any deterministic 1-local 0-memory routing algorithm has information about direct neighbors only. Hence, it cannot distinguish between the two instances when routing out of w. This means that it routes to the same neighbor of w in both instances, and either choice of neighbor leads to a nonoptimal route in one of the two instances. The smallest loss occurs when the choice is toward the closest corner of T_{uw}, for which Figure 11a is the bad instance. If we let the side of T_{uw} be the unit of length, this gives a lower bound of $(1/2-|wm|)+1+1=5/2-|wm|$, since the points in the corners of T_{uw} can be moved arbitrarily close to the corners while keeping their relative positions. Using that $|wm|=|uw|\cdot\sin\alpha$ and $\sqrt{3}/2=|um|=|uw|\cdot\cos\alpha$, the lower bound reduces to $(5/\sqrt{3}\cdot\cos\alpha-\sin\alpha)\cdot|uw|$. By appropriately adding $\Omega(k)$ points close to the corners such that u is not in the k-neighborhood of w, the lower bound holds for any deterministic k-local 0-memory routing algorithm. ∎

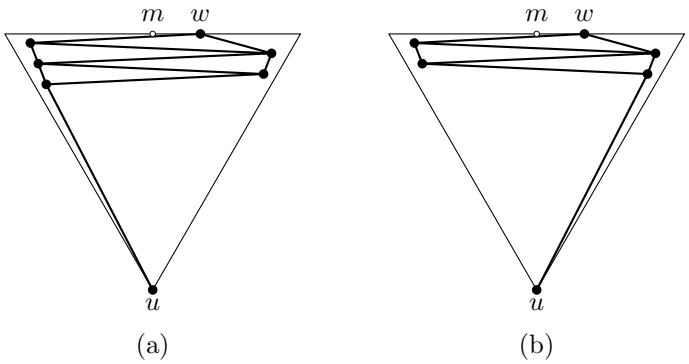

FIG. 11. *The lower bound instances for routing to a vertex in a negative cone.*

Routing algorithm. The only difference with the routing algorithm we used for positive routing lies in the initial case. Since our destination is in a negative cone, we start in one of the negative cases. This time, in addition to cases B and C, where both or one of X_1 and X_2 are empty, we also need case D, where neither is empty. Recall that in the previous section, we showed that a routing step starting in case A, B, or C can never result in case D. Thus, if the routing process starts in case D, it never returns there once it enters case A, B, or C.

In case D, the routing algorithm first tries to follow an edge (s,v) with $v\in X_0$. If several such edges exist, an arbitrary one of these is followed. If no such edge exists, the routing algorithm follows the single edge (s,v) with v in the smaller of X_1 and X_2. In short, the routing algorithm favors moving toward the closest corner of T_{ts} when

it is not able to move toward t. Note that, in the instances of Figure 11, this choice ensures that the first routing step incurs the smallest loss in the worst case, making it possible to meet the lower bound of Lemma 9. We now show that our algorithm achieves this lower bound in all cases.

Upper bound. The potential in case D is given in Figure 12. It mirrors the lower bound path in that it allows walking toward the closest corner, crossing the triangle, and then walking down to t. This is the highest potential among the four cases.

$$\text{Case D:} \quad \phi = |ta| + |ab| + \min(|as|, |sb|)$$

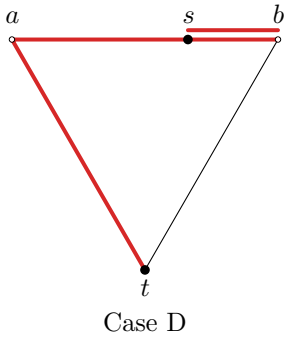

FIG. 12. *The potential ϕ in case D.*

As before, we want to show that for any routing step, the reduction in ϕ is at least as large as the length of the edge followed. Since we already did this for cases A, B, and C, all that is left is to prove it for case D.

Case D. A routing step starting in case D cannot lead to case A, as the step stays within T_{ts}, but it may lead to case B, C, or D. There are two situations, depending on whether edges (s, v) with $v \in X_0$ exist or not. These are illustrated in Figure 13.

