Competitive Online Routing in Geometric Graphs ⋆

Prosenjit Bose and Pat Morin

School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, Canada, K1S 5B6.

Abstract

We consider online routing algorithms for finding paths between the vertices of plane graphs. Although it has been shown in Bose et al. [4] 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. We then generalize our results to show that the $O(1)$-memory c-competitive routing strategy works for all plane graphs possessing both the diamond property and the good convex polygon property.

Key words: Online routing, competitive routing, geometric graph, minimum weight triangulation, Delaunay triangulation, greedy triangulation, spanner, spanning ratio, planar graph, good polygon.

1 Introduction

Path finding, or routing, is central to a number of fields including geographic information systems, urban planning, robotics, and communication networks.

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

Email address: {jit,morin}@scs.carleton.ca (Prosenjit Bose and Pat Morin).

URL: www.scs.carleton.ca/{~jit,~morin} (Prosenjit Bose and Pat Morin).

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* [3] routing algorithms.

In this paper we consider online routing in the following abstract setting: 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 $N(v)$.

Network routing has a rich history and has been studied in many different contexts (see the handbook [2] for a comprehensive review). The starting point for this paper is the paper [6] where online geometric routing algorithms are classified 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 must forward the packet and the *only* information available to the algorithm is the coordinates of v, t and $N(v)$. An $O(1)$-memory routing algorithm decides where to move a packet when the only information available to the algorithm is the coordinates of v, t, $N(v)$, and the content of its constant size memory. Henceforth, we assume that a constant size memory can hold a constant number of vertex identifiers, distances, and $O(\log n)$ bit integers [1] .

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

[1] In certain contexts, this is sometimes referred to as log memory, however in our setting, we want to emphasize that the memory only holds a *constant* number of words each consisting of at most $O(\log n)$ bits.

in geometric settings. Kalyanasundaram and Pruhs [10] 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. [11] 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 [6] 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. [7] experimentally evaluate the performance of routing algorithms very similar to those described by Kranakis et al. [11] and Bose and Morin [6]. When considering the Euclidean distance travelled during point-to-point routing, their results show that the GREEDY routing algorithm [6] performs better than the COMPASS routing algorithm [6,11] 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. [4] 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 [6] 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. We then generalize this to show that in fact, the routing strategy works for all plane graphs possessing both the diamond property and the *good convex polygon property*.

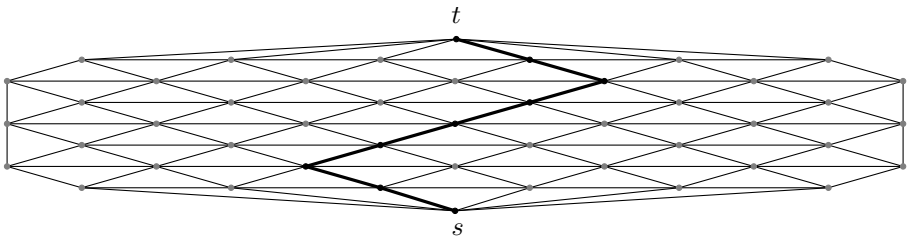
The remainder of the paper is organized as follows: In Section 2, we review the lower bound construction of Bose et al. [4]. In Section 3 we present a de-

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

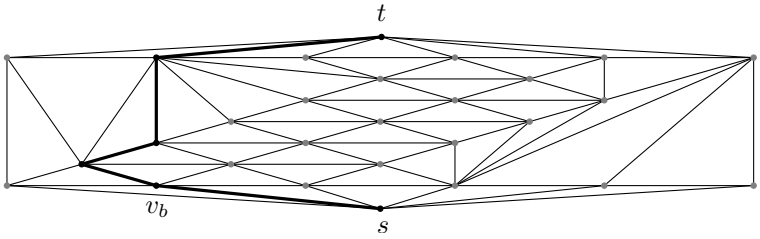
terministic competitive online routing algorithm for routing on triangulated polygons with two ears. Section 4 presents our results for routing on triangulations that possess the diamond property. Finally, Section 6 summarizes and concludes with open problems.

