Competitive Online Routing in Geometric Graphs[*]

Prosenjit Bose Pat Morin

Abstract

We consider online routing algorithms for finding paths between the vertices of plane graphs. Although it has been shown in Bose *et al.* [3] that there exists *no* competitive routing scheme that works on all triangulations, we show that there exists a simple online $O(1)$-memory c-competitive routing strategy that approximates the shortest path in triangulations possessing the *diamond property*, i.e. the total distance travelled by the algorithm to route a message between two vertices is at most a constant c times the shortest path. Our results imply a competitive routing strategy for certain classical triangulations such as the Delaunay, greedy, or minimum-weight triangulation, since they all possess the diamond property.

keywords: Online routing, competitive routing, geometric graph, minimum weight triangulation, delaunay triangulation, greedy triangulation

1 Introduction

Path finding, or routing, is central to a number of fields including geographic information systems, urban planning, robotics, and communication networks. In many cases, knowledge about the environment in which routing takes place is not available beforehand, and the vehicle/robot/packet must learn this information through exploration. Algorithms for routing in these types of environments are referred to as *online* [2] routing algorithms.

In this paper we consider online routing in the following abstract setting [4]: The environment is a plane graph, G (i.e., the planar embedding of G) with n vertices and whose edges are weighted with the Euclidean distance between their endpoints. The source s and destination t are vertices of G, and a packet can only travel on edges of G. Initially, a packet only knows the coordinates of s, t, and $N(s)$, where $N(v)$ denotes the set of vertices adjacent to a node v. When a packet visits a node v, it learns the coordinates of $N(v)$.

Bose and Morin [4] classify routing algorithms based on their use of memory. A deterministic routing algorithm is *memoryless* or *oblivious* if, given a packet currently at vertex v and destined for node t, the algorithm decides where to forward the packet based only on the coordinates of v, t and $N(v)$. An $O(1)$-memory routing algorithm decides where to move a packet based only on the coordinates of v, t, $N(v)$, and the content of its constant size memory M.[1]

[*]This research was partly funded by the Natural Sciences and Engineering Research Council of Canada.

[1]A constant size memory can hold a constant number of vertex identifiers, distances, and $O(\log n)$ bit integers.

We say that a routing algorithm $\mathcal{A}$ is *defeated* by a graph G if there exists a pair of vertices $s, t \in G$ such that a packet stored at s will never reach t when being routed using $\mathcal{A}$. Otherwise, we say that $\mathcal{A}$ *works* for G.

Let $\mathcal{A}(G, s, t)$ denote the length of the walk taken by routing algorithm $\mathcal{A}$ when travelling from vertex s to vertex t of G, and let $SP(G, s, t)$ denote the length of the shortest path, in G, between s and t. We say that $\mathcal{A}$ is *c-competitive* for a class of graphs $\mathcal{G}$ if

$$\frac{\mathcal{A}(G, s, t)}{SP(G, s, t)} \leq c$$

for all graphs $G \in \mathcal{G}$ and all $s, t \in G$, $s \neq t$. We say that $\mathcal{A}$ is simply *competitive* if $\mathcal{A}$ is c-competitive for some constant c.

Recently, several papers have dealt with online routing and related problems in geometric settings. Kalyanasundaram and Pruhs [7] give a 16-competitive algorithm to *explore* any unknown plane graph, i.e., visit all of its nodes. This online exploration problem makes the same assumptions as those made here, but the goal of the problem is to visit all vertices of G, not just t. This difference leads to inherently different solutions.

Kranakis *et al.* [8] give a deterministic oblivious routing algorithm that works for any Delaunay triangulation, and give a deterministic $O(1)$ memory algorithm that works for any connected plane graph.

Bose and Morin [4] also study online routing in geometric settings, particularly triangulations. They give a randomized oblivious routing algorithm that works for any triangulation, and ask whether there is a deterministic oblivious routing algorithm for all triangulations. They also give a competitive $O(1)$-memory routing algorithm for Delaunay triangulations.