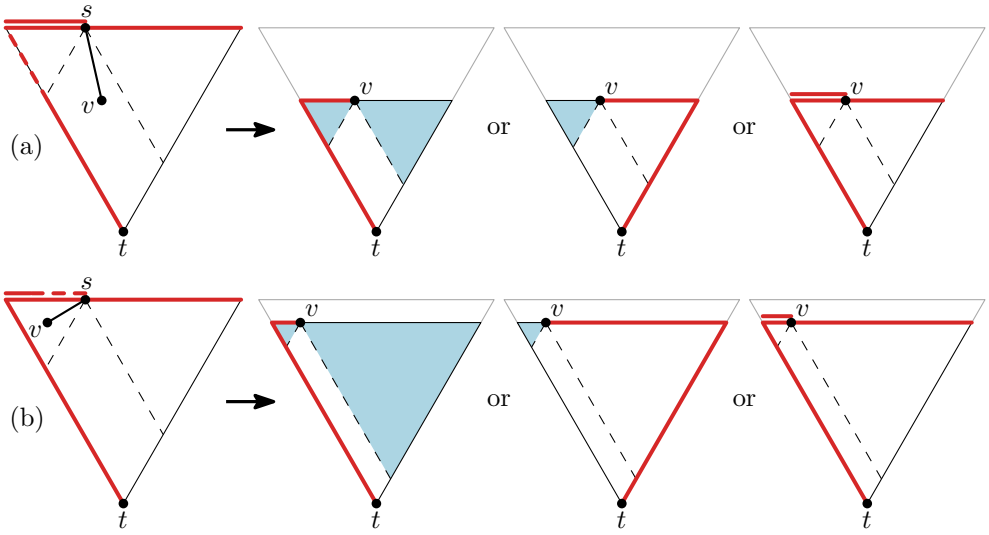

FIG. 13. *Routing in case D. The endpoint v of the edge followed lies in X_0* (a)*, or the smaller of X_1 and X_2* (b).

In the first situation, where we follow an edge (s, v) with $v \in X_0$, the reduction of

the vertical part of ϕ is at least as large as $|sv|$ by Observation 1. The horizontal part of ϕ can only decrease, as T_{tv} is fully contained in T_{ts} and v lies in X_0. In the second situation, where the endpoint of our edge lies in the smaller of X_1 and X_2, these roles switch, with the reduction of the horizontal part of ϕ being at least as large as $|sv|$ and the vertical part of ϕ only decreasing. In both situations, the statement is proven.

LEMMA 10 (upper bound for negative routing). *Let u and w be two vertices, with w in a positive cone of u. Let m be the midpoint of the side of T_{uw} opposing u, and let α be the unsigned angle between uw and um. There is a deterministic, 1-local, 0-memory routing algorithm on the half-θ_6-graph for which every path followed has length at most $(5/\sqrt{3} \cdot \cos\alpha - \sin\alpha) \cdot |uw|$ when routing from w to u.*

Proof. Since the choices that the routing algorithm makes are completely determined by the neighbors of s and the location of s and t, the algorithm is indeed deterministic, 1-local, and 0-memory. To bound the length of the resulting path, we again showed that for any routing step, the reduction in ϕ is at least as large as the length of the edge followed. As in the proof of Lemma 8, this implies that the routing algorithm terminates and that the total length of the path followed is bounded by the initial value of ϕ. Since our destination lies in a negative cone, we start in one of the cases B, C, or D. Of these three cases, case D has the largest initial potential of $|ta| + |ab| + \min(|as|, |sb|)$. Taking the side of T_{uw} as the unit of length reduces this to $1 + 1 + 1/2 - |wm| = 5/2 - |wm|$, and, using the same analysis as in Lemma 9, we obtain the desired bound of $(5/\sqrt{3} \cdot \cos\alpha - \sin\alpha) \cdot |uw|$. ∎

As Theorem 4 follows from Lemmas 7, 8, 9, and 10, this concludes our proof.

6. A stateful algorithm. Next we present a slightly different routing algorithm from the one in the previous section. The main difference between the two algorithms is that this one maintains one piece of information as state, making it $O(1)$-memory instead of 0-memory. The information that is stored is a *preferred side*, and it is either nil, X_1, or X_2. Intuitively, the new algorithm follows the original algorithm until it is routing negatively and determines that either X_1 or X_2 is empty. At that point, the algorithm sets the empty side as the preferred side and picks the rest of the edges in such a way that the preferred side remains empty. Thus, the algorithm maintains as invariant that if the preferred side is set (not nil), that region is empty. Furthermore, once the preferred side is set, it stays fixed until the algorithm reaches the destination. This algorithm simplifies the cases a little, but, more importantly, it allows the algorithm to check far fewer edges while routing. This is crucial, as the new algorithm forms the basis for routing algorithms on versions of the half-θ_6-graph with some edges removed to bound the maximum degree, as described in the next section.