2 Lower Bounds

By modifying a proof of Papadimitriou and Yannakakis [12], Bose et al. [4] showed that under the Euclidean metric, no deterministic routing algorithm is $o(\sqrt{n})$-competitive for all triangulations. The idea behind their argument is depicted in Fig. 1. Given any deterministic routing algorithm, observe the path it obtains between s and t on the n vertex triangulation in Fig. 1 (a). If each row in the triangulation has $\Omega(\sqrt{n})$ vertices and the length of every horizontal segment is n, then the length of the path is at least $\Omega(n\sqrt{n})$ since the shortest path from s to t has length $\Omega(n\sqrt{n})$. Next, construct a new graph which keeps this path and every vertex adjacent to this path intact but has a shortcut of length $O(n)$ from s to t. The path found by the deterministic algorithm on the new graph will still have length $\Omega(n\sqrt{n})$ since all of the vertices visited by the algorithm look identical.



(a)



(b)

Fig. 1. (a) The triangulation T with the path found by $\mathcal{A}$ indicated. (b) The resulting triangulation T' with the "almost-vertical" path shown in bold.

Theorem 1 *[4] Under the Euclidean distance metric, no deterministic routing algorithm is $o(\sqrt{n})$ competitive for all triangulations.*

Intuitively, the main problem with being able to route competitively online is the existence of long skinny triangles. Essentially, it is impossible to visit too many long skinny triangles while searching for a path. Note that the triangulation in Fig. 1(a) has many long skinny triangles making it possible to hide a shortcut as was done in Fig. 1(b) once a deterministic algorithm has computed a path in (a). A natural question to ask is what additional property would allow the existence of a competitive routing scheme. In [6], we showed that if a triangulation is Delaunay, then an $O(1)$-memory online competitive routing scheme exists. In this paper, we show that a weaker geometric property is sufficient, namely the diamond property for triangulations and additionally the good convex polygon property for plane graphs. All of these properties essentially remove long skinny triangles.

3 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 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).

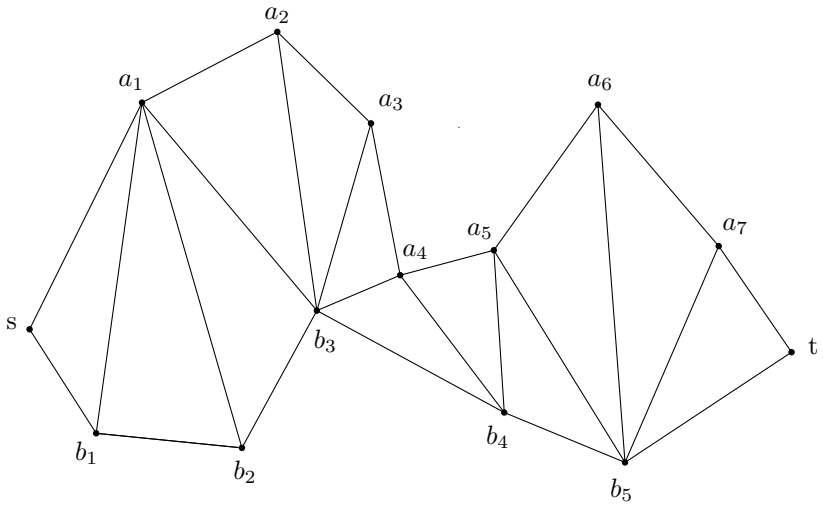


Fig. 2. The ears s and t partition P into an upper and lower chain.

The two ears naturally divide the outer face of P into two chains (see Figure 2). Let $\{s = a_0, a_1, \ldots, a_m = t\}$ be the sequence of vertices in the upper chain

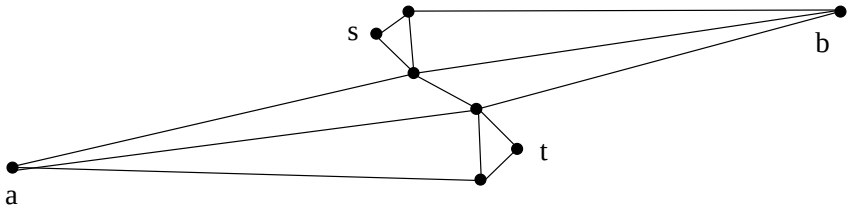


Fig. 3. The paths along the lower and upper chains of P can be arbitrarily long.