Cucka et al. [5] experimentally evaluate the performance of routing algorithms very similar to those described by Kranakis *et al.* [8] and Bose and Morin [4]. When considering the Euclidean distance travelled during point-to-point routing, their results show that the GREEDY routing algorithm [4] performs better than the COMPASS routing algorithm [4, 8] on random graphs, but does not do as well on Delaunay triangulations of random point sets.[2] However, when one considers not the Euclidean distance, but the number of edges traversed (link distance), then the COMPASS routing algorithm is slightly more efficient for both random graphs and Delaunay triangulations. Recently, Bose *et al.* [3] provide a deterministic oblivious routing strategy that works for all triangulations. However, they also show that there is *no competitive online routing algorithm* under the Euclidean distance metric in arbitrary triangulations. In light of this fact, it is interesting to classify which types of triangulations admit competitive routing algorithms since it was shown in [4] that there exist $O(1)$-memory competitive routing strategies for the Delaunay triangulation.

In this paper we explore this question further and present an $O(1)$-memory competitive routing strategy that works for the class of triangulations possessing the *diamond property*. This class is fairly large as it includes such classical triangulations as the Delaunay, greedy and minimum-weight triangulations.

The remainder of the paper is organized as follows: In Section 2 we present a deterministic competitive online routing algorithm for routing on triangulated polygons with two ears. Section 3 presents our results for routing on triangulations that possess the diamond property. Finally, Section 4 summarizes and concludes with open problems.

[2] Cucka *et al.* call these algorithms P-DFS and D-DFS, respectively.

2 Competitive Routing in Triangulated Polygons with Two Ears

Before addressing the problem of routing on plane graphs, we first study the problem in a specific setting that will prove to be quite useful in the sequel. A triangulated simple polygon is a geometric outer-planar graph P where every face except the outer face is a triangle. A vertex of degree two in P is known as an *ear*. In this section, we study triangulated simple polygons with only two ears, s and t. Given such a graph P, we devise a simple online $O(1)$-memory routing strategy that finds a path from s to t such that the total distance travelled by the algorithm when routing from s to t is at most $9 \cdot SP(P, s, t)$ (i.e. the shortest path from s to t in P).

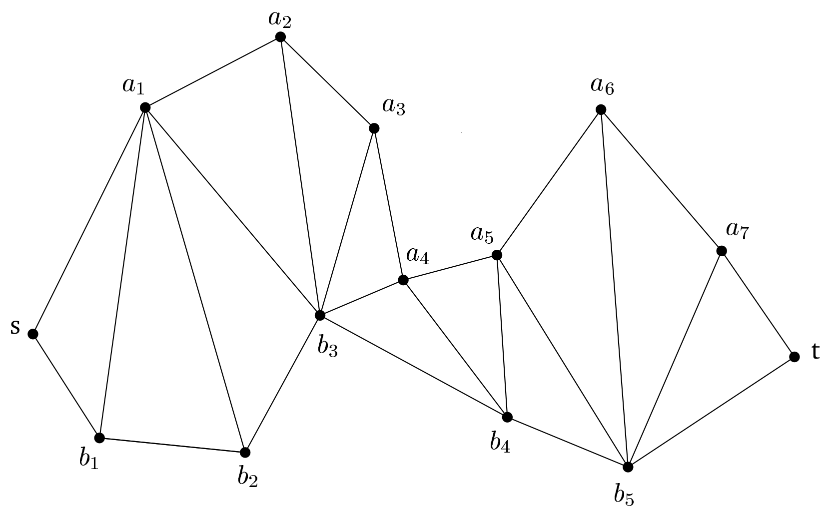

Figure 1: The ears s and t partitions P into an upper and lower chain.

The two ears naturally divide the outer face of P into two chains (see Figure 1). Let $\{s = a_0, a_1, \ldots, a_m = t\}$ be the sequence of vertices in the upper chain and $\{s = b_0, b_1, \ldots, b_n = t\}$ be the sequence of vertices in the lower chain. If the shortest path from s to t happened to be one of these two chains, then one could devise a simple online routing strategy by directly apply a result of Baeza-Yates et al. [1]. Baeza-Yates et al. studied the following problem: given a two-way infinite line and a searcher starting at the origin, the searcher must find a goal that lies at some unknown distance d from the origin. The searcher can only move in unit steps and the objective is to minimize the ratio of the distance traversed to the true distance d. The strategy proposed in [1] is to have the searcher alternate her search between the two sides of the origin and each time the searcher travels a certain distance on one side of the origin, she doubles the distance travelled on the other side of the origin. This results in the searcher travelling at most $9d$ steps to find the goal. By having the upper and lower chain represent each of the two sides of the origin, applying this technique would result in a 9-competitive search strategy. Unfortunately, the shortest path need not be one of the two chains. In fact, the ratio between the length of the shortest path and either of the two chains can be unbounded (see Figure 2).

