Robot Map Verification of a Graph World

Xiaotie Deng

Department of Computer Science

City University of Hong Kong

Kowloon, Hong Kong SAR, China

deng@cs.cityu.edu.hk

Evangelos Milios, Andranik Mirzaian

Department of Computer Science, York University

Toronto, Ontario, Canada, M3J 1P3

{eem,andy}@cs.yorku.ca.

Abstract: In the map verification problem, a robot is given a (possibly incorrect) map M of the world G with its position and orientation indicated on the map. The task is to find out whether this map, for the given robot position and its orientation in the map, is correct for the world G. We consider the world model of a graph $G = (V_G, E_G)$ in which, for each vertex, edges incident to the vertex are ordered cyclically around that vertex. (This also holds for the map $M = (V_M, E_M)$.) The robot can traverse edges and enumerate edges incident on the current vertex, but it cannot distinguish vertices (and edges) from each other. To solve the verification problem, the robot uses a portable edge marker, that it can put down at an edge of the graph world G and pick up later as needed. The robot can recognize the edge marker when it encounters it in the world G. By reducing the verification problem

to an exploration problem, verification can be completed in $O(|V_G| \times |E_G|)$ edge traversals (the *mechanical cost*) with the help of a single vertex marker which can be dropped and picked up at vertices of the graph world [DJMW1, DSMW2]. In this paper, we show a strategy that verifies a map in $O(|V_M|)$ edge traversals only, using a single edge marker, when M is a *plane* embedded graph, even though G may not be planar (e.g., G may contain overpasses, tunnels, etc.).

KEW WORDS: robot exploration and map verification, on-line algorithm, topological graph, face tracing.

1 Introduction

There are several different facets for robot navigation which are important in real world environments. One crucial issue is how to deal with cumulative errors, which may cause the robot to lose track of its position in the real world. Several different approaches are suggested to relate the robot position with the external features of the environment using a map [BCR, BRS, GMR, KB, LL, PY]. This leads to the task of map construction (mapping), i.e., learning the cognitive map from observations, as summarized by Kuipers and Levitt [KL]. Much research has been done on the robot mapping problem for different external environments [AH, DJMW1, DM, DP, DKP, Kw, PP, RS].

Naturally, one major class of maps used in robot navigation is geometric. However, it remains an important and difficult problem how to utilize the map and match it with the enormous amount of observed geometric information for robot decision makings. Alternatively, qualitative maps, such as topological graphs, are proposed to model robot environments which usually require much less information in comparison with geometric models [DJMW1, DM, DP, KB, LL, RS]. This approach often focuses on a small set of characteristic locations in the environment and the routes between them to reduce information necessary for robot navigation. Consequently, it simplifies the task of robot decision making. Kuipers and Levitt have proposed a spatial hierarchical representation of environments consisting of four levels: *sensorimotor* level (a robot uses sensors to detect local features of the environment); *procedural* level where the robot applies its knowledge of the world to find its place in the world and to follow specified routes; *topological* level which describes places and their connecting paths, usually with the graph model; and *metric* level which includes necessary geometric information related to the topological representation [KL, also see KB, LL]. These approaches, neither purely metric nor purely qualitative, leave certain features of the environment out but keep necessary information helpful for robot motion planning.

The robot's perception of the world can also be different as a result of the different sensors it carries. In many situations, it is assumed that nodes or edges traversed previously can all be distinguished. In contrast, it is assumed in [RS] that nodes are divided into a small number of classes, for example, white and black colors, and can only be recognized as such. Dudek et al. [DJMW1] apply the world model introduced by Kuipers and Levitt to a specific situation in which no global orientation information is possible. They divide the world into places represented by nodes in a topological (i.e., embedded) graph and use an edge between two nodes to represent a connecting path between the corresponding two places. The robot is assumed not be able to distinguish nodes from each other but can recognize a special local geometric feature: *a cyclic order of incident edges at each node*. This emulates the fact that, at the crossroads, paths form a cyclic order because of the local planar geometric nature of the surface. Dudek et al. [DJMW1] show that it is impossible to learn the graph in general if the robot uses only information collected under the above restriction. For instance, this happens when every node in the graph, representing the world, has the same number of incident edges. On the other hand, they show that in a total of $O(|V| \times |E|)$ traversals of edges, the map can be constructed with the help of a single marker which can be put down on nodes and picked up by the robot. Deng and Mirzaian [DM] showed that using $|V|$ identical node-markers the robot can construct a map of a plane embedded graph G by traversing each edge at most a constant number of times (i.e., a total of $O(|V|)$ edge traversals).

