Competitive Local Routing with Constraints*

Prosenjit Bose[1], Rolf Fagerberg[2], André van Renssen[3], and Sander Verdonschot[1]

[1] School of Computer Science, Carleton University, Ottawa, Canada.
jit@scs.carleton.ca, sander@cg.scs.carleton.ca
[2] Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark. rolf@imada.sdu.dk
[3] National Institute of Informatics (NII), Tokyo, Japan; JST, ERATO, Kawarabayashi Large Graph Project. andre@nii.ac.jp

Abstract. Let P be a set of n points in the plane and S a set of non-crossing line segments between vertices in P, called constraints. Two vertices are visible if the straight line segment connecting them does not properly intersect any constraints. The constrained θ_m-graph is constructed by partitioning the plane around each vertex into m disjoint cones with aperture $\theta = 2\pi/m$, and adding an edge to the 'closest' visible vertex in each cone. We consider how to route on the constrained θ_6-graph. We first show that no deterministic 1-local routing algorithm is $o(\sqrt{n})$-competitive on all pairs of vertices of the constrained θ_6-graph. After that, we show how to route between any two visible vertices using only 1-local information, while guaranteeing that the returned path has length at most 2 times the Euclidean distance between the source and destination. To the best of our knowledge, this is the first local routing algorithm in the constrained setting with guarantees on the path length.

1 Introduction

A fundamental problem in any graph is the question of how to route a message from one vertex to another. What makes this more challenging is that often this must be *locally*, i.e. it can only use 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 that is routed on 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 [10] and [11] for surveys of the area) since they offer routing strategies that use the coordinates of the vertices to guide the search, instead of the more traditional routing tables.

Most of the research has focused on the situation where the network is constructed by taking a subgraph of the complete Euclidean graph. We study this problem in a more general setting with the introduction of line segment

* Research supported in part by NSERC, Carleton University's President's 2010 Doctoral Fellowship, and the Danish Council for Independent Research, Natural Sciences.

constraints. Specifically, let P be a set of points in the plane and let S be a set of line segments between vertices in P, with no two line segments intersecting properly. The line segments of S are called *constraints*. Two vertices u and v can *see each other* if and only if either the line segment uv does not properly intersect any constraint or uv is itself a constraint. If two vertices u and v can see each other, the line segment uv is a *visibility edge*. The *visibility graph* of P with respect to a set of constraints S, denoted $\text{Vis}(P, S)$, has P as vertex set and all visibility edges as edge set. In other words, it is the complete graph on P minus all non-constraint edges that properly intersect one or more constraints in S.

This setting has been studied extensively within the context of motion planning amid obstacles. Clarkson [8] was one of the first to study this problem and showed how to construct a $(1 + \epsilon)$-spanner of $\text{Vis}(P, S)$ with a linear number of edges. A subgraph H of G is called a t-spanner of G (for $t \geq 1$) if for each pair of vertices u and v, the shortest path in H between u and v has length at most t times the shortest path in G between u and v. The smallest value t for which H is a t-spanner is the *spanning ratio* of H. Following Clarkson's result, Das [9] showed how to construct a spanner of $\text{Vis}(P, S)$ with constant spanning ratio and constant degree. Bose and Keil [6] showed that the Constrained Delaunay Triangulation is a 2.42-spanner of $\text{Vis}(P, S)$. Recently, the constrained half-θ_6-graph (which is identical to the constrained Delaunay graph whose empty visible region is an equilateral triangle) was shown to be a plane 2-spanner of $\text{Vis}(P, S)$ [5] and all constrained θ-graphs with at least 6 cones were shown to be spanners as well [7].

However, though it is known that these graphs contain short paths, it is not known how to route in a local fashion. To address this issue, we look at k-local routing algorithms in the constrained setting, i.e. routing algorithms that must decide which vertex to forward a message to based solely on knowledge of the source and destination, the current vertex and all vertices that can be reached from the current vertex by following at most k edges. Furthermore, we require our algorithms to be *competitive*, i.e. the length of the returned path needs to be related to the length of the shortest path in the graph. We first show that no deterministic 1-local routing algorithm is $o(\sqrt{n})$-competitive on all pairs of vertices of the constrained θ_6-graph, regardless of the amount of memory it is allowed to use. Two other lower bounds were provided by Bose *et al.* [4] for the *unconstrained* setting: First, no deterministic k-local routing algorithm can be better than 2-competitive on the θ_6-graph. Secondly, no deterministic k-local routing algorithm can be better than $5/\sqrt{3}$-competitive on the half-θ_6-graph (the θ_6-graph is the union of two half-θ_6-graphs). In the same paper, the authors also provided a 1-local 0-memory routing algorithm that matches these lower bounds.

Despite our lower bound, we describe a 1-local 0-memory routing algorithm between any two *visible* vertices of the constrained θ_6-graph that guarantees that the length of the path traveled is at most 2 times the Euclidean distance between the source and destination. Additionally, we provide a 1-local $O(1)$-memory 18-competitive routing algorithm between any two visible vertices in the constrained half-θ_6-graph. To the best of our knowledge, these are the first local routing algorithms in the constrained setting with guarantees on the path length.

2 Preliminaries

We define a *cone* C to be the region in the plane between two rays originating from a single vertex, the apex of the cone. We let six rays originate from each vertex, with angles to the positive x-axis being multiples of $\pi/3$ (see Figure 1). Each pair of consecutive rays defines a cone. We write C_i^u to indicate the i-th cone of a vertex u, or C_i if the apex is clear from the context. For ease of exposition, we only consider point sets in general position: no two points define a line parallel to one of the rays that define the cones and no three points are collinear.

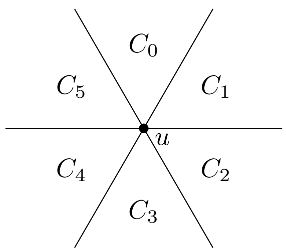

Fig. 1. The cones having apex u in the θ_6-graph

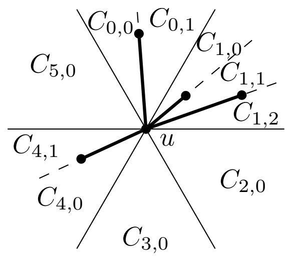

Fig. 2. The subcones having apex u in the constrained θ_6-graph. Constraints are shown as thick line segments

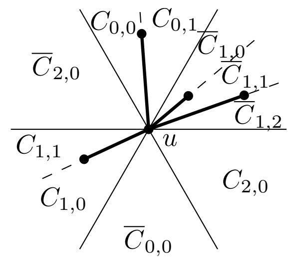

Fig. 3. The subcones having apex u in the constrained half-θ_6-graph. Constraints are shown as thick line segments

Let vertex u be an endpoint of a constraint and let the other endpoint lie in cone C_i^u. The lines through all such constraints split C_i^u into several *subcones* (see Figure 2). We use $C_{i,j}^u$ to denote the j-th subcone of C_i^u. When a constraint $c = (u, v)$ splits a cone of u into two subcones, we define v to lie in both of these subcones. We consider a cone that is not split to be a single subcone.

The constrained θ_6-graph is constructed as follows: for each subcone $C_{i,j}$ of each vertex u, add an edge from u to the closest visible vertex in that subcone, where distance is measured along the bisector of the original cone, not the subcone. More formally, we add an edge between two vertices u and v if v can see u, $v \in C_{i,j}$, and for all vertices $w \in C_{i,j}$ that can see u, $|uv'| \leq |uw'|$, where v' and w' denote the orthogonal projection of v and w on the bisector of C_i. Note that our general position assumptions imply that each vertex adds at most one edge per subcone to the graph.

Next, we define the constrained half-θ_6-graph. This is a generalized version of the half-θ_6-graph as described by Bonichon *et al.* [2]. The constrained half-θ_6-graph is similar to the constrained θ_6-graph with one major difference: edges are only added in every second cone. More formally, its cones are categorized as positive and negative. Let $(\overline{C}_1, C_0, \overline{C}_2, C_1, \overline{C}_0, C_2)$ be the sequence of cones in counterclockwise order starting from the positive x-axis. The cones C_0, C_1, and

C_2 are called *positive* cones and $\overline{C}_0$, $\overline{C}_1$, and $\overline{C}_2$ are called *negative* cones. We add edges only in the positive cones (and their subcones). We use C_i^u and $\overline{C}_i^u$ to denote cones C_i and $\overline{C}_i$ with apex u. For any two vertices u and v, $v \in C_i^u$ if and only if $u \in \overline{C}_i^v$. Analogous to the subcones defined for the θ_6-graph, constraints can split cones into subcones. We call a subcone of a positive cone a positive subcone and a subcone of a negative cone a negative subcone (see Figure 3).

Given a vertex w in a positive cone C_i^u of vertex u, we define the *canonical triangle* T_{uw} to be the triangle defined by the borders of C_i^u (not the borders of the subcone of u that constains w) and the line through w perpendicular to the bisector of C_i^u. Note that for each pair of vertices there exists a unique canonical triangle. We say that a region is *empty* if it does not contain any vertices of P.

Next, we define our routing model. A routing algorithm is a deterministic k-local, m-memory routing algorithm, if the vertex to which a message is forwarded from the current vertex u is a function of s, t, $N_k(u)$, and M, where s and t are the source and destination vertex, $N_k(u)$ is the k-neighborhood of u and M is a memory of size m, stored with the message. The k-neighborhood of a vertex u is the set of vertices in the graph that can be reached from u by following at most k edges. For our purposes, we consider a unit of memory to consist of $\log_2 n$ bits or a point in $\mathbb{R}^2$. Our model also assumes that the only information stored at each vertex of the graph is $N_k(u)$. Since our graphs are geometric, we identify each vertex by its coordinates in the plane. Unless otherwise noted, all routing algorithms we consider in this paper are deterministic 0-memory algorithms.