We now present the details of this stateful version of the routing algorithm. Recall that we are trying to find a path from a current vertex s to a destination vertex t. For ease of description, we again assume without loss of generality that t lies in C_0 or $\overline{C}_0$ of s. If t lies in $\overline{C}_0$, the cones around s split T_{ts} into three regions X_0, X_1, and X_2, as in Figure 6. For brevity, we use "an edge in X_0" to denote an edge incident to s with the other endpoint in X_0. The cases are as follows:

- If t lies in a positive cone of s, we are in case $\mathcal{A}$.
- If t lies in a negative cone of s and no preferred side has been set yet, we are in case $\mathcal{B}$.
- If t lies in a negative cone of s and a preferred side has been set, we are in case $\mathcal{C}$.

These cases are closely related to the cases in the stateless algorithm. Cases $\mathcal{A}$

and $\mathcal{B}$ correspond to cases A and D, respectively, while case $\mathcal{C}$ merges cases B and C from the original algorithm into a single case, where only one side's emptiness is tracked. This is reflected in the routing strategy for each case:

- In case $\mathcal{A}$, follow the unique edge (s, v) in the positive cone containing t. If t lies in a negative cone of v, set the preferred side to the region (X_1 or X_2 of v) that is contained in T_{sv}, as this is now known to be empty (see Figure 8b).
- In case $\mathcal{B}$, if there are edges in X_0, follow an arbitrary one. Otherwise, if there is an edge in the smaller of X_1 and X_2, follow that edge. Otherwise, follow the edge in the larger of X_1 and X_2, and set the other as the preferred side. By Theorem 1, at least one of these edges must exist.
- In case $\mathcal{C}$, if there are edges in X_0, follow the one closest to the preferred side in cyclic order around s. Otherwise, follow the edge in the positive cone that is not on the preferred side. Again, at least one of these edges must exist.

The proof in section 5 can be adapted to show that this routing algorithm achieves the same upper bounds. In short, the proof is simplified to only use a potential as defined for cases A, C, and D, and only a subset of the illustrations in Figures 8, 10, and 13 are relevant. We omit the repetitive details.

7. Bounding the maximum degree. Each vertex in the half-θ_6-graph has at most one incident edge in each positive cone, but it can have an unbounded number of incident edges in its negative cones. In this section, we describe two transformations that allow us to bound the total degree of each vertex. The transformations are adapted from Bonichon et al. [5].

The first transformation discards all edges in each negative cone, except for three: the first and last edges in clockwise order around the vertex and the edge to the "closest" vertex, meaning the vertex whose projection on the bisector of the cone is closest (see Figure 14a). This results in a subgraph with maximum degree 12, which we call G_{12}.

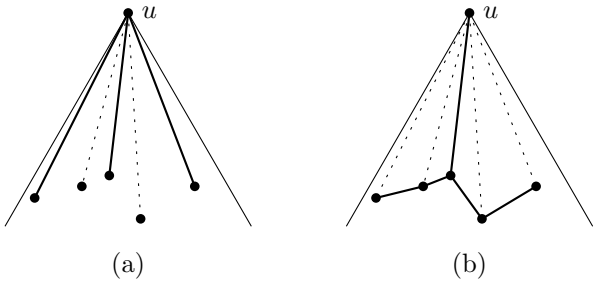

FIG. 14. *The construction for G_{12}(a) and G_9(b). Solid edges are kept, while dotted edges are discarded if no other vertex wants to keep them.*

To reduce the degree even further, we note that since the half-θ_6-graph is internally triangulated, consecutive neighbors of u within a negative cone are connected by edges. We call the path formed by these edges the *canonical path*. Instead of keeping three edges per negative cone, we now keep only the edge to the closest vertex but force the edges of the canonical path to be kept as well (see Figure 14b). We call the resulting graph G_9. Bonichon et al. [5] showed that all edges on the canonical path are either first or last in a negative cone, making G_9 a subgraph of G_{12}. Note that since the half-θ_6-graph is planar, both subgraphs are planar as well. They also proved that G_9 is a 3-spanner of the half-θ_6-graph with maximum degree 9. Since the

half-θ_6-graph is a 2-spanner and G_9 is a subgraph of G_{12}, this shows that both G_9 and G_{12} are 6-spanners of the complete Euclidean graph. We give an adapted version of the proof of the spanning ratio of G_9 below.