The world model introduced by Dudek, et al., will be adopted for our discussion [DJMW1]. We consider the map verification problem: The robot is given a single directed edge-marker and a plane embedded n-vertex map M and the initial position and orientation of the robot in the map. The task is to verify the correctness of the map in this setting. The environment graph G may or may not be planar (e.g., it may contain tunnels, overpasses, etc.). We demonstrate that the verification of plane embedded graph maps can be done very efficiently with a single edge marker by introducing a map verification strategy which traverses each edge at most 4

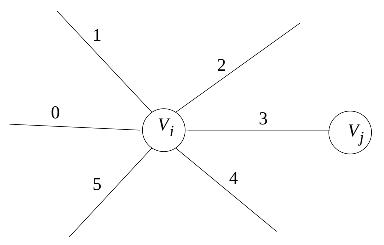

Figure 1: A cyclic order of edges incident to a node.

times.

2 The World Model

As discussed above, the world is an undirected connected graph $G = (V, E)$ embedded with no edge crossings on a (not necessarily planar) surface [DJMW1]. At each node, the incident edges are embedded locally in the plane (Figure **??**). The local embedding forms a natural cyclic order for the edges incident to the node. When we have specified an edge incident to the node as the reference edge for the node, we can name other edges incident to this node with respect to the reference edge according to this cyclic order (e.g., in clockwise order).

The Robot's Map of the Graph World. In the graph world model, nodes usually correspond to intersection of passage ways in the real world. To deal with general situations where landmarks may look similar by the robot's sensors, Dudek et al. assume the worst possible situation: nodes in the graph are indistinguishable to the robot. Therefore, the complete map of the graph is a triple $(V, E, \mathcal{S})$, where V and E are the node set and the edge set of the graph, and $\mathcal{S}$ is the collection of the local planar embeddings of edges incident to each node.

Note that in Figure **??**, we intentionally give an example, where, for several nodes, the cyclic orderings of incident edges are different from those in the topological graph to emphasize the general situation where local orientations of edges at each node can be arbitrary. In fact, given any graph $G = (V, E)$ and given any set $\mathcal{S}$ of

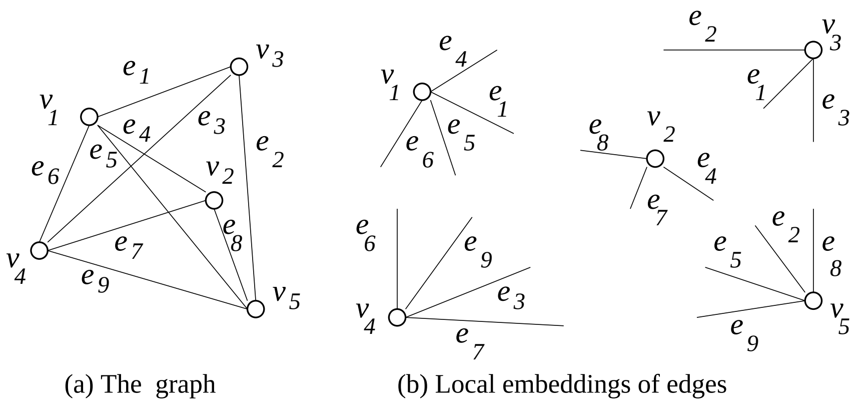

Figure 2: A map.

cyclic orders of edges incident to a node, there is a surface on which the graph can be embedded such that the local planar embedding of edges incident to each node follows the cyclic order of $\mathcal{S}$ [HR]. Even in the real world, these may happen because tunnels and overpasses can create generally embedded surfaces.

An alternative representation of the map is its *cyclically ordered Adjacency List Structure*. That is, for each node v in the map we associate a cyclically ordered list of the nodes adjacent to v. The cyclical ordering is the clockwise ordering of the corresponding incident edges around node v.

In Figure **??**, we give several maps of K_4 to illustrate different graph world maps that can be generated from K_4. Notice that (a) is embedded in a planar surface and the others must be embedded in surfaces of higher genus. with careful examination of the maps, one can see that (b) and (d) correspond to the same combinatorially embedded map.

Robot Navigation with a Correct Map. Once the correctness of the map is guaranteed, the current location of the robot is matched to a node u_0 on the map, and a path from the robot location is matched to an edge e_0 incident to u_0 on the map, the robot can match all the paths at its current location to edges on the map incident to u_0. If the robot moves to another location through one of the paths, it knows which map node matches the next location it reaches. The edge that leads