There are essentially two notions of *competitiveness* of a routing algorithm. One is to look at the Euclidean distance between the two vertices and the other is to compare the routing path to the shortest path in the graph. A routing algorithm is *c-competitive with respect to the Euclidean distance (resp. shortest path)* provided that the total distance traveled by the message is not more than c times the Euclidean distance (resp. shortest path) between source and destination. Analogous to the spanning ratio, the *routing ratio* of an algorithm is the smallest c for which it is c-competitive.

Since the length of the shortest path between two vertices is at least the Euclidean distance between the two vertices, an algorithm that is c-competitive with respect to the Euclidean distance is also c-competitive with respect to the shortest path. We use competitiveness with respect to the Euclidean distance when proving upper bounds and we use competitiveness with respect to the shortest path when proving lower bounds.

3 Lower Bound on Local Routing

We modify the proof by Bose *et al.* [3] (that shows that no deterministic routing algorithm is $o(\sqrt{n})$-competitive for all triangulations) to show the following lower bound.

Theorem 3.1. *No deterministic 1-local routing algorithm is $o(\sqrt{n})$-competitive with respect to the shortest path on all pairs of vertices of the θ_6-graph, regardless of the amount of memory it is allowed to use.*

Due to space constraints, we present a shortened version of the proof of this theorem. The full proof can be found in Appendix A.

Proof. Consider an $n \times n$ grid and shift every second row to the right by half a unit. We stretch the grid, such that each horizontal edge has length n (see Figure 4). Next, we replace each horizontal edge by a constraint to prevent vertical visibility edges. Finally, we add two additional vertices, origin s and destination t, centered horizontally at one unit below the bottom row and one unit above the top row, respectively.

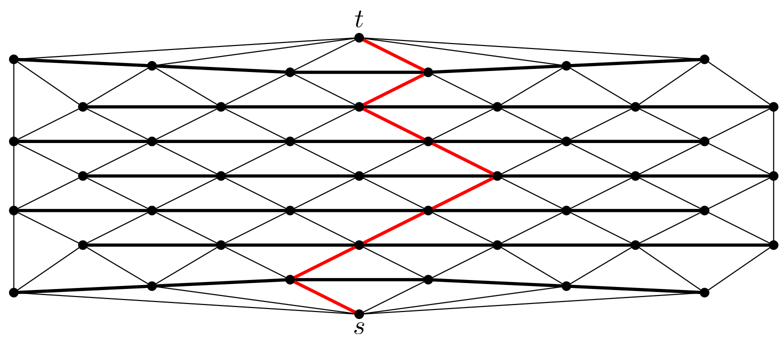

Fig. 4. The constrained θ_6-graph starting from a grid, using horizontal constraints to block vertical edges, and the red path of the routing algorithm

We move all vertices by at most some arbitrarily small amount ϵ, such that no two points define a line parallel to one of the rays that define the cones and no three points are collinear. In particular, we ensure that all vertices on the bottom row have s as the closest vertex in one of their subcones and all vertices on the top row have t as the closest vertex in one of their subcones. On this point set and these constraints, we build the constrained θ_6-graph G.

Consider any deterministic 1-local ∞-memory routing algorithm and let π be the path this algorithm takes when routing from s to t. If π consists of at least $n\sqrt{n}$ non-vertical steps, the total length of the path is $\Omega(n^2\sqrt{n})$. However, G contains a path of length $O(n^2)$ between s and t: the path that follows a diagonal edge to the left, followed by a diagonal edge to the right, until it reaches t. Hence, in this case, the local routing algorithm is not $o(\sqrt{n})$-competitive.

Now, assume that π consists of $f(n)$ non-vertical steps, for $f(n) < n\sqrt{n}$. Consider the $2\sqrt{f(n)}$ neighbors of s at horizontal distance at most $n\sqrt{f(n)}$ from s. Next, consider the vertical lines through these $2\sqrt{f(n)}$ neighbors of s and let π' be the routing path π minus vertices s and t. We say that a vertex of π' *touches* a vertical line if it has a neighbor on that line. Hence, any vertex along π' touches at most 2 vertical lines. Thus, the total number of lines touched by the vertices along π' is at most $2f(n)$. Hence, there exists a vertical line that is touched at most $\sqrt{f(n)}$ times. Let u be the neighbor of s on the vertical line that is touched the fewest number of times.

We now create a new constrained θ_6-graph G' such that the deterministic 1-local routing algorithm follows the same path, but G' contains a short 'almost vertical' path via u. We start with s, t, and all vertices of π. Next, we add all vertices and constraints connected to these vertices in G. On this point set and these constraints, we build the constrained θ_6-graph G' (see Figure 5).

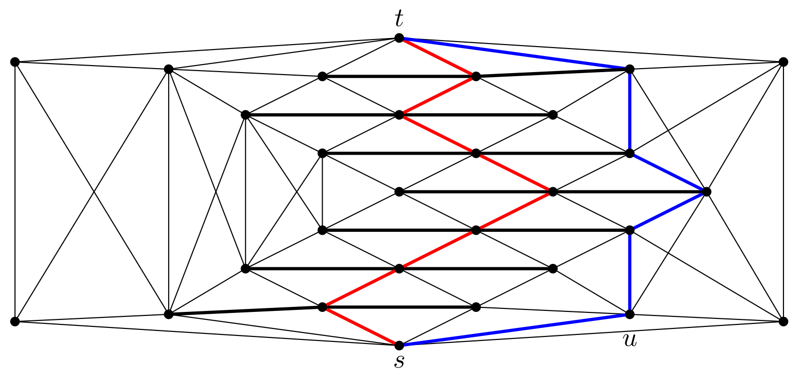

Fig. 5. The constrained θ_6-graph that looks the same from the red path of the routing algorithm, but has an almost vertical blue path

Since the horizontal distance between vertices is far larger than their vertical distance, an 'almost vertical' path from u to the top row of G' is formed. This almost vertical path is a path that is vertical whenever possible and uses detours to avoid path π (see Figure 13): If π arrives at a vertex v that has a neighbor on the vertical line through u, we avoid π by following one edge away from π, followed by an edge back to the vertical line through u (see Figure 6a). If π arrives at a vertex on the vertical line through u, we avoid the vertex before and after v on π as before, and meet π at v (see Figure 6b). Since no edge along the left and right boundary of G touches the vertical line through u, this vertical line is touched by at most $\sqrt{f(n)}$ vertices of π and only $O\left(\sqrt{f(n)}\right)$ of these detour edges are required. Hence, G' contains a path from s to t of length $O\left(n\sqrt{f(n)}\right)$.

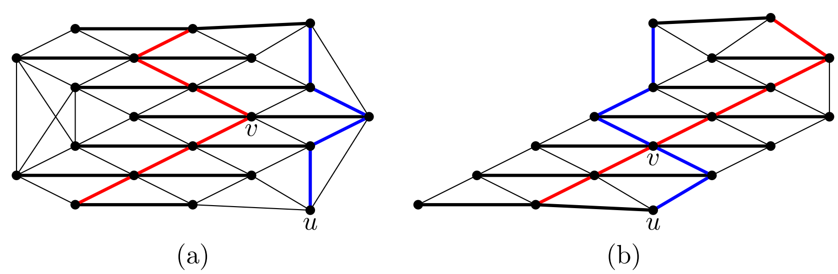

Fig. 6. The two types of detour: (a) when π does not visit the vertical line through u, (b) when π visits the vertical line through u

Since the 1-local routing algorithm is deterministic and the 1-local information of the vertices of π in G' is the same as in G, the algorithm follows the same path. Since most edges in G' have length at least n, π has length $\Omega(nf(n))$. This implies that π is not $o(\sqrt{n})$-competitive, as $f(n) \geq n+1$. Hence, since G' can be constructed for any deterministic 1-local routing algorithm, we have shown that no deterministic 1-local routing algorithm is $o\left(\sqrt{n}\right)$-competitive on all pairs of vertices. □

4 Routing on the Constrained θ_6-Graph

In this section, we provide a 1-local routing algorithm on the constrained θ_6-graph for any pair of visible vertices. Since the constrained θ_6-graph is the union of two constrained half-θ_6-graphs, we describe a routing algorithm for the constrained half-θ_6-graph for the case where the destination t lies in a positive subcone of the origin s. After describing this algorithm and proving that it is 2-competitive, we describe how to use it to route 1-locally on the constrained θ_6-graph. Throughout this section, we use the following auxiliary lemma proven by Bose *et al.* [5].

Lemma 4.1. *Let u, v, and w be three arbitrary points in the plane such that uw and vw are visibility edges and w is not the endpoint of a constraint intersecting the interior of triangle uvw. Then there exists a convex chain of visibility edges from u to v in triangle uvw, such that the polygon defined by uw, wv and the convex chain is empty and does not contain any constraints.*

4.1 Positive Routing on the Constrained Half-θ_6-Graph

Before describing how to route when t lies in a positive subcone of s, we first show that there exists a path in canonical triangle T_{st}.

Lemma 4.2. *Given two vertices u and w such that u and w see each other and w lies in a positive subcone $C^u_{i,j}$, there exists a path between u and w in the triangle T_{uw} in the constrained half-θ_6-graph.*

The proof of this lemma is a straightforward modification of Theorem 1 in [5].

Positive Routing Algorithm for the Constrained Half-θ_6-Graph
Next, we describe how to route from s to t, when s can see t and t lies in a positive subcone $C^s_{i,j}$ (see Figure 7): When we are at s, we follow the edge to the closest vertex in the subcone that contains t. When we are at any other vertex u, we look at all edges in the subcones of C^u_i and all edges in the subcones of the adjacent negative cone $\overline{C}^u$ that is intersected by st. An edge in a subcone of $\overline{C}^u$ is considered only if it does not cross st. For example, in Figure 7, we do not consider the edge to v_1 since it lies in $\overline{C}^u$ and crosses st. It follows that we can cross st only when we follow an edge in C^u_i.