THEOREM 11. *G_9 is a 3-spanner of the half-θ_6-graph.*

Proof. Consider an edge (s, v) in the half-θ_6-graph, and assume, without loss of generality, that v lies in a negative cone of s (if not, we can swap the roles of s and v). Now consider the path between them in G_9 consisting of the edge from s to the vertex closest to s, followed by the edges on the canonical path between the closest vertex and v. We will refer to this path as the *approximation path*, and we show that it has length at most $3 \cdot |sv|$.

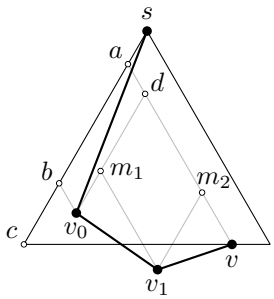

FIG. 15. *The approximation path.*

Let v_0 be the closest vertex, and let $v_1, \ldots, v_k = v$ be the other vertices on the approximation path. We assume without loss of generality that s lies in C_0 of v and that v lies to the right of v_0. We shoot rays parallel to the boundaries of C_0 from each vertex on the approximation path. Let m_i be the intersection of the right ray of v_{i-1} and the left ray of v_i (see Figure 15). These intersections must exist, as s is the closest vertex in $C_0^{v_i}$, for each v_i. Let a and b be the intersections of the left boundary of $\overline{C}_0^s$ with the left rays of v and v_0, respectively, and let c be the intersection of this left boundary with the horizontal line through v. Finally, let d be the intersection of the right ray of v_0 and the left ray of v. We can bound the length of the approximation path as follows:

$$
\begin{aligned}
& |sv_0| + \sum_{i=1}^{k} |v_{i-1}v_i| \\
\leq\ & |sb| + |bv_0| + \sum_{i=1}^{k} |v_{i-1}m_i| + \sum_{i=1}^{k} |m_i v_i| \\
=\ & |sb| + |bv_0| + |ab| + |dv| \quad \{\text{by projection}\} \\
=\ & |sb| + |ab| + |av| \\
\leq\ & |sc| + 2 \cdot |cv|.
\end{aligned}
$$

The last inequality follows from the fact that v_0 is the closest vertex to s. Let α be $\angle csv$. Some basic trigonometry gives us that $|sc| = \frac{2}{\sqrt{3}} \cdot \sin\left(\alpha + \frac{\pi}{3}\right) \cdot |sv|$ and $|cv| = \frac{2}{\sqrt{3}} \cdot \sin\alpha \cdot |sv|$. Thus the approximation path is at most $\frac{2}{\sqrt{3}} \cdot \left(\sin\left(\alpha + \frac{\pi}{3}\right) + 2 \cdot \sin\alpha\right)$ times as long as (s, v). Since this function is increasing in $[0, \frac{\pi}{3}]$, the maximum is achieved for $\alpha = \pi/3$, where it is 3. Therefore every edge of the half-θ_6-graph can

be approximated by a path that is at most three times as long, and the theorem follows. □

Note that the part of the approximation path that lies on the canonical path has length at most $2 \cdot |cv| = \frac{4}{\sqrt{3}} \cdot \sin\alpha \cdot |sv|$. This function is also increasing in $[0, \frac{\pi}{3}]$, and its maximal value is 2, so the total length of this part is at most $2 \cdot |sv|$.

7.1. Routing in G_{12}. The stateful algorithm in section 6 constructs a path between two vertices in the half-θ_6-graph. We cannot directly follow this path in G_{12}, as some of the edges may have been removed. Hence, we need to find a new path in G_{12} that approximates the path in the half-θ_6-graph, taking the missing edges into account. This often amounts to following the approximation path for edges that are in the path in the half-θ_6-graph but were removed to create G_{12}. In addition, some of the information the algorithm uses to decide which edge to follow relies on the presence or absence of edges in the half-θ_6-graph. Since the absence of these edges in G_{12} does not tell us whether or not they were present in the half-θ_6-graph, we need to find a new way to make these decisions.