We circumvent this problem by uncovering some key properties of the shortest-path tree of P rooted at s, denoted $T(s)$. The tree $T(s)$ is the tree formed by taking the union of the shortest paths from s to all the vertices in P. The shortest path from s to a node x in P consists of a

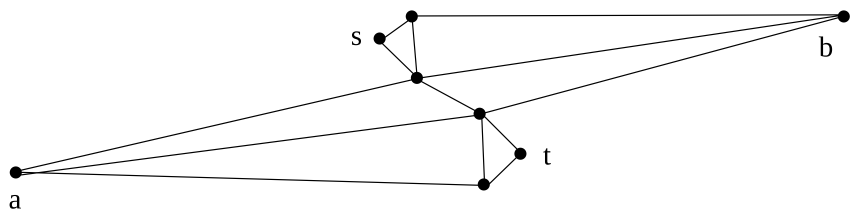

Figure 2: The paths along the lower and upper chains of P can be arbitrarily long.

sequence of nodes from the upper and lower chain. This sequence cannot have a node from the lower chain between two consecutive nodes in the upper chain or vice versa.

We refer to nodes of degree 1 in $T(s)$ as *leaves*, nodes of degree 2 as *internal nodes* and all other nodes as *branching nodes*. The crucial observation is that the shortest path from s to t visits every branching node.

Lemma 1. *Given a triangulated simple polygon P with two ears s and t, the shortest path from s to t in P visits every branching node of the shortest path tree rooted at s.*

Proof. The proof is by induction on the number of vertices in P. The lemma holds trivally when P has 4 vertices. Assume that the lemma holds when P has $4 \leq k < n$ vertices. Let us consider the inductive step when P has $k = n$ vertices. Vertex t has degree 2 and is adjacent to one vertex, a_i from the upper chain and one vertex b_j from the lower chain. Remove t from P to form a triangulated polygon P'. Now, P' has only two ears, s and one of a_i or b_j. Without loss of generality, assume b_j is an ear of P'. By the inductive hypothesis, the shortest path from s to b_j visits every branching node of the shortest path tree rooted at s in P'. By re-introducing t, the shortest path tree does not change very much. In fact, t is a leaf in the tree and it is adjacent either to a_i or b_j. However, the least common ancestor of a_i and b_j in the tree is the first branching node in path from b_j to s. Therefore, t visits every branching node of $T(s)$. □

Branching nodes can be identified locally with only a constant amount of extra information. Consider the node a_i in the upper chain. Let $b_j, b_{j+1}, \dots, b_k$ be the sequence of nodes in the lower chain adjacent to a_i. If we know the length of $SP(P, s, a_{i-1})$ and $SP(P, s, b_j)$, then we can identify whether a_i or any of its adjacent vertices on the lower chain are branching nodes. For example, the node b_j is a branching node if $|SP(P, s, b_j)| + dist(b_j, a_i) < |SP(P, s, a_{i-1})| + dist(a_{i-1}, a_i)$, where $dist(p, q)$ represents the Euclidean distance between points p and q.

The approach to finding a competitive routing algorithm is to move from branching node to branching node in a competitive fashion. Notice that to find a short path between two consecutive branching nodes, we only need to explore two paths, one consisting solely of upper chain vertices and the other of lower chain vertices. The following algorithm, which we call NEXT-BRANCH starts at a branching node x and moves to the next branching node y travelling a total of $9 \cdot SP(P, x, y)$. Since x is a branching node, there are two paths of $T(s)$ leading out of x. One of them leads to y and the other ends at a leaf. Without loss of generality, let $P_1 = x, a_i, a_{i+1}, \dots, a_j, y$ be one of the paths and $P_2 = x, b_k, b_{k+1}, \dots, b_l$ be the other.

NEXT-BRANCH

1: $d = \min dist(x, a_i), dist(x, b_k)$
2: **repeat**
3: travel along P_1 until reaching a branching node or until reaching a vertex a_z such that the length of the path from x to $a_{z+1} > d$. When travelling from one vertex to the next, retain the length of the shortest path to the one upper chain and one lower chain vertex required to compute all shortest path information at the next step. If a branching node is reached quit, otherwise return to x.
4: $d \leftarrow 2d$
5: travel along P_2 until reaching a branching node or until reaching a vertex b_z such that the length of the path from x to $a_{b+1} > d$. When travelling from one vertex to the next, retain the length of the shortest path to the one upper chain and one lower chain vertex required to compute all shortest path information at the next step. If a branching node is reached quit, otherwise return to x.
6: $d \leftarrow 2d$
7: **until** y is reached

We note that in steps 3 and 5 of the algorithm, it is not necessary to return to x upon an unsucessful search. In fact, it would be easier to simply follow an edge connecting a vertex of the upper (resp. lower) chain to a vertex on the lower (resp. upper) chain. Although this shortcutting will likely reduce the distance travelled in practice, it is difficult to reduce the upper bound when these short cuts are taken and the analysis is much simpler if one returns to x after each unsuccesful search.