Let z be the intersection of st and the boundary of $\overline{C}^u$ that is not a boundary of C_i^u. We follow the edge uv that minimizes the unsigned angle $\angle zuv$. For example, in Figure 7, when we are at vertex u we follow the edge to v_2 since, out of the two remaining edges uv_2 and uv_3, $\angle zuv_2$ is smaller than $\angle zuv_3$. We also note that during the routing process, t does not necessarily lie in C_i^u. Finally, since the algorithm uses only information about the location of s and t and the neighbors of the current vertex, it is a 1-local routing algorithm.

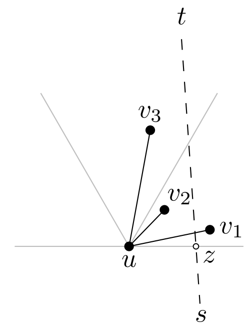

Fig. 7. An example of routing from s to $t \in C_0^s$. The dashed line represents the visibility line between s and t

We proceed by proving that the above routing algorithm can always perform a step, i.e. at every vertex reached there exists an edge that is considered by the algorithm.

Lemma 4.3. *The routing algorithm can always perform a step in the constrained half-θ_6-graph.*

Due to space constraints, the proof of this lemma can be found in Appendix B.

Theorem 4.4. *Given two vertices s and t in the half-θ_6-graph such that s and t can see each other and t lies in a positive subcone of s, there exists a 1-local routing algorithm that routes from s to t and is 2-competitive with respect to the Euclidean distance.*

Proof. We assume without loss of generality that $t \in C_0^s$. The routing algorithm will thus only take steps in $C_0^{v_i}$, $\overline{C}_1^{v_i}$, and $\overline{C}_2^{v_i}$, where v_i is an arbitrary vertex along the routing path. Let a and b be the upper left and right corner of T_{st}. To bound the length of the routing path, we first bound the length of each edge. We consider three cases: (a) edges in subcones of $\overline{C}_1^{v_i}$ or $\overline{C}_2^{v_i}$, (b) edges in subcones of $C_0^{v_i}$ that do not cross st, (c) edges in subcones of $C_0^{v_i}$ that cross st. For ease of notation we use v_0 and v_k to denote s and t.

Case (a): If edge $v_i v_{i+1}$ lies in a subcone of $\overline{C}_1^{v_i}$, let a_i be the upper corner of $T_{v_{i+1} v_i}$ (see Figure 8a). By the triangle inequality, we have that $|v_i v_{i+1}| \leq |v_i a_i| + |a_i v_{i+1}|$. The case where $v_i v_{i+1}$ lies in $\overline{C}_2^{v_i}$ is analogous.

Case (b): If edge $v_i v_{i+1}$ lies in a subcone of $C_0^{v_i}$ and does not cross st, let a_i and b_i be the upper left and right corner of $T_{v_i v_{i+1}}$ (see Figure 8b). If v_i lies to the left of st, we use that $|v_i v_{i+1}| \leq |v_i a_i| + |a_i v_{i+1}|$. If v_i lies to the right of st, we use that $|v_i v_{i+1}| \leq |v_i b_i| + |b_i v_{i+1}|$.

Case (c): If edge $v_i v_{i+1}$ lies in a subcone of $C_0^{v_i}$ and crosses st, we split it into two parts, one for each side of st (see Figure 8c). Let x be the intersection of st and $v_i v_{i+1}$. If u lies to the left of st, let a_i be the upper left corner of $T_{v_i x}$ and let b_i be the upper right corner of $T_{x v_{i+1}}$. By the triangle inequality, we have that $|v_i v_{i+1}| \leq |v_i a_i| + |a_i x| + |x b_i| + |b_i v_{i+1}|$. If u lies to the right of st, let a_i

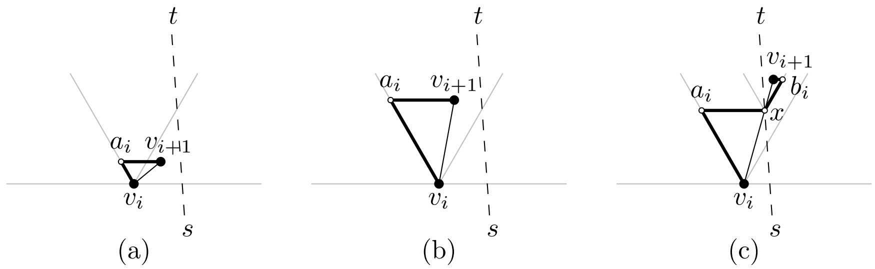

Fig. 8. Bounding the edge lengths: (a) an edge in a subcone of $\overline{C}_1^u$, (b) an edge in a subcone of C_0^u that does not cross st, and (c) an edge in a subcone of C_0^u that crosses st

be the upper left corner of $T_{xv_{i+1}}$ and let b_i be the upper right corner of T_{v_ix}. By triangle inequality, we have that $|v_iv_{i+1}| \leq |v_ib_i| + |b_ix| + |xa_i| + |a_iv_{i+1}|$.

To bound the length of the full path, let x and x' be two consecutive points where the routing path crosses st and let v_iv_{i+1} be the edge that crosses st at x and let $v_{i'}v_{i'+1}$ be the edge that crosses st at x'. Let a_x and b_x be the upper left and right corner of $T_{xx'}$. If the path between x and x' lies to the left of st, this part of the path is bounded by $|xa_i| + \sum_{j=i}^{i'-1} |a_jv_{j+1}| + \sum_{j=i+1}^{i'} |v_ja_j| + |a_{i'}x'|$. Since xa_i and all v_ja_j are parallel to xa_x and all a_xv_{j+1} are horizontal, we have that $|xa_i| + \sum_{j=i+1}^{i'} |v_ja_j| = |xa_x|$. Similarly, since $a_{i'}x'$ and all a_jv_{j+1} are parallel and have disjoint projections onto a_xx', we have that $\sum_{j=i}^{i'-1} |a_jv_{j+1}| + |a_{i'}x'| = |a_xx'|$. Thus, the length of a path to the left of st is at most $|xa_x| + |a_xx'|$. If the path between x and x' lies to the right of st, this part of the path is bounded by $|xb_i| + \sum_{j=i}^{i'-1} |b_jv_{j+1}| + \sum_{j=i+1}^{i'} |v_jb_j| + |b_{i'}x'| = |xb_x| + |b_xx'|$ (see Figure 9a).

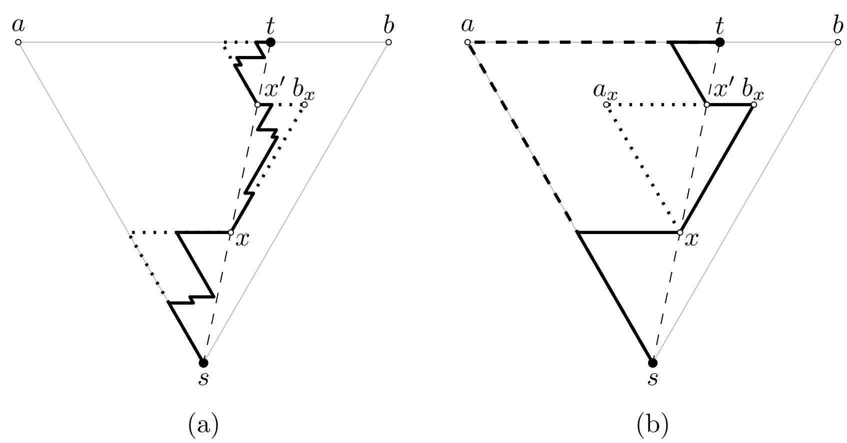

Fig. 9. Bounding the total length: (a) the bounds (solid lines) are unfolded (dotted lines) and (b) the unfolded bounds (solid lines) are flipped to the longer of the two sides (dotted lines) and unfolded again (dashed lines)

Next, we flip all unfolded bounds to the longer of the two sides at and bt: if $|at| \geq |bt|$, we replace all bounds of the form $|xb_x| + |b_x x'|$ by $|xa_x| + |a_x x'|$ and if $|at| < |bt|$, we replace all bounds of the form $|xa_x| + |a_x x'|$ by $|xb_x| + |b_x x'|$ (see Figure 9b). Note that this can only increase the length of the bounds. Finally, we sum these bounds and get that the sum is equal to $\max\{|sa| + |at|, |sa| + |bt|\}$, which is at most $2 \cdot |st|$. □

4.2 Routing on the Constrained θ_6-Graph

To route on the constrained θ_6-graph, we split it into two constrained half-θ_6-graphs: the constrained half-θ_6-graph oriented as in Figure 3 and the constrained half-θ_6-graph where positive and negative cones are inverted. When we want to route from s to t, we pick the constrained half-θ_6-graph in which t lies in a positive subcone of s, referred to as G^+ in the remainder of this section, and apply the routing algorithm described in the previous section. Since this routing algorithm is 1-local and 2-competitive, we obtain a 1-local and 2-competitive routing algorithm for the constrained θ_6-graph, provided that we can determine locally, while routing, whether an edge is part of G^+. When at a vertex u, we consider the edges in order of increasing angle with the horizontal halfline through u that intersects st.