First, note that the only information needed to determine which of the three cases we are in is the coordinates of s and t and whether the preferred side has been set or not. Therefore we can still make this distinction in G_{12}. The following five headlines refer to steps of the stateful algorithm on the half-θ_6-graph, and the text after a headline describes how to simulate that step in G_{12}. We discuss modifications for G_9 in section 7.2.

Follow an edge (s, v) in a positive cone C. If the edge of the half-θ_6-graph is still present in G_{12}, we simply follow it. If it is not, the edge was removed because s is on the canonical path of v and it is not the closest, first, or last vertex on the path. Since G_{12} is a supergraph of G_9, we know that all of the edges of the canonical path are kept and every vertex on the path originally had an edge to v in C. Therefore it suffices to traverse the canonical path in one direction until we reach a vertex with an edge in C, and follow this edge. Since the edges connecting v to the first and last vertices on the path are always kept, the edge we find in this way must lead to v. Note that the edges of the canonical path are easy to identify, as they are the closest edges to C in cyclic order around s (one on either side of C).

This method is guaranteed to reach v, but we want to find a *competitive* path to v. Therefore we use exponential search along the canonical path: we start by following the shorter of the two edges of the canonical path incident to s. If the endpoint of this edge does not have an edge in C, we return to s and travel twice the length of the first edge in the other direction. We keep returning to s and doubling the maximum travel distance until we find a vertex x that does have an edge in C. If x is not the closest to v, by the triangle inequality, following its edge to v is shorter than continuing our search until we reach the closest and following its edge. So for the purpose of bounding the distance traveled, we can assume that x is closest to v. Let d be the distance between s and x along the canonical path. By using exponential search to find x, we travel at most nine times this distance [2], and afterward we follow (x, v). From the proof of Theorem 11, we know that $d \leq 2 \cdot |sv|$ and $d + |xv| \leq 3 \cdot |sv|$. Thus the total length of our path is at most $9 \cdot d + |xv| = 8 \cdot d + (d + |xv|) \leq 16 \cdot |sv| + 3 \cdot |sv| = 19 \cdot |sv|$.

Determine if there are edges in X_0. In the regular half-θ_6-graph we can look at all our neighbors and see if any of them lie in X_0. However, in G_{12}, these edges may have been removed. Fortunately, we can still determine if they existed in the original half-θ_6-graph. To do this, we look at the vertices of the canonical path in this cone that are first and last in clockwise order around s. If these vertices do not exist, s did

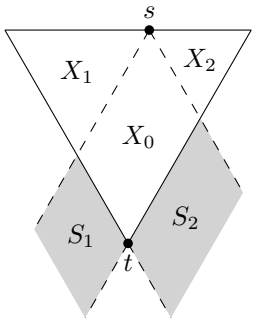

FIG. 16. *Possible regions for the first and last vertices.*

not have any incoming edges in this cone, so there can be no edges in X_0. If the first and last are the same vertex, this was the only incoming edge to s from this cone, so we simply check if its endpoint lies in X_0. The interesting case is when the first and last exist and are distinct. If either of them lies in X_0, we have our answer, so assume that both lie outside of X_0. Since they were connected to s, they cannot have t in their positive cone, so they must lie in one of two regions, which we call S_1 and S_2 (see Figure 16).

If both the first and last lie in S_2, there can be no edge in X_0, since any vertex of the canonical path in X_0 either lies in cone C_0 of the last vertex or would come after the last vertex in clockwise order around s. Both yield a contradiction. If both lie in S_1, a similar argument using the first vertex applies.

On the other hand, if the first lies in S_2 and the last in S_1, both X_1 and X_2 have to be empty, since both vertices are connected to s. Now we are in one of two cases: either X_0 is also empty, or it is not. If there are no vertices in X_0 (different from t and s), t must have had an edge to s. On the other hand, if there are other vertices in X_0, the topmost of these vertices must have had an edge to s. In either case, there must have been an edge in X_0. This shows that we can check whether there was an edge in X_0 in the half-θ_6-graph using only the coordinates of the first and last vertices.