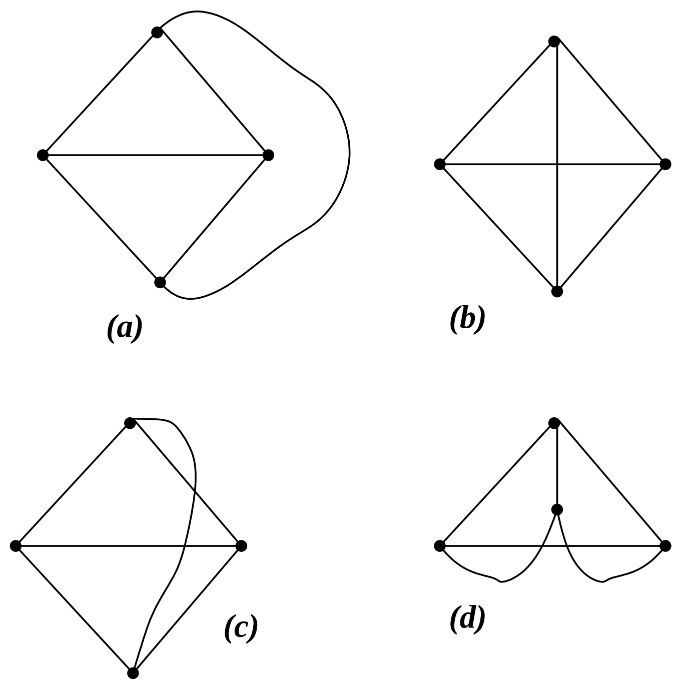

Figure 3: Some combinatorial embeddings of K_4.

the robot to the next location becomes its reference edge at the new location. From the local embedding of edges incident to the new node, with the help of the reference edge, it can again match edges incident to this new node with paths from its new location. This allows the robot to use the map in navigating from one location to another until it reaches its destination.

The Map Verification Problem. The robot is initially positioned at a certain node of an unknown embedded graph environment G and oriented along one of its incident edges. It is also given an embedded map M (say, in the form of the cyclically ordered adjacency list structure). We let the *augmented map* M to mean the map M plus the initial position and orientation of the robot on the map. Let the *augmented* G be defined similarly. The map verification problem is: for an augmented pair (M, G), verify whether M is a correct augmented map of the augmented environment G by navigating in the environment and comparing the local observations with the given map.

3 The Plane Map Verification Algorithm

A notion of *face tracing* from topological graph theory [GT] is crucial in our verification algorithm. The general idea in our algorithm is to trace the faces of the augmented map M one by one, mimic the same actions on the environment graph, and compare the local observations with the map. For the world model as discussed above, the local observations are the degree of the node visited, and the presence or absence of a marker at the current node or edge. The intricacy of this approach is reflected by the fact that the sequence of these local observations for any proper subset of faces is not at all enough even for a partial map but the complete sequence uniquely verifies the map.

In Algorithm 1, a single edge-marker is used. When we refer to the map M, we will use phrases such as to put down or to pick up the edge-marker or to move along an edge. The actual action is carried out on the graph environment G, but a corresponding symbolic operation is performed on M.

Remark 1: Instead of considering an embedding on the plane, it is preferable to think of the embedding on the sphere. In this way, there is no distinguished "outer" face. $\square$

ALGORITHM 1: We have an augmented pair (M, G) and a directed edge-marker. M is a given planar embedded map represented, say, by its cyclically ordered adjacency list structure, and G is the yet unknown augmented environment graph.

The cases when M has a single node or edge can be handled and proved trivially. Now consider the general case.

Assume that the robot is initially positioned at node n_o of M and oriented along edge $e_o = (n_o, n_1)$. Place an edge-marker on edge e_o in the direction of robot's orientation. In general, suppose the robot is currently at node n_i and is oriented along edge $e_i = (n_i, n_{i+1})$. Record the degree of node n_i and let us denote that by d_i. The robot moves to node n_{i+1}. Now let $e_{i+1} = (n_{i+1}, n_{i+2})$ be the edge clockwise next to edge e_i at node n_{i+1}. Increment i by one and repeat the same process. The robot stops before iteration, say, l, when the directed edge $e_l = (n_l, n_{l+1})$ it is about to traverse contains the edge-marker in the direction of the traversal.

At that point, the robot picks up the marker. That is, the marker on the edge $e = (u, v)$ is directed from u to v and it is picked up when we are about to move from u to v again and realize this by noticing that the marker is along this direction. The robot has completed tracing a face, say, f of M. Let $\mathcal{D}(f) = (d_o, d_1, \cdots, d_l)$ denote *the node degree sequence* the robot observed during the tracing of face f. In our map verification algorithm we will use $\mathcal{D}(f)$ as the signature for face f. Note that the signature of a single face by itself does not uniquely identify the topological structure of the face. We will show, however, that the collection of all signatures together will.

The robot then *backtracks* along the most recent edges traversed (during forward tracing of faces) until it reaches the first edge, say, e of M that it has traversed only once. (Note that during the backtracking, we consult the map M, not G, to figure out the edge e.) The robot then uses this edge e as the starting edge of the next face to trace. It places the edge-marker on that edge in the direction not yet traversed (according to the map M) and follows the same face tracing procedure described above.

