Constructing Competitive Tours From Local Information

Bala Kalyanasundaram[1]

Computer Science Department, University of Pittsburgh, PA. 15260, USA

Kirk R. Pruhs[2]

Computer Science Department, University of Pittsburgh, PA. 15260, USA

Abstract

We consider the problem of a searcher exploring an initially unknown weighted planar graph G. When the searcher visits a vertex v, it learns of each edge incident to v. The searcher's goal is to visit each vertex of G, incurring as little cost as possible. We present a constant competitive algorithm for this problem.

1 Introduction

In this paper we consider the following situation. A salesperson is assigned to visit all the towns in some rural state that he/she knows nothing about. Of course, the salesperson wishes to accomplish this with as little time spent traveling as possible. The salesperson, however, is not given the benefit of having a map. Hence, when the salesperson visits a town, the only information that he/she may be able to glean about other cities is from the road signs on the roads leaving that town. Each road sign gives the name and the distance to the next city down that road. As the salesperson visits towns, new information may reveal shorter routes and may cause the salesperson to modify the order in which he/she plans to visit the remaining unvisited towns.

We call this problem the *online traveling salesperson problem*, online TSP for short, and model it graph theoretically in the following manner. We assume that the roads form an edge-weighted planar connected graph $G = (V, E)$. Then G is learned by a searcher under what we call the *fixed graph scenario*. That is, when the searcher visits a vertex v, it learns of each vertex w adjacent to v in G, as well as the length $|vw|$ of the edge vw. Note that $|vw|$ need not be the Euclidean distance

[1]Supported in part by NSF under grants CCR-9009318 and CCR-9202158

[2]Supported in part by NSF under grant CCR-9209283

between v and w in the planar embedding. We only require that the distances are nonnegative. So, for example, the distances need not satisfy the triangle inequality. In addition, n, the number of vertices of G, is not known in advance to the searcher. As in the standard traveling salesperson problem [19], the salesperson's goal is to visit all of the vertices, traveling only on the edges, with his/her path being as short as possible.

Since the searcher/salesperson lacks complete information, it is generally not possible to construct the optimal tour. Instead, the searcher's goal is to construct a tour that is as close to optimal as possible. We take as our measure of closeness the ratio of the length of the searcher's tour to the length of the optimal tour. This ratio is called the *competitiveness* of the tour. An online algorithm is *α-competitive*, or alternatively, has a *competitive factor* of α, if the supremum, over all possible instances, of this ratio is α. For us a "good" algorithm is one that has a competitive factor that is bounded by a constant. We simply say that such an algorithm is *competitive*.

The main result of this paper is a competitive algorithm, ShortCut, for online TSP in a planar graph. ShortCut is described in section 2. In section 3, we show that the competitive factor for ShortCut is at most 18. We also show how this algorithm can be extended to planar graphs that additionally have vertex weights. The total computation time for ShortCut is asymptotically equivalent to the time required by the standard algorithms for solving the all-pairs shortest path problem on a sparse graph[9].

It is well known that the offline traveling salesperson problem, in which G is known in advance, is NP-hard [19]. At this point we should note that the standard heuristics for approximating offline TSP do not seem to be applicable in the online setting. These heuristics consider the points in some order $v_1, \ldots, v_n$, and at each stage construct a partial tour T_i of the first i points. We define such a heuristic to be *local* if v_i follows v_{i-1} in $T_i, T_{i+1}, \ldots, T_n$. By necessity an algorithm in the online setting must be local. The competitive offline approximation heuristics, eg. nearest insertion, cheapest insertion, the minimum spanning tree algorithm, and Christofides algorithm [19, 24], are not local. The proofs that these heuristics are competitive involve finding a one-to-one correspondence between the edges in the approximate tour and the edges in the minimum spanning tree. Besides being non-local, these heuristics, when applied in the online setting, do not take into account all costs incurred by the salesperson. In contrast, the heuristics that are local, eg.

the nearest neighbor algorithm [19], and the space filling curve algorithm [22], are not competitive. (The competitive factor of the nearest neighbor algorithm on an arbitrary graph is $\Theta(\log n)$. [24]) Since our algorithm is both local and competitive, the proof that the resulting tours are competitive seems to require a more general technique than finding a one-to-one correspondence with edges in the minimum spanning tree.