Follow an arbitrary edge in X_0. If the half-θ_6-graph has edges in X_0, we simulate following an arbitrary one of these by first following the edge to the closest vertex in the negative cone. If this vertex is in X_0, we are done. Otherwise, we follow the canonical path in the direction of X_0 and stop once we are inside. This traverses exactly the approximation path of the edge,and hence travels a distance of at most three times the length of the edge.

Determine if there is an edge in X_1 or X_2. Since these regions are symmetric, we will consider only the case for X_1. Since X_1 is contained in a positive cone of s, it contains at most one edge incident to s. If the edge is present in G_{12}, we can simply test whether the other endpoint lies in X_1. However, if s does not have a neighbor in this cone (see Figure 17), we need to find out whether it used to have one in the original half-θ_6-graph and, if so, whether it was in X_1. Since this step is only needed in case $\mathcal{B}$ after we determine that there are no edges in X_0, we can use this information to guide our search. Specifically, we know that if we find an edge, we should follow it.

Therefore we simply attempt to follow the edge in this cone, using the exponential search method for following an edge in a positive cone described earlier. Let x be the first vertex we encounter that still has an edge (x, w) in C_1. If, in the half-θ_6-graph, s had an edge (s, v) in X_1, then we know (from the arguments presented earlier for

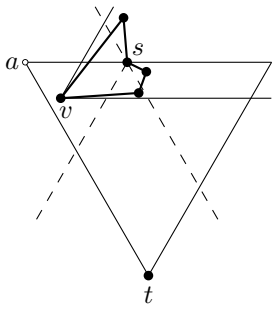

FIG. 17. *An example where s had an edge in X_1 in the half-θ_6-graph, which was removed during the construction of G_{12}.*

following an edge in a positive cone) that w is v. As such, w must lie in X_1. We also know (from the proof of Theorem 11) that the distance along the canonical path from s to x is at most $2 \cdot |sv|$, which is bounded by $2 \cdot |as|$ since v lies in X_1. In this case, we follow the edge from x to v. Conversely, if we do not find any vertex with an edge in C_1 within a distance of $2 \cdot |as|$ from s, or we do, but the endpoint w of the edge does not lie in X_1, then we can return to s and conclude that it did not have an edge in X_1 in the half-θ_6-graph and therefore X_1 must be empty.

If there was an edge in X_1, we traveled the same distance as if we were simply following the edge: at most $19 \cdot |sv|$. If we return to s unsuccessfully, we traveled at most $20 \cdot |as|$: 9 times $2 \cdot |as|$ during the exponential search and $2 \cdot |as|$ to return to s.

Follow the edge in X_0 closest to the preferred side in clockwise order. To follow this edge, we first follow the edge to the closest vertex. If this lands us in X_0, we then follow the canonical path toward the preferred side and stop at the last vertex on the canonical path that is in X_0. If the closest is not in X_0, we follow the canonical path toward X_0 and stop at the first or last vertex in X_0, depending on which side of X_0 we started on. This follows the approximation path of the edge, so the distance traveled is at most three times the length of the edge.

Routing ratio. This shows that we can simulate the stateful routing algorithm on G_{12}. The state we need to store in the message includes not only the preferred side but also information for the exponential search, including distance traveled. The exact routing ratios are as follows.

THEOREM 12. *Let u and w be two vertices, with w in a positive cone of u. There exists a deterministic 1-local $O(1)$-memory routing algorithm on G_{12} with routing ratio*

(i) $19 \cdot 2 = 38$ *when routing from u to w,*

(ii) $19 \cdot 5/\sqrt{3} = 54.848\ldots$ *when routing from w to u.*

Proof. As shown above, we can simulate every edge followed by the algorithm by traveling at most 19 times the length of the edge. The only additional cost is incurred in case $\mathcal{B}$, when we try to follow an edge in the smaller of X_1 and X_2, but this edge does not exist. In this case, we travel an additional $20 \cdot |as|$, where a is the corner closest to s. Fortunately, this can happen at most once during the execution of the algorithm, as it prompts the transition to case $\mathcal{C}$, after which the algorithm never returns to case $\mathcal{B}$. Looking at the proof for the upper bound in section 5 (specifically, the second case in Figure 13b), we observe that in the transition from case D to C, there is $2 \cdot |as|$ of unused potential. Since we are trying to show a routing ratio of 19 times the original, we can charge the additional $20 \cdot |as|$ to the $38 \cdot |as|$ of unused potential. ∎

7.2. Routing in G_9. In this subsection, we explain how to modify the previously described simulation strategies so that they work for G_9, where the first and last edges are not guaranteed to be present. We discuss only those steps that rely on the presence of these edges. To route successfully in this setting, we need to change our model slightly. We now let every vertex store a constant amount of information in addition to the information about its neighbors.