Lemma 2. *Starting at x,* NEXT-BRANCH *reaches y after travelling a total of* $9 \cdot SP(P, x, y)$.

Proof. Let $c = \min\{dist(x, a_i), dist(x, b_k)\}$. Let $d_f = 2^k c$ be the value of d during the final exploration step (Line 3 or Line 5) of the algorithm. Therefore, the total distance travelled by the algorithm is equal to

$$\begin{aligned} D &= 2 \cdot \sum_{i=1}^{k-1} 2^i c + L \\ &\leq 2^{k+1} c + L \end{aligned}$$

where L is the distance travelled during the last exploration step. There are now two cases to consider.

Case 1: The algorithm terminated while exploring the shorter of the two paths P_1 or P_2. Then $d_f \leq 4 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\}$, otherwise the algorithm would have reached y in the previous iteration of the algorithm. Therefore

$$\begin{aligned} D &\leq 8 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\} + L \\ &= 9 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\} \end{aligned}$$

Case 2: The algorithm terminated while exploring the longer of the two paths P_1 or P_2. Then $x \leq d_f \leq 2 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\}$, otherwise the algorithm would have reached y in the previous exploration step. Then

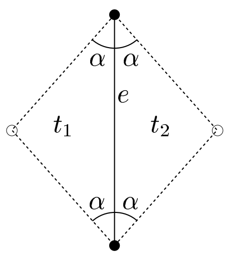

Figure 3: The edge e satisfies the diamond property if at least one of t_1 and t_2 does not contain any point of V in its interior.

$$\begin{aligned} D &\leq 4 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\} + L \\ &\leq 6 \cdot \min\{\text{length}(P_1), \text{length}(P_2)\} \end{aligned}$$

In both cases, the conditions of the lemma are satisfied. □

Putting Lemma 1 and Lemma 2 together, we devise FIND-SHORT-PATH, an online competitive $O(1)$ memory routing strategy to move from s to t in P. Starting at vertex s, repeatedly invoke NEXT-BRANCH until t is reached.

Theorem 1. FIND-SHORT-PATH *reaches* t *after having travelled at most* 9 *times* $SP(P, s, t)$.

Proof. By Lemma 1, the shortest path from s to t must visit every branching node. Since each of these steps is 9-competitive, by Lemma 2, the theorem follows. □

3 Competitive Routing in Triangulations

Although it was shown in [3] that there is *no competitive online routing algorithm* under the Euclidean distance metric in arbitrary triangulations, in this section we provide an $O(1)$-memory competitive algorithm for the class of triangulations possessing the *diamond* property. Das and Joseph [6] showed these triangulations approximate the complete Euclidean graph in terms of the shortest path length. We elaborate on the precise definition of this property.

Let α be any angle $0 < \alpha \leq \pi/2$. For an edge e of a triangulation $T = (V, E)$, consider the two isosceles triangles t_1 and t_2 whose base is e and with base angle α. Refer to Fig. 3. The edge e satisfies the *diamond property* with parameter α if one of t_1 or t_2 does not contain any point of V in its interior. A triangulation T satisfies the *diamond property* with parameter α if every edge of T satisfies the diamond property with parameter α. Das and Joseph prove the following.

Lemma 3. *[6] Given a triangulation* $T = (V, E)$ *satisfying the diamond property with parameter* α*, there exists a constant* d_α *(depending on* α*), such that* $\forall x, y \in V, SP(T, x, y)/dist(x, y) \leq d_\alpha$.

They showed that the diamond property is not an obscure property that is possessed by only a few triangulations but that the class of triangulations possessing the diamond property is fairly rich and includes some of the classical triangulations.

