Constrained Routing Between Non-Visible Vertices[⋆]

Prosenjit Bose[1], Matias Korman[2], André van Renssen[3,4], and Sander Verdonschot[1]

[1] School of Computer Science, Carleton University, Ottawa, Canada. `jit@scs.carleton.ca`, `sander@cg.scs.carleton.ca`
[2] Tohoku University, Sendai, Japan. `mati@dais.is.tohoku.ac.jp`
[3] National Institute of Informatics, Tokyo, Japan. `andre@nii.ac.jp`
[4] JST, ERATO, Kawarabayashi Large Graph Project.

Abstract. Routing is an important problem in networks. We look at routing in the presence of line segment constraints (i.e., obstacles that our edges are not allowed to cross). Let P be a set of n vertices in the plane and let S be a set of line segments between the vertices in P, with no two line segments intersecting properly. We present the first 1-local $O(1)$-memory routing algorithm on the *visibility graph* of P with respect to a set of constraints S (i.e., it never looks beyond the direct neighbours of the current location and does not need to store more than $O(1)$-information to reach the target). We also show that when routing on any triangulation T of P such that $S \subseteq T$, no $o(n)$-competitive routing algorithm exists when only considering the triangles intersected by the line segment from the source to the target (a technique commonly used in the unconstrained setting). Finally, we provide an $O(n)$-competitive 1-local $O(1)$-memory routing algorithm on any such T, which is optimal in the worst case, given the lower bound.

1 Introduction

Routing is a fundamental problem in graph theory and networking. What makes this problem challenging is that often in a network the routing strategy must be *local*, i.e. the routing algorithm must decide which vertex to forward a message to based solely on knowledge of the current vertex, its neighbors and a constant amount of additional information (such as the source and destination 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 vertices and weighted by the Euclidean distance between their endpoints. Geometric routing algorithms are important in wireless sensor networks

[⋆] P. B. is supported in part by NSERC. M. K. was partially supported by MEXT KAKENHI Nos. 12H00855, 15H02665, and 17K12635. A. v. R. was supported by JST ERATO Grant Number JPMJER1305, Japan. S. V. is supported in part by NSERC, the Ontario Ministry of Research and Innovation, and the Carleton-Fields Postdoctoral Award.

(see [11] and [12] 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 on this problem has focused on the situation where the network is constructed by taking a subgraph of the complete Euclidean graph, i.e. the graph that contains an edge between every pair of vertices and the length of this edge is the Euclidean distance between the two vertices. We study this problem in a more general setting with the introduction of *line segment constraints* S. Specifically, let P be a set of n vertices in the plane and let S be a set of line segments between the vertices in P, with no two line segments properly intersecting (i.e., anywhere except at the endpoints). 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 $Vis(P, S)$, 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 in the context of motion planning amid obstacles. Clarkson [8] was one of the first to study this problem. He showed how to construct a $(1 + \epsilon)$-spanner of $Vis(P, S)$ 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 between u and v in G. The smallest value t for which H is a t-spanner is the *spanning ratio* or *stretch factor* of H. Following Clarkson's result, Das [9] showed how to construct a spanner of $Vis(P, S)$ with constant spanning ratio and constant degree. Bose and Keil [4] showed that the Constrained Delaunay Triangulation is a 2.42-spanner of $Vis(P, S)$. 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 $Vis(P, S)$ [2] and all constrained Θ-graphs with at least 6 cones were shown to be spanners as well [7].

Spanners of $Vis(P, S)$ are desirable because they are sparse and the bounded stretch factor certifies that paths do not make large detours. However, it is not known how to route locally on them. To address this issue, we look at 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 vertex, the current vertex, all vertices that can be seen from the current vertex and a constant amount of memory. We define this model formally in the next section. Furthermore, we study *competitiveness* of our routing algorithms, i.e. the ratio of the length of the path followed by the routing algorithm and the length of the shortest path in the graph.