Lemma 4.5. *While executing the positive routing algorithm for two visible vertices s and t, we can determine locally at a vertex u for any edge uv in the constrained θ_6-graph whether it is part of G^+.*

Proof. Suppose we color the edges of the constrained θ-graph red and blue such that red edges form G^+ and blue edges form the constrained half-θ_6-graph, where t lies in a negative subcone of s. At u, we need to determine locally whether uv is red. Since an edge can be part of both constrained half-θ_6-graphs, it can be red and blue at the same time. This makes it harder to determine whether an edge is red, since determining that it is blue does not imply that it is not red.

If v lies in a positive subcone of u, we need to determine if it is the closest vertex in that subcone. Since by construction of the constrained half-θ_6-graph, u is connected to the closest vertex in this subcone, it suffices to check whether this vertex is v. Note that if uv is a constraint, v lies in two subcones of u and hence we need to check if it is the closest vertex in at least one of these subcones.

If v lies in a negative subcone of u, we know that if it is not the closest visible vertex in that subcone, uv is red. Hence, it remains to determine for the edge to the closest vertex whether it is red: If it is the closest visible vertex, it is blue, but it may be red as well if u is also the closest visible vertex to v. Hence, we need to determine whether u is the closest vertex in $C^v_{i,j}$, a subcone of v that contains u. We consider two cases: (a) uv is a constraint, (b) uv is not a constraint.

Case (a): Since uv is a constraint, it cannot cross st. Since we are considering uv, all edges that make a smaller angle with the horizontal halfline through u that intersects st are not red. Hence, uv is either part of the boundary of the routing path or the constraint is contained in the interior of the region bounded by the routing path and st. However, by the invariant of Lemma 4.3, the region bounded by the routing path and st does not contain any constraints in its interior. Thus, uv is part of the boundary of the routing path and uv is red.

Case (b): If uv is not a constraint, let regions A and B be the intersection of C_i^v and the two subcones of u adjacent to $\overline{C}_i^u$ and let C be the intersection of $C_{i,j}^v$ and the negative subcone of u that contains v (see Figure 10). We first note that since uv lies in a negative subcone of u, the invariant of Lemma 4.3 implies that B is empty. Furthermore, since v is the closest visible vertex to u, C does not contain any vertices that can see u or v.

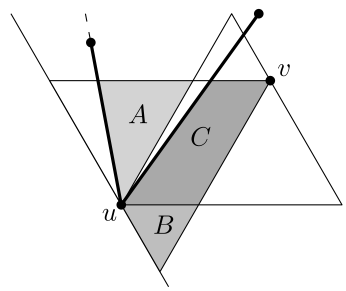

Fig. 10. Determining whether an edge is part of the constrained half-θ_6-graph

Since C does not contain any vertices that can see u or v, any constraint in $\overline{C}_i^u$ that has u as an endpoint and lies above uv, ensures that v cannot see A, i.e. it cannot block visibility of this region only partially. Hence, if such a constraint exists, u is the closest visible vertex to v in $C_{i,j}^v$, since neither B nor C contain any vertices visible to v. Therefore, uv is red.

If v can see A, we show that uv is red, if and only if the closest visible vertex in the subcone of u that contains A does not lie in A. We first show that if uv is red, then the closest visible vertex in the subcone of u that contains A does not lie in A. We prove the contrapositive of this statement. Since A is visible to v, u is not the endpoint of a constraint in $\overline{C}_i^u$ above uv. Hence, we have two visibility edges uv and ux and u is not the endpoint of a constraint intersecting the interior of triangle uxv. Thus, by Lemma 4.1, we have a convex chain between x and v. Let y be the vertex adjacent to v along this chain. Since the polygon defined by ux, uv, and the convex chain is empty and does not contain any constraints, y lies in $C_{i,j}^v$. Thus, u is not the closest visible vertex in $C_{i,j}^v$ and uv is not red.

Next, we show that if the closest visible vertex x in the subcone of u that contains A does not lie in A, then uv is red. We prove this by contradiction, so assume that uv is not red. This implies that there exists a vertex $y \in C_{i,j}^v$ that is visible to v and closer than u. Since B is empty and C does not contain any vertices that can see v, y lies in A. Since uv and vy are visibility edges and v is not the endpoint of a constraint intersecting the interior of triangle uyv, by Lemma 4.1 there exists a convex chain of visibility edges between u and y.

Furthermore, since C does not contain any vertices that can see u, the vertex adjacent to u along this chain lies in A. Since any vertex in A is closer to u than x, this leads to a contradiction, completing the proof. □

Routing Algorithm for the Constrained θ_6-Graph
Hence, to route on the constrained θ_6-graph, we apply the positive routing algorithm on G^+, while determining which edges are part of this constrained half-θ_6-graph. The latter can be determined as follows: If v lies in a positive subcone, we need to check whether it is the closest vertex in that subcone. If v lies in a negative subcone and it is not the closest vertex, it is part of the constrained half-θ_6-graph. Finally, if v is the closest vertex in a negative subcone, it is part of the constrained half-θ_6-graph if it is a constraint or the intersection of the cone of v that contains u and the subcone of C_{i-1}^u adjacent to $\overline{C}_i^u$ is empty.

4.3 Negative Routing on the Constrained Half-θ_6-Graph

To complement the positive routing algorithm on the constrained half-θ_6-graph, we also provide a negative routing algorithm on this graph. Due to space constraints, we refer the reader to Appendix C for details on the routing algorithm and only state the main result.

Theorem 4.6. *There exists an $O(1)$-memory 1-local 18-competitive routing algorithm for negative routing in the constrained half-θ_6-graph.*

References

1. R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins. Searching in the plane. *Inform. Comput.*, 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. P. Bose, A. Brodnik, S. Carlsson, E. D. Demaine, R. Fleischer, A. López-Ortiz, P. Morin, and I. J Munro. Online routing in convex subdivisions. *Int. J. Comput. Geom. App.*, 12(04):283–295, 2002.
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*, volume 7256 of *LNCS*, pages 85–96, 2012.
6. P. Bose and J. M. Keil. On the stretch factor of the constrained Delaunay triangulation. In *ISVD*, pages 25–31, 2006.
7. P. Bose and A. van Renssen. Upper bounds on the spanning ratio of constrained theta-graphs. In *LATIN*, volume 8392 of *LNCS*, pages 108–119, 2014.
8. K. Clarkson. Approximation algorithms for shortest path motion planning. In *STOC*, pages 56–65, 1987.
9. G. Das. The visibility graph contains a bounded-degree spanner. In *CCCG*, pages 70–75, 1997.
10. S. Misra, S. C. Misra, and I. Woungang. *Guide to Wireless Sensor Networks.* Springer, 2009.
11. H. Räcke. Survey on oblivious routing strategies. In *Math. Theo. Comput. Prac.*, volume 5635 of *LNCS*, pages 419–429, 2009.

A Proof of Theorem 3.1

Theorem 3.1. *No deterministic 1-local routing algorithm is $o(\sqrt{n})$-competitive, with respect to the shortest path, on all pairs of vertices of the θ_6-graph, regardless of the amount of memory it is allowed to use.*

Proof. Consider an $n \times n$ grid with $n \geq 16$ and shift every second row to the right by half a unit. We stretch the grid, such that each horizontal edge has length n (see Figure 11). Next, we replace each horizontal edge by a constraint to prevent vertical visibility edges. Finally, we add two additional vertices, origin s and destination t, centered horizontally at one unit below the bottom row and one unit above the top row, respectively.

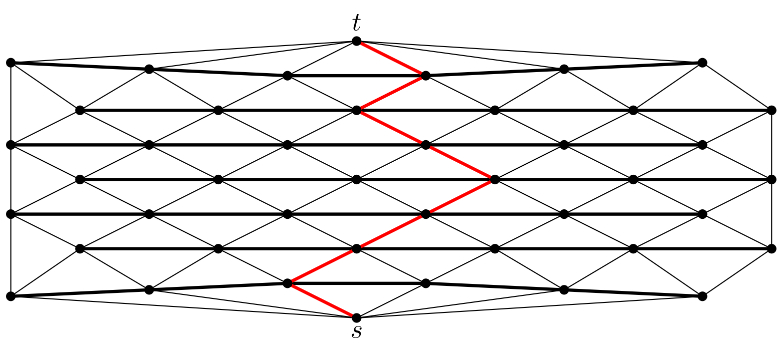

Fig. 11. The constrained θ_6-graph starting from a grid, using horizontal constraints to block vertical edges, and the red path of the routing algorithm

To conform to our general position assumption, we move all vertices by at most some arbitrarily small amount ϵ, such that no two points define a line parallel to one of the rays that define the cones and no three points are collinear. In particular, we ensure that all vertices on the bottom row have s as the closest vertex in one of their subcones and all vertices on the top row have t as the closest vertex in one of their subcones. This can be achieved by placing the bottom row on the upper hull of an ellipse and placing the top row on the lower hull of an ellipse. On this point set and these constraints, we build the constrained θ_6-graph G.

Consider any deterministic 1-local ∞-memory routing algorithm and let π be the path this algorithm takes when routing from s to t. We note that by construction, π consists of at least $n+1$ steps. If π consists of at least $n\sqrt{n}$ non-vertical steps, the total length of the path is $\Omega(n^2\sqrt{n})$, since we stretched the grid such that each edge has length $\Theta(n)$. However, G contains a path of length $O(n^2)$ between s and t: the path that follows a diagonal edge to the left, followed by a diagonal edge to the right, until it reaches t. Hence, in this case, the local routing algorithm is not $o(\sqrt{n})$-competitive.