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 applying 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 3).

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 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 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, by the triangle inequality.

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 2 *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: Without loss of generality, consider an arbitrary branching vertex, a_i, of $T(s)$ on the upper chain of P. The argument is symmetric for a branching vertex on the lower chain. Since a_i has degree at least 3, it must be adjacent to at least one vertex b_j on the lower chain. The two vertices a_i and b_j form

a cut set of the graph P. Therefore, every path from s to t in P goes through either a_i or b_j. In particular, the path in $T(s)$ from s to t must go through one of these two vertices. We have two cases to consider. The first case is when b_j is a child of a_i in $T(s)$. In this case, the unique path from s to t in $T(s)$ must go through a_i.

The second case is when a_i is a child of b_j. In this case, we need to show that the subtree rooted at a_i contains t. Since P is triangulated, we have that either a_i is adjacent to b_{j-1} or b_j is adjacent to a_{i-1} in P. The former contradicts that the shortest path from s to a_i goes through b_j, therefore, we must have the latter. The latter implies that a_i cannot be adjacent to a_{i-1} in $T(s)$, otherwise $T(s)$ would contain a cycle. Therefore, since a_i has degree at least 3 in $T(s)$, it must be adjacent to another vertex on the lower chain, b_k. Now, k must be greater than j, since we established that the edge $b_j a_{i-1}$ exists. Again, since P is a triangulation, a_i must be adjacent to all the vertices from b_j to b_k on the lower chain in P. This implies that b_j has two children in $T(s)$. One of them is a_i and the other is b_{j+1}. The subtree rooted at b_{j+1} is a path on the lower chain that ends at b_{k-1}. Therefore, the subtree rooted at a_i must contain t, thereby implying that the shortest path from s to t goes through a_i.

□

Branching nodes can be identified locally with only a constant amount of extra information. Consider the node a_i in the upper chain. Let $b_j, b_{j+1}, \ldots, b_k$ 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. Therefore, the only information required to determine if a_i or any of its neighbors is a branching node is $SP(P, s, a_{i-1})$ and $SP(P, s, b_j)$. This is precisely the information that we will maintain while routing.

The approach to finding a competitive routing algorithm is to move from branching node to branching node in a competitive fashion. 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 solely 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 $P_1 = x, a_i, a_{i+1}, \ldots, a_j, y$ be one of the paths and $P_2 = x, b_k, b_{k+1}, \ldots, b_l$ be the other. The algorithm is outlined in Fig. 4.

NEXT-BRANCH

The input is either the starting point or a branching node denoted x. The output is either the final destination t or the next branching node on the shortest path from s to t along with its furthest adjacent neighbors on the upper and lower chains.

The loop invariants maintained are: C contains the current position, a_n is the furthest vertex on the upper chain adjacent to C, a_p is its predecessor on the upper chain, b_n is the furthest vertex on the lower chain adjacent to C, b_p is its predecessor on the lower chain, S_a is the length of the shortest path from x to a_n and S_b is the length of the shortest path from x to b_n.

1. $d = \min dist(x, a_i), dist(x, b_k),$
2. $C, a_p, b_p = x$
3. $a_n = a_i$, $b_n = b_k$, $S_a = dist(x, a_i)$, $S_b = dist(x, b_k)$
4. While (TRUE) {
    5. While ($S_a < d$) {
        6. $C = a_n$. (i.e. move current vertex to a_n.)
        7. If C is an ear, then the destination is reached. Move to C and exit.
        8. If C or any vertex in $N(C)$ is a branching node, move to that node, output its furthest neighbor on the upper and lower chain and exit.
        9. Set a_n to the next vertex on upper chain adjacent to C and update S_a and a_p. This can be done since a_p, b_n, $N(a_n)$, S_a and S_b are available.
        10. Set b_n to the furthest vertex on the lower chain adjacent to C and update S_b and b_p. }
    11. $d \leftarrow 2d$
    12. While ($S_b < d$) {
        13. $C = b_n$
        14. If C is an ear, then the destination is reached. Move to C and exit.
        15. If C or any vertex in $N(C)$ is a branching node, move to that node, output its furthest neighbor on the upper and lower chain and exit. Again, this can be done since we know S_a and S_b.
        16. Set b_n to the next vertex on upper chain adjacent to C and update S_b and b_p. This can be done since b_p, a_n, $N(b_n)$, S_a and S_b are available.
        17. Set a_n to the furthest vertex on the lower chain adjacent to C and update S_a and a_p. }
    18. $d \leftarrow 2d$
19. }

