Competitive Routing on a Bounded-Degree Plane Spanner

Prosenjit Bose* Rolf Fagerberg§ André van Renssen* Sander Verdonschot*

Abstract

We show that it is possible to route locally and competitively on two bounded-degree plane 6-spanners, one with maximum degree 12 and the other with maximum degree 9. Both spanners are subgraphs of the empty equilateral triangle Delaunay triangulation. First, in a weak routing model where the only information stored at each vertex is its neighbourhood, we show how to find a path between any two vertices of a 6-spanner of maximum degree 12, such that the path has length at most $95/\sqrt{3}$ times the straight-line distance between the vertices. In a slightly stronger model, where in addition to the neighbourhood of each vertex, we store $O(1)$ additional information, we show how to find a path that has length at most $15/\sqrt{3}$ times the Euclidean distance both in a 6-spanner of maximum degree 12 and a 6-spanner of maximum degree 9.

1 Introduction

A t-spanner of a weighted graph G is a connected subgraph H with the property that for all pairs of vertices, the weight of the shortest path between the vertices in H is at most t times the weight of the shortest path in G, for some fixed constant $t \geq 1$. The constant t is referred to as the *spanning ratio*. The graph G is referred to as the *underlying graph*. In our setting, the underlying graph is the complete graph on a set of n points in the plane and the weight of an edge is the Euclidean distance between its endpoints (see [9] for a comprehensive overview of spanners).

In communication networks, in addition to being a constant spanner, a desirable property is the ability to route messages on the network such that the total distance travelled by the message is at most a constant times the spanning ratio. Inability to route effectively defeats the purpose of building a spanner in the first place. Network routing strategies such as Dijkstra's algorithm [7] require knowledge of the whole network topology to compute a short route. In many settings, this assumption is impractical. As such, we focus on *local* routing strategies (see [8] for a discussion of various models wrt. local routing). In a local routing strategy, the decision to forward a message depends on information stored at the node where the message currently resides, location of the source, location of the destination and the contents of a small memory. Typically, the information stored at a node is the set of direct neighbours.

Formally, an algorithm A is a k-memory routing algorithm, if the vertex to which a message is forwarded from the current vertex s is a function of s, t, $N(s)$, and M, where t is the destination vertex, $N(s)$ is the set of direct neighbours of s and M is a memory of size k, stored with the message. 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 $N(s)$. The algorithm A is *d-competitive* provided that the total distance travelled by the message is not more than d times the Euclidean distance between source and destination. We refer to the constant d as the *routing ratio*.

We present the first competitive k-memory routing algorithm to route on a bounded-degree plane spanner. Our algorithm routes on a 6-spanner with maximum degree 12, which is a subgraph of the empty equilateral triangle Delaunay triangulation [4]. We then present another competitive k-memory routing algorithm that routes on a subgraph with maximum degree 9. However, for this algorithm, we need to slightly enhance our model by storing a constant number of bits and points at each vertex (in addition to the neighbourhood of the vertex) to help guide the routing process.

2 Graphs

The empty equilateral triangle Delaunay triangulation was one of the first plane graphs that was shown to be a spanner [6]. It is internally triangulated and has a spanning ratio of 2. Recently, Bonichon *et al.* showed that it is equivalent to the half-θ_6-graph [2] and that it contains a bounded-degree spanner as a subgraph [3]. In this section, we describe the construction of the half-θ_6-graph and two of its bounded-degree subgraphs.

Given a set P of points in the plane, we consider each point $v \in P$ and partition the plane into 6 cones with apex v, each defined by two rays at consecutive multiples of $\pi/3$ radians from the positive x-axis. We

*School of Computer Science, Carleton University. Research supported in part by NSERC. Email: jit@scs.carleton.ca, andre@cg.scs.carleton.ca, sander@cg.scs.carleton.ca.

§Department of Mathematics and Computer Science, University of Southern Denmark. Email: rolf@imada.sdu.dk. Partially supported by the Danish Council for Independent Research, Natural Sciences.

label the cones $\overline{C}_1, C_0, \overline{C}_2, C_1, \overline{C}_0$ and C_2, in counterclockwise order around v, 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*.