Follow an edge (s, v) in a positive cone. Because the first and last edges are not always kept, we cannot guarantee that the first vertex we reach with an edge in this positive cone is still part of the same canonical path. This means that the edge could connect to some arbitrary vertex, far away from v. Therefore our original exponential search solution does not work. Instead, we store one bit of information at s (per positive cone), namely, in which direction we have to follow the canonical path to reach the closest vertex to v. Knowing this, we just follow the canonical path in the indicated direction until we reach a vertex with an edge in this positive cone. This vertex must be the closest, so it gives us precisely the approximation path, and therefore we travel at most $3 \cdot |sv|$.

Determine if there are edges in X_0. In G_{12}, this test was based on the coordinates of the endpoints of the first and last edges. Since these might be missing in G_9, we store the coordinates of these vertices at s. This allows us to perform the check without increasing the distance traveled.

Determine if there is an edge in X_1 or X_2. As in the positive routing simulation, we now know where to go to find the closest. Therefore we simply follow the canonical path in this direction from s and stop when we reach a vertex with an edge in the correct positive cone, or when we have traveled $2 \cdot |as|$. If there is an edge, we follow exactly the approximation path, giving us three times the length of the edge. If there is no edge, we travel $2 \cdot |as|$ back and forth, for a total of $4 \cdot |as|$.

Routing ratio. Since the other simulation strategies do not rely on the presence of the first or last edges, we can now analyze the routing ratio obtained on G_9.

THEOREM 13. *Let u and w be two vertices, with w in a positive cone of u. By storing $O(1)$ additional information at each vertex, there exists a deterministic 1-local $O(1)$-memory routing algorithm on G_9 and G_{12} with routing ratio*

(i) $3 \cdot 2 = 6$ *when routing from u to w,*

(ii) $3 \cdot 5/\sqrt{3} = 8.660\ldots$ *when routing from w to u.*

Proof. The simulation strategy for G_{12} followed the approximation path for each edge, except when following an edge in a positive cone. Since our new strategy follows the approximation path there as well, our new routing ratio is only three times the one for the half-θ_6-graph. Note that this is still sufficient to charge the additional $4 \cdot |sa|$ traveled to the transition from case $\mathcal{B}$ to $\mathcal{C}$, which has $3 \cdot 2 \cdot |as|$ of otherwise unused potential. Since G_9 is a subgraph of G_{12}, this strategy works on G_{12} as well. □

8. Conclusions. We presented a competitive deterministic 1-local 0-memory routing algorithm on the half-θ_6-graph. We also presented matching lower bounds on the routing ratio for any deterministic, k-local, 0-memory algorithm, showing that our algorithm is optimal. Since any triangulation can be embedded as a half-θ_6-graph using Schnyder's embedding [21], this shows that any triangulation has an embedding that admits a competitive routing algorithm. An interesting open problem here is whether this approach can be extended to other theta-graphs. In particular, we recently extended the proof for the spanning ratio of the half-θ_6-graph to theta-graphs with $4k + 2$ cones for integer $k > 0$ [7]. It would be interesting to see if it is possible

to find optimal routing algorithms for these graphs as well.

We further extended our routing algorithm to work on versions of the half-θ_6-graph with bounded maximum degree. As far as we know, these are the first competitive routing algorithms on bounded-degree plane graphs. There are several problems here that are still open. For example, while we found a matching lower bound for negative routing in the regular half-θ_6-graph, we do not have one for the version with bounded degree. Can we find this, or is it possible to improve the routing algorithm further? And can we extend the algorithm to the version with maximum degree 6, introduced by Bonichon et al. [5]?

REFERENCES

[1] P. Angelini, F. Frati, and L. Grilli, *An algorithm to construct greedy drawings of triangulations*, J. Graph Algorithms Appl., 14 (2010), pp. 19–51.