In the constrained setting, routing has not been studied much. Bose *et al.* [3] showed that it is possible to route locally and 2-competitively between any two visible vertices in the constrained Θ_6-graph. Additionally, an 18-competitive routing algorithm between any two visible vertices in the constrained half-Θ_6-

graph was provided. While it seems like a serious shortcoming that these routing algorithms only route between pairs of visible vertices, in the same paper the authors also showed that no deterministic local routing algorithm is $o(\sqrt{n})$-competitive between all pairs of vertices of the constrained Θ_6-graph, regardless of the amount of memory it is allowed to use.

In this paper, we develop routing algorithms that work between any pair of vertices in the constrained setting. We provide a non-competitive 1-local routing algorithm on the visibility graph of P with respect to a set of constraints S. We also show that when routing on any triangulation T of P such that $S \subseteq T$, no $o(n)$-competitive routing algorithm exists when only considering the triangles intersected by the line segment from the source to the target (a technique commonly used in the unconstrained setting). Finally, we provide an $O(n)$-competitive 1-local routing algorithm on T, which is optimal in the worst case, given the lower bound. Prior to this work, no local routing algorithms were known to work between any pair of vertices in the constrained setting.

2 Preliminaries

The Θ_m-graph plays an important role in our routing strategy. We begin with their definitions. Define a *cone* C to be the region in the plane between two rays originating from a vertex referred to as the apex of the cone. When constructing a (constrained) Θ_m-graph, for each vertex u consider the rays originating from u with the angle between consecutive rays being $2\pi/m$. Each pair of consecutive rays defines a cone. The cones are oriented such that the bisector of some cone coincides with the vertical halfline through u that lies above u. Let this cone be C_0 of u and number the cones in clockwise order around u (see Fig. 1). The cones around the other vertices have the same orientation as the ones around u. We write C_i^u to indicate the i-th cone of a vertex u, or C_i if u is clear from the context. For ease of exposition, we only consider point sets in general position: no two points lie on a line parallel to one of the rays that define the cones, no two points lie on a line perpendicular to the bisector of a cone, and no three points are collinear.

Let vertex u be an endpoint of a constraint c and let the other endpoint v that lies in cone C_i^u (if any). The lines through all such constraints c split C_i^u into several *subcones* (see Fig. 2). We use $C_{i,j}^u$ to denote the j-th subcone of C_i^u (again, numbered in clockwise order). 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.

We now introduce the constrained Θ_m-graph: for each subcone $C_{i,j}$ of each vertex u, add an edge from u to the closest vertex in that subcone that can see u, 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, $v \in C_{i,j}^u$, and for all points $w \in C_{i,j}^u$ that can see u, $|uv'| \leq |uw'|$, where v' and w' denote the projection of v and w on the bisector of C_i^u and $|xy|$ denotes

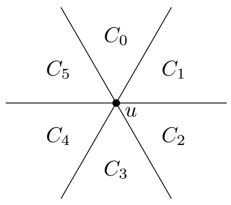

Fig. 1. The cones whit apex u in the Θ_6-graph. All points of S have exactly six cones.

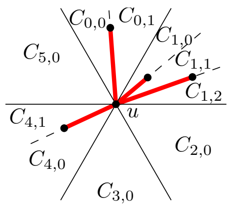

Fig. 2. The subcones with apex u in the constrained Θ_6-graph (constraints denoted as red thick segments).

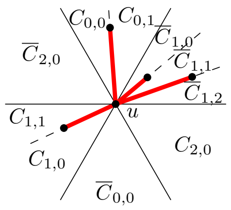

Fig. 3. If we consider the half-Θ_6-graph instead, we have the same amount of cones, but different notation.

the length of the line segment between two points x and y. Note that our general position assumption implies that each vertex adds at most one edge per subcone.