Fig. 4. Algorithm NEXT-BRANCH

Clearly, the algorithm uses only a constant amount of memory as outlined by the constant number of loop variables maintained. A subtle detail that remains to be clarified is how to determine what is the next vertex on the upper (resp. lower) chain, given that the current vertex on the upper (resp. lower) chain (see Lines 8, 9,10, 14, 15 and 16). We address the situation in Line 9. All other cases are similar. In Line 9, we wish to compute the next neighbor to a_n on the upper chain. This vertex is a_n's clockwise neighbor after a_p in $N(a_n)$.

Lemma 3 *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 4 or Line 11) of the algorithm. Therefore, the total distance travelled by the algorithm is bounded by

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

where L is the distance travelled during the last exploration step. Note that $2^i c$ is strictly an upper bound on the distance travelled since we do not actually return to x prior to the invocation of Line 4 and Line 11. 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

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

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

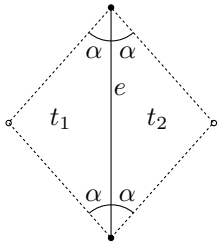


Fig. 5. The edge e satisfies the diamond property if at least one of t_1 and t_2 does not contain any point of V in its interior.

Putting Lemma 2 and Lemma 3 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 4 FIND-SHORT-PATH *is an online competitive* $O(1)$ *memory routing strategy. Given triangulated polygon* P *and ears* s *and* t*, the algorithm reaches* t *after having travelled at most* 9 *times* $SP(P, s, t)$*.*

Proof: By Lemma 2, the shortest path from s to t must visit every branching node. Since each of these steps is 9-competitive, by Lemma 3, the theorem follows. □

4 Competitive Routing in Triangulations

Although 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 [8], whose work is a generalization of the seminal paper by Dobkin, Friedman and Supowit [9], showed these triangulations approximate the complete Euclidean graph in terms of the shortest path length. A graph G approximates the complete Euclidean graph in terms of shortest path length if for every pair of vertices, u, v in G, the ratio between $SP(G, u, v)$ and the Euclidean distance between u and v is a constant. Such graphs are known as *spanners*.

We elaborate on the precise definition of the diamond 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. 5. 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 5 *[8] 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.

Lemma 6 *[8] The set of triangulations satisfying the diamond property include such classical triangulations as the Delaunay triangulation, the minimum weight triangulation and the greedy triangulation*[3] *.*

Given two vertices s, t in a triangulation T, consider the set S_{st} of triangles of T whose interiors intersect the line segment $[s, t]$. Define T_{st} as the subgraph of T containing only those edges of T bounding triangles of S_{st}. An example is shown in Fig. 6.

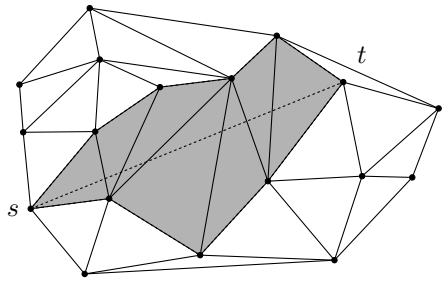


Fig. 6. The graph T_{st} (shaded).

At this point, one may be tempted to assume that T_{st} is a triangulated simple polygon with two ears and simply apply Theorem 4. However, the polygon is not necessarily simple as can be seen in Fig. 7. A careful examination reveals that the region is a triangulated *weakly simple* polygon. Many definitions of *weakly simple* polygon exist in the literature. The simplest is the following: A polygon is weakly simple provided that the graph defined by its vertices and edges is plane, the outer face is a cycle, and one bounded face is adjacent to all vertices.