We should also note that we have essentially determined how much local information is necessary to construct online a competitive tour of a planar graph. If all edges incident to a vertex v do not need to be revealed when v is visited, then no competitive algorithm exists. To see this consider the following situation. The edge weights in G are the Euclidean distances between the points. Initially the searcher only knows of one vertex v_1 far from the starting vertex s. After visiting v_1 the searcher learns of another vertex v_2 near s, and after visiting v_2 learns of another vertex v_3 near v_1, etc. Since the searcher is forced to repeatedly travel back and forth between two points, the resulting competitive factor is $n-1$, where n is the number of vertices in G.

Our original motivation for considering online TSP arose from an online mapping problem, apparently first proposed in the literature by Deng and Papadimitriou [11]. In this problem a robot searcher inhabits a plane littered with opaque polygonal obstacles. The searcher learns about the environment only through visual information. More precisely, the searcher only learns about a part of an obstacle when it comes into the searcher's line of sight. An example of part of such a setting is shown in figure 1. In the version of mapping proposed by Deng and Papadimitriou, the robot's goal is to construct a short path P with the property that every point in the plane, which is not covered by an obstacle, is observable from some point on P.

It has been shown that any online algorithm for this mapping problem must have a competitive factor that is $\omega(1)$, ie. asymptotically larger than any constant[12, 17]. Notice that in this mapping problem the searcher is allowed to map arbitrarily minute details from arbitrarily large distances. At least in some situations, it seems more realistic to assume that one must be close to an object to completely map it. This assumption leads us to define the following problem, which we call the *visual traveling salesperson problem*, or visual TSP for short. In visual TSP the searcher's goal is to visit and traverse the perimeter of each object.

In section 4, we modify our algorithm for online TSP to obtain a competitive algorithm for visual TSP. The main difficultly in developing the algorithm for visual TSP is that the visibility graph of the objects is not necessarily planar. This result shows that the ability of the adversary to map from a distance is the reason that competitive algorithms can not be achieved for the Deng and Papadimitriou's mapping problem.

As far as we know this paper is the first one to examine constructing short tours online under the fixed graph scenario. However, we will now briefly survey some related work to place our results in perspective. Baeza-Yates, Culberson, and Rawlins [3] seem to have initiated the recent line of research into problems involving searching with incomplete information. They studied several problems that deal with finding a short path to an unknown destination in some simple types of metrics.

Papadimitriou and Yannakakis [21] introduced the problem of finding a short path to some vertex under the fixed graph scenario. Related results can be found in [6, 13]. One application cited by Bar-noy and Schieber [6] is the problem of establishing a point to point connection in a communication network with unreliable links. Computing a tour online under the fixed graph scenario has some relation to broadcasting in a network with unknown topology. Note that the combinatorics of finding shortest paths under the fixed graph scenario differs significantly from the combinatorics of finding tours under the fixed graph scenario.

Recently, there has been much interest in problems involving online searching and mapping using visual information [5, 7, 11, 12, 16, 17, 18, 21]. Papadimitriou and Yannakakis [21] introduced and popularized the problem of finding online a short path to a known destination in a scene of polygonal obstacles. Later results on this problem can be found in [5, 7, 17]. Kalyanasundaram and Pruhs [16] and Klein [18] considered the case when the location of the destination is also initially unknown, but is recognizable once it is seen. Deng, Kameda, and Papadimitriou [12] gave a competitive algorithm for mapping the interior, or exterior, of a simple polygon.