To help the backtracking process, the robot can stack up all its forward movements by pushing an appropriate id of the edge of M just traversed onto a stack of edges traversed only once. Then, during the backtracking, it pops the top id from the stack and performs the reverse move. During the backtracking no id's are pushed onto the stack and we do not need to check node degrees.

If during the backtracking the stack becomes empty, the robot terminates its traversal: all faces of M have been traced, and we have the signatures of each face in the order they have been traced.

To complete the description of our map verification algorithm we should add that the robot mimics the same edge tracing actions on the embedded environment graph G as it performed on the map M (eg, taking the next edge clockwise). If ever during the process it notices a mismatch of local observations between M and G, it immediately halts and declares the augmented map is incorrect. The local observations that the robot matches are during the forward movements (tracing faces) and that is observing and matching the degree of the current node and the presence or absence of the edge-marker on the current edge and its direction on that edge.

If a mismatch occurs, it is obvious that the augmented map is incorrect. The crux of the matter is to prove that if no mismatch occurs, then indeed the augmented map is correct, and hence the algorithm always gives the correct answer. □

The verification algorithm We now summarize the algorithm described above with the following notes.

- M is the map and G is the graph-like world.
- All robot actions take place "mentally" on M and physically on G.
- Node degrees are measured in G, and counted on M at the same time. A clockwise order is assumed available at each node of G and M.
- As the robot moves, it checks whether its perception in G agrees with its expectation based on M. If a discrepancy occurs, then verification fails.
- Discrepancy of type 1: node degree observed on G not the same as expected based on M.
- Discrepancy of type 2: edge-marker not found where expected (or found where not expected) based on M.
- For each edge e of M, $trace(e)$ indicates how many times has edge e been traversed in either direction during the forward face tracing traversals.

Main algorithm:

```
Set ES (edge stack) to empty.
For each (undirected) edge $e$ in $M$, set $trace(e) \leftarrow 0$.
Select edge $e_0 = (n_0, n_1)$ out of initial node $n_0$.
Place edge-marker on $e_0$ in the direction $n_0 \rightarrow n_1$.
**ForwardTraverseFace** F of M defined by the directed edge-marker.
**Loop** until ES is empty.
    Pop ES to obtain edge $e = (n_i, n_{i+1})$.
    **if** $trace(e) = 2$ **then** Backtrack along edge $e$.
    **else** /* $trace(e) = 1$ */
        Place the edge marker on $e$ in the direction not yet traversed $n_{i+1} \rightarrow n_i$.
        **ForwardTraverseFace** defined by the directed edge-marker.
**end Loop**


Procedure:
**ForwardTraverseFace** defined by edge $e = (n_i, n_{i+1})$ and direction $dir = (n_i \rightarrow n_{i+1})$
    **Repeat**
            Verify degree of current node of M against G (type 1).
            Traverse edge $e$ in the direction $dir$.
            Push $e$, in the direction of traversal, on ES.
            set $trace(e) \leftarrow trace(e) + 1$.
            Select the next edge $e_1$ out of the current node.
            (clockwise wrt to the previous edge traversed).
            set $e \leftarrow e_1$ and update $dir$ accordingly.
            Verify whether presence/absence of edge-marker on $e$ agrees on
            M and G (type 2).
    **until** edge-marker is detected on $e$.
    pick up the edge-marker.
**end ForwardTraverseFace**
```

Examples To appreciate some of the subtleties of this seemingly simple algorithm, we present several illustrative examples.

Example 1: In this example we consider only the actions taken by Algorithm 1 regarding the map. Figure **??** shows a map M with 3 faces, indicated by f_1 through f_3 in the order they are traced by the algorithm. The edge indicated by a directed arrow and labeled m_i is the position and orientation of the marker at the starting edge of face f_i. The remaining forward moves are shown by dotted arrows. However, the backtracking traversals are not shown on the figure. The robot tracing face f_1 visits edges $(ab, bc, cd, de, ec, cb, ba, ag, gh, hi, ig, ga)$, with the signature $\mathcal{D}(f_1) = (2, 2, 3, 2, 2, 3, 2, 2, 3, 2, 2, 3, 2)$. Then it picks up the marker and backtracks along edge ag. Positions the marker along edge gi and starts tracing the second face $f_2 = (gi, ih, hg)$ with signature $\mathcal{D}(f_2) = (3, 2, 2, 3)$. Picks up the marker and backtracks along edges $(gh, hi, ig, gi, ih, hg, ga, ab, bc)$ and places the marker along edge ce. It then starts tracing the third face $f_3 = (ce, ed, dc)$ with signature $(3, 2, 2, 3)$. It then backtracks along edges $(cd, de, ec, ce, ed, dc, cb, ba)$. At this point the stack is empty. The process is complete. □