[2] R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins, *Searching in the plane*, Inform. and Comput., 106 (1993), pp. 234–252.

[3] L. Barba, P. Bose, M. Damian, R. Fagerberg, W. L. Keng, J. O'Rourke, A. van Renssen, P. Taslakian, S. Verdonschot, and G. Xia, *New and improved spanning ratios for Yao graphs*, J. Comput. Geom., 6 (2015), pp. 19–53.

[4] N. Bonichon, C. Gavoille, N. Hanusse, and D. Ilcinkas, *Connections between theta-graphs, Delaunay triangulations, and orthogonal surfaces*, in Proceedings of the 36th International Conference on Graph Theoretic Concepts in Computer Science (WG 2010), Lecture Notes in Comput. Sci. 6410, Springer, Berlin, 2010, pp. 266–278.

[5] N. Bonichon, C. Gavoille, N. Hanusse, and L. Perkovic, *Plane spanners of maximum degree six*, in Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP 2010 (1)), Lecture Notes in Comput. Sci. 6198, Springer, Berlin, 2010, pp. 19–30.

[6] P. Bose, A. Brodnik, S. Carlsson, E. D. Demaine, R. Fleischer, A. López-Ortiz, P. Morin, and J. Ian Munro, *Online routing in convex subdivisions*, Internat. J. Comput. Geom. Appl., 12 (2002), pp. 283–296.

[7] P. Bose, J.-L. De Carufel, P. Morin, A. van Renssen, and S. Verdonschot, *Optimal bounds on theta-graphs: More is not always better*, in Proceedings of the 24th Canadian Conference on Computational Geometry (CCCG 2012), Charlottetown, Prince Edward Island, Canada, 2012, pp. 305–310.

[8] P. Bose and P. Morin, *Online routing in triangulations*, SIAM J. Comput., 33 (2004), pp. 937–951.

[9] P. Bose, A. van Renssen, and S. Verdonschot, *On the spanning ratio of theta-graphs*, in Proceedings of the 13th Algorithms and Data Structures Symposium (WADS 2013), Lecture Notes in Comput. Sci. 8037, Springer, Berlin, 2013, pp. 182–194.

[10] P. Chew, *There are planar graphs almost as good as the complete graph*, J. Comput. System Sci., 39 (1989), pp. 205–219.

[11] R. Dhandapani, *Greedy drawings of triangulations*, Discrete Comput. Geom., 43 (2010), pp. 375–392.

[12] M. B. Dillencourt, *Realizability of Delaunay triangulations*, Inform. Process. Lett., 33 (1990), pp. 283–287.

[13] M. T. Goodrich and D. Strash, *Succinct greedy geometric routing in the Euclidean plane*, in Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC 2009), Lecture Notes in Comput. Sci. 5878, Springer, Berlin, 2009, pp. 781–791.

[14] X. He and H. Zhang, *A simple routing algorithm based on Schnyder coordinates*, Theoret. Comput. Sci., 494 (2013), pp. 112–121.

[15] X. He and H. Zhang, *On succinct greedy drawings of plane triangulations and* 3*-connected plane graphs*, Algorithmica, 68 (2014), pp. 531–544.

[16] J. M. Keil and C. A. Gutwin, *Classes of graphs which approximate the complete Euclidean graph*, Discrete Comput. Geom., 7 (1992), pp. 13–28.

[17] T. Leighton and A. Moitra, *Some results on greedy embeddings in metric spaces*, Discrete Comput. Geom., 44 (2010), pp. 686–705.

[18] S. Mishra, I. Woungang, and S. C. Misra, eds., *Guide to Wireless Sensor Networks*, Springer, Berlin, 2009.

[19] C. H. Papadimitriou and D. Ratajczak, *On a conjecture related to geometric routing*, Theoret. Comput. Sci., 344 (2005), pp. 3–14.

[20] H. Räcke, *Survey on oblivious routing strategies*, in Mathematical Theory and Computational Practice: Proceedings of the 5th Conference on Computability in Europe (CiE 2009), Lecture Notes in Comput. Sci. 5635, Springer, Berlin, 2009, pp. 419–429.

[21] W. Schnyder, *Embedding planar graphs on the grid*, in Proceedings of the 1st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 1990), ACM, New York, SIAM, Philadelphia, 1990, pp. 138–148.