Now, assume that π consists of $f(n)$ non-vertical steps, for $f(n) < n\sqrt{n}$. Consider the $2\sqrt{f(n)}$ neighbors of s at horizontal distance at most $n\sqrt{f(n)}$ from s. Since $f(n)$ is less than $n\sqrt{n}$ and G is a modified $n \times n$ grid with $n \geq 16$, these $2\sqrt{f(n)}$ neighbors exist. Next, consider the vertical lines through these $2\sqrt{f(n)}$ neighbors of s and let π' be the routing path π minus vertices s and t. We say that a vertex of π' *touches* a vertical line if it has a neighbor on that line. Hence, any vertex along π' touches at most 2 vertical lines. This implies that the total number of lines touched by the vertices along π' is at most $2f(n)$. Hence, on average, a line is touched no more than $2f(n)/2\sqrt{f(n)} = \sqrt{f(n)}$ times. This implies that there exists a vertical line that is touched at most $\sqrt{f(n)}$ times. Let u be the neighbor of s on the vertical line that is touched the fewest number of times.

We now create a new constrained θ_6-graph G' such that the deterministic 1-local routing algorithm follows the same path, but G' contains a short 'almost vertical' path via u. We start with s, t, and all vertices of π. Next, we add all vertices and constraints connected to these vertices in G. On this point set and these constraints, we build the constrained θ_6-graph G' (see Figure 12).

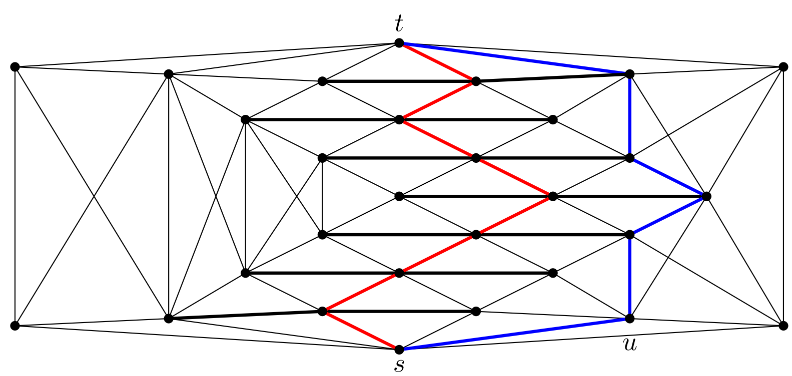

Fig. 12. The constrained θ_6-graph that looks the same from the red path of the routing algorithm, but has an almost vertical blue path

Since the horizontal distance between vertices is far larger than their vertical distance, an 'almost vertical' path from u to the top row of G' is formed. This almost vertical path is a path that is vertical whenever possible and uses detours to avoid path π (see Figure 13): If π arrives at a vertex v that has a neighbor on the vertical line through u, we avoid π by following one edge away from π, followed by an edge back to the vertical line through u (see Figure 13a). If π arrives at a vertex on the vertical line through u, we avoid the vertex before and after v on π as before, and meet π at v (see Figure 13b).

Since no edge along the left and right boundary of G touches the vertical line through u, this vertical line is touched by at most $\sqrt{f(n)}$ vertices of π and only $O\left(\sqrt{f(n)}\right)$ of these detour edges are required. Hence, the length of the path from u to the top row of G' has length $O\left(n\sqrt{f(n)}\right)$. Thus, G' contains a path from s to t of length $O\left(n\sqrt{f(n)}\right)$: Follow the edge from s to u, follow the almost vertical path from u to the top row of G', and follow the edge to t.

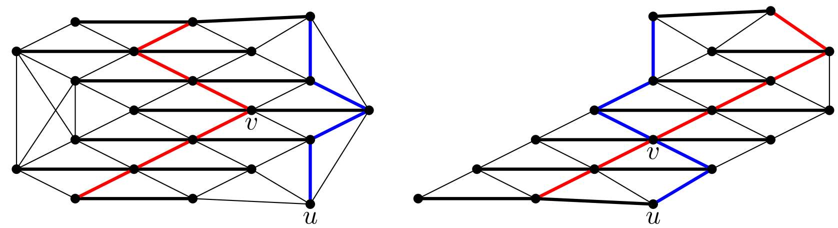

Fig. 13. The two types of detour: (a) when π does not visit the vertical line through u, (b) when π visits the vertical line through u

Since the 1-local routing algorithm is deterministic and the 1-local information of the vertices of π in G' is the same as in G, the algorithm stores the same data in its memory and follows the same path. In order to analyse the length of π, we consider two cases: π never follows a vertical edge along the left and right boundary of G, or π follows at least one edge along this boundary. If π follows no vertical edge along the boundary of G, it follows that the total length of this path is $\Omega(n \cdot f(n))$, since it does not follow any of the shortcut edges. Hence, the deterministic 1-local routing algorithm is not $o\left(\sqrt{f(n)}\right)$-competitive on G'. Thus, since π does not follow any vertical edges, $f(n)$ is at least $n+1$ and π is not $o(\sqrt{n})$-competitive.

If π follows at least one edge along this boundary, it follows that π has length at least $\Omega(n^2 + n \cdot f(n))$, as the left and right boundary of G are at distance $\Omega(n^2)$ from s and each of the $f(n)$ non-vertical edges has length $\Omega(n)$. However, since none of these vertical edges touch the vertical line through u, the almost vertical path still contains at most $O\left(\sqrt{f(n)}\right)$ detours. Thus, the length of the shortest path is at most $O\left(n\sqrt{n} + n \cdot \sqrt{f(n)}\right)$.

To complete the proof, we look at the number of non-vertical edges of π, i.e. $f(n)$. If $f(n) \leq n$, the length of π reduces to $\Omega(n^2)$ and the length of the almost vertical path reduces to $O(n\sqrt{n})$, which implies that π is not $o(\sqrt{n})$-competitive. If $f(n) > n$, the length of π reduces to $\Omega(n \cdot f(n))$ and the length of the almost vertical path reduces to $O\left(n \cdot \sqrt{f(n)}\right)$, which implies that π is not $o\left(\sqrt{f(n)}\right)$-competitive. Finally, since $f(n) > n$, this implies that π is not

$o(\sqrt{n})$-competitive. Hence, since G' can be constructed for any deterministic 1-local routing algorithm, we have shown that no deterministic 1-local routing algorithm is $o(\sqrt{n})$-competitive on all pairs of vertices. □

B Proof of Lemma 4.3

Lemma 4.3. *The routing algorithm can always perform a step in the constrained half-θ_6-graph.*

Proof. Given two vertices s and t such that s and t can see each other, we assume without loss of generality that $t \in C_0^s$. We maintain the following invariant (see Figure 14):

> **Invariant** Let x be the last intersection of an edge of the routing path with st (initially x is s), let $v_0, ..., v_k$ denote the endpoints of the edges following x as selected by the algorithm, and let x' be the intersection of st and the horizontal line through v_k. The simple polygon defined by $x, v_0, ..., v_k, x'$ is empty and does not contain any constraints.

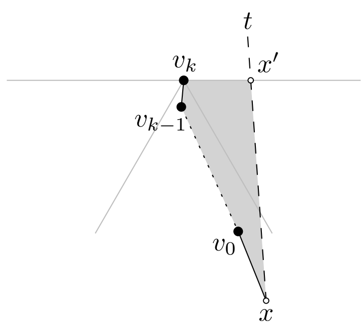

Fig. 14. By the invariant, the gray region is empty and does not contain any constraints

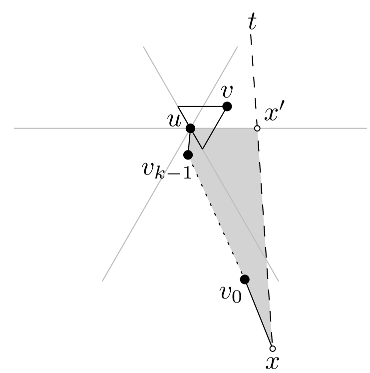

Fig. 15. By the invariant, the gray region is empty, so the path between u and v lies inside $T_{vu} \cap (C_0^u \cup \overline{C}_1^u)$

When the routing algorithm starts at s, it looks at the subcone that contains t. Since t is visible from s, this subcone contains at least one visible vertex. Hence, it also contains a closest visible vertex v_0 and by construction, s has an edge to v_0. Therefore, when the routing algorithm starts at s, it can follow an edge.

To see that the invariant is satisfied, we need to show that triangle sv_0x' is empty and does not contain any constraints in its interior. By construction s cannot be the endpoint of any constraints in the interior of sv_0x', hence since sx' and sv_0 are visibility edges, any constraint has at least one endpoint in sv_0x'.

Thus, it suffices to show that sv_0x' is empty. We prove this by contradiction, so assume that it is not empty. Since sv_0 and sx' are visibility edges and by construction s is not the endpoint of a constraint intersecting the interior of sv_0x', Lemma 4.1 gives us a convex chain of visibility edges between v_0 and x'. Since the region bounded by sv_0, sx', and this chain is empty and does not contain any constraints, the vertex along this chain that is closest to s is visible to s. However since every point in sv_0x' is closer to s than v_0, this contradicts the fact that v_0 is the closest visible vertex to s. Hence, triangle sv_0x' must be empty and the invariant is satisfied.

When the routing algorithm is at vertex u ($u \neq s$), we assume without loss of generality that u lies to the left of st. Let h be the halfplane below the horizontal line through t and let h' be the halfplane to the left of st. We need to show that u has at least one edge in the union of $C_0^u \cap h$ and $\overline{C}_1^u \cap h \cap h'$. We first show that there exists a vertex that is visible to u in the union of $C_0^u \cap h$ and $\overline{C}_1^u \cap h \cap h'$, by showing that such a vertex exists in the union of $C_0^u \cap h \cap h'$ and $\overline{C}_1^u \cap h \cap h'$. Since t lies in this region, we know that it is not empty. Consider all vertices in this region and let v be the vertex in this region that minimizes $\angle x'uv$. Note that we did not require there to be an edge between u and v. Since v minimizes $\angle x'uv$ and no constraint can cross st or ux', v is visible from u. We consider two cases: v lies in a subcone of C_0^u and v lies in a subcone of $\overline{C}_1^u$.