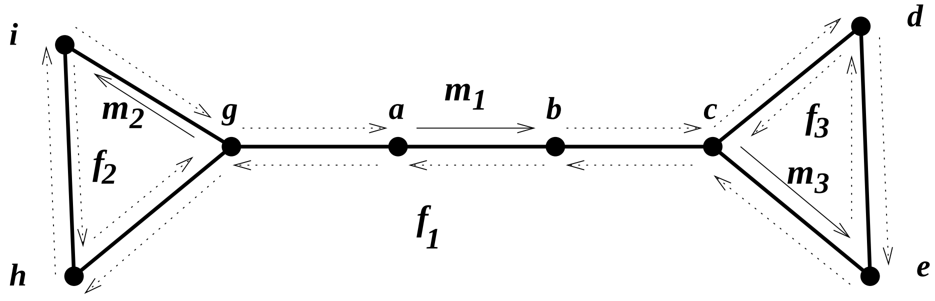

Figure 4: Tracing faces of M by the algorithm.

Example 2: In this example M is an *incorrect* augmented map of G as shown in Figure **??**. The arrows indicate the starting edge of each face. Algorithm 1 does not notice the difference between M and G until the fourth face of the map is traversed. □

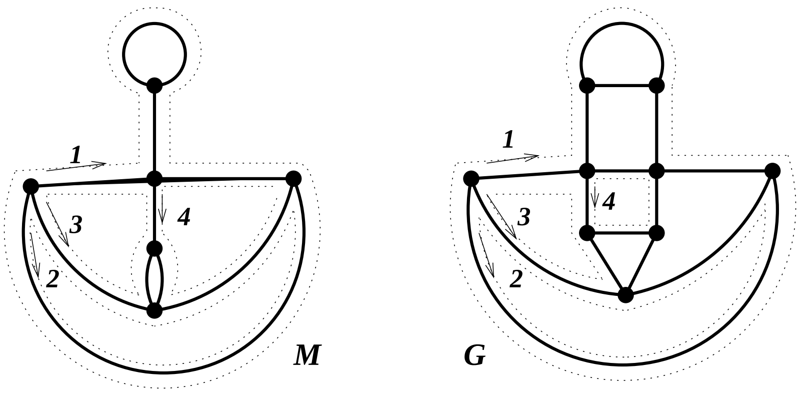

Figure 5: An incorrect map.

Example 3: We can simulate the effect of an edge-marker by two distinct node-markers, placing them at the two ends of the edge, or three homogeneous node-markers by placing one at one end and two at the other end. However, in this example we show that using a single node-marker which is placed at the starting node of each face does not work correctly. In Figure **??** we see a situation where we have an incorrect map, but the robot will not detect the error. The arrows indicate starting edges of each face. The node-marker is placed at the start of the corresponding starting edge. While tracing face f_1 of M, the robot will double-trace the "outer" face of G. Also, while tracing faces f_2 and f_3 of M, the robot will trace the same portion of the "inner" face of G. It will never visit the two leaf nodes in G and their incident edges. The robot will observe the same signatures (and presence or absence of the node-marker) as it traces the faces of M and copies the same actions on G. □

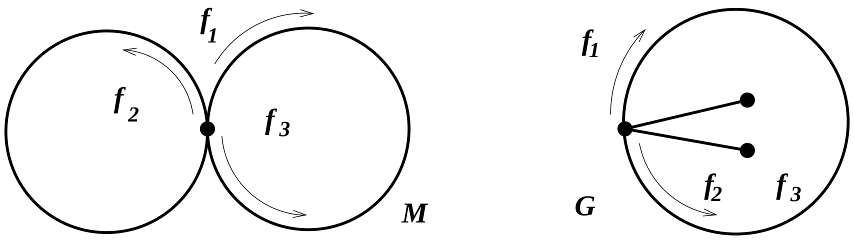

Figure 6: A single node-marker is not enough.

Example 4: Algorithm 1 does not work when the map is not embedded in the plane. For instance, consider the example in Figure **??**. The augmented Map M has the single face with signature $2, 2, 3, 3, 3, 2, 2, 3, 3, 3, 2$. The corresponding face of G has the same signature. Map M in this example can be embedded on a torus, and has combinatorially embedded genus 1. Algorithm 1 will fail to observe any mismatch.