Deng and Papadimitriou [11] investigated the problem of visiting all of the edges in an unweighted graph revealed under a variant of the fixed graph scenario.

Several researchers have considered constructing spanning trees and Steiner trees online under what we call the *point by point* scenario. In this scenario the points are revealed one at a time. When the ith point is revealed, all edges to previously revealed points are also revealed, and the online algorithm must extend (without the deletion of any edges) the previous tree to include the new point. Chandra and

Vishwanathan [8], and Imase and Waxman [14] showed that it is possible to maintain a $O(\log n)$ competitive spanning tree under the point by point scenario. Alon and Azar [1] give an $\Omega(\log n/\log\log n)$ lower bound on the competitiveness achievable for constructing a Steiner tree point by point in the plane. Further results can be found in [2, 4, 14]. The main difference between this scenario and the fixed graph scenario is in what information in known to the online algorithm about "explored" vertices. In the point by point scenario the online algorithm may not be aware of all edges incident to revealed points. More precisely, the online algorithm is not aware of those edges whose other endpoint has not been revealed. While in the fixed graph scenario, the online algorithm is aware of every edge incident to a visited vertex. As our earlier example illustrates, the best competitive factor achievable for constructing tours under the point by point scenario is $\Omega(n)$ if one assumes that new points must be added to the end of the previous tour.

We denote an edge between vertices x and y by xy, with $|xy|$ being the length of xy. We think of graphs as being multisets of edges and perform set operations accordingly. If S is a multiset of edges then $|S|$ is the aggregate length of the edges in S. We say a vertex v is a member of a graph S if an edge in S is incident to v. We use OPT for the optimal offline path, and MST for the minimum spanning tree of G.

2 The Algorithm ShortCut

Intuitively, the algorithm ShortCut performs depth first search on different localities in G, with occasional jumps from one locality to another. Before being more specific we need some definitions.

Definition 1 *Throughout the algorithm each vertex will be classified in one of three mutually exclusive ways:*

1. *Visited: A visited vertex is one that has been visited by the searcher.*
2. *Boundary: A boundary vertex is an unvisited vertex adjacent to a visited vertex.*
3. *Unknown: An unknown vertex is one that the searcher has not yet seen.*

Definition 2 *Throughout the algorithm each edge in G will be classified in one of three mutually exclusive ways:*

1. *Explored: An edge is explored if both endpoints are visited.*

2. Boundary: A boundary edge is one for which exactly one endpoint has been visited.

3. Unknown: An unknown edge is one for which neither endpoint has been visited.

As mentioned previously we will need to occasionally shift our search from one portion of G to another via a known path in G. Conceptually, this shift can be viewed as traversing a new edge, which we will call a *jump edge*, that is added to G. Throughout the rest of this paper whenever we refer to a boundary edge, say vw, we will always list the visited vertex first. So, in this case v would be a visited vertex, and w would be a boundary vertex.

Definition 3 *At any particular time, let $d(v, w)$ denote the length of the shortest path known between vertices v and w using only explored and boundary edges.*

The following definition is the crucial one for understanding the algorithm ShortCut. We will define the constant $\delta > 0$ later so as to minimize the competitive factor.

Definition 4 *A boundary edge xy blocks a boundary edge vw if $|xy| < |vw|$ and $d(v, x) + |xy| < (1 + \delta)|vw|$. A boundary edge vw is a shortcut if no other boundary edge blocks vw.*

We are now ready to continue the explanation of ShortCut. The searcher begins as if it were performing a standard depth first search on G. Assume that the searcher is at a vertex v and is considering whether to traverse a boundary edge vw. If vw is a shortcut then vw is traversed at this time. In our later analysis, we will say that vw is a *charged edge*. Otherwise, the traversal of vw is delayed, perhaps indefinitely.