Since one bounded face is adjacent to all the vertices, a weakly-simple polygon can be symbolically transformed into a simple polygon by essentially doubling degenerate edges and vertices. A vertex is degenerate if it is not on the outer face or if it is on the outer face and has degree greater than 2. An edge is degenerate if it is not on the outer face. Once the degenerate edges and vertices are doubled, running an Euler tour gives the simple polygon. An

[3] The greedy triangulation is obtained by starting with a set of points and adding edges in non-decreasing order as long as the graph remains planar.

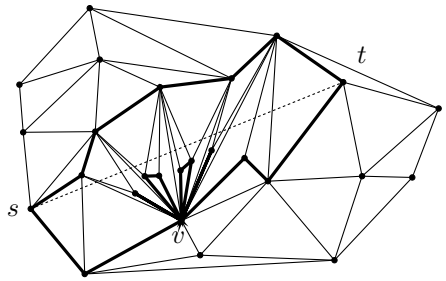


Fig. 7. Note that the lower chain is not simple. Observe vertex v. The graph T_{st} is a triangulated weakly simple polygon. The boundary of the weakly simple polygon is highlighted in bold.

illustration is given in Fig. 8 (see also [5] for another description of this well-known technique.). For the remainder of this section, we assume that T_{st} represents the simple polygon obtained by symbolic transformation. Since T_{st} is triangulated and has the diamond property, we can prove the following.

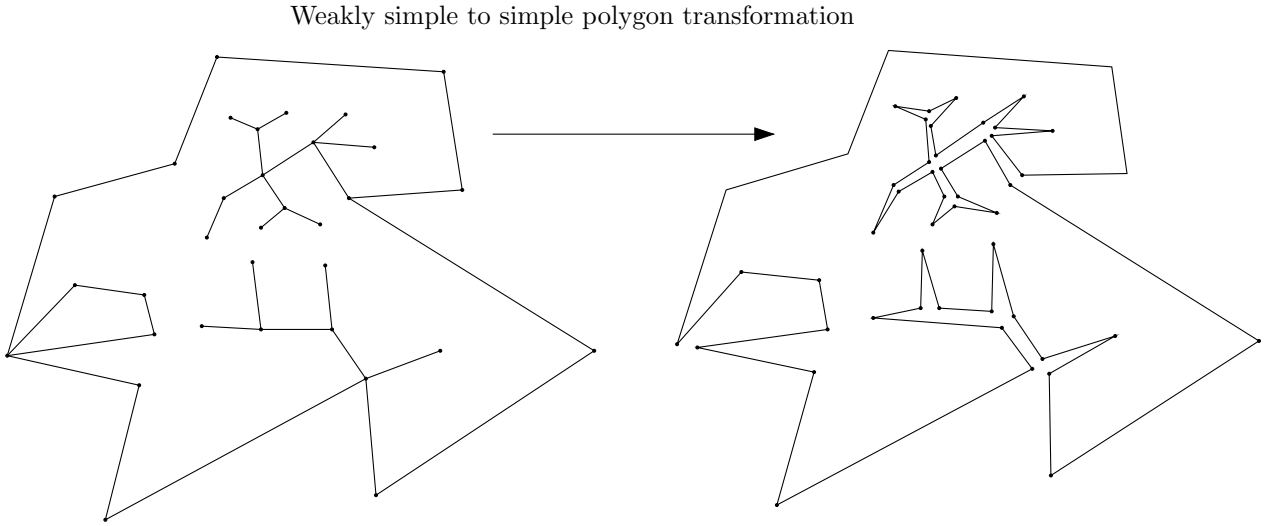


Fig. 8. Transformation from weakly simple to simple polygon

Lemma 7 *Given a triangulation $T = (V, E)$ satisfying the diamond property with parameter α, and two vertices $s, t \in V$, the shortest path between s and t in T_{st} is at most d_α times dist(s, t).*

Proof: In the proof of Lemma 5 in [8], the authors show that for every triangulation satisfying the diamond property with parameter α, the shortest path between s and t in the triangulation has length at most d_α times the line segment $[s, t]$. The proof is constructive and provides several different methods for finding such a path depending on the different cases.