If v lies in $C_0^u \cap h \cap h'$, it follows from Lemma 4.2 and the fact that v is visible from u that there exists a path between u and v that lies inside T_{uv}. Since T_{uv} is contained in $C_0^u \cap h$, there exists an edge in $C_0^u \cap h$ and the routing algorithm can perform a step.

If v lies in $\overline{C}_1^u \cap h \cap h'$, it follows from Lemma 4.2 and the fact that v is visible from u that there exists a path between u and v that lies inside T_{vu}. Canonical triangle T_{vu} intersects three cones of u (see Figure 15): C_0^u, $\overline{C}_1^u$, and C_2^u. Since the routing algorithm follows edges in C_0^u or $\overline{C}_1^u$, the routing path reaches u by following edge $v_{k-1}u$ that lies in either $\overline{C}_0^u$ or C_1^u. This implies that $T_{vu} \cap C_2^u$ is contained in the region of the invariant and is therefore empty. Hence, the first edge on the path from u to v lies in either $C_0^u \cap h$ or $\overline{C}_1^u \cap h \cap h'$ and the algorithm can perform a step.

It remains to show that after the algorithm takes a step, the invariant is satisfied at the new vertex v. Let uv be the edge that the algorithm followed and let x'' be the intersection of st and the horizontal line through v. We consider three cases (see Figure 16): (a) v lies in a subcone of $\overline{C}_1^u$, (b) v lies in a subcone of C_0^u and uv does not cross st, and (c) v lies in a subcone of C_0^u and uv crosses st.

Case (a): If v lies in a subcone of $\overline{C}_1^u$, we need to show that the quadrilateral $uvx''x'$ is empty and does not contain any constraints (see Figure 16a). We first show that u cannot be the endpoint of a constraint intersecting the interior of $uvx''x'$. We prove this by contradiction, so assume it is and let y be the other endpoint of the constraint. We first note that $\angle x'uy < \angle x'uv$. We look at $C_{1,j}^y$, the subcone of C_1^y that lies below uy, and let z be the lowest vertex in this subcone. If u is the closest visible vertex in this subcone, uy would be an edge, which contradicts that v minimizes $\angle x'uv$. Otherwise, since z is the lowest vertex in $C^y_{1,j}$, the visible region of T_{zu} is empty and uz is an edge. However, since $\angle x'uz < \angle x'uy < \angle x'uv$, we have a contradiction. Thus u cannot be the endpoint of a constraint intersecting the interior of $uvx''x'$.

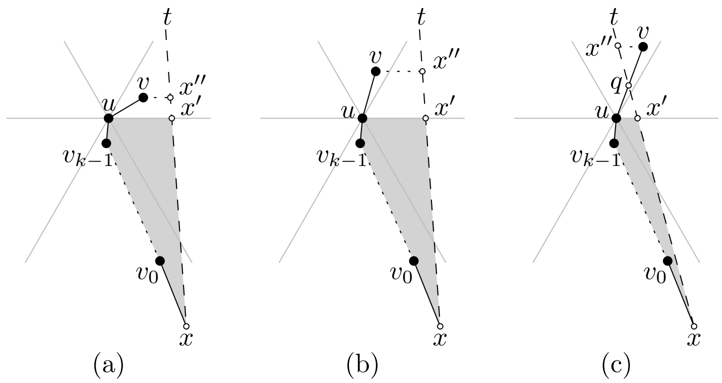

Fig. 16. The three types of steps the algorithm can take: (a) v lies in $\overline{C}^u_1$, (b) v lies in C^u_0 and uv does not cross st, and (c) v lies in C^u_0 and uv crosses st

Since u is not the endpoint of a constraint intersecting the interior of $uvx''x'$, and uv, ux', and $x'x''$ are visibility edges, any constraint intersecting the interior of $uvx''x'$ has at least one endpoint in $uvx''x'$. Thus it suffices to show that $uvx''x'$ is empty. We prove this by contradiction, so assume that $uvx''x'$ is not empty and let y be the lowest vertex in $uvx''x'$. Let $C^y_{1,j}$ be the subcone of C^y_1 that contains u. Vertex u is visible to y, since any constraint crossing uy has an endpoint in $\overline{C}^u_1$ below y, contradicting that y is the lowest vertex, or in the region bounded by $x, v_0, ..., v_{k-1}, u, x'$ which contradicts the invariant. Hence y has an edge in $C^y_{1,j}$. This edge cannot be to u since $\angle x'uy < \angle x'uv$. Since y is the lowest vertex in $uvx''x'$, it cannot have an edge to a vertex in $uvx''x'$. Since by the invariant the region bounded by $x, v_0, ..., v_{k-1}, u, x'$ is empty, the edge of y in $C^y_{1,j}$ must cross uv. However, this contradicts the fact that the constrained half-θ_6-graph is plane. Thus, $uvx''x'$ is empty of both vertices and constraints.

Case (b): If v lies in a subcone of C^u_0 and uv does not cross st, we again need to show that the quadrilateral $uvx''x'$ is empty and does not contain any constraints (see Figure 16b). We first show that $uvx''x'$ is empty. We prove this by contradiction, so assume that $uvx''x'$ is not empty and let y be the lowest vertex in $uvx''x'$. We consider two cases: y lies in $\overline{C}^u_1$ and y lies in C^u_0.

If y lies in $\overline{C}^u_1$, let $C^y_{1,j}$ be the subcone of y that contains u. Vertex u is visible to y, since any constraint crossing uy has an endpoint in $\overline{C}^u_1$ below y, contradicting that y is the lowest vertex, or in the region bounded by $x, v_0, ..., v_{k-1}, u, x'$ which contradicts the invariant. Hence y has an edge in $C^y_{1,j}$. This edge cannot be to u since $\angle x'uy$ is smaller than $\angle x'uv$. Furthermore, since y is the lowest vertex in $uvx''x'$, it cannot have an edge to a vertex in $uvx''x'$. Finally, since by the invariant the region bounded by $x, v_0, ..., v_{k-1}, u, x'$ is empty, the edge of y in $C^y_{1,j}$ must cross uv, which contradicts the fact that the constrained half-θ_6-graph is plane.

If y lies in a subcone of C^u_0 and y is visible to u, uy would be an edge and $\angle x'uy < \angle x'uv$. So, assume that y is not visible from u. This means that there is a constraint that crosses uy. Since the line st and the edges of the region bounded by $x, v_0, ..., v_{k-1}, u, x'$ are visibility edges, the lower endpoint of this constraint must lie in $x, v_0, ..., v_{k-1}, u, v, x''$. By the invariant, it cannot lie in $x, v_0, ..., v_{k-1}, u, x'$, so it must lie in $uvx''x'$ and below y. However, this contradicts that y is the lowest vertex in $uvx''x'$. Since we arrived at a contradiction in both cases, we conclude that quadrilateral $uvx''x'$ is empty.

Next, we show that $uvx''x'$ does not contain any constraints. Since $uvx''x'$ is empty, a the only way a constraint can intersect it, is when u is one of its endpoints. Hence, it remains to show that u cannot be the endpoint of a constraint intersecting the interior of $uvx''x'$. We prove this by contradiction, so assume it is and let y be the other endpoint of the constraint. Since $uvx''x'$ is empty, uy crosses vx''. Since st is a visibility edge, uy cannot cross it. Vertex y cannot lie in $\overline{C}^u_1 \cap h'$, since this would imply that either uy is an edge or there exists a vertex z in the subcone of y below uy that contains u, which in combination with Lemma 4.2 implies that there exists a path between y and u that lies below uy. Since both alternatives contradict that v minimizes $\angle x'uv$, y cannot lie in $\overline{C}^u_1 \cap h'$. Hence, it remains to consider the case where y lies in a subcone of C^u_0. Let $C^u_{0,j}$ be the subcone of C^u_0 to the right of uy. If y lies below t, $C^u_{0,j}$ contains a closest visible vertex whose angle with ux' is less than $\angle x'uv$, contradicting that the routing algorithm routes to v.

If y lies above t, let z be the lowest vertex in the union of $C^u_{0,j}$ and $\overline{C}^u_1 \cap h'$. Since this region contains t, it is not empty and such a vertex z exists. If $z \in C^u_{0,j}$, it is the closest vertex in $C^u_{0,j}$. If $z \in \overline{C}^u_1$, u is the closest vertex to z. We note that in both cases z is visible to u, since any constraint blocking it would have an endpoint below z. Hence, both cases result in an edge uz. However, since $\angle x'uz < \angle x'uv$, this contradicts that the routing algorithm routed to v. Thus, u cannot be the endpoint of a constraint intersecting the interior of $uvx''x'$.

Case (c): If v lies in a subcone of C^u_0 and uv crosses st, let q be the intersection of uv and st. We need to show that the triangles uqx' and $qx''v$ are empty and do not contain any constraints (see Figure 16c). The proof that uqx' empty and does not contain any constraints is analogous to the previous case.

We prove that $qx''v$ is empty by contradiction, so assume that $qx''v$ is not empty. Since qx'' and qv are visibility edges, we can apply Lemma 4.1 and we obtain a vertex y in $qx''v$ that is visible from q. If y is visible from u, v is not the closest vertex and edge uv would not exist. If y is not visible from u, we note that uq is visible and apply Lemma 4.1 on triangle uyq. This gives us a vertex z that is visible to u and closer to u than v, again contradicting the existence of edge uv. Hence, triangle $qx''v$ is empty.