Assume that the searcher just traversed a boundary edge xy, causing y to become visited and xy to become explored. It may then be the case that some other boundary edge vw, whose traversal was delayed at some previous point in time, now becomes a shortcut. In this case a jump edge is added from y to w. Conceptually, the searcher can traverse this jump edge like any other edge. If at some time ShortCut directs the searcher to traverse this jump edge, then the searcher will actually traverse the shortest path that it is aware of from y to w. We will prove later that $d(y, w) \leq (2 + \delta)|vw|$. In this case we will say that vw is a *charged edge*, and it will pay for the cost of moving from y to w.

We now give pseudo-code for ShortCut. For each vertex v, ShortCut maintains a list, $Incident(v)$, of edges incident to v. For each boundary edge e, ShortCut maintains a list, $Block(e)$, of boundary edges that block e.

```
Procedure ShortCut(x, y: Vertices; G:Graph);
Comment: Traveling from x, the searcher visits y for the first time
begin
For each boundary edge vw do
        If visiting y caused Block(vw) to became empty then
            add a jump edge yw at the end of Incident(y) and Incident(w)
EndFor
For each edge yz ∈ Incident(y) do
    If z is a boundary vertex and yz is a jump edge then
        Traverse the shortest known path from y to z
        ShortCut(y, z, G)
    EndIf
    If z is a boundary vertex and yz is a shortcut then
        Traverse the edge yz
        ShortCut(y, z, G)
    EndIf
EndFor
Return to x along the shortest known path
end;
```

3 Algorithm Analysis

In this section we prove that ShortCut constructs a competitive tour, and analyze the total computation time required. It is important to mention the possibility that there is no one-to-one correspondence between the edges used by the algorithm to the edges in minimum spanning trees. Proofs found in [10, 20] also face similar problem. The proof that the tour is competitive (as well as proofs found in [10, 20]) overcome this problem by performing an amortized cost-accounting.

Theorem 1 *The searcher, using ShortCut, visits all the vertices.*

Proof: Assume to reach a contradiction that upon termination the algorithm does not visit all the vertices. At the time of termination let xy be the boundary edge minimizing $|xy|$. Hence, during the visit of some vertex z (which need not be x), xy

will be a shortcut. Therefore, y will be visited either through the jump edge from z or directly from x. $\diamond$

Observation 1 *If at some point in time a boundary edge xy blocks a boundary edge vw, then xy will continue to block vw until either y or w is visited.*

This observation follows since both xy and vw remain boundary edges until either y or w are visited.

Lemma 1 *Assume that after traversing a boundary edge xy another boundary edge vw became a shortcut and a jump edge was added from y to w. Then $d(y,w) \leq (2+\delta)|vw|$.*

Proof: After visiting y no edge blocks vw. By observation 1, before y was visited it must be the case that there was a boundary edge incident on y that blocked vw. Hence by definition 4, $d(y,v) < (1+\delta)|vw|$. Therefore, $d(y,w) < (2+\delta)|vw|$. $\diamond$

Theorem 2 *The algorithm ShortCut is 18-competitive.*

Proof: Let P be the set of *charged edges*. Recall that that an edge xy is said to be charged if it is the shortcut edge that allowed y to be visited for the first time (though xy need not have been traversed on this visit). We know that the total length of the tour is at most $2(2+\delta)|P|$, with the factor of $(2+\delta)$ coming from lemma 1, and the factor of 2 coming from the fact that depth first search traverses each edge once in each direction. Let MST be the minimum spanning tree that minimizes the number of edges in $P - MST$ (this assumption is unnecessary if edge lengths are distinct). Note $|MST| \leq |OPT|$. We show that $|P| \leq (1+\frac{2}{\delta})|MST|$. The theorem then follows by selecting $\delta = 2$.

Consider a fixed planar embedding of $MST \cup P$ (note that each edge in $P \cap MST$ is only included once in this embedding). We call an edge in $P - MST$ a *chord*. Let R be the closed walk obtained by walking around the planar embedding of MST. Note that each edge in MST is included exactly twice in R. We then give each vertex in R a new identifier. It will likely aid the reader's intuition to imagine blowing R up like a balloon so that R becomes the perimeter of a polygon, and the chords are embedded on the outside of R. Let z be any vertex on the exterior face of the embedding. Now imagine making a straight line cut that passes through z from the interior of R to the exterior face. We henceforth consider this planar embedding. Note that R is now an open curve. For example, the graph in the upper of figure 2 shows a planar graph, where MST is shown with solid edges, and edges in $P - MST$ are shown as dashed lines. To its right the closed walk is shown in solid lines. On the bottom is the blown-up version of R and the chords. By cutting at d, we can get the open curve $R = dc'ec''b'fb''abcd'$.

To prove $|P| \leq (1 + \frac{2}{\delta})|MST|$ it is sufficient to prove that the cumulative length of the chords is at most $|R|/\delta$. We say that a chord xy is *inside* a chord vw if in traversing R we encounter these points in the order $vxyw$, $vyxw$, $wyxv$, or $wxyv$. We now recursively consider the chords from inside out, changing R in the process.

Let xy be the chord under consideration. Denote by $R(x, y)$ the portion of R between x and y. We first prove that $|R(x, y)| \geq (1 + \delta)|xy|$. Suppose to reach a contradiction that $|R(x, y)| < (1 + \delta)|xy|$. We say an edge vw is *big* if $vw \in R(x, y)$ and $|xy| \leq |vw|$. We first show that there is at least one big edge. Consider the time that xy was charged, or equivalently, the time that y was first visited. At that time there must be one other boundary edge in $R(x, y)$ since $R(x, y)$ is a path from a visited vertex to a boundary vertex. Let vw be the first such boundary edge encountered when traveling from x to y on $R(x, y)$. Since at that time $d(x, v) + |vw| < (1 + \delta)|xy|$, it must be the case that $|xy| \leq |vw|$, or xy would not have been charged when y was visited.

Now notice that every big edge must be charged. If a big edge vw is not charged, then $vw \in MST$. We can then derive a contradiction since either $MST - \{vw\} + \{xy\}$ has smaller cost than MST, or has fewer chords than MST.

We now reach our contradiction by consider the big edge vw that is charged last. Assume, without loss of generality that v is visited before w, and consider the time that the boundary edge vw was charged. Let ab be the first boundary edge encountered when traversing the path $R(x, y) + \{xy\} - \{vw\}$ from v to w. Note that $d(v, a) + |ab| \leq |R(x, y)| + |xy| - |vw|$. Then using the fact that $|vw| \geq |xy|$, and the assumption that $|R(x, y)| < (1 + \delta)|xy|$, we can conclude that $d(v, a) + |ab| < (1 + \delta)|vw|$. Hence, ab blocks vw and vw, contradicting the fact that this is the time that vw was charged.

We can now assume that $|R(x, y)| \geq (1 + \delta)|xy|$. We let R be the curve formed by replacing $R(x, y)$ by xy, and repeat the argument recursively. Let $T(|R|, k)$ be the supremum, over all open simple curves R, with length $|R|$, and over all ways to add k chords to R such that the resulting graph is planar and each chord satisfies $|R(x, y)| \geq (1 + \delta)|xy|$ when the above argument is applied recursively, of the total length of these k chords. Then $T(|R|, k)$ then satisfies the following recurrence relation:

$$T(|R|, 0) = 0$$

$$T(|R|, k) \leq T(|R| - (1+\delta)|xy| + |xy|) + |xy|$$

The solution to this recurrence relation is $T(|R|, k) \leq |R|/\delta$. $\diamond$

Using a standard trick [19] we now briefly explain how to extend ShortCut to handle graphs with vertex costs in addition to edge costs. That is each vertex x has a cost $w(x)$ and the searcher incurs a cost of $w(x)$ each time it visits x. Furthermore, when the searcher is at a vertex x it additionally knows the cost of each adjacent vertex. We modify G to create a new graph G' by increasing the cost on each edge xy to $|xy| + (w(x) + w(y))/2$. Note that G' can be constructed online, and ShortCut does not require that the graph satisfy the triangle inequality. Based on the observation that in the analysis of ShortCut each edge traversed by ShortCut is charged twice, it can then be seen that ShortCut applied to G' yields a competitive tour.

We finish this section by briefly considering the total computation time required by ShortCut. First note that the number of edges in a planar graph is $O(n)$ [23]. In order to efficiently maintain the *Block* lists, for each boundary edge vw, we maintain a list, $BlockedBy(vw)$, of edges that vw blocks. So, $xy \in Block(vw)$ if and only if $vw \in BlockedBy(xy)$. Furthermore, we maintain cross pointers between the two corresponding entries in these two lists. When a boundary edge xy is first encountered, $Block(xy)$ and $BlockedBy(xy)$ can be initialized using Dijkstra's single-source shortest path algorithm from y in time $O(n \log n)$. Hence at most $O(n^2 \log n)$ time is needed to initialize the *Block* lists. When a jump edge is traversed, the shortest path can be computed in time $O(n \log n)$ using Dijkstra's algorithm. Since there will be at most $O(n)$ jump edges traversed, the total time required for computing all such shortest paths is $O(n^2 \log n)$. Finally, when an edge xy becomes an explored edge, it will no longer block other boundary edges. Then for each edge vw in $BlockedBy(xy)$ we can remove xy from $Block(vw)$ in constant time using the cross pointers. Since any edge is added to and deleted from each *Block* and *BlockedBy* list at most once, the total time for all such modifications to these lists is $O(n^2)$. This gives a total computation time of $O(n^2 \log n)$, which is the same as the time required for the standard all-pairs shortest path algorithms for sparse graphs [9].

4 Visual TSP

We now show how to use the algorithm ShortCut, presented in section 2, to develop a competitive algorithm for visual TSP. Notice that our analysis of ShortCut holds only when the underlying graph is planar. In addition we require that the planar graph used by the searcher must contain a minimum spanning tree as a subgraph. In this section, we show how to construct online such a planar subgraph.

Let V be the set of all the vertices of the obstacles. In the *visibility graph*, VG for short, two vertices $v, w \in V$ are adjacent if they are mutually visible. We consider adjacent corners of an object are mutually visible since we are dealing with polygonal objects. The shortest obstacle avoiding path between two obstacle vertices is the shortest path between these vertices in VG. We use the notation $d(x, y)$ to denote the distance between vertices x and y in VG, and $|xy|$ for the Euclidean distance between x and y.

Definition 5 *For two points $v, w \in V$, the $lune(v, w)$ consists of those points contained strictly inside both the circle with radius $|vw|$ centered at v, and the circle with radius $|vw|$ centered at w. Hence, $x \in lune(v, w)$ is equivalent to $|vx| < |vw|$ and $|wx| < |vw|$.*

Definition 6 *An edge $vw \in VG$ is an edge in the object neighborhood graph, denoted ONG, if v and w are consecutive vertices on the perimeter of some object, or if there is no vertex $x \in lune(v, w)$ that is mutually visible from both v and w. In the first case, vw is called a perimeter edge.*

An object neighborhood graph is a generalization of a relative neighborhood graph [25].

Definition 7 *For a nonperimeter edge vw in ONG, the $Olune(v, w)$ is the intersection of $lune(v, w)$ with the region of the plane mutually visible from v and w.*

Lemma 2 *Let vw be a nonperimeter edge in ONG and assume that ONG is drawn so that vw is horizontal. Then $Olune(v, w)$ consists of $lune(v, w)$ minus possibly two straight line cuts, one above vw and one below.*