Although not explicitly stated, a careful but fairly straightforward analysis of the proof reveals that none of the path construction methods presented in [8] use any edges that are not in T_{st}. Furthermore, all of the paths constructed

consist of alternating portions of the upper and lower boundary connected by edges crossing the segment $[st]$. The portions of the upper and lower boundary paths appear in the same order as they appear on the boundary in T_{st}. That is, in the case of a degenerate vertex, the vertex may appear multiple times in the path constructed.

The lemma follows. □

Note that the above lemma shows that the path is related not only to the shortest path between s and t but in fact to the Euclidean distance between s and t. In order to route in a competitive fashion between vertices s, t in a triangulation T possessing the diamond property, attention can be restricted to the subgraph T_{st}. Actually, we want to route using FIND-SHORT-PATH on the triangulated simple polygon obtained by symbolic perturbation of the triangulated weakly simple polygon. In order to do this, we need to modify NEXT-BRANCH slightly. First, we need to show how to determine if a vertex is on the upper chain or lower chain. Consider the line L through s and t. For simplicity assume the line is horizontal. A vertex v is on the upper (resp. lower) chain if it is above (resp. below) L and is adjacent to a vertex below (resp. above) L. Thus, the only local information needed to decide this is the line L. Next, we need to show how to determine what is the next vertex on the upper (resp. lower) chain, given that the current vertex on the upper (resp. lower) chain in this new context (this affects Lines 8, 9,10, 14, 15 and 16 in NEXT-BRANCH). All the cases are similar so we address the one in Line 9. Here, we wish to compute the next neighbor to a_n on the upper chain. Now, a_n may or may not be degenerate, however, since T_{st} is triangulated, we know that a_p, a_n, b_n form a triangle. The next vertex on the upper chain adjacent to a_n is the first vertex in $N(v)$ that is above L counter-clockwise from b_n.

Therefore, we can route using FIND-SHORT-PATH and at each step in the algorithm only local tests are used. We conclude with the following:

Theorem 8 *Given a triangulation $T = (V, E)$ satisfying the diamond property with parameter α, and two vertices $s, t \in V$, a modified* FIND-SHORT-PATH *is an $O(1)$-memory online competitive routing algorithm that moves a packet from s to t after travelling a total of at most $9 \cdot d_\alpha \cdot dist(s, t)$.*

5 Competitive Routing in Plane Graphs with the Diamond and Good Convex Polygon Property

Intuitively, the diamond property ensures that an edge does not act as a barrier between two vertices that are relatively close (i.e. within the diamond).

A graph possessing the diamond property does not necessarily need to be a triangulation in order to be a spanner. Das and Joseph showed that if every bounded face of a plane graph also has the *good polygon property*, then the graph is a spanner. A plane graph has the good polygon property provided that for every pair s, t of visible vertices on the boundary of a face, the shortest distance from s to t around the boundary of the face is at most a constant times $|st|$.

We need a slightly stronger condition than the one defined in [8], namely the *good convex polygon property*. A plane graph possesses the *good convex polygon property* if every bounded face of G is convex and has the property that for every pair of vertices, s, t on the boundary of the face, the shortest distance from s to t around the boundary of the face is at most $g*|st|$, for some constant g. Notice that the good convex polygon property trivially holds for triangles. Let G be a plane graph satisfying the diamond property and the good convex polygon property. Consider a non-adjacent pair of vertices s, t in G. Let F_{st} be the faces of G whose interiors intersect the line segment $[st]$. Define G_{st} as the subgraph of G induced by all vertices of F_{st}. Note that the segment $[st]$ only intersects a face once since each face is convex. Thus, by removing the edges that intersect $[st]$ we have an upper chain and lower chain after modifying for degenerate edges and vertices as in the previous section. This is precisely the reason we need this stronger condition since a segment can intersect a face multiple times if the graph only has the good polygon property. With a segment intersecting multiple edges, there no longer necessarily exists an upper chain and lower chain when routing.

Lemma 9 *Let $G = (V, E)$ be a plane graph satisfying the diamond property with parameter α, and the good convex polygon property with constant g. Given two vertices $s, t \in V$, the shortest path between s and t in G_{st} is at most gd_α times $dist(s, t)$ for constants g, d_α.*

Proof: As in the proof of Lemma 7, this lemma follows from a careful analysis of Lemma 5 in [8]. The authors prove the result for plane graphs satisfying the diamond and good polygon properties.