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.* [1]. 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 $(C_0, \overline{C_2}, C_1, \overline{C_0}, C_2, \overline{C_1})$ be the sequence of cones in counterclockwise order starting from the positive y-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. Note that the positive cones coincide with the even cones of the constrained Θ_6-graph and the negative cones coincide with the odd ones. 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. Note that, by the way in which cones are labeled, for any two vertices u and v, it holds that $v \in C_i^u$ if and only if $u \in \overline{C_i^v}$. Analogous to the subcones defined for the Θ_6-graph, constraints 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 Fig. 3). We look at the undirected version of these graphs, i.e. when an edge is added, both vertices are allowed to use it. This is consistent with previous work on Θ-graphs.

Finally, we define the constrained Delaunay triangulation. Given any two visible vertices p and q, the constrained Delaunay triangulation contains an edge between p and q if and only if pq is a constraint or there exists a circle O with p and q on its boundary such that there are no vertices of P in the interior of O is visible to both p and q. For simplicity, we assume that no four vertices lie on the boundary of any circle.

There are two notions of *competitiveness* of a routing algorithm. One is to look at the Euclidean shortest path between the two vertices, i.e. the shortest path in $\mathit{Vis}(P, S)$, and the other is to compare the routing path to the shortest path in the subgraph of $\mathit{Vis}(P, S)$. A routing algorithm is *c-competitive with respect to the Euclidean shortest path (resp. shortest path in the graph)* provided that the total distance traveled by the message is not more than c times the

Euclidean shortest path length (resp. shortest path length) between the source and the destination. The *routing ratio* of an algorithm is the smallest c for which it is c-competitive. Since the shortest path in the graph between two vertices is at least as long as the Euclidean shortest path between them, an algorithm that is c-competitive with respect to the Euclidean shortest path is also c-competitive with respect to the shortest path in the graph. We use competitiveness with respect to the Euclidean shortest path when proving upper bounds and with respect to the shortest path in the graph when proving lower bounds.

We now define our routing model. Formally, a routing algorithm A is a deterministic k-local, m-memory routing algorithm, if the vertex to which a message is forwarded from the current vertex s is a function of s, t, $N_k(s)$, and M, where t is the destination vertex, $N_k(s)$ is the k-neighborhood of s and M is a memory of size m, stored with the message. The k-neighborhood of a vertex s is the set of vertices in the graph that can be reached from s by following at most k edges. For our purposes, we consider a unit of memory to consist of a $\log_2 n$ bit integer or a point in $\mathbb{R}^2$. Our model also assumes that the only information stored at each vertex of the graph is $N_k(s)$. Unless stated otherwise, when we say "local", we will assume that $k = 1$ and that $|M| \in O(1)$, i.e. our algorithms are 1-local and use a constant amount of memory. Since our graphs are geometric, we identify each vertex by its coordinates in the plane.

We say that a region R *contains* a vertex v if v lies in the interior or on the boundary of R. We call a region *empty* if it does not contain any vertex of P.

Lemma 2.1. [2] *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.*

3 Local Routing on the Visibility Graph

In the unconstrained setting, local routing algorithms have focused on subgraphs of the complete Euclidean graph such as Θ-graphs. There exists a very simple local routing algorithm that works on all Θ-graphs with at least 4 cones and is competitive when the graph uses at least 7 cones. This routing algorithm (often called the Θ-routing) always follows the edge to the closest vertex in the cone that contains the destination. In the constrained setting, however, a problem arises if we try to apply this strategy: even though a cone contains the destination it need not contain any visible vertices, since a constraint may block its visibility. Hence, the Θ-routing algorithm will often get stuck since it cannot follow any edge in that cone. In fact, given a set P of points in the plane and a set S of disjoint segments, no local routing algorithm is known for routing on $\mathit{Vis}(P, S)$.

When the destination t is visible to the source s, it is possible to route locally by essentially "following the segment st", since no constraint can intersect st. This approach was used to give a 2-competitive 1-local routing algorithm on the constrained half-Θ_6-graph, provided that t is in a positive cone of s [3]. In the case where t is in a negative cone of s, the algorithm is much more involved and the competitive ratio jumps to 18.

The stumbling block of all known approaches is the presence of constraints. In a nutshell, the problem is to determine how to "go around" a constraint in such a way as to reach the destination and not cycle. This poses the following natural question: does there exist a deterministic 1-local routing algorithm that always reaches the destination when routing on the visibility graph? In this section, we answer this question in the affirmative and provide such a 1-local algorithm. The main idea is to route on a planar subgraph of $Vis(P, S)$ that can be computed locally.

In [10] it was shown how to route locally on a plane geometric graph. Subsequently, in [6], a modified algorithm was presented that seemed to work better in practice. Both algorithms are described in detail in [6], where the latter algorithm is called FACE-2 and the former is called FACE-1. Neither of the algorithms is competitive. FACE-1 reaches the destination after traversing at most $\Theta(n)$ edges in the worst case and FACE-2 traverses $\Theta(n^2)$ edges in the worst case. Although FACE-1 performs better in the worst case, FACE-2 performs better on average in random graphs generated by points uniformly distributed in the unit square.

Coming back to our problem of routing locally from a source s to a destination t in $Vis(P, S)$, the main difficulty for using the above strategies is that the visibility graph is not plane. Its seems counter-intuitive that having more edges renders the problem of finding a path more difficult. Indeed, almost all local routing algorithms in the literature that guarantee delivery do so by routing on a plane subgraph that is computed locally. For example, in [6], a local routing algorithm is presented for routing on a unit disk graph and the algorithm actually routes on a planar subgraph known as the Gabriel graph. However, none of these algorithms guarantee delivery in the presence of constraints. In this section, we adapt the approach from [6] by showing how to locally identify the edges of a planar spanning subgraph of $Vis(P, S)$, which then allows us to use FACE-1 or FACE-2 to route locally on $Vis(P, S)$.

The graph in question is the constrained half-Θ_6-graph, which was shown to be a plane 2-spanner of $Vis(P, S)$ [2]. Therefore, if we can show how to locally identify the edges of this graph, we can apply FACE-1 or FACE-2. If we are at a vertex v and we know all visibility edges incident to v, then identifying the edges in v's positive cones is easy: they connect v to the endpoints in this cone whose projection on the bisector is closest. Thus, the hard part is deciding which of the visibility edges in v's negative cone are added by their endpoints. Next, we show that v has enough information to find these edges locally.

Lemma 3.1. *Let u and v be vertices such that $u \in \overline{C_0^v}$. Then uv is an edge of the constrained half-Θ_6-graph if and only if v is the vertex whose projection on the bisector of C_0^u is closest to u, among all vertices in C_0^u visible to v and not blocked from u by constraints incident on v.*

Proof. First, suppose that v is not closest to u among the vertices in C_0^u visible to v and not blocked by constraints incident on v (see Fig. 4a). Then there are such vertices whose projection on the bisector is closer to u. Among those vertices, let x be the one that minimizes the angle between vx and vu. Now v cannot be the endpoint of a constraint intersecting the interior of triangle uvx, since the endpoint of that constraint would lie inside the triangle, contradicting our choice of x. Since both uv and vx are visibility edges, Lemma 2.1 tells us that there is a convex chain of visibility edges connecting u and x inside triangle uvx. In particular, the first vertex y from u on this chain is visible from both u and v and is closer to u than v is (in fact, $y = x$ by our choice of x). Moreover, v must be in the same subcone of u as y, since the region between v and the chain is completely empty of both vertices and constraints. Thus, uv cannot be an edge of the half-Θ_6-graph.

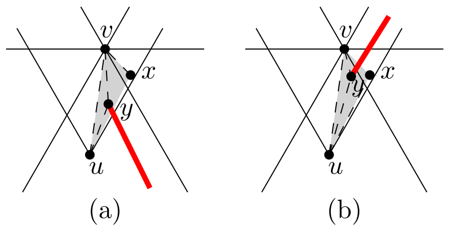

Fig. 4. (a) If v is not closest to u among the vertices visible to v, then uv is not in the half-Θ_6-graph. (b) If v is closest to u among the vertices visible to v, then uv must be in the half-Θ_6-graph.

Second, suppose that v is closest to u among the vertices visible to v and not blocked by constraints incident on v, but uv is not an edge of the half-Θ_6-graph. Then there is a vertex $x \in C_0^u$ and in the same subcone as v, who is visible to u, but not to v, and whose projection on the bisector is closer to u (see Fig. 4b). Since x and v are in the same subcone, u is not incident to any constraints that intersect the interior of triangle uvx, so we again apply Lemma 2.1 to the triangle formed by visibility edges uv and ux. This gives us that there is a convex chain of visibility edges connecting v and x, inside triangle uvx. In particular, the first point y from v on this chain must be visible to both u and v. And since y lies in triangle uvx, it lies in C_0^u and its projection is closer to u. But this contradicts our assumption that v was the closest vertex. Thus, if v is the closest vertex, uv must be an edge of the half-Θ_6-graph. □

With Lemma 3.1, we can compute 1-locally which of the edges of $Vis(P, S)$ incident on v are also edges of the half-Θ_6-graph. Therefore, we can apply FACE-1 or FACE-2 in order to route on $Vis(P, S)$ by routing on the half-Θ_6-graph.

Corollary 3.2. *We can 1-locally route on $Vis(P, S)$ by routing on the constrained half-Θ_6-graph.*

Although it was shown in [3] that no deterministic 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, the caveat to the above local routing algorithm is that the competitive ratio is not bounded by any function of n. In fact, by applying FACE-1, it is possible to visit almost every edge of the graph four times before reaching the destination. It is worse with FACE-2, where almost every edge may be visited a linear number of times before reaching the destination. In the next section, we present a 1-local routing algorithm that is $O(n)$-competitive and provide a matching worst-case lower bound.

4 Routing on Constrained Triangulations

Next, we look at routing on a given constrained triangulation: a graph in which all constraints are edges and all faces are triangles. Hence, we do not have to check that the graph is a triangulation and we can focus on the routing process.

4.1 Lower Bound

Given a triangulation G and a source vertex s and a destination vertex t, let H be the subgraph of G that contains all edges of G that are part of a triangle that is intersected by st. We first show that if G is a constrained Delaunay triangulation or a constrained half-Θ_6-graph, the shortest path in H can be a factor of $n/4$ times longer than that in G. This implies that any local routing algorithm that considers only the triangles intersected by st cannot be $o(n)$-competitive with respect to the shortest path in G on every constrained Delaunay triangulation or constrained half-Θ_6-graph on every pair of points. In the remainder of this paper, we use $\pi_G(u, v)$ to denote the shortest path from u to v in a graph G.

Lemma 4.1. *There exists a constrained Delaunay triangulation G with vertices s and t such that $|\pi_H(s,t)| \geq \frac{n}{4} \cdot |\pi_G(s,t)|$.*

Proof. We construct a constrained Delaunay graph with this property. For ease of description and calculation, we assume that the size of the point set is a multiple of 4. Note that we can remove this restriction by adding 1, 2, or 3 vertices "far enough away" from the construction so it does not influence the shortest path.

We start with two columns of $n/2 - 1$ points each, aligned on a grid. We add a constraint between every horizontal pair of points. Next, we shift every other row by slightly less than half a unit to the right (let $\varepsilon > 0$ be the small amount that we did not shift). We also add a vertex s below the lowest row and a vertex t above the highest row, centred between the two vertices on said row. Note that this placement implies that st intersects every constraint. Finally, we stretch the point set by an arbitrary factor $2x$ in the horizontal direction, for some arbitrarily large constant x. When we construct the constrained Delaunay triangulation on this point set, we get the graph G shown in Fig. 5.

In order to construct the graph H, we note that all edges that are part of H lie on a face that has a constraint as an edge. In particular, H does not contain any of the vertical edges on the left and right boundary of G. Hence, all that remains is to compare the length of the shortest path in H to that in G.

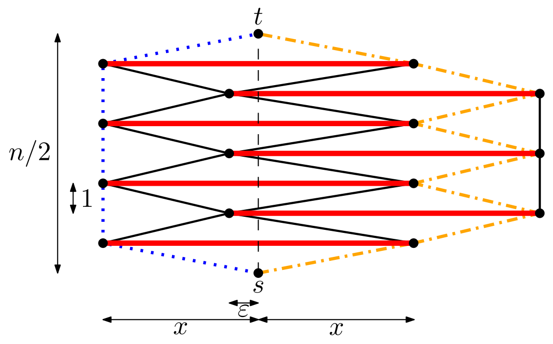

Fig. 5. Lower bound construction: the constraints are shown in thick red, the shortest path in G is shown in blue (dotted), and the shortest path in H is shown in orange (dash dotted). The remaining edges of G are shown in black (solid).

Ignoring the terms that depend on ε, the shortest path in H uses $n/2$ edges of length x, hence it has length $x \cdot n/2$. Graph G on the other hand contains a path of length $2x + n/2 - 1$ (again, ignoring small terms that depend on ε), by following the path to the leftmost column and following the vertical path up. Hence, the ratio $|\pi_H(s,t)|/|\pi_G(s,t)|$ approaches $n/4$, since $\lim_{x\to\infty} \frac{x \cdot \frac{n}{2}}{2x + \frac{n}{2} - 1} = \frac{n}{4}$. $\square$

Note that the above construction is also the constrained half-Θ_6-graph of the given points and constraints.

Corollary 4.2. *There exist triangulations G such that no local routing algorithm that considers only the triangles intersected by st is $o(n)$-competitive when routing from s to t.*

4.2 Upper Bound

Next, we provide a simple local routing algorithm that is $O(n)$-competitive. To make it easier to bound the length of the routing path, we use an auxiliary graph H' defined as follows: let H' be the graph H, augmented with the edges of the convex hull of H and all visibility edges between vertices on the same internal face (after the addition of the convex hull edges). For these visibility edges, we only consider constraints with both endpoints in H. The different graphs G, H, and H' are shown in Fig. 6. Note that the gray region in Fig. 6c is one of the regions where visibility edges are added and note that edge uv is not added, since visibility is blocked by a constraint that has both endpoints in H. We first show that the length of the shortest path in H' is not longer than that in G.

Lemma 4.3. *For any triangulation G, we have $|\pi_{H'}(s,t)| \leq |\pi_G(s,t)|$.*

Proof. If every vertex along $\pi_G(s,t)$ is part of H', we claim that every edge of $\pi_G(s,t)$ is also part of H'. Consider an edge uv of $\pi_G(s,t)$. If uv is part of a

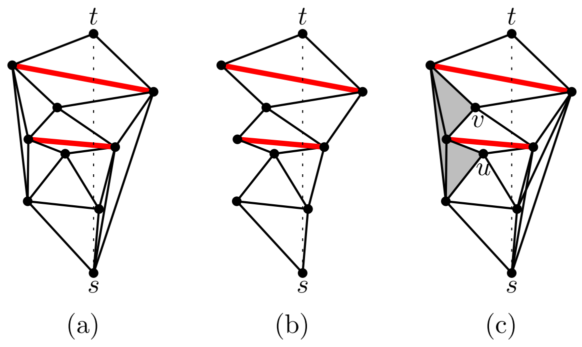

Fig. 6. The three different graphs: (a) The original triangulation G, (b) the subgraph H, (c) graph H' constructed by adding edges to H.

triangle intersected by st, it is already part of H and therefore of H'. If uv is not part of a triangle intersected by st, then u and v must lie on the same face of H' before we add the visibility edges, since otherwise the edge uv would violate the planarity of G. Furthermore, since uv is an edge of G, u and v can see each other. Hence, the edge uv is added to H' when the visibility edges are added. Therefore, every edge of $\pi_G(s,t)$ is part of H' and thus $|\pi_{H'}(s,t)| \leq |\pi_G(s,t)|$.

If not every vertex along $\pi_G(s,t)$ is part of H', we create subsequences of the edges of $\pi_G(s,t)$ such that each subpath satisfies either (i) all vertices are in H', or (ii) only the first and last vertex of the subpath are in H'. Using an argument analogous to the previous case, it can be shown that subpaths of $\pi_G(s,t)$ that satisfy (i) only use edges that are in H'.

To complete the proof, it remains to show that given a subpath π' that satisfies (ii), there exists a different path in H' that connects the two endpoints of H' and has length at most $|\pi'|$. Let u and v be the first and last vertex of this π' (see Fig. 7). If u and v lie on the same face of H' before the visibility edges are added, H' contains the geodesic path $\pi_{H'}$ between u and v with respect to the constraints using only vertices in H'. Note that this path can cross constraints that have at most 1 endpoint in H'. Path π' uses only edges of G which by definition do not cross any constraints. Hence, π' cannot be shorter than $\pi_{H'}$.

Finally, we consider the case in which u and v do not lie on the same face before the visibility edges are added. We construct $\pi_{H'}$ by following the geodesic paths in the faces of u and v, and using the convex hull edges for the remainder of the path. Since u and v do not lie on the same face, a convex hull vertex x lies on this path. Next, we look at π'. Recall that no vertex along π' (other than u and v) is in H'. This implies that π' cannot cross st and it must cross any ray originating from x that does not intersect the convex hull. Hence, π' goes around $\pi_{H'}$ and therefore H' contains a path from u to v of length at most $|\pi'|$. □

Lemma 4.4. *For any triangulation G, we have $|\pi_H(s,t)| \leq (n-1) \cdot |\pi_{H'}(s,t)|$.*

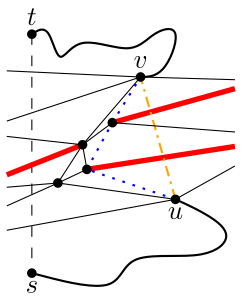

Fig. 7. A subpath of $\pi_G(s,t)$ (dotted blue) that satisfies condition (*ii*): no vertex other than its endpoints are in H'.

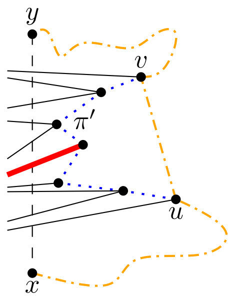

Fig. 8. The path π' (dotted blue) simulates uv on the shortest path in H' (dot dashed orange) and lies on the boundary of a pocket of H.

Proof. To prove the lemma, we show that for every edge uv on the shortest path in H', there is a path in H from u to v whose length is at most $|\pi_{H'}(s,t)|$. Consider an edge uv on the shortest path in H'. If uv is also an edge of H we use it in our path. Since uv is part of the shortest path in H' we have $|uv| \leq |\pi_{H'}(s,t)|$.

It remains to consider the case in which uv is not part of H. Note that this implies that uv is either an edge of the convex hull of H or a visibility edge between two vertices of the same internal face. Instead of following uv, we *simulate* uv by following the path π' along the pocket of H from u to v (the part of the boundary that does not visit both sides of st; see Fig. 8).

The path $\pi_{H'}(s,t)$ must cross the segment st several times (at least once at s and once at t). Let x be the last intersection before u in $\pi_{H'}(s,t)$. Similarly, let y be the first intersection after v. Since π' lies on the boundary of a pocket, it cannot cross st and therefore it is contained in the polygon defined by the segment xy, and the portion of $\pi_{H'}(s,t)$ that lies between x and y. All edges of π' lie inside this polygon. In particular, each such edge has length at most the length of $\pi_{H'}(s,t)$ from x to y, which is at most $|\pi_{H'}(s,t)|$ (since a segment inside a polygon has length at most half the perimeter of the polygon).