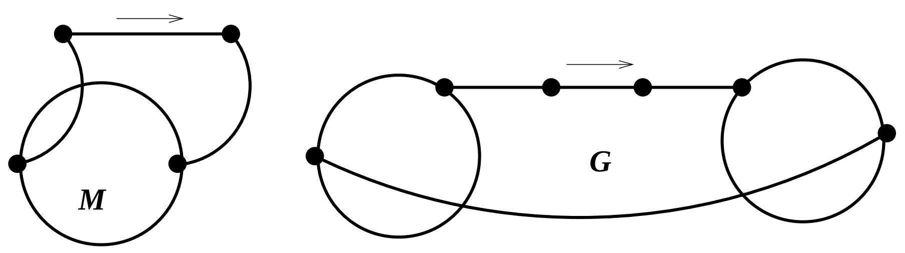

Figure 7: A non-plane map.

4 Proof of Correctness of Algorithm 1 for Plane Maps

We will show Algorithm 1 of the previous section works correctly when the given map M is embedded in the plane (even though the environment graph G may not be). We will also show that the total number of edges traversed by the robot during the algorithm (forward and backtracking included) is at most 4 times the total number of edges of the map.

Theorem 1 *Algorithm 1 of the previous section correctly verifies any augmented plane embedded map using a single directed edge-marker. Furthermore, the total number of edges traversed by the robot is at most 4 times the number of edges in the map.*

Proof: Suppose M is an $n-$node augmented map and G is the augmented environment graph. It is not hard to see that each edge of the map will be traversed at most 4 times: Each edge is forwardly traversed twice, and with each such move the robot pushes an id of the edge onto the stack. Each edge backtracked corresponds to popping a previously pushed id from the stack.

Now let us consider the correctness of the algorithm. If the algorithm finds a mismatch, then clearly the returned answer that M is not a valid augmented map is correct. Now suppose the algorithm returns the answer that M is a correct augmented map. We will prove the correctness of the answer by induction on the number of faces of M.

Basis: In this case M has only one face and since it is embedded in the plane, it must be a tree. However, G may be any embedded graph. Suppose to the contrary that M is not a correct augmented map of G, but the algorithm makes the incorrect conclusion. Consider the smallest counter-example, that is, one in which M has fewest possible number of nodes. We will reach a contradiction by showing that there is even a smaller counter-example. From the description of the algorithm, M is not a single node or edge. Since each edge of M is a bridge (i.e., an edge whose removal will disconnect the graph), each edge of M will be traversed twice (in opposite directions) during the tracing of the single face of M. In other words, the length of the face of M, that is the number of edges incident to the face, is $2n - 2$. While tracing the face of M, suppose the robot is tracing a face f of G. Since it will check the presence or absence of the edge-marker along edges, it will make exactly one complete round around face f of G. Hence, length of f is also $2n - 2$.

Since M is a tree, it must contain a leaf node other than the two ends of the starting edge. Let x be the first such leaf of M that the robot reaches during the tracing of the face of M. When it reaches node x of M, suppose the robot is at node y of G. Since the degrees match, y must be a leaf of G, and the unique nodes adjacent to x in M and to y in G must also have equal degrees. As the robot continues the tracing of the face of M, it will encounter the leaf node x only once. Similarly, since it traces face f of G exactly one round, it will encounter leaf node

y of G only once. Now consider the smaller counter-example pair (M', G') in which G' is the same as G with leaf node y and its incident edge removed, and M' is M with leaf node x and its incident edge removed. Furthermore, consider the same initial position-orientation of the robot. When the algorithm is applied to (M', G') it will observe the same signature and hence return the incorrect answer that M' is a correct augmented map of G'. This contradicts the minimality of M.

Induction: In this case M has at least two faces. Suppose f_1 and f_2 are respectively the first and second faces of M traced by the algorithm. Suppose $e_i = (v_i, u_i)$, $i = 1, 2$, is the starting directed edge of face f_i. Let $\hat{f}_i$, $\hat{e}_i = (\hat{v}_i, \hat{u}_i)$, $i = 1, 2$, be the corresponding objects of G. Because of the use of the edge-marker, length of f_i is the same as length of $\hat{f}_i$, $i = 1, 2$, and they have a matching signature. Because of the backtracking process, edge e_2 is the first edge, in opposite direction, incident to face f_1 that was traversed only once during the tracing of f_1. Hence, faces f_1 and f_2 are incident to edge e_2. Thus, e_2 is not a bridge of M. Therefore, faces f_1 and f_2 of M are distinct and share edge e_2.

As it makes the backtracking in M, the robot makes the same number of backtracking moves to trace back from edge $\hat{e}_1$ to edge $\hat{e}_2$. Now we show that $\hat{e}_2$ cannot be a bridge of G. At the start of tracing of the second face, the edge-marker is placed at the starting edges e_2 and $\hat{e}_2$ of faces f_2 and $\hat{f}_2$ of M and G, respectively. Since e_2 is not a bridge of M, the edge-marker on e_2 will be seen only once at the start and once at the end of the tracing of f_2. Therefore, the same must hold in G, that is, the edge-marker on $\hat{e}_2$ will be seen only once at the start and once at the end of the tracing of face $\hat{f}_2$. We conclude that $\hat{e}_2$ cannot be a bridge of G either. If it were, then the edge-marker would have been seen once more, in the reverse direction, in the middle of the tracing of $\hat{f}_2$. Therefore, faces $\hat{f}_1$ and $\hat{f}_2$ are distinct and share the edge $\hat{e}_2$.

Figure **??**(a) shows the situation in M. Now we will perform a local surgery on M to construct a map M' with one fewer face than M. Simply remove edge e_2 from M. This will reduce the degrees of nodes v_2 and u_2 by 1. The surgery is

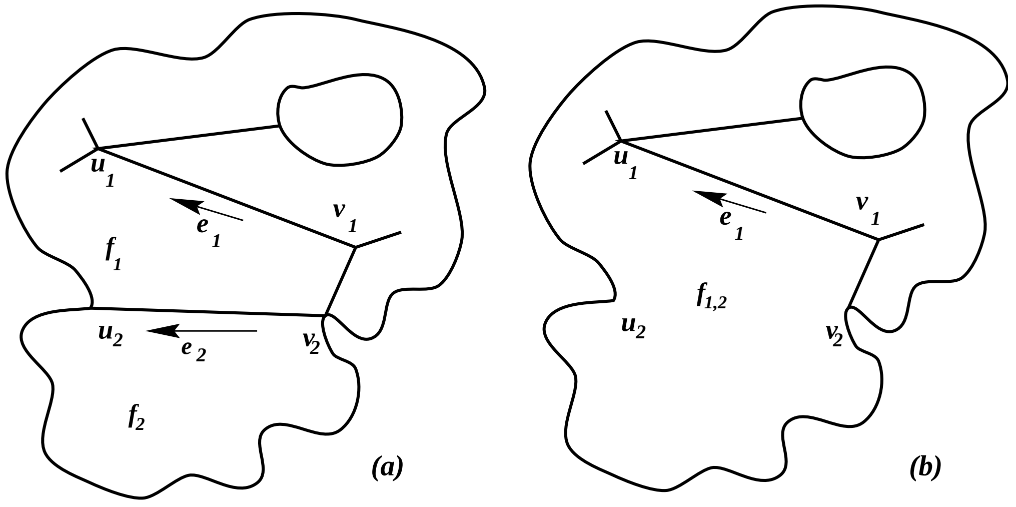

Figure 8: (a) Before the surgery, (b) after the surgery.

depicted in Figure **??**(b). This surgery effectively merges the two faces f_1 and f_2 into a new face denoted by $f_{1,2}$. Similarly, let G' be obtained from G be removing edge $\hat{e}_2$. This merges the two faces $\hat{f}_1$ and $\hat{f}_2$ into a new face $\hat{f}_{1,2}$. Now consider the instance (M', G') with the same initial position-orientation of the robot as in the instance (M, G). (In case e_2 is the same as e_1, in reverse, in M, take the initial edge in the modified map to be the one next to e_1 on f_1. Apply the same idea on G.) Since f_i and $\hat{f}_i$ had the same length and the same signature, for $i = 1, 2$, the new faces $f_{1,2}$ and $\hat{f}_{1,2}$ also have the same length and signature. Furthermore, the remaining faces, if any, traced by the robot will be exactly as before the surgery (except that end nodes of e_2 and $\hat{e}_2$ have reduced degrees). Thus, if the instance (M', G') was the input, the robot would still answer that M' is a correct augmented map of G'. Since M' has one fewer face than M, by the induction hypothesis, M' is indeed a correct map of G'. Furthermore, since before the surgery we had matching signatures and matching backtracking length, in (M', G') the pair of node v_2 and u_2 will respectively match the pair of nodes $\hat{v}_2$ and $\hat{u}_2$. Therefore, M is a correct map of G. This completes the proof. $\square$.

5 Implementation

We have an animated simulation program for our verification algorithm for the case of plane embedded maps. The program is written in C embedded in an OpenGL/GLUT program to produce the graphics. The program runs on a Sun Ultra 2 running Solaris version 2.5.1.

The input to the program is a list of the vertices in the map and graph and for each vertex, a list of the edges eminating from that vertex specified in clockwise order. The data is held in an augmented adjacency list structure. The program was tested successfully on a variety of simple map/graph pairs. The animation shows the progress of the (simulated) robot on the graph and its corresponding state and position on the map. A snapshot of the animation is shown in Figure **??**.

6 Remarks and Discussion

In the paper we presented an efficient algorithm for planar map verification by a robot in a graph world, with the aid of one edge marker. We assumed that the robot is told its initial position on the map. If the robot is not told this information, then the problem is one of self-location [DJMW2]: the robot is required to locate itself on the map, while verifying the map at the same time. It is an open question whether this self-location problem in a planar graph-like world can be solved in $O(|E|)$ edge traversals by a robot with a single edge marker. Another open question is whether a map with combinatorial-genus g can be verified in $O(|E|)$ edge traversals using only $O(g)$ markers.

Acknowledgement Authors' research was partly supported by NSERC grants. Xiaotie Deng would also like to acknowledge the financial support by a CERG grant from the Research Grants Council of Hong Kong Special Administration Region, China (Project No. CityU 1074/00E). The authors would like to thank Arlene Ripsman for producing the animated simulation program.

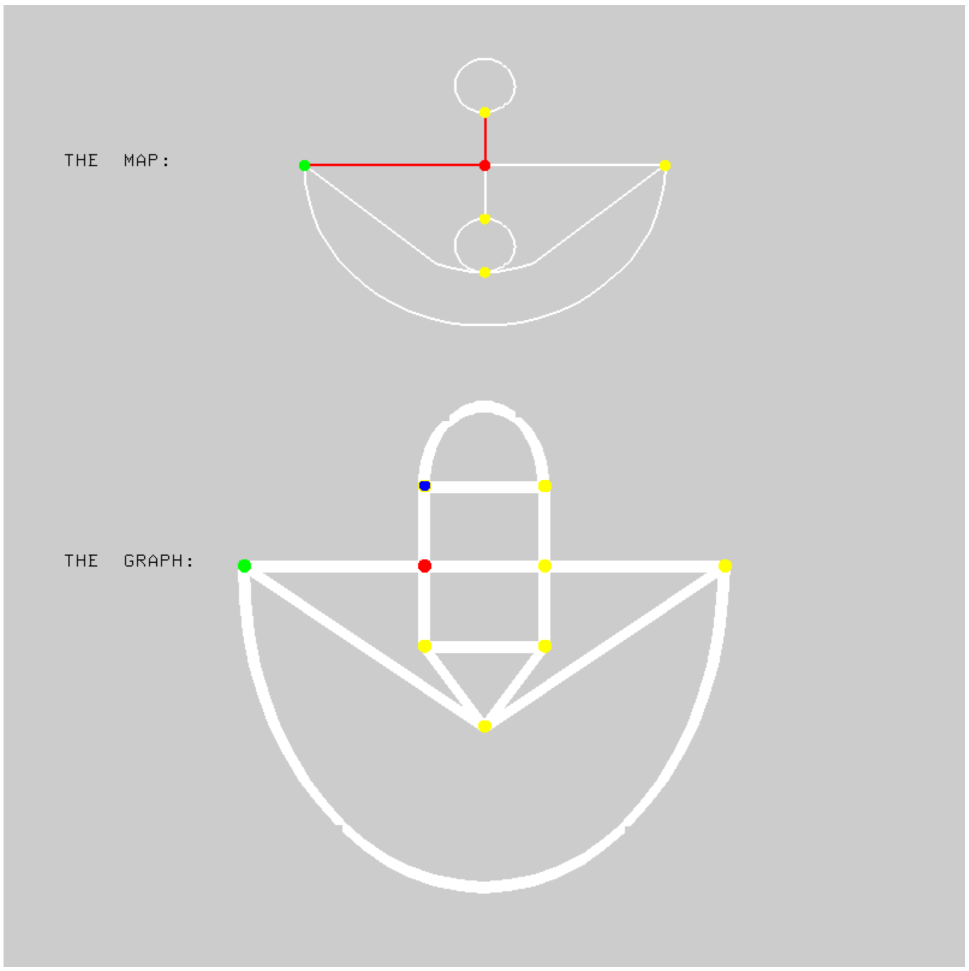

Figure 9: The animation snapshot.

References

[AH] S. Albers and M.R. Henzinger, "Exploring Unknown Environments", *Proc. of the 29th Annual ACM Symposium on Theory of Computing* (STOC'97), pp. 416-425.

[AHU] A.V. Aho, J.E. Hopcroft, J.D. Ullman, "The Design and Analysis of Computer Algorithms," Addison-Wesley Pub. Comp., 1976.

[BCR] R.A. Baeza-Yates, J.C. Culberson, and G.J.E. Rawlins, "Searching in the Plane," *Information and Computation*. **106**, 1993, pp.234-252.

[BRS] A. Blum, P. Raghavan and B. Schieber, "Navigating in Unfamiliar Geometric Terrain," *Proc. of the 23rd Annual ACM Symposium on Theory of Computing* (STOC'91), pp. 494-504.

[DJMW1] G.Dudek, M, Jenkin, E. Milios, D. Wilkes, "Robotic Exploration as Graph Construction," *IEