Local Routing on Ordered Θ-Graphs

André van Renssen
The University of Sydney, Australia

Shuei Sakaguchi
The University of Sydney, Australia

Abstract

The problem of locally routing on geometric networks using limited memory is extensively studied in computational geometry. We consider one particular graph, the ordered Θ-graph, which is significantly harder to route on than the Θ-graph, for which a number of routing algorithms are known. Currently, no local routing algorithm is known for the ordered Θ-graph.

We prove that, unfortunately, there does not exist a deterministic memoryless local routing algorithm that works on the ordered Θ-graph. This motivates us to consider allowing a small amount of memory, and we present a deterministic $O(1)$-memory local routing algorithm that successfully routes from the source to the destination on the ordered Θ-graph. We show that our local routing algorithm converges to the destination in $O(n)$ hops, where n is the number of vertices. To the best of our knowledge, our algorithm is the first deterministic local routing algorithm that is guaranteed to reach the destination on the ordered Θ-graph.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Ordered Θ-graph, Local routing, Computational geometry

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.52

Related Version *Full Version*: https://arxiv.org/abs/2506.16021

Funding *André van Renssen*: This research was partially funded by the Australian Government through the Australian Research Council (project number DP240101353).

1 Introduction

A *geometric network* $G = (V, E)$ is a weighted graph where each vertex $v \in V$ is a point in the Euclidean plane $\mathbb{R}^2$ and each edge $(u, v) \in E$ is a straight line segment, connecting $u \in V$ and $v \in V$, weighted by the Euclidean distance $|uv|$. We define the *weighted distance* $\delta_G(u, v)$ between two vertices u and v on G to be the sum of the weights of the edges along the weighted shortest path from u to v in G.

A subgraph $H = (V, E')$ of a geometric network $G = (V, E)$ is called a c-spanner of G, if, for every pair of vertices $s, t \in V$, we have $\delta_H(s, t) \leq c \cdot \delta_G(s, t)$, where $c \in \mathbb{R}_{\geq 1}$. Here, G is called the *underlying network* of H. The smallest constant c such that H is a c-spanner of G is called the *spanning ratio* or *stretch factor* of H. We consider the situation where the underlying network G is the *complete Euclidean graph* of V – i.e., a c-spanner H of G approximates the Euclidean distance $|st| = \delta_G(s, t)$ between any pair of points $s, t \in V$ by a constant factor c. The spanning ratio of a class of spanners $\mathcal{G}$ is the spanning ratio of the worst-case instance $G \in \mathcal{G}$ maximizing the spanning ratio. See the textbook by Narasimhan and Smid [15] and the survey by Bose and Smid [9] for a comprehensive overview of spanners and their open problems.

The study of geometric spanners is closely related to the design and analysis of efficient routing algorithms that forward a message between a pair of vertices. Given a source vertex s and a target vertex t on a graph G, a *routing algorithm* aims to find a short path from s to t. If the information of the entire graph can be known and kept track during routing, several

classical path-finding algorithms, including Dijkstra's algorithm [11] and the Bellman-Ford-Moore algorithm [14], can be used on spanners to find a short path. However, when we have restricted knowledge of the graph and limited memory, the routing problem becomes more challenging.

A routing algorithm is called *h-local* if it only knows the h-neighbourhood of the current vertex u and the information of the destination t, where the h-neighbourhood of u is the set of vertices that can be reached from u by taking at most h distinct edges in the graph. A routing algorithm is called *m-memory* if a memory of size m is stored with the message of routing.[1]

Two major classes of local routing algorithms have been actively researched in the literature: (1) deterministic 1-local memoryless routing algorithms and (2) deterministic 1-local $O(1)$-memory routing algorithms. Borrowing the notation used by Bose et al. [2], these two classes of local routing algorithms can be formalised as follows:

- A deterministic 1-local memoryless routing algorithm on a graph $G = (V, E)$ is a *routing function* $f : V \times V \times \mathcal{P}(V) \to V$, where $\mathcal{P}(\cdot)$ denotes the power set. We express the parameters of the function as $f(u, t, N(u))$, where $u \in V$ is the "current" vertex during routing (i.e., u currently holds the message we want to route from the source to the destination), $t \in V$ is the destination vertex, and $N(u) \subseteq V$ is the 1-neighbourhood of u (which simply means the set of immediate neighbours of u). The output of the routing function with the given parameters, $v = f(u, t, N(u)) \in N(u)$, is the neighbour of u to which the message is forwarded.
- A deterministic 1-local $O(1)$-memory routing algorithm on a graph $G = (V, E)$ is a routing function $g : V \times V \times \mathcal{P}(V) \times \mathcal{I} \to V$, which is similar to the routing function f above, except the parameters of the function $g(u, t, N(u), i)$ has an additional parameter $i \in \mathcal{I}$, modelling some information stored in the constant-sized memory of g that can be used to make routing decisions.

A routing algorithm performed on a graph H is called *c-competitive* with respect to the underlying network G if the total weight of the path traveled by the routing algorithm between any pair of vertices in H is at most c times the total weight of the shortest path in G [4]. The minimum possible value c for which the given routing algorithm is c-competitive with respect to G is called the *routing ratio* of the routing algorithm in H.

The Θ-graph, introduced independently by Clarkson [10] and Keil [13], is a geometric network studied extensively in the literature. Given a point set $V \subset \mathbb{R}^2$, the Θ_k-graph $G = (V, E)$ is defined and constructed as follows: For each vertex $u \in V$, we let equally spaced k rays emerge from u, resulting in k cones, each with the aperture $\theta = \frac{2\pi}{k}$. For each cone of every vertex u, we add an edge from u to the nearest neighbour v within that cone, with distance measured along the bisector of the cone (see Figure 1).

It is known that Θ_k-graphs are spanners (see Table 1 for a summary). In addition, various local routing algorithms with good routing ratios have been developed for Θ_k-graphs. A prime example is Θ*-routing*, also known as *cone-routing*. Θ-routing from s to t on a Θ_k-graph is defined as follows: Let u be the current vertex visited during the routing from s to t. If there exists a direct edge from u to t, follow this edge. Otherwise, follow the edge to the closest vertex in the cone of u containing t. This is repeated until the destination is reached.

[1] In this paper, we use the standard *real RAM* model of computation used in computational geometry. In this model, each *machine word* holds a real number or a $O(\log n)$-bit integer (such as a pointer or an index), and we assume that arithmetic operations (excluding the floor and ceiling functions for real numbers) on machine words take $O(1)$ time. Consequently, a $O(1)$-memory routing algorithm may store $O(\log n)$ bits.

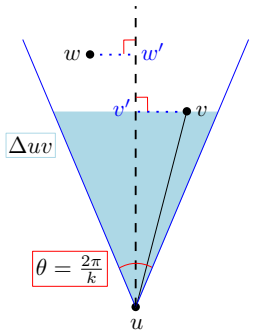

Figure 1 The creation of an edge (u, v) during the construction of a Θ_k-graph. The orthogonal projection v' of v is the closest to u in the cone.

Θ-routing is known to be competitive when used on Θ_k-graphs for $k \geq 7$. In addition to Θ-routing, various local routing algorithms with good routing ratios have been developed for Θ-graphs with smaller numbers of cones (see Table 1 for a summary).

Table 1 The current upper bounds on the spanning and routing ratio of Θ_k-graphs, with $\theta = \frac{2\pi}{k}$.

	Spanning Ratio	Routing Ratio
Θ_2 and Θ_3	Not spanners [12]	Not spanners [12]
Θ_4	17 [3]	17 [3]
Θ_5	$\frac{\sin(3\pi/10)}{\sin(2\pi/5)-\sin(3\pi/10)}$ [7]	-
Θ_6	2 [1]	2 [5]
$\Theta_{(4i+2)}$ for $i \geq 2$	$1 + 2\sin(\frac{\theta}{2})$ [4]	$\frac{1}{1-2\sin(\theta/2)}$ [16]
$\Theta_{(4i+3)}$ for $i \geq 1$	$\frac{\cos(\theta/4)}{\cos(\theta/2)-\sin(3\theta/4)}$ [4]	$1 + \frac{2\sin(\theta/2)\cdot\cos(\theta/4)}{\cos(\theta/2)-\sin(\theta/2)}$ [4]
$\Theta_{(4i+4)}$ for $i \geq 1$	$1 + \frac{2\sin(\theta/2)}{\cos(\theta/2)-\sin(\theta/2)}$ [4]	$1 + \frac{2\sin(\theta/2)}{\cos(\theta/2)-\sin(\theta/2)}$ [4]
$\Theta_{(4i+5)}$ for $i \geq 1$	$\frac{\cos(\theta/4)}{\cos(\theta/2)-\sin(3\theta/4)}$ [4]	$1 + \frac{2\sin(\theta/2)\cdot\cos(\theta/4)}{\cos(\theta/2)-\sin(\theta/2)}$ [4]

Unfortunately, the Θ-graph has a number of undesirable properties, including potentially a high maximum degree and a large diameter. The *ordered* Θ*-graph* alleviated these issues. It was introduced by Bose et al. [6] in 2004 to achieve logarithmic maximum degree and logarithmic diameter, in addition to the spanner property. An ordered Θ_k-graph is constructed by inserting the vertices one by one and connecting each vertex to the "closest" *previously* inserted vertex in each of the k cones.

However, achieving these additional properties comes at a price. For example, ordered Θ-graphs with 4, 5, or 6 cones are not spanners [8], despite the fact that their unordered counterparts are. In addition, ordered Θ-graphs have worse spanning ratios compared to Θ-graphs. The best known upper bounds and lower bounds on the spanning ratio of ordered Θ_k-graphs are summarized in Table 2.

Although we have good local routing algorithms for the Θ-graph, no local routing algorithm is known for the ordered Θ-graph. A major obstacle is that even if there are vertices in a cone, there may still not be any edges, due to the insertion order (see Figure 2). This makes the ordered Θ-graph much more challenging to route on than the Θ-graph – for example, Θ-routing fails on the ordered Θ-graph.

Table 2 The current upper and lower bounds on the spanning ratio of ordered Θ_k-graphs, with $\theta = \frac{2\pi}{k}$.

	Upper Bound	Lower Bound
Θ_3, Θ_4, Θ_5, and Θ_6	Not constant spanners [8]	Not constant spanners [8]
$\Theta_{(4i+2)}$ for $i \geq 2$	$\frac{1}{1-2\sin(\theta/2)}$ [6]	$\frac{1}{1-2\sin(\theta/2)}$ [8]
$\Theta_{(4i+3)}$ for $i \geq 1$	$\frac{1}{1-2\sin(\theta/2)}$ [6]	$\frac{\cos(\theta/4)+\sin(\theta)}{\cos(3\theta/4)}$ [8]
$\Theta_{(4i+4)}$ for $i \geq 1$	$1+\frac{2\sin(\theta/2)}{\cos(\theta/2)-\sin(\theta/2)}$ [8]	$1+\frac{2\sin(\theta/2)}{\cos(\theta/2)-\sin(\theta/2)}$ [8]
$\Theta_{(4i+5)}$ for $i \geq 1$	$\frac{1}{1-2\sin(\theta/2)}$ [6]	$1+\frac{2\sin(\theta/2)\cdot\cos(\theta/4)}{\cos(\theta/2)-\sin(3\theta/4)}$ [8]

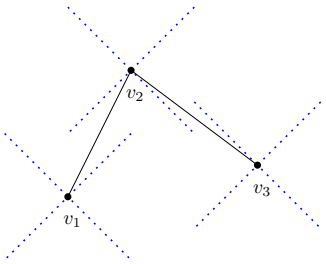

Figure 2 Three vertices v_1, v_2, and v_3 are inserted in the order v_1, v_2, v_3. When inserting v_2, an edge to v_1 is added. When inserting v_3, only an edge to v_2 is added, resulting in v_1 having vertex v_3 in a cone with no edges.

2 Technical overview

Our contribution is twofold: First, we show why so far no deterministic *memoryless h-local* routing algorithms have been found. Specifically, we prove that there does not exist a deterministic memoryless h-local routing algorithm on the ordered Θ_k-graph, for $h \geq 1$ and $k \geq 2$. We construct two ordered Θ_k-graphs L_h^k and R_h^k that cannot be distinguished in this routing model, which implies that any such algorithm fails on one of L_h^k and R_h^k. By doing so, we obtain the following theorem.

▶ **Theorem 1.** *There is no deterministic memoryless h-local routing algorithm (for any integer $h \geq 1$) capable of finding a path between every pair of source vertex s and destination vertex t on the ordered Θ_k-graph (for any integer $k \geq 2$).*

We then present a deterministic $O(1)$-memory 1-local routing algorithm $\mathcal{A}$ and prove that it converges to the destination in $O(n)$ hops (i.e., taking at most $O(n)$ edges). Our algorithm $\mathcal{A}(s,t)$ works in two phases: The first routes from the source s to the first inserted vertex v_1 by repeatedly moving to a neighbour with a smaller order. In the second phase, our algorithm routes from v_1 to the destination t by systematically exploring vertices that might lead to t. By exploring "candidate" vertices in a systematic way, we ensure that we do not need to keep track of the path. Instead, we can use a simple memoryless 1-local routing algorithm we call *ordered* Θ*-routing*, which allows us to implicitly keep track of the vertices we need to backtrack to by utilizing the geometric property of the ordered Θ-graph.

To prove that our algorithm $\mathcal{A}(s,t)$ converges, we consider the path P travelled by ordered Θ-routing from t to v_1 and show that our algorithm explores this path in the reverse direction. By arguing that any vertex is explored at most once and backtracked from at most once, we conclude that our algorithm reaches the destination in at most $O(n)$ hops and $O(kn)$ time. Consequently, we obtain the following theorem.

▶ **Theorem 2.** *Given an arbitrary source vertex $s \in V$ and the destination vertex $t \in V$ of an ordered Θ_k-graph $G = (V, E)$ (for an integer $k \geq 2$), the algorithm $\mathcal{A}(s, t)$ is a deterministic $O(1)$-memory 1-local routing algorithm which successfully routes from s to t in $O(n)$ hops. In particular, $\mathcal{A}(s, t)$ takes $O(kn)$ time in the worst-case.*

3 Preliminaries

Given a set of vertices $V \subset \mathbb{R}^2$ in the Euclidean plane, the ordered Θ_k-graph $G = (V, E)$ is a geometric network, with edges being undirected straight line segments, constructed as follows. For each vertex $u \in V$, we define $\rho(u)$ to be the order of insertion of u – i.e., if v_i is the i-th vertex inserted into G, for some positive integer $i \leq n$, where $n = |V|$, we let $\rho(v_i) = i$. We incrementally insert the vertices from the first to the last according to the pre-defined order ρ. Upon inserting each vertex $u \in V$, we partition the plane around u into k disjoint regions, by projecting k equally spaced rays from u, such that the angle between each pair of consecutive rays is $\theta = \frac{2\pi}{k}$ (see Figure 3(a)). We define a *cone* C^u of u to be a region in the plane between two consecutive rays originating from u. We orient the k cones such that the bisector of one of them coincides with the vertical half-line through u lying above u. Let the cone with such a bisector be C_0^u, from which we start numbering the cones in clockwise order around u. For the insertion of every other vertex $v \in V$, we orient and number the cones around v in the same way. We write C_i^v to denote the i-th cone of v, for $0 \leq i < k$. We say that v is the *apex* of C_i^v. We also define $C^v(w)$ to be the cone of v containing a vertex w.

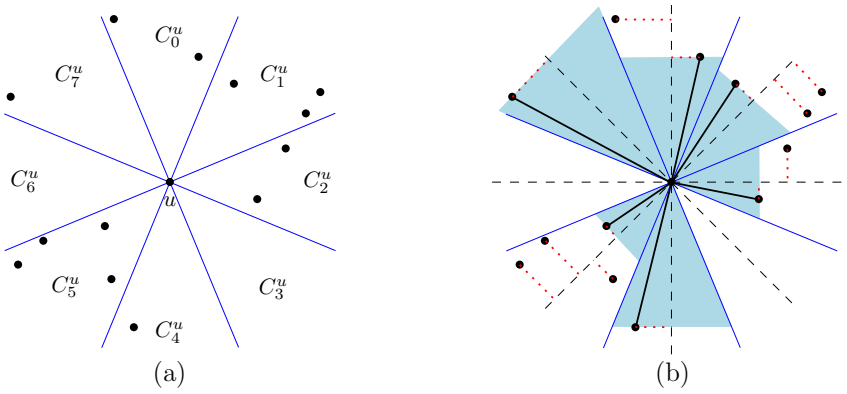

Figure 3 (a) the numbered cones around u and (b) the edges created in each of them.

Upon the insertion of each vertex $u \in V$, an edge is created to the "closest" *previously* inserted vertex in each cone C_i^u of u (see Figure 3(b)). More formally, an edge is created from u to v if (1) u is the apex of the cone that contains v such that $\rho(v) < \rho(u)$ and (2) for every vertex w within the cone such that $\rho(w) < \rho(u)$, $|uv'| \leq |uw'|$, where x' denotes the orthogonal projection of a vertex x onto the bisector of the cone of u that contains x. The canonical triangle Δuv corresponding to an edge created from u to v is defined by the boundary of the cone of u containing v and the line through v and its orthogonal projection onto the bisector. It is important to note that only the vertices inserted before u are considered for the edge creation process. This is why different orderings of vertex insertions result in different ordered Θ_k-graphs, even if the vertex set is the same.

For the insertion of each vertex $u \in V$, we store its order $\rho(u)$ on the vertex, in addition to the x- and y-coordinates of u. Since $1 \leq \rho(u) \leq n$ (where $n = |V|$), storing the order $\rho(u)$ requires at most $\lceil \log n \rceil$ extra bits per vertex, fitting in a single machine word.

It is important to note that a canonical triangle of the ordered Θ-graph is not necessarily empty, despite the fact that any canonical triangle in the Θ-graph is always empty of vertices in its interior. This is because the vertices are inserted incrementally and we can always insert a vertex in a canonical triangle between a pair of vertices previously inserted: Suppose, during the construction of an ordered Θ-graph, we have just inserted a vertex $v \in V$ with order $\rho(v) = k$. Let v' with order $\rho(v') < \rho(v) = k$ be the closest neighbour of a cone of v. At this stage, the canonical triangle $\Delta vv'$ is empty. However, we can insert the next vertex v'' with order $\rho(v'') = k + 1$ into the interior of $\Delta vv'$, making the canonical triangle non-empty.

For simplicity, in order to avoid tie-breaking for edge creation, we assume that V is in *general position*. Specifically, we will assume that, upon inserting each vertex during the construction of the ordered Θ-graph, it is clear which vertices lie in which cone (i.e., no previously inserted vertex lies on a boundary of a cone of the currently inserted vertex) and it is clear which vertex is the "closest" in each cone of a vertex (i.e., no vertex w lies on the boundary of the canonical triangle Δuv of the current vertex u and a neighboring vertex v, where $w \neq u$ and $w \neq v$). This assumption can be removed using standard techniques.

4 Local Routing on Ordered Θ_k-graphs

In this section, we will show that no deterministic *memoryless* local routing algorithm is guaranteed to reach the destination on the ordered Θ_k-graph, for any integer $k \geq 2$ (Section 4.1). As this motivates us to allow a small amount of memory, we will then present a deterministic $O(1)$-*memory* local routing algorithm that is guaranteed to reach the destination (Section 4.2).

We first define an important operation used by local routing algorithms.

▶ **Definition 3** (Hop). *We define a* hop *to be a single move or a forwarding operation from one vertex to its neighbour. Namely, taking a hop means we either (1) move from a vertex u to another vertex v through an edge, or (2) forward a message from u to v via an edge between them.*

4.1 Impossibility result for memoryless local routing

We show that no deterministic memoryless local routing algorithm works on the ordered Θ-graph. This impossibility result naturally motivates us to consider the next weakest class of local routing algorithms – deterministic $O(1)$-memory 1-local routing algorithm.

▶ **Theorem 1.** *There is no deterministic memoryless h-local routing algorithm (for any integer $h \geq 1$) capable of finding a path between every pair of source vertex s and destination vertex t on the ordered Θ_k-graph (for any integer $k \geq 2$).*

Proof. We present two ordered Θ_k-graphs such that any deterministic memoryless h-local routing algorithm A fails to route from s to t on one of these graphs. Each of these two graphs has $2h + 3$ vertices and $2h + 2$ edges, forming a simple path (see Figure 4).

Intuitively, we design two graphs, where one has a path from s to t to the left of s and t (L_h^k) and one to the right of s and t (R_h^k). By ensuring that these graphs are identical up to h hops from s, no h-local routing algorithm can tell them apart, and since it is deterministic and has no memory, it will always make the same choice at s, regardless of what path of the graph it has explored in the past.

Formally, we specify the location of the $2h + 3$ vertices of L_h^k as follows: $v_1 = (0, 0)$, $v_{2i} = (\frac{i}{h}, -i\epsilon)$ and $v_{2i+1} = (-\frac{i}{h}, -i\epsilon)$ for $i \in \{1, 2, \ldots, h\}$, $v_{2h+2} = (-2, -2h\epsilon)$, and $v_{2h+3} = (0, -\frac{2}{\tan(\theta/2)} - 3h\epsilon)$. Note that the y-coordinate of v_{2h+3} is given in terms of θ, ϵ, and h

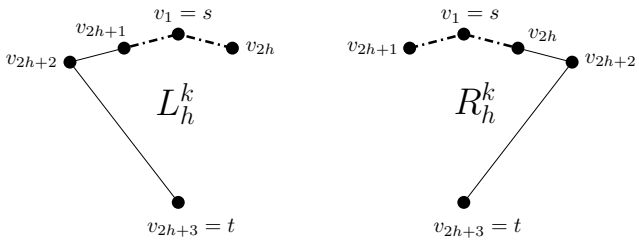

Figure 4 A pair of ordered Θ_k-graphs where any deterministic memoryless h-local routing algorithm fails on one of them.

such that all previously inserted vertices belong to $C_0^{v_{2h+3}}$. Here, ϵ is an arbitrarily small non-negative real number, such that for $j \in \{2, 3, \ldots, 2h+3\}$, a cone of v_j contains the vertices $v_{j'}$ for $j' \in \{1, 2 \ldots, j-1\}$. In particular, if no boundary of the cones of a vertex u coincides with the horizontal line through u, ϵ can simply be 0. We insert these vertices in ascending order: v_1 and then v_j for $j \in \{2, \ldots, 2h+3\}$. The creation of R_h^k is very similar to that of L_h^k. The only difference is that v_{2h+2} is placed at $(2, -2h\epsilon)$ in R_h^k.

We set s to be v_1 and t to be v_{2h+3} for both L_h^k and R_h^k, then we run A on both graphs. Consider the current vertex $u = s = v_1$ of routing. From the perspective of A, the local configuration at v_1 in L_h^k and the local configuration at v_1 in R_h^k are indistinguishable: The coordinate and ordering information of $u = v_1$, $t = v_{2h+3}$, and the h-neighbourhood of $u = v_1$ are identical between both graphs. Hence, any deterministic memoryless h-local routing algorithm A must make the same choice at v_1 both in L_h^k and R_h^k. If A decides to forward the message to v_2 from v_1, A can never reach v_3 in L_h^k, since A always deterministically makes the same choice at v_1 to forward the message to v_2 because the algorithm is memoryless, thereby going back and forth between v_1 and v_2 indefinitely. Similarly, if A decides to forward the message to v_3 from $s = v_1$, A cannot reach v_2 and thus t on R_h^k. ◀

4.2 $O(1)$-memory local routing

Now that we know we cannot route locally without using some memory, we shift to having a limited amount of it. In particular, our approach will store $O(1)$ information.

Our algorithm works in two phases, first to the lowest order vertex v_1 (Section 4.2.1) then to t (Section 4.2.2). The first phase repeatedly moves from the current vertex to a neighbour with a smaller order, which results in reaching v_1 from s without knowing v_1 in advance. The second phase routes from v_1 to t by systematically exploring vertices that might lead to t while implicitly keeping track of the vertices we need to backtrack to by utilizing the geometric property of the ordered Θ-graph.

4.2.1 Reaching vertex v_1

We consider a special vertex in the graph $G = (V, E)$ – the first inserted vertex $v_1 \in V$ with $\rho(v_1) = 1$. We can easily route from s to v_1 using the following algorithm that we call $\mathcal{A}^{down}$.

▶ **Algorithm 1** ($\mathcal{A}^{down}(s)$)**.** Given a source s, we initialize the current vertex u to be s. Then, we repeat the following process until we have $u = v_1$: Pick $v \in N(u)$ such that $\rho(v) < \rho(u)$ (if there are multiple options, pick an arbitrarily one among them, say the minimum) and update $u \leftarrow v$. Note that we do not know v_1 in advance before reaching it.

Below, we analyze the correctness and complexity of $\mathcal{A}^{down}$.

▶ **Lemma 4.** *Given an arbitrary source vertex $s \in V$ and the destination vertex being the first inserted vertex $v_1 \in V$ of an ordered Θ_k-graph $G = (V, E)$ (for an integer $k \geq 2$), the algorithm $\mathcal{A}^{down}(s)$ is a deterministic memoryless 1-local routing algorithm which successfully routes from s to v_1 in $O(n)$ hops. In particular, $\mathcal{A}^{down}(s)$ takes $O(kn)$ time in the worst-case.*

Now that we can route from s to v_1 using $\mathcal{A}^{down}(s)$, we move on to showing how to route from v_1 to t.

4.2.2 Finding the destination t

Starting from v_1, we will systematically explore vertices that might lead to t. Our strategy has two main components: exploration and backtracking. For the former we have to be able to tell which vertices have already been explored, while for the latter we have to know which vertex to backtrack to when the current vertex has no more valid neighbours to explore.

We first present the core component of the backtracking strategy: *ordered Θ-routing*. Given an ordered Θ-graph G and an arbitrary pair of source and destination vertices $s, t \in V$, we define *ordered Θ-routing* as follows.

▶ **Algorithm 2** (Ordered Θ-routing)**.** Initialize the current vertex $u \leftarrow s$. Then, we repeat this forwarding operation until reaching t or "getting stuck": If there exists an edge from u to t, take that edge – we are done. Otherwise, take the edge to the closest neighbour $v \in \Delta ut$ such that $\rho(v) < \rho(u)$. We call this single hop an *ordered Θ-routing step*. If both conditions are negative – i.e., (1) there does not exist an edge from u to t and (2) there does not exist an edge from u to a vertex $v \in \Delta ut$ with a smaller order – ordered Θ-routing is considered "stuck" and we terminate.

The ordered Θ-routing scheme we presented is inspired by the Θ-routing scheme developed for the regular Θ-graph. The main difference is that ordered Θ-routing considers the ordering information when determining which neighbour to forward the message to – it always chooses the closest neighbour with a smaller order.

Note that ordered Θ-routing from the source s to the destination t can only forward the message to the next vertex if either (1) there exists an edge from the current vertex u to the destination t or (2) there exists a vertex $v \in \Delta ut$ such that $\rho(v) < \rho(u)$. Thus, it can fail to reach the destination when neither condition is met. However, as we will see below, ordered Θ-routing is guaranteed to reach the destination if the destination is v_1, the first inserted vertex during the construction of the ordered Θ-graph.

▶ **Lemma 5.** *Given an arbitrary source $s \in V$ of any ordered Θ_k-graph $G = (V, E)$, ordered Θ-routing towards the first inserted vertex v_1 always reaches v_1.*

Proof. We prove the lemma by routing from s to v_1 using ordered Θ-routing (see Figure 5). We initialize the current vertex $u \leftarrow s$, and consider the two possible routing steps:

- If there is an edge from u to v_1, we can take that edge and reach v_1 – we are done.
- Otherwise, consider the canonical triangle Δuv_1. The fact that there does not exist an edge from u to v_1 implies that, when the current vertex u was inserted during the construction of the graph, there existed a vertex $v \in \Delta uv_1$ inserted before u (i.e., we have $\rho(u) > \rho(v)$), such that v was the closest neighbour in the cone $C^u(v_1)$. This means that there exists an edge from u to $v \in \Delta uv_1$, such that $\rho(u) > \rho(v)$. We take that edge. After moving from u to v, we update the current vertex u to be v.

Ordered Θ-routing from s to v_1 eventually converges. Since the order $\rho(v_1) = 1$ is the smallest among all vertices in V, the current vertex u has an edge either to v_1 or to a vertex $v \in \Delta uv_1$ such that $\rho(u) > \rho(v)$ by construction. So, the order $\rho(u)$ of the current vertex u

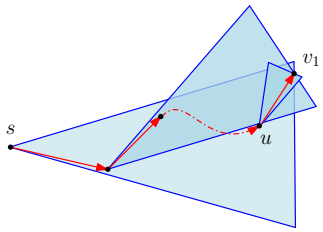

Figure 5 Ordered Θ-routing from s to v_1.

keeps decreasing by at least 1 for each step. Hence, within $n-1$ hops, the canonical triangle Δuv_1 becomes empty of vertices whose order is smaller than that of u, which implies there exists an edge from u to v_1. ◀

We can route from any vertex to v_1 using ordered Θ-routing. In particular, observe that we can replace $\mathcal{A}^{down}(s)$ with ordered Θ-routing from s to v_1 if every vertex stores v_1. However, to minimize space, we use $\mathcal{A}^{down}(s)$ instead.

In order to route from v_1 to t, we look for the reverse of the path traveled by ordered Θ-routing from t to v_1. To see which vertices are "worth exploring" to find such a path, we formally define the *exploration space* below. Conceptually, the exploration space builds a set of vertices by considering the ordered Θ-routing paths from all vertices to v_1, in order to cover the reverse of the ordered Θ-routing path from t to v_1.

▶ **Definition 6** (Exploration Space). *Given an ordered Θ-graph $G=(V,E)$, the first inserted vertex $v_1 \in V$, and an arbitrary destination vertex $t \in V$, the* exploration space $\mathcal{S}(v_1) \subseteq V$ *is a set of vertices defined recursively as follows:*

$$\mathcal{S}(v_1) := \{v_1\} \cup \bigcup_{v \in N^*(v_1)} \left(\{v\} \cup \bigcup_{v' \in N^*(v)} \left(\{v'\} \cup \bigcup_{v'' \in N^*(v')} (\cdots) \right) \right),$$

where we define the exploration candidate neighbours $N^*(u) \subseteq N(u)$ *of any vertex $u \in V$ to be the set of neighbours u' of u satisfying these two conditions:*

1. $\rho(u) < \rho(u') \leq \rho(t)$.
2. *Let $C^{u'}(v_1)$ be the cone of u' containing v_1 in its interior. Then, $C^{u'}(v_1)$ also contains u in its interior – i.e., we have $u \in C^{u'}(v_1)$.*

In particular, these two conditions are equivalent to saying that an ordered Θ-routing step from u' to v_1 leads to u (see Lemma 7 below for the proof).

Note that the *implication* of the two conditions (that an ordered Θ-routing step from u' to v_1 leads to u) is what we actually want to check, because we want to find the reverse of the path travelled by ordered Θ-routing from t to v_1. However, directly checking the implication would require 2-local information – given a current vertex u, accessing a neighbour $u' \in N(u)$ is 1-local, and accessing a neighbour $u'' \in N(u')$ of u', in order to check whether an ordered Θ-routing step from u' to v_1 leads to u'' such that $u''=u$, is 2-local. So, instead we check the two conditions (1) and (2), which is 1-local – we just have to compare the order of u, u', and t, and check the geometric statement $u \in C^{u'}(v_1)$, both of which are 1-local.

▶ **Lemma 7.** *Given an ordered Θ-graph $G=(V,E)$, the first inserted vertex $v_1 \in V$, an arbitrary destination vertex $t \in V$, an arbitrary current vertex $u \in V$, and a neighbour $u' \in N(u)$ of u, we have the following: $\rho(u) < \rho(u') \leq \rho(t)$ and $u \in C^{u'}(v_1)$ if and only if an ordered Θ-routing step from u' to v_1 leads to u and $\rho(u') \leq \rho(t)$.*

Proof. The fact that the ordered Θ-routing step and the inequality $\rho(u') \leq \rho(t)$ imply the two conditions follows by definition. Hence, we focus on the other proof direction. Suppose there exists $u' \in N(u)$ satisfying the first and second conditions. Then, $\rho(u') \leq \rho(t)$ trivially holds, and since $u' \in N(u)$, we have an edge $\{u, u'\} \in E$. The edge $\{u, u'\}$ must have been created upon the insertion of u' into G (when we inserted u into G, u' did not exist yet). This means that, when u' was inserted into G, u was the closest vertex in $C^{u'}(v_1)$. It follows that there does not exist a vertex $w \neq u$ such that w lies in the canonical triangle $\Delta u'u$ and the order of w is smaller than the order of u', because the existence of such vertex w contradicts the fact that u was the closest vertex in $C^{u'}(v_1)$ when u' was inserted into G. Note that we have $\Delta u'u \subset C^{u'}(v_1)$ by the second condition $u \in C^{u'}(v_1)$. Hence, an ordered Θ-routing step from u' to v_1 leads to u and $\rho(u') \leq \rho(t)$. ◀

It is important to ensure that the exploration space $\mathcal{S}(v_1)$ contains t, which we will argue through the lemma given below.

▶ **Lemma 8.** *Given an ordered Θ-graph $G = (V, E)$, the first inserted vertex $v_1 \in V$, an arbitrary destination vertex $t \in V$, the exploration space $\mathcal{S}(v_1)$ contains t.*

Consequently, our problem of locally routing from v_1 to t can be reduced to systematically exploring all vertices belonging to the exploration space $\mathcal{S}(v_1)$ defined above. Building upon the lemmas we have, we now present a deterministic $O(1)$-memory 1-local routing algorithm $\mathcal{A}^{up}$ that routes from v_1 to t, where $v_1 \in V$ is the first inserted vertex and $t \in V$ is an arbitrary destination vertex of the given ordered Θ_k-graph $G = (V, E)$ (for an integer $k \geq 2$). Conceptually, $\mathcal{A}^{up}$ mimics Depth-First Search (DFS) over the vertices in the exploration space $\mathcal{S}(v_1)$. However, unlike how DFS keeps track of the current exploration path from the source by storing a sequence of vertices in a stack so that backtracking can be done by popping the top vertex from the stack, $\mathcal{A}^{up}$ does not explicitly store the exploration path – our local routing algorithm delegates it to the geometric property of the ordered Θ-graph, as backtracking can be done with an ordered Θ-routing step. Below is the full description.

▶ **Algorithm 3** ($\mathcal{A}^{up}(v_1, t)$)**.** Do the following:

1. Initialize the previous vertex u^p to be *null*.
2. Initialize the current vertex u to be v_1.
3. Initialize a bit named *state* to be 1, indicating whether the previous hop was exploration or backtracking.
4. Repeat this block until $u = t$:
   - If $state = 1$ (the previous hop was exploration):
     Let l be the horizontal half-line originating from u towards the right. Sweep l in counter-clockwise direction (see Figure 6). Let $v \in N(u)$ be the first neighbour that touches l. Check whether $v \in N^*(u)$ by checking:
     a. $\rho(u) < \rho(v) \leq \rho(t)$.
     b. $u \in C^v(v_1)$.

     If indeed $v \in N^*(u)$, then we explore v from u by updating the current vertex u, i.e., we set $u \leftarrow v$. Otherwise, we keep sweeping l in counter-clockwise direction, checking whether each $v \in N(u)$ is an exploration candidate neighbour in $N^*(u)$ – if one is found, explore that exploration candidate neighbour, updating u. If l reaches its original position, we perform an ordered Θ-routing step from u to v_1 to backtrack to a vertex u' (observe that $u \in N^*(u')$), updating the current vertex $u \leftarrow u'$, the previous vertex $u^p \leftarrow u$, and the state bit $state \leftarrow 0$.

- Else, i.e., $state = 0$ (the previous hop was backtracking):
  Let l be the half-line originating from u through u^p. Sweep l in counter-clockwise direction as usual, starting from this position, until it becomes the half-line pointing to the right. For each $v \in N(u)$ that touches l, check whether it is an exploration candidate neighbour by checking the two conditions in the same way as above. If an exploration candidate neighbour is found, explore it and update the variables accordingly, setting u to be the neighbour and *state* to be 1. If l becomes the half-line pointing to the right, we perform an ordered Θ-routing step from u to v_1 to backtrack to a vertex u', updating the variables as in the previous case, setting $u \leftarrow u'$, $u^p \leftarrow u$, and $state \leftarrow 0$.

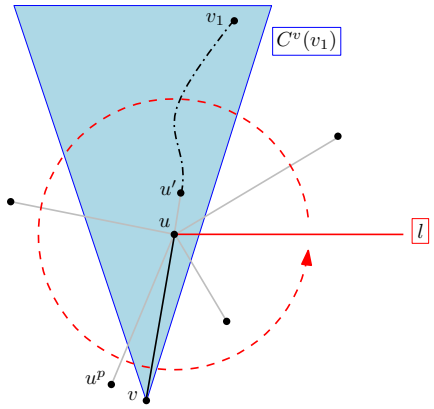

Figure 6 Illustration of $\mathcal{A}^{up}(v_1, t)$.

Before analyzing the correctness of $\mathcal{A}^{up}$, we will first argue that $\mathcal{A}^{up}$ is a deterministic $O(1)$-memory 1-local routing algorithm: There is no randomness involved in the algorithm, so it is deterministic. It only stores v_1, u^p, and the *state* bit. The routing algorithm does not have to store the current vertex u because we are at said vertex, which means we can trivially access the current node without storing it. Hence, the algorithm only uses $O(1)$-memory. The algorithm only accesses the 1-neighbourhood $N(u)$ of u, so it is a 1-local routing algorithm. Therefore, we have the following lemma.

▶ **Lemma 9.** *$\mathcal{A}^{up}$ is a deterministic $O(1)$-memory 1-local routing algorithm.*

Next, we prove the convergence of $\mathcal{A}^{up}$, by showing that the algorithm, starting from v_1, reaches t in $O(n)$ hops. To deduce this, we will prove three lemmas.

▶ **Lemma 10.** *Let u be the current vertex of $\mathcal{A}^{up}(v_1, t)$. Then, $u \in \mathcal{S}(v_1)$.*

▶ **Lemma 11.** *Any vertex $v \in \mathcal{S}(v_1)$ is explored by $\mathcal{A}^{up}(v_1, t)$ at most once.*

Proof. We prove the lemma by induction.

The base case is when $v = v_1$. At the beginning, the algorithm is at v_1. Clearly, v_1 cannot be explored again from a neighbour $v \in N(v_1)$, since the first condition $(\rho(v) < \rho(v_1) \leq \rho(t))$ for v_1 being in $N^*(v)$ cannot be met, because $\rho(v_1) = 1$ is the minimum among all vertices of the graph. Hence, v_1 is explored at most once.

The inductive case is when $v \neq v_1$. Suppose v was explored for the first time from u. Our inductive hypothesis is that u was explored at most once. We will prove by contradiction that v cannot be explored again: For the sake of contradiction, suppose $\mathcal{A}^{up}(v_1, t)$ explored

the vertex v for the first time from u and then for a second time from u' (see Figure 7). Then, we have $v \in N^*(u)$ and $v \in N^*(u')$, since the algorithm explicitly checks whether v is an exploration candidate neighbour of u and u'. Firstly, we will show that $u \neq u'$. Having $u = u'$ means that we visited v from u twice. By induction hypothesis, u was explored at most once. So, to visit v twice from u, after exploring v from u for the first time, we have to backtrack from v to u through an ordered Θ-routing step and then explore v again from u. However, this is impossible since the counter-clockwise sweep of l, which determines the exploration order of the vertices in $N^*(u)$, is done only once. Hence, we cannot explore $v \in N^*(u)$ twice from u, meaning that $u \neq u'$.

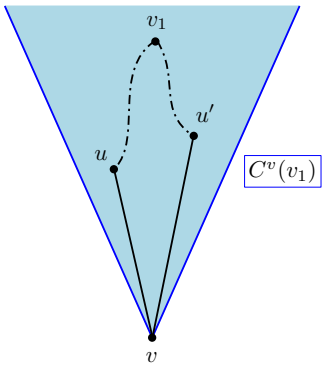

Figure 7 Illustration of a situation where v is explored from u and from u', which leads to a contradiction.

Consider $C^v(v_1)$. Since we explored v from u and then from u', both u and u' lie in the cone $C^v(v_1)$ by the second condition of Definition 6. Also, both u and u' are inserted before v by the first condition of Definition 6. By the general position assumption, only one of u or u' is the closest to v in $C^v(v_1)$. Consequently, during the construction of the ordered Θ-graph, an edge is created from v to only one of u or u', meaning that we have either $v \notin N(u)$ or $v \notin N(u')$ – i.e., v is not a neighbour of either u or u'. This contradicts to the fact that v was explored from u and then from u'. Hence, $v \neq v_1$ can be explored at most once.

Therefore, any vertex $v \in \mathcal{S}(v_1)$ is explored by $\mathcal{A}^{up}(v_1, t)$ at most once. ◀

Observe that, since the algorithm is guaranteed to visit any vertex at most once, we do not have to keep track of which vertices have been explored and which have not.

▶ **Lemma 12.** *For any vertex $v \in \mathcal{S}(v_1)$ explored by $\mathcal{A}^{up}(v_1, t)$, an ordered Θ-routing step is performed at v towards v_1 at most once – i.e, the algorithm backtracks from v at most once.*

By Lemma 10, the local routing algorithm $\mathcal{A}^{up}(v_1, t)$ only explores vertices that belong to $\mathcal{S}(v_1)$. By Lemma 11 and Lemma 12, there can be at most $O(n)$ exploration and backtracking hops because $\mathcal{S}(v_1) \subseteq V$. In addition, we know that $t \in \mathcal{S}(v_1)$ by Lemma 8. Therefore, $\mathcal{A}^{up}(v_1, t)$ requires $O(n)$ hops overall to reach t, implying the following lemma.

▶ **Lemma 13.** *$\mathcal{A}^{up}(v_1, t)$ reaches t in $O(n)$ hops.*

Next, we show that $\mathcal{A}^{up}(v_1, t)$ performs a linear number of operations to determine all exploration candidate neighbours.

▶ **Lemma 14.** *Given any current vertex u during the local routing procedure of $\mathcal{A}^{up}(v_1, t)$, let a* candidate verification *of $v \in N(u)$ be the operations required for checking the two conditions required for having $v \in N^*(u)$. Then, $\mathcal{A}^{up}(v_1, t)$ performs at most $O(kn)$ candidate verifications before reaching t, where k is the number of cones of the ordered Θ-graph.*

4.2.3 Routing from s to t

As $\mathcal{A}^{down}$ routes from s to v_1 and $\mathcal{A}^{up}$ routes from v_1 to t, we obtain our full algorithm.

▶ **Algorithm 4** ($\mathcal{A}(s,t)$)**.** Given the source vertex $s \in V$ and the destination $t \in V$, the algorithm routes from s to t by performing the following two steps:

1. Execute $\mathcal{A}^{down}(s)$ to route from s to v_1.
2. Execute $\mathcal{A}^{up}(v_1, t)$ to route from v_1 to t.

Consequently, we obtain the following theorem.

▶ **Theorem 2.** *Given an arbitrary source vertex $s \in V$ and the destination vertex $t \in V$ of an ordered Θ_k-graph $G = (V, E)$ (for an integer $k \geq 2$), the algorithm $\mathcal{A}(s,t)$ is a deterministic $O(1)$-memory 1-local routing algorithm which successfully routes from s to t in $O(n)$ hops. In particular, $\mathcal{A}(s,t)$ takes $O(kn)$ time in the worst-case.*

5 Conclusion

In this paper, we addressed the following problem with a positive result: *Does there exist a local routing algorithm that works on the ordered Θ-graph?* We first proved there does not exist a deterministic memoryless h-local routing on the ordered Θ_k-graph for any integers $h \geq 1$ and $k \geq 2$. Then, we presented a deterministic $O(1)$-memory local routing algorithm that is guaranteed to reach the destination in $O(n)$ hops and $O(kn)$ time on the ordered Θ_k-graph for any $k \geq 2$. To the best of our knowledge, the algorithm we provided is the first local routing strategy that is successful on the ordered Θ_k-graph.

Although our local routing algorithm $\mathcal{A}(s,t)$ can route between any source and destination, we remark that the total routing path length can be arbitrarily high, which means our local routing algorithm is not c-competitive.

Therefore, one natural open question is whether there exists a c-competitive local routing algorithm using a small amount of memory on the ordered Θ-graph. Things that make finding such an algorithm difficult include the fact that ordered Θ-graphs are generally not plane, the proofs used to bound the spanning ratio by Bose et al. [8] and Bose et al. [6] are highly non-local, and the commonly used properties (such as empty canonical triangles) do not hold for these graphs.

References

1 N. Bonichon, C. Gavoille, N. Hanusse, and D. Ilcinkas. Connections between theta-graphs, Delaunay triangulations, and orthogonal surfaces. In *Proceedings of the 36th International Workshop on Graph Theoretic Concepts in Computer Science*, pages 266–278. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-16926-7_25.

2 P. Bose, J.-L. De Carufel, and O. Devillers. Expected complexity of routing in θ_6 and half-θ_6 graphs. *Journal of Computational Geometry*, 11(1):212–234, 2020. doi:10.20382/jocg.v11i1a9.

3 P. Bose, J.-L. De Carufel, D. Hill, and M. Smid. On the spanning and routing ratio of the directed theta-four graph. *Discrete & Computational Geometry*, 71(3):872–892, 2024. doi:10.1007/s00454-023-00597-8.

4 P. Bose, J.-L. De Carufel, P. Morin, A. van Renssen, and S. Verdonschot. Towards tight bounds on theta-graphs: More is not always better. *Theoretical Computer Science*, 616:70–93, 2016. doi:10.1016/j.tcs.2015.12.017.

5 P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot. Optimal local routing on Delaunay triangulations defined by empty equilateral triangles. *SIAM Journal on Computing*, 44(6):1626–1649, 2015. doi:10.1137/140988103.

6 P. Bose, J. Gudmundsson, and P. Morin. Ordered theta graphs. *Computational Geometry*, 28(1):11–18, 2004. doi:10.1016/j.comgeo.2004.01.003.

7 P. Bose, D. Hill, and A. Ooms. Improved spanning ratio of the theta-5 graph. *Journal of Computational Geometry*, 15(1):66–87, 2024. doi:10.20382/jocg.v15i1a3.

8 P. Bose, P. Morin, and A. van Renssen. The price of order. *International Journal of Computational Geometry & Applications*, 26(03n04):135–149, 2016. doi:10.1142/S0218195916600013.

9 P. Bose and M. Smid. On plane geometric spanners: A survey and open problems. *Computational Geometry*, 46(7):818–830, 2013. doi:10.1016/j.comgeo.2013.04.002.

10 K. Clarkson. Approximation algorithms for shortest path motion planning. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 56–65, 1987. doi:10.1145/28395.28402.

11 E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.

12 N. M. El Molla. Yao spanners for wireless ad hoc networks. Master's thesis, Villanova University, 2009. URL: http://www.csc.villanova.edu/~mdamian/StudentTheses/NawarThesis.pdf.

13 J. M. Keil. Approximating the complete Euclidean graph. In *Proceedings of the 1st Scandinavian Workshop on Algorithm Theory*, pages 208–213, 1988. doi:10.1007/3-540-19487-8_23.

14 E. F. Moore. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*, pages 285–292, 1959.

15 G. Narasimhan and M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007. doi:10.1017/CBO9780511546884.

16 J. Ruppert and R. Seidel. Approximating the d-dimensional complete Euclidean graph. In *Proceedings of the 3rd Canadian Conference on Computational Geometry*, pages 207–210, 1991. URL: https://cccg.ca/proceedings/1991/paper50.pdf.