We note that all of the paths constructed consist of alternating portions of the upper and lower boundary connected by edges crossing the segment $[st]$. The portions of the upper and lower boundary paths appear in the same order as they appear on the boundary in G_{st}. So in the case of a degenerate vertex, the vertex may appear multiple times in the path constructed. □

Now the challenge is to route competitively in G_{st} using only $O(1)$ memory. G_{st} may have some edges missing between the upper and lower chains. The algorithm developed in the previous section is no longer directly applicable.

Two main difficulties arise because of the missing edges. First, the branching vertices in the shortest path tree rooted at s can no longer be identified locally since edges may be absent. Second, and more importantly, the shortest path from s to t no longer necessarily visits all of the branching vertices.

We need to generalize Lemma 2 to account for missing edges by exploiting the properties of G_{st}. Given the shortest path tree $T(s)$ rooted at s, we define *a crossing vertex* to be a vertex u on an upper (resp. lower chain) of G_{st} with at least one child in $T(s)$ that is in the lower (resp. upper chain) of G_{st}. This property is precisely what is required for the shortest path from s to t to go through u.

Lemma 10 *Given the graph G_{st}, the shortest path from s to t in G_{st} visits every crossing node of the shortest path tree rooted at s.*

Proof: Without loss of generality, consider an arbitrary crossing vertex, u, of $T(s)$ on the upper chain of G_{st}. The argument is symmetric for a crossing vertex on the lower chain. By definition, u has a child v in the lower chain. These two vertices form a cut set of the graph G_{st} whose removal puts s and t in different components. Therefore, every path from s to t visits either u or v. Since v is a child of u in $T(s)$, the lemma follows. □

The above lemma permits us to identify the important vertices to consider when routing from s to t similar to the branching vertices in the previous section. If we could easily identify the crossing vertices, then we could apply an algorithm similar to NEXT-BRANCH to walk from one crossing vertex to the next. However, the main problem is to identify crossing vertices locally. For a given vertex v, computing the exact length of the shortest path from s to every vertex in $N(v)$ is no longer a local operation since there are missing edges, which makes it difficult to determine locally if a vertex is a crossing vertex. The crucial idea is that we do not need to precisely identify crossing vertices, but by exploiting the diamond and good convex polygon properties, we can identify approximations of crossing vertices. We elaborate below.

The line through the vertices s and t naturally divides the vertices of G_{st} into two chains. 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. The only candidates for crossing vertices are the vertices on the upper chain adjacent to at least one vertex in the lower chain and vice versa. Let $C = \{c_1, \ldots, c_j\}$ and $D = \{d_1, \ldots, d_k\}$ be the ordered subsequence of vertices of the upper and lower chains that are candidates. Consider a vertex c_i on the upper chain adjacent to a vertex d_j on the lower chain. What information is needed to determine if c_i is a crossing vertex? The vertex c_i is a crossing vertex provided that $|SP(G_{st}, s, c_{i-1})| + |SP(G_{st}, c_{i-1}, c_i)| +$

$|c_i d_j| \leq |SP(G_{st}, s, d_{j-1})| + |SP(G_{st}, d_{j-1}, d_j)|$. However, we cannot maintain this shortest path information locally. We now show how to exploit the diamond and good convex polygon property in order to approximately compute crossing vertices.

First, we are no longer so ambitious as to maintain exact shortest path information as was done in the previous sections, but we maintain approximate shortest path information. The approximation factor is the factor given by the diamond and good convex polygon properties in Lemma 9. For any pair of vertices x, y, we can compute an upper bound on the length of the shortest path (namely the path approximate shortest path constructed by Das and Joseph has length at most $gd_\alpha \cdot dist(x, y)$). We shall refer to their path as the approximate shortest path. Therefore, Lemma 9 allows us to compute an approximation to the length of the shortest path between two vertices by simply knowing their coordinates. The approximation is even better ($g \cdot dist(x, y)$) if x and y are on the same face. We use this approximate information to determine if c_i is an approximate crossing vertex. Let $SP'(G_{st}, x, y)$ represent the length of the approximate shortest path from x to y in G_{st}. By Lemma 9, we set $|SP'(G_{st}, x, y)| = gd_\alpha|xy|$. The vertex c_i is an approximate crossing vertex provided that $|SP'(G_{st}, s, c_{i-1})|+g|c_{i-1}c_i|+|c_i d_j| \leq |SP'(G_{st}, s, d_{j-1})|+g|d_{j-1}d_j)|$. Note that d_{j-1} and d_j are on a common face and c_{i-1} and c_i are also on a common face. We now show that the approximate shortest path from s to t must contain c_i.