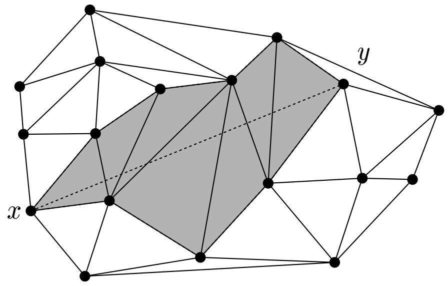

Figure 4: The graph T_{xy} (shaded).

Lemma 4. *[6] The set of triangulations satisfying the diamond property include such classical triangulations as the Delaunay triangulation, the minimum weight triangulation and the greedy triangulation.*

Given two vertices x, y in a triangulation T, consider the set S_{xy} of triangles of T whose interiors intersect the line segment $[x, y]$. Define T_{xy} as the subgraph of T containing only those edges of T bounding triangles of S_{xy}. By construction, T_{xy} is a triangulated simple polygon with two ears, x and y.[3] An example is shown in Fig. 4.

Lemma 5. *Given a triangulation $T = (V, E)$ satisfying the diamond property with parameter α, and two vertices $x, y \in V$, the shortest path between x and y in T_{xy} is at most d_α times $dist(x, y)$.*

Proof. Essentially, although it is not stated explicitly, a careful analysis of the proof of Lemma 3 reveals that in fact, for every triangulation satisfying the diamond property with parameter α, the shortest path between x and y in T_{xy} has length at most d_α times the line segment $[x, y]$. This is stronger than simply the shortest path, but the distance is related to the Euclidean distance. □

Note that the above lemma shows that the path is related not only to the shortest path between x and y but in fact to the Euclidean distance between x and y. In order to route in a competitive fashion between vertices x, y in a triangulation T possessing the diamond property, attention can be restricted to the subgraph T_{xy}. Given a vertex z of T, a local test determines whether or not z lies in T_{xy} by testing whether any of the edges adjacent to z intersect the segment $[x, y]$. This leads to a routing scheme based on algorithm FIND-SHORT-PATH. Basically, starting at vertex x, run FIND-SHORT-PATH until y is reached. At each step in the algorithm, a local test allows one to determine which vertices form the upper and lower chain. We conclude with the following:

Theorem 2. *Given a triangulation $T = (V, E)$ satisfying the diamond property with parameter α, and two vertices $x, y \in V$, a modified* FIND-SHORT-PATH *is an $O(1)$-memory online competitive routing algorithm that moves a packet from x to y after travelling a total of at most $9 \cdot d_\alpha \cdot dist(x, y)$.*

4 Conclusions

Given that no competitive routing strategy works for all triangulations, in this paper we presented an $O(1)$-memory competitive routing strategy that works for the class of triangulations possessing

[3]There are degenerate cases in which vertices other than x and y lie on the segment $[x, y]$. In such cases, T_{xy} is a sequence of triangulated simple polygons joined at the vertices on $[x, y]$.

the *diamond property*. This class is fairly large as it includes such classical triangulations as the Delaunay, greedy and minimum-weight triangulations. The routing strategy is based on a simple online competitive strategy for routing on triangulated simple polygons.

These results are in contrast with results for the *link distance metric*, where the length of a path is the number of edges it uses. It is known [3] that no competitive algorithm exists for greedy, minimum-weight, or Delaunay triangulations under this metric. This raises the question: For what classes of geometric graphs do competitive routing algorithms exist under the link distance metric?

References

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

[2] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[3] P. Bose, A. Brodnik, S. Carlsson, E. Demaine, R. Fleischer, A. Lopez, P. Morin, and I. Munro. Online routing in convex subdivisions. In *Proceedings of the 11th International Symposium on Algorithms and Computation (ISAAC'00)*, Springer LNCS, 2000.

[4] P. Bose and P. Morin. Online routing in triangulations. In *Proceedings of the Tenth International Symposium on Algorithms and Computation (ISAAC'99)*, volume 1741 of *Springer LNCS*, pages 113–122, 1999.

[5] P. Cucka, N. S. Netanyahu, and A. Rosenfeld. Learning in navigation: Goal finding in graphs. *International Journal of Pattern Recognition and Artificial Intelligence*, 10(5):429–446, 1996.

[6] G. Das and D. Joseph. Which triangulations approximate the complete graph? In *Proceedings of the International Symposium on Optimal Algorithms*, pages 168–192, 1989.

[7] B. Kalyanasundaram and K. R. Pruhs. Constructing competitive tours from local information. *Theoretical Computer Science*, 130:125–138, 1994.

[8] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG'99)*, 1999.