Proof: See figure 3 for an example of an $Olune$. First observe that VG is connected. Consider the perimeter edges that intersect $lune(v, w)$ and have both endpoints outside of $lune(v, w)$. Let U be the lowest such line segment above vw, and B be the highest such line segment below vw. If the area between U and B in $lune(v, w)$ is not completely obstacle free then there is some vertex x mutually visible from v and w. To see this, sweep one horizontal line segment from vw upwards, and sweep one horizontal line segment from vw downwards. The first vertex encountered will be visible from both v and w. $\diamond$

Lemma 3 *When the searcher is at a vertex v it can determine all of the edges from ONG incident to v from visual information.*

Proof : To determine whether an edge $(v, w) \in VG$ is in ONG, the searcher need only verify that for every other edge of the form vx, with $x \neq w$, that x is not in $Olune(v, w)$. $\diamond$

Lemma 4 *ONG is a planar graph that contains the minimum spanning tree of VG as a subgraph.*

Proof: We first show that ONG contains the minimum spanning tree of VG as a subgraph. Let S and $V - S$ be a partition of V. Let $v \in S$ and $w \in V - S$ be the pair of points in the different partitions that minimize $d(v, w)$. Then vw must be in ONG. Assume not, to reach a contradiction. Then vw is not a perimeter edge and there is a point $x \in lune(v, w)$ that is mutually visible to v and w. If $x \in S$ this contradicts the choice of v, and if $x \in V - S$ this contradicts the choice of w. Therefore ONG contains the minimum spanning tree as a subgraph [9].

We now show that ONG is a planar graph. If two edges vw and xy is in ONG and cross at a point c then neither edge can be a perimeter edge. Without loss of generality assume, $|cv| \leq \min\{|cw|, |cx|, |cy|\}$. Hence, $v \in lune(x, y)$. For the edge xy to be in ONG, v must be outside $Olune(x, y)$. Therefore, either v is not visible from x or from y. Suppose v is not visible from x due to an object O. Then, either there is a corner of O that is mutually visible to both x and y (so xy is not an edge in ONG) or the object O cuts the line segment vw (so vw is not an edge in VG and hence not in ONG). Therefore, ONG is planar. $\diamond$

If the searcher wishes to visit each obstacle vertex it now need only apply ShortCut to ONG. In visual TSP, however, we are also asked to traverse each perimeter edge. This can be accomplished by modifying ShortCut so that when the searcher visits an object for the first time it circumnavigates the perimeter of that object.

This circumnavigation does not change the status of any of the edges or vertices, and ShortCut then continues as if it had not performed this circumnavigation.

Theorem 1 *There is a 19-competitive algorithm for visual TSP.*

Proof: The proof is the same as theorem 2, except that the cost of the circumnavigations adds one to the competitive factor. $\diamond$

The total time required to compute ONG online from VG is $O(n^2)$. Thus the total time required for the algorithm for visual TSP is $O(n^2 \log n)$. We should note that for planar graphs, such as ONG, for which the edge lengths are the actual Euclidean distances, it is possible to lower the competitive factor slightly by changing the constants in the definition of blocking.

5 Conclusion

This paper is the first to investigate constructing tours online under the fixed graph scenario. The outstanding open question seems to be whether there is a competitive algorithm for online TSP on a general weighted graph under the fixed graph scenario. Note that the competitive factor of the greedy algorithm Nearest Neighbor is $O(\log n)$ for arbitrary weighted graphs [24].

Acknowledgments: We would like to thank Sundar Vishwanathan, Gautam Das, Giri Narasimhan, and Harry Plantinga for helpful discussions. Finally, we would like to thank an anonymous referee for an observation that allowed us to reduce the competitive factor from 24 to 18.

References

[1] N. Alon, and Y. Azar, On-line Steiner trees in the Euclidean plane, in: *Proceedings of the 8th ACM Symposium on Computational Geometry* (1992).

[2] Y. Azar, Lower bounds for insertion methods for TSP, Manuscript.