Concatenate all our simulating paths and shortcut the resulting path from s to t such that every vertex is visited at most once. The result is a path which consists of at most $n-1$ edges, each of length at most $|\pi_{H'}(s,t)|$. □

Theorem 4.5. *For any triangulation G, we have $|\pi_H(s,t)| \leq (n-1) \cdot |\pi_G(s,t)|$.*

In order to route on the graph H, we apply the Find-Short-Path routing algorithm by Bose and Morin [5]. This routing algorithm is designed precisely to route on the graph created by the union of the triangles intersected by the line segment between the source and destination. The algorithm reaches t after having travelled at most 9 times the length of the shortest path from s to t in this union of triangles. Hence, applying Find-Short-Path to graph H yields a routing path of length at most $9(n-1) \cdot |\pi_G(s,t)|$.

Theorem 4.6. *For any triangulation, there exists a 1-local $O(n)$-competitive routing algorithm that visits only triangles intersected by the line segment between the source to destination.*

5 Conclusions

We presented the first local routing algorithm to route on the visibility graph. We then showed a local $O(n)$-competitive routing algorithm for any triangulation and showed that this is optimal when restricted to routing on the set of triangles intersected by the segment from the source to the destination. The competitiveness of our routing algorithm on $Vis(S, P)$ is not bounded by any function of n. On the other hand, our local $O(n)$-competitive routing algorithms require a triangulated subgraph of $Vis(S, P)$. Unfortunately, it is not known how to compute such a triangulation locally, which naturally leads to the following open problem: Can one locally compute a triangulation of $Vis(S, P)$? It is known that the constrained Delaunay triangulation cannot be computed locally and the constrained half-Θ_6-graph is not necessarily a triangulation.

Acknowledgements
We thank Luis Barba, Sangsub Kim, and Maria Saumell for fruitful discussions.

References

1. Bonichon, N., Gavoille, C., Hanusse, N., Ilcinkas, D.: Connections between theta-graphs, Delaunay triangulations, and orthogonal surfaces. In: WG. pp. 266–278 (2010)
2. Bose, P., Fagerberg, R., van Renssen, A., Verdonschot, S.: On plane constrained bounded-degree spanners. In: LATIN. LNCS, vol. 7256, pp. 85–96 (2012)
3. Bose, P., Fagerberg, R., van Renssen, A., Verdonschot, S.: Competitive local routing with constraints. In: ISAAC. LNCS, vol. 9472, pp. 23–34 (2015)
4. Bose, P., Keil, J.M.: On the stretch factor of the constrained Delaunay triangulation. In: ISVD. pp. 25–31 (2006)
5. Bose, P., Morin, P.: Competitive online routing in geometric graphs. TCS 324(2), 273–288 (2004)
6. Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless networks. Wireless Networks 7(6), 609–616 (2001)
7. Bose, P., van Renssen, A.: Upper bounds on the spanning ratio of constrained theta-graphs. In: LATIN. LNCS, vol. 8392, pp. 108–119 (2014)
8. Clarkson, K.: Approximation algorithms for shortest path motion planning. In: STOC. pp. 56–65 (1987)
9. Das, G.: The visibility graph contains a bounded-degree spanner. In: CCCG. pp. 70–75 (1997)
10. Kranakis, E., Singh, H., Urrutia, J.: Compass routing on geometric networks. In: CCCG. pp. 51–54 (1999)
11. Misra, S., Misra, S.C., Woungang, I.: Guide to Wireless Sensor Networks. Springer (2009)
12. Räcke, H.: Survey on oblivious routing strategies. In: Math. Theory Comput. Prac. LNCS, vol. 5635, pp. 419–429 (2009)