Finally, we show that $qx''v$ does not contain any constraints. Since qx'' and qv are visibility edges and $qx''v$ is empty, any constraint intersecting the interior of $qx''v$ must have q as an endpoint. However, since q is not a vertex, it cannot be the endpoint of a constraint. □

C Negative Routing on the Constrained Half-θ_6-Graph

In this section, we provide an $O(1)$-memory 1-local routing algorithm for the case where the destination t lies in a negative subcone of the origin s. For ease of exposition, we assume that t lies in a subcone of $\overline{C}_0^s$. The $O(1)$-memory 1-local routing algorithm finds a path from s to t of length at most $2 \cdot |st|$ and travels a total distance of at most $18 \cdot |st|$ to do so.

The remainder of this section is structured as follows: First, we identify a set of conditions that edges need to meet in order to be considered by the routing algorithm. Unfortunately, one of these conditions cannot be checked 1-locally. Therefore, we replace it with a set of conditions that exclude edges that are guaranteed not to satisfy the original condition and can be checked 1-locally.

We proceed to describe the edges considered by the negative routing algorithm. Given a vertex v and all neighbors of v whose projection along the bisector of C_0^t is closer to t than the projection of v, we number the neighbors $u_0, ..., u_k$ of v in counterclockwise order, starting from the horizontal half-line to the left of v (see Figure 17). We create $k + 1$ regions around v:

- We create $k - 1$ triangular regions vu_iu_{i+1} for $0 \leq i \leq k - 1$.
- We create one unbounded region using edge vu_0 and the two horizontal half-lines starting at v and u_0 directed towards the left.
- We create one unbounded region using edge vu_k and the two horizontal half-lines starting at v and u_k directed towards the right.

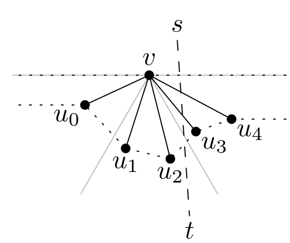

Fig. 17. Triangle vu_2u_3 is the last region of v intersected by st

The *last region* of v intersected by st is defined as the last of these regions that is encountered when following st from s to t. In Figure 17, the region defined by v, u_2, and u_3 is the last region of v intersected by st.

We consider an edge uv for our routing algorithm when it satisfies the following three conditions:

- Vertices u and v lie inside or on the boundary of T_{ts}.
- Edge uv is part of the last region of v that is intersected by st.
- Edge uv is the edge that the positive routing algorithm picks at u when routing from t to s. Note that for this condition, we do not require that u is part of the positive routing path, but only that should the positive routing path reach u, edge uv is the edge it would select for its next step.

Given s and t, the first two requirements can be checked using only the location of s and t and 1-local information, i.e. the neighbors of the current vertex. The last requirement, on the other hand, may need 2-local information as it involves the neighbors of the neighbors of v. Hence, instead of using this last requirement, we ignore the edges that can never satisfy it and show that we can route competitively and 1-locally on the graph G formed by the edges that meet the first two requirements.

Since t lies in a subcone of $\overline{C}_0^s$, the edges that define the last intersected region of a vertex v can lie in three cones: C_1^v, $\overline{C}_0^v$, and C_2^v. Since edges in the C_1^v and C_2^v of the negative routing algorithm correspond to edges in $\overline{C}_1^u$ and $\overline{C}_2^u$ of the positive routing algorithm (applied from t to s), the positive routing algorithm never follows these edges if they intersect st. Hence, these edges need not be considered by the negative routing algorithm (see Figure 18a).

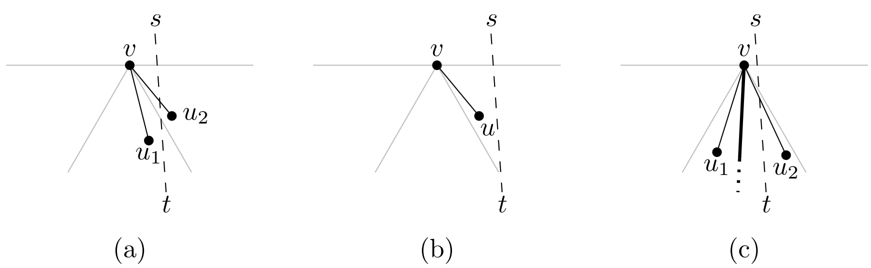

Fig. 18. The edges ignored by the negative routing algorithm: (a) edge u_2v is ignored since it intersects st, (b) edge uv is ignored since C_2^v is intersected by st, (c) edge u_1v is ignored since it lies in a subcone that is not intersected by st and u_1vu_2 is intersected by a constraint that has v as an endpoint

We also do not need to consider edges in C_1^v and C_2^v when that cone is intersected by st (see Figure 18b): Assume C_1^v is intersected by st. Since we are considering edge uv, it cannot cross st. Hence, st intersects cone C_1^u, but more importantly st intersects $\overline{C}_2^u$. Hence, if the positive routing algorithm reaches u, it continues by following an edge in $\overline{C}_2^u$ or C_0^u. Since C_1^v corresponds to $\overline{C}_1^u$, no edge in this cone is followed by the positive routing algorithm, and we can ignore it.

Finally, we ignore edges that lie in a subcone that is not intersected by st when v is the endpoint of a constraint that intersects the interior of the last region of v that is intersected by st (see Figure 18c): If v is the endpoint of a constraint that intersects the interior of the last region of v that is intersected by st, we do not consider the edge that is not intersected by st. We can ignore this edge, since by the invariant, the region between the routing path and st does not contain any constraints.

Since these conditions can be checked using only s, t, v, the neighbors of v, and the constraints incident to v, we can determine 1-locally whether to consider an edge. Hence, the graph G on which we route is the graph formed by all edges uv such that:

- Vertices u and v lie inside or on the boundary of T_{ts}.
- Edge uv is part of the last region of v that is intersected by st.
- Edge uv does not meet any of the following three conditions:
  - Edge uv lies in C_1^v or C_2^v and crosses st.
  - Edge uv lies in C_1^v or C_2^v and this cone is intersected by st.
  - Edge uv lies in a subcone that is not intersected by st and v is the endpoint of a constraint that intersects the interior of the last region of v that is intersected by st.

In the remainder of this section, for ease of exposition, we consider each edge of G to be oriented upward: Let u' and v' be the projections of u and v along the bisector of C_0^t. Edge uv is oriented from u to v if and only if $|tu'| \leq |tv'|$. Note that this does not imply that u lies in a negative cone of v. We proceed to prove that every vertex with two incoming edges is part of the positive routing path when routing from t to s.

Lemma C.7. *Every vertex with in-degree 2 in G that is reached by the negative routing algorithm is part of the positive routing path from t to s.*

Proof. Let v be a vertex of in-degree 2 that is reached by the negative routing algorithm. Let u and w be the other endpoints of these edges to v, such that the projection of u along the bisector of T_{ts} is closer to t than the projection of w (see Figure 19). Since both uv and wv are part of the last intersected region of v, vertices u and w must lie on opposite sides of st. This implies that the positive routing algorithm reaches at least one of them when routing from t to s, since by the invariant the region between the routing path and st is empty. Thus it suffices to show that from both u and w the positive routing algorithm eventually reaches v.

If the positive routing algorithm reaches w, we show that it would follow the edge to v. Let x be the intersection of uv and the horizontal line through w (see Figure 19). First, we show that triangle vwx is empty. If w lies in a subcone of C_1^v or C_2^v, u lies in a subcone of $\overline{C}_0^v$, since otherwise one of the two edges would cross st and be ignored. Since vw and vx are visibility edges and v is not the endpoint of a constraint intersecting the interior of vwx, it follows from Lemma 4.1 that if vwx is not empty, there exists a convex chain of visibility edges between w and x and the region bounded by this chain, vw, and vx is empty. Let y be the topmost vertex along this convex chain and note that y is visible to v. If y lies

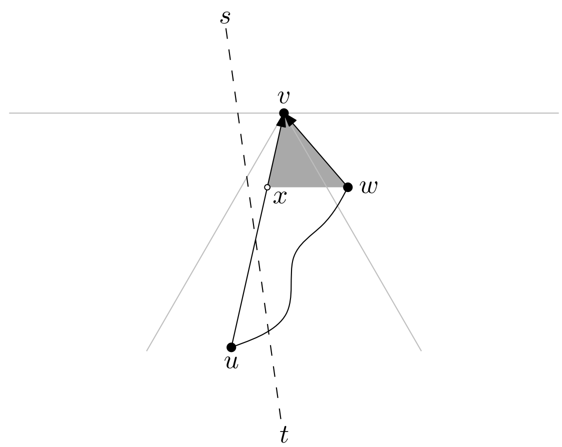

Fig. 19. Vertex v has in-degree 2

in the same cone of v as w, it also lies in the same subcone of v as w, since v is not the endpoint of a constraint intersecting the interior of vwx. However, this implies that w is not the closest visible vertex to v in this subcone, contradicting that vw is an edge. If y lies in $\overline{C}_0^v$, y has an edge in its subcone that contains v, since v is a visible vertex in that subcone. This edge cannot cross vw and vu, since the constrained half-θ_6-graph is plane, and it cannot be connected to a vertex in the region bounded by the convex chain, vw, and vx, since it is empty. Finally, since y is the topmost vertex along the convex chain, the edge cannot connect y to another vertex of the convex chain. Hence, y would have an edge to v, contradicting that vu and vw are consecutive edges around v. We conclude that triangle vwx is empty.

Using an analogous argument, it can be shown that if u lies in a subcone of C_1^v or C_2^v, w lies in $\overline{C}_0^v$ and the existence of a vertex in vwx would contradict that uv is an edge or that u and w are consecutive edges around v. If both u and w lie in a subcone of $\overline{C}_0^v$, the argument reduces to the case where y lies in $\overline{C}_0^v$, again contradicting that u and w are consecutive edges around v. Hence, since vwx is empty, the positive routing algorithm routes to v when it reaches w, since it minimizes angle $\angle xwv$.

Next, we look at the case where the positive routing path reaches u. If it follows edge uv, we are done. If it does not follow edge uv, let z be the other endpoint of the edge the positive routing algorithm follows at u. By construction of the positive routing path, we know that the projection of z on the bisector of C_0^t lies further from t than the projection of u. Since the constrained half-θ_6-graph is plane, the path from z to s cannot cross uv or wv, and since the positive routing path is monotone with respect to the bisector of C_0^t, it cannot go down and around or through u. Furthermore, since the region enclosed by the positive routing path and st is empty, the path also cannot go around w without passing through w. Finally, since uv and wv are consecutive edges around v, the path from z to s cannot reach v by arriving from an edge between uv and wv. Hence, w must lie on the path from z to s. Thus, since we previously showed that when the positive routing algorithm reaches w, it routes to v, vertex v is also reached when the positive routing path reaches u. □

Negative Routing Algorithm for the Constrained Half-θ_6-Graph
Routing from s to t now comes down to searching for a vertex that has in-degree 2 on one of the two paths leaving s. When such a vertex v is found, we need to find the next vertex that has in-degree 2 on one of the two paths leaving v. This process is repeated until we reach t. A single instance of this problem, i.e. finding the next vertex has in-degree 2 from another vertex can be viewed as searching for a specific point on a line. This problem has been studied extensively and a search strategy that is 9-competitive was presented by Baeza-Yates *et al.* [1]: We start by following the shorter of the two edges connected to s and call this distance 1. If we reached a vertex with in-degree 2, we are done. Otherwise, we go back to s and follow the other path up to distance 2 from s. Again, if we reached a vertex with in-degree 2, we are done. Otherwise, we go back to s and follow the first path up to distance 4 from s. This process of backtracking and doubling the allowed travel distance is repeated until a vertex with in-degree 2 is reached. Since this strategy needs to keep track of the distance traveled, it uses $O(1)$-memory. Hence, we apply this search strategy and perform the following actions when we reach an unvisited vertex v:

– If v has in-degree 2, v is part of the positive routing path and we restart the searching strategy from v.
– If v has in-degree 1, we proceed to its neighbor u if we have enough budget left to traverse the edge. At u we check whether the positive routing algorithm would follow edge uv. If this is not the case, we know that v was a dead end and the path on the opposite side of st is part of the positive routing path. Hence, we backtrack and follow the path on the opposite side of st to the last visited vertex on that side.
– If v has in-degree 0, it is a dead end and we backtrack like in the previous case.

We conclude this section by showing that the above $O(1)$-memory 1-local routing algorithm has a routing ratio of at most 9 times the length of the positive routing path, which implies an 18-competitive 1-local routing algorithm for negative routing in the constrained half-θ_6-graph.

Theorem C.8. *There exists an $O(1)$-memory 1-local 18-competitive routing algorithm for negative routing in the constrained half-θ_6-graph.*

Proof. Let p be the last vertex where the search strategy was restarted — initially p is s. We prove the theorem by showing that when we restart the search strategy at vertex q, we traveled at most 9 times the distance along the positive routing path between p and q. If we restart the search strategy because we reached a vertex of in-degree 2, this follows directly from the fact that the search strategy is 9-competitive, i.e. we found the vertex we are looking for and we spent at most 9 times the distance along the positive routing path between p and q.

If we reach a vertex v with in-degree 0 or we traverse an edge vu and the positive routing algorithm would not have routed from u to v, we backtrack to p and traverse the path on the opposite side of st. We follow this path until we reach w, the last vertex traversed on this side of st. Unfortunately, w is too close to p to prove that the total length traveled is at most 9 times the distance along the positive routing path between p and w. However, w must have in-degree 1: Since w is part of the positive routing path, it cannot have in-degree 0, and since we did not restart the search strategy when we reached w the previous time, it cannot have in-degree 2. Hence, it has in-degree 1 and it follows that the vertex q to which w is connected is also part of the positive routing path. Since the distance along the positive routing path between p and v is at most 2 times the distance along the positive routing path between p and q, an argument analogous to the one used by Baeza-Yates *et al.* [1] shows that we traversed at most 9 times the distance along the positive routing path between p and q to reach q. □

D Lower Bound on the Negative Routing Algorithm

In this section we show that the negative routing algorithm described in the previous section cannot be guaranteed to reach t while traveling less than $2\sqrt{39} \cdot |st| \approx 12.48 \cdot |st|$. This situation is shown in Figure 20: We place a vertex r_1 almost horizontally to the right of s at distance 1, followed by a vertex l_1 almost horizontally to the left of s at distance 2, followed by a vertex r_2 almost horizontally to the right of s at distance 4. Once we reach the corners of T_{ts} at l_2 and r_3, we proceed down along the boundary of T_{ts} and place vertices l_3 and r_4 such that the distance between s and l_3 via l_2 is 8 and the distance between s and r_4 via r_3 is 16. Finally, we place vertices l_4 and r_5 arbitrarily close to t. The positive routing path from t to s would route to r_5, r_4, r_3, r_2, r_1, and finally s.

The negative routing algorithm on the other hand would try both sides, going back through s each time it switches sides: go to r_1, go to l_1, go to r_2, go to l_3 (via l_2), go to r_5 (via r_3), go to l_4 (via l_2), and finally go to t (via s and r_5). We can pick the edge lengths between the vertices in such a way that each time the next vertex along one of the two sides is reached (other than l_4), the negative routing algorithm runs out of budget and needs to backtrack to try the other side. The total length traveled this way is the sum of:

- $2 \cdot \delta(s, r_5)$, for going back and forth from s until the step before r_5 is reached for the first time,
- $2 \cdot \delta(s, r_5)$, for going to r_5 and back to s when the negative routing algorithm almost reaches t,
- $2 \cdot \delta(s, l_4)$, for going down the wrong path (and back up) after reaching r_5,
- $\delta(s, t)$, for finally reaching t,

where $\delta(x, y)$ is the distance along the negative routing path between x and y. Since r_5 can be arbitrarily close to t, this sums up to $5 \cdot \delta(s, t) + 2 \cdot \delta(s, l_4)$.

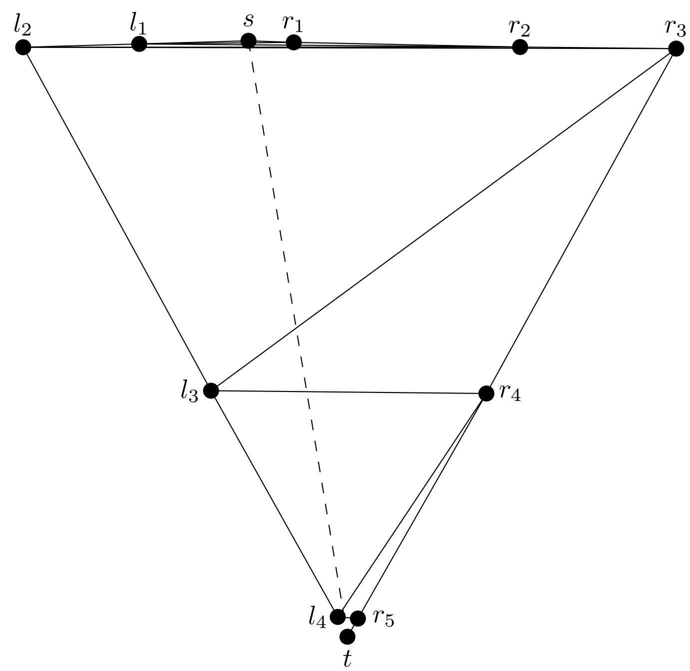

Fig. 20. The situation where the negative routing algorithm uses $2\sqrt{39} \cdot |st|$ to reach t

Let α be the angle between the bisector of T_{ts} and ts. Using the law of sines, we can express $\delta(s,t)$ and $\delta(s,l_4)$ as follows:

$$\begin{aligned}
\delta(s,t) &= |sr_3| + |r_3t| \\
&= \left(\frac{\sin\left(\frac{\pi}{6}+\alpha\right)}{\sin\left(\frac{\pi}{3}\right)} + \frac{\sin\left(\frac{\pi}{2}-\alpha\right)}{\sin\left(\frac{\pi}{3}\right)}\right) \cdot |st| \\
&= \left(\sqrt{3}\cdot\cos\alpha + \sin\alpha\right) \cdot |st|
\end{aligned}$$

$$\begin{aligned}
\delta(s,l_4) &= |sl_2| + |l_2l_4| \\
&= \left(\frac{\sin\left(\frac{\pi}{6}-\alpha\right)}{\sin\left(\frac{\pi}{3}\right)} + \frac{\sin\left(\frac{\pi}{2}-\alpha\right)}{\sin\left(\frac{\pi}{3}\right)}\right) \cdot |st| \\
&= \left(\sqrt{3}\cdot\cos\alpha - \sin\alpha\right) \cdot |st|
\end{aligned}$$

Thus, the total distance traveled by the negative routing algorithm becomes:

$$\begin{aligned} &5 \cdot \delta(s,t) + 2 \cdot \delta(s,l_4) \\ &= 5 \cdot \left(\sqrt{3} \cdot \cos\alpha + \sin\alpha\right) \cdot |st| + 2 \cdot \left(\sqrt{3} \cdot \cos\alpha - \sin\alpha\right) \cdot |st| \\ &= \left(7\sqrt{3} \cdot \cos\alpha + 3\sin\alpha\right) \cdot |st| \end{aligned}$$

When maximizing this function over α, with $0 \leq \alpha \leq \pi/6$, we find the maximum at $\alpha \approx 0.2425$, where the function has value $2\sqrt{39} \cdot |st| \approx 12.48 \cdot |st|$.