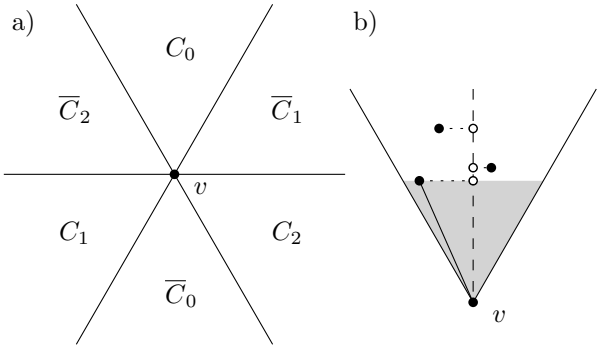

Figure 1: a) The cones around a vertex v. b) The construction of the half-θ_6-graph. In each positive cone, v connects to the vertex with the closest projection on the bisector of that cone.

To build the half-θ_6-graph, we consider each vertex v and add an edge between v and the 'closest' vertex in each of its positive cones. However, instead of using the Euclidean distance, we measure distance by projecting each vertex onto the bisector of the cone. We call the vertex in this cone whose projection is closest to v the *closest vertex* and connect it to v with an edge (see Figure 1b). For simplicity, we assume that no two points lie on a line parallel to a cone boundary.

Given two points a and b such that b lies in a positive cone of a, we define their *canonical triangle* T_{ab} to be the triangle bounded by the cone of a that contains b and the line through b perpendicular to the bisector of that cone. For example, the shaded region in Figure 1b is the canonical triangle of v and its closest vertex. 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.

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. We describe two transformations that allow us to bound the total degree of each vertex. The transformations are adapted from Bonichon *et al.* [3].

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 (see Figure 2a). This results in a subgraph with maximum degree 12, which we call G_{12}.

To reduce the degree even further, we note that since the half-θ_6-graph is internally triangulated, consecutive neighbours of v within a negative cone are connected by edges. We call the path formed by these edges the

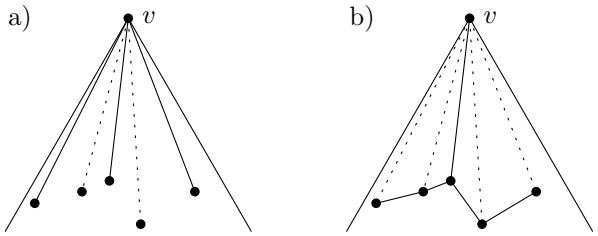

Figure 2: The construction for G_{12} (a) and G_9 (b). Solid edges are kept, while dotted edges are discarded if no other vertex wants to keep them.

canonical path. So instead of keeping three edges per negative cone, we keep only the edge to the closest vertex, but force the edges of the canonical path to be kept as well (see Figure 2b). We call the resulting graph G_9. Since the half-θ_6-graph is planar, both subgraphs are planar as well. In a previous paper [5], we showed that G_9 is a 6-spanner with maximum degree 9. We give an adapted version of the proof for the spanning ratio below.

Theorem 1 *G_9 is a 3-spanner of the half-θ_6-graph.*

Proof. We show that for every edge (s, v) in the half-θ_6-graph, there is a path of length at most $3 \cdot |sv|$ in G_9. This path consists of the edge to the closest vertex, followed by the edges on the canonical path between the closest vertex and v. We will refer to it as the *approximation path*.

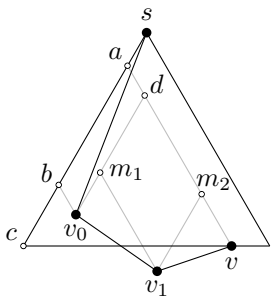

Figure 3: The approximation path.

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. Let m_i be the intersection of the right ray of v_{i-1} and the left ray of v_i (see Figure 3). Let a and b be the intersections of the left boundary of $\overline{C}_0$ of 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:

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

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 $\frac{2}{\sqrt{3}} \cdot \left(\sin\left(\alpha + \frac{\pi}{3}\right) + 2\sin(\alpha)\right)$ 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 3 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 $2 \cdot |sv|$.

Since the half-θ_6-graph is a 2-spanner, this shows that G_9 is a 6-spanner. Bonichon *et al.* [3] also showed that all edges on the canonical path are either first or last in their respective negative cones, making G_9 a subgraph of G_{12}. Hence G_{12} is a 6-spanner as well.

3 Routing on G_{12}

We now turn our attention to finding a competitive path from a current vertex s to a given destination t in G_{12}. In a previous paper, we showed that it is possible to route competitively on the half-θ_6-graph [4].

Theorem 2 ([4], Corollary 4.1) *Let u and w be two vertices with w in a positive cone of u. There exists a 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 best possible for deterministic local routing schemes.

Next we present a slightly modified version of the routing algorithm for the half-θ_6-graph. The difference lies in the fact that the original algorithm does not keep state. However, the same proofs can be used to show that the stateful version in this paper finds a path with the same routing ratio.

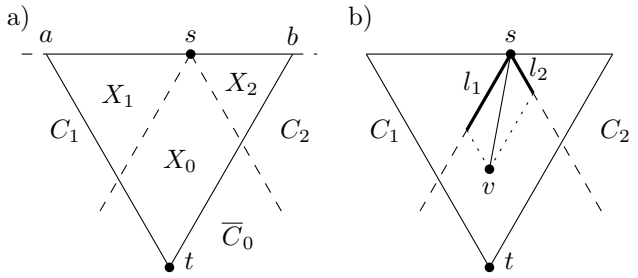

Figure 4: a) The points and regions involved in negative routing. b) The projected length of an edge.

Before we can describe the actual algorithm, we need a few definitions. We 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, which we call X_0, X_1 and X_2, as shown in Figure 4a. Formally, let $X_0 = \overline{C}_0 \cap T_{ts}$, $X_1 = C_1 \cap T_{ts}$ and $X_2 = C_2 \cap T_{ts}$. Further, we let a be the corner of T_{ts} that is on the boundary of C_1 and b the corner on the boundary of C_2. For brevity, we use "an edge in X_0" to denote an edge incident to s with the other endpoint in X_0.

We also need the concept of the *projected length* of an edge onto a neighbouring cone. For an edge (s, v) in $\overline{C}_0$, the neighbouring cones of s are C_1 and C_2. Let $\vec{e_1}$ and $\vec{e_2}$ be unit vectors parallel to the boundary of $\overline{C}_0$ with C_1 and C_2, respectively. Since $\vec{e_1}$ and $\vec{e_2}$ are linearly independent, the vector $\vec{sv}$ can be uniquely written as $l_1 \cdot \vec{e_1} + l_2 \cdot \vec{e_2}$. We define the projected length of (s, v) on C_1 as l_1 and on C_2 as l_2 (see Figure 4b).

Our algorithm distinguishes three cases and keeps track of a *preferred side*, which is one of the positive cones, or undefined. The preferred side is stored as state in the message. If t lies in a positive cone of s, we are in case A. If t lies in a negative cone of s and no preferred side has been set yet, we are in case B. If t lies in a negative cone of s and a preferred side has been set, we are in case C. The algorithm works as follows on the half-θ_6-graph.

- In case A, follow the unique edge in the positive cone containing t.
- In case 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. At least one of these edges must exist [4].
- In case C, if there are edges in X_0, follow the one with the largest projected distance on the preferred side. Otherwise, follow the edge in the positive cone that is not on the preferred side. Again, at least one of these edges must exist [4].

This algorithm constructs a path between two vertices in the half-θ_6-graph. To approximate this path in G_{12} and G_9, we simulate each step of the algorithm. Note that we can decide which case we are in based solely on the coordinates of s and t and whether the preferred side has been set. The following five headlines refer to original steps of the 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 4.

Follow an edge (s, v) in a positive cone. If the edge is still there, 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 the same positive cone. Therefore it suffices to search the canonical path for any vertex with an edge in this positive cone 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.

This method is guaranteed to reach v, but we want to find a *competitive* path to v. Therefore we will 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 our positive cone, 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 our positive cone. 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 travelled, 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 9 times this distance [1] and afterwards we follow (s, x). From the spanning proof, 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 neighbours 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 first and last vertex along the canonical path in this cone. If these vertices do not exist, s did not have any incoming edges in this cone, so there can be no edges in X_0. If 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 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 cannot have t in their positive cone, they must lie in one of two regions, which we call S_1 and S_2 (see Figure 5a).

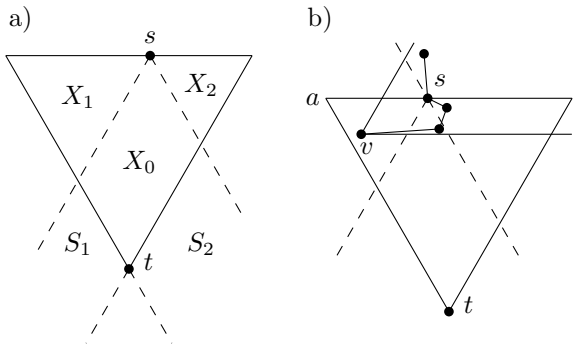

Figure 5: a) Possible regions for the first and last vertex. b) A vertex v in X_1.

If both first and last lie in the same region (say S_1), there can be no edge in X_0, since any vertex on the canonical path between them in X_0 would lie in C_0 of the last vertex. This would prevent the last vertex from having an edge to s, which is a contradiction.

On the other hand, if first lies in S_1 and last in S_2, both X_1 and X_2 have to be empty, since s was the closest vertex to both. Thus if there are no vertices in X_0 (different from t and s), t must have an edge to s, which gives us an edge in X_0. On the other hand, if there are vertices in X_0, the same holds for the topmost vertex in X_0, so in either case there must be 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 vertex.

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 at most 3 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. It is contained in a positive cone of s, so it contains at most one edge incident to s. If the edge is still there, we can simply test whether it is in X_1 or not. However, if s does not have a neighbour in this cone, 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 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. If there is a vertex v in X_1 and the edge (s, v) was removed (see Figure 5b), we must encounter the closest vertex after travelling at most $2 \cdot |sv|$ along the canonical path. Since all edges in X_1 have length at most $|as|$, we use exponential search, travelling at most $2 \cdot |as|$ from s. Once we explored both sides of the path to a distance of $2 \cdot |as|$ without encountering a vertex with an edge in the correct positive cone, we return to s and conclude that there was no edge in X_1. If we do find such a vertex, we test whether its edge leads into X_1 and follow it if it does. If the edge does not lead into X_1, either the edge of s in C_1 had its endpoint outside of X_1, or s did not have an edge in C_1. Either way, we return to s and conclude that there was no edge in X_1.

If there was an edge in X_1, we travelled the same distance as if we were simply following the edge: at most $19 \cdot |sv|$. If we return to s unsuccessfully, we travelled 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 with the largest projected distance on the preferred side. In the half-θ_6-graph, we have sufficient information about our neighbours to simply compute their projected distances. However, a lot of these edges might have been removed in the construction of G_{12}. To help find the correct edge, we first prove the following property.

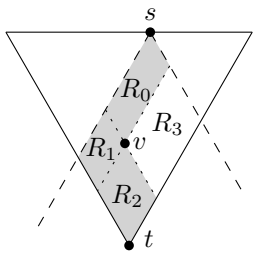

Figure 6: Situation around the first vertex in X_0.

Lemma 3 *In the half-θ_6-graph, the first or last edge in X_0 in counter-clockwise order around s has the largest projected distance on the preferred side.*

Proof. We consider only the case where the preferred side is C_1. The case for C_2 is analogous. Let v be the endpoint of the first edge in X_0 in counter-clockwise order around s. The lines through v parallel to the boundaries of $\overline{C}_0$ partition X_0 into four regions. In counter-clockwise order, starting at the top, we call these R_0, R_1, R_2 and R_3 (see Figure 6). Now let us consider the possible locations of other vertices on the canonical path in this negative cone of s.

Since v has an edge to s, R_0 must be empty. There can also be no neighbours of s in R_1, as these would have come before v in the counter-clockwise ordering around s. Finally, for vertices in R_2, v will always be closer than s, so there can be no neighbours of s in R_2 either. Thus all other vertices of the canonical path must either be outside X_0 or in R_3. Since the projected distance of (s, v) is at least as large as the projected distance to any vertex in R_3, (s, v) has the largest projected distance among all edges in X_0. □

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 towards 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 towards 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 travelled is at most 3 times the length of the edge.

Routing ratio. This shows that we can simulate the routing algorithm on G_{12}. Note that in contrast to the routing algorithm on the half-θ_6-graph, we maintain state in the message. We need to store not only the preferred side, but also information for the exponential search, including distance travelled. The exact routing ratios are as follows.

Theorem 4 *Let u and w be two vertices with w in a positive cone of u. There exists an $O(1)$-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 travelling at most 19 times the length of the edge. The only additional cost is incurred in case 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 C, after which the algorithm never returns to case B. Looking at the original proof for the routing ratio [4], we observe that in the transition from case B 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. □

4 Routing on G_9

In this section, we explain how to modify the 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.

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. Therefore our original exponential search solution does not work. Instead, we need to store one bit of information at s, 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 this 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 edge. 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 travelled.

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 travelled $2 \cdot |as|$. If there is an edge, we follow exactly the approximation path, giving us 3 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 5 *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 an $O(1)$-memory routing algorithm on G_{12} and G_9 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 3 times the one for the half-θ_6-graph. Note that this is still sufficient to charge the additional $4 \cdot |sa|$ travelled to the transition from case B to C. Since G_9 is a subgraph of G_{12}, this strategy works on G_{12} as well. □

5 Conclusion

We presented two competitive $O(1)$-memory routing algorithms for bounded-degree subgraphs of the half-θ_6-graph. To the best of our knowledge, these are the first competitive routing algorithms on bounded-degree plane spanners. The first strategy works on G_{12} and achieves a routing ratio of 19 times the routing ratio on the half-θ_6-graph. The second algorithm works on G_9 as well as G_{12} and reduces the routing ratio to 3 times the original. However, it achieves this only by storing information at vertices. Note that this routing ratio is optimal for the positive case, as the spanning ratio of 6 is tight for both graphs.

An interesting open problem is whether the routing ratio for the negative case can be improved, or if, as for the half-θ_6-graph, we can find a matching lower bound. And is it possible to improve the routing ratio on G_{12} without storing information at the vertices? Or does there exist a 0-memory routing algorithm for these graphs? It would also be interesting to see if there are other bounded-degree plane spanners that are easier to route on or allow better routing ratios. For example, a slight modification transforms G_9 into a plane 6-spanner with maximum degree 6 [3]. Is it possible to route competitively on that graph as well?

References

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

[2] N. Bonichon, C. Gavoille, N. Hanusse, and D. Ilcinkas. Connections between theta-graphs, Delaunay triangulations, and orthogonal surfaces. In *WG*, pages 266–278, 2010.

[3] N. Bonichon, C. Gavoille, N. Hanusse, and L. Perkovic. Plane spanners of maximum degree six. In *ICALP (1)*, pages 19–30, 2010.

[4] P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot. Competitive routing in the half-θ_6-graph. In *SODA*, pages 1319–1328, 2012.

[5] P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot. On plane constrained bounded-degree spanners. In *LATIN*, pages 85–96, 2012.

[6] P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989.

[7] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[8] X. Li. *Wireless Ad Hoc and Sensor Networks*. Cambridge University Press, 2008.

[9] G. Narasimhan and M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.