Lemma 11 *If c_i is an approximate crossing vertex, the approximate shortest path from s to t contains vertex c_i.*

Proof: Since c_i is an approximate crossing vertex, we have that $|SP'(G_{st}, s, c_{i-1})|+ g|c_{i-1}c_i| + |c_i d_j| \leq |SP'(G_{st}, s, d_{j-1})| + g|d_{j-1}d_j)|$. Recall that the vertices c_i and d_j form a cut set which means all paths from s to t must go through at least one of the two vertices. Since the approximate shortest path from s to d_j contains c_i by the above relation, the lemma follows.

□

We now have a way of locally determining if a vertex is an approximate crossing node. To determine if c_i is an approximate crossing node, we need to have the coordinates of c_{i-1}, d_j, d_{j-1}, s and the approximation factors g and d_α. We can determine if a node is on the upper or lower chain by its relation to the line through s and t. Since there are edges missing between the upper and lower chains in G_{st}, we do not need to remember the furthest and second furthest vertices seen on the upper and lower chains but the furthest and second furthest vertices in the sets C and D, as required to determine if vertices are approximate crossing or not. These simple modifications to the algorithm

NEXT-BRANCH incorporating the above produce an algorithm NEXT-CROSS to walk from one approximate crossing vertex to the next. This in turn allows us to define a modified FIND-SHORT-PATH.

Theorem 12 *Given a plane graph $G = (V, E)$ satisfying the diamond property with parameter α and the good convex polygon property with constant g, and two vertices $s, t \in V$, a modified* FIND-SHORT-PATH *is an $O(1)$-memory online competitive routing algorithm that moves a packet from s to t after travelling a total of at most $9g \cdot d_\alpha \cdot dist(s, t)$.*

Proof: Similar to the proof of Lemma 3 and Theorem 4 □

6 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 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. We then generalized this result to show that the routing strategy works on all plane triangulations possessing both the diamond property and the good convex polygon property.

One question that immediately comes to mind is whether or not these two geometric properties are necessary for the existence of a competitive online strategy. The lower bound construction implies that some additional property is necessary for a competitive algorithm to exist. Is the diamond property and the good convex polygon property combined the weakest properties that still guarantee the existence of a competitive routing algorithm?

Also, 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 [4] 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] M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, editors. *Network Routing*. North-Holland, Amsterdam, 1995.

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

[4] P. Bose, A. Brodnik, S. Carlsson, E. Demaine, R. Fleischer, A. Lopez, P. Morin, and I. Munro. Online routing in convex subdivisions. *International Journal of Computational Geometry*, 12(4):283–295, 2002. See also *Proceedings of the 11th International Symposium on Algorithms and Computation (ISAAC'00)*, Springer LNCS, 2000, pp. 57–79.

[5] P. Bose, J. Gudmundsson, and M. Smid. Constructing plane spanners of bounded degree. In *Proceedings of the European Symposium on Algorithms (ESA'02)*, Springer LNCS, pages 234–246, 2002.

[6] P. Bose and P. Morin. Online routing in triangulations. *SIAM Journal of Computing*, to appear. See also *Proceedings of the Tenth International Symposium on Algorithms and Computation (ISAAC'99)*, Springer LNCS, 1999, pp. 113–122.

[7] 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.

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

[9] D. Dobkin, S. J. Friedman, and K. J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete and Computational Geometry*, 5:399–407, 1990. See also *28th Symp. Found. Comp. Sci.*, 1987, pp. 20–26.

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

[11] 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.

[12] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.