[3] R. Baeza-Yates, J. Culberson, and G. Rawlins, Searching with uncertainty, to appear in: *Information and Computation.*

[4] V. Bafna, B. Kalyanasundaram, and K. Pruhs, Not all insertion methods yield constant approximate tours in the plane, Technical Report, Computer Science Department, University of Pittsburgh, 1992.

[5] E. Bar-Eli, P. Berman, A. Fiat, and P. Yan, On-line navigation in a room, in: *Proceedings of the Second Annual SIAM/ACM Conference on Discrete Algorithms* (1991) 237–249.

[6] A. Bar-Noy, and B. Schieber, The Canadian travelers problem, in: *Proceedings of the Second Annual ACM/SIAM Symposium on Discrete Algorithms* (1991) 261–270.

[7] A. Blum, P. Raghavan, and B. Schieber, Navigating in unfamiliar geometric terrain, in: *Proceedings of the Twenty Third Annual ACM Symposium of Theory of Computing* (1991) 494–504.

[8] B. Chandra, and S. Vishwanathan, Constructing reliable communication networks of small weight online, Manuscript.

[9] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms* (McGraw-Hill, New York, 1990).

[10] G. Das and D. Joseph, Which triangulations approximate the complete graph, in: *Proceedings of International Symposium on Optimal Algorithms* (1989) 168–192.

[11] X. Deng, and C. Papadimitriou, Exploring an unknown graph, in: *Proceedings of the Thirty First Annual Symposium on Foundations of Computer Science* (1990) 355–361.

[12] X. Deng, Kameda, and C. Papadimitriou, How to learn an unknown environment, in: *Proceedings of the Thirty Second Annual Symposium on Foundations of Computer Science* 1991, 298–303.

[13] A. Fiat, D. Foster, H. Karloff, Y. Rabani, Y. Ravid, S. Vishwanathan Competitive algorithms for layered graph traversal, in: *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science* (1991), 288–297.

[14] M. Imase, and B. Waxman, Dynamic Steiner tree problem, *SIAM Journal of Discrete Mathematics* **4** (1991) 369–384.

[15] B. Kalyanasundaram, and K. Pruhs, Constructing competitive tours from local information, Technical Report, Computer Science Department, University of Pittsburgh, 1992.

[16] B. Kalyanasundaram, and K. Pruhs, A competitive analysis of nearest neighbor algorithms for searching unknown scenes, in: *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science* (1992) 147–157.

[17] H. Karloff, Y. Rabani, and Y. Ravid, Lower bounds for randomized k-server and motion planning algorithms, in: *Proceedings of the Twenty Third Annual ACM Symposium of Theory of Computing* (1991) 278–288.

[18] R. Klein, Walking an unknown street with bounded detour, in: *Proceedings of the Thirty Second Annual Symposium on Foundations of Computer Science* (1991) 304–313.

[19] E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Schmoys, *The Traveling Salesman Problem* (Wiley, New York, 1985).

[20] C. Levcopoulos, and A. Lingas, There are planar graphs almost as good as the complete graphs and as short as minimum spanning trees, in: *Proceedings of International Symposium on Optimal Algorithms* (1989) 9–13.

[21] C. Papadimitriou, and M. Yannakakis, Shortest paths without a map, in: *Proceedings of the Sixteenth Annual Internation Colloquium on Automata, Languages, and Programming* (1989) 610–620.

[22] L. Platzman, and J. Bartholdi, Spacefilling curves and the planar traveling salesman problem", *Journal of the ACM* **36** (1989) 719–737.

[23] F. Preparata, and M. Shamos, *Computational Geometry: An Introduction* (Springer-Verlag, New York, 1985).

[24] D. Rosenkrantz, R. Stearns, and P. Lewis, An analysis of several heuristics for the traveling salesman problem, *SIAM Journal of Computing* **6** (1977) 563–581.

[25] G. Toussaint, The relative neighborhood graph of a finite planar set, *Pattern Recognition* **12** (1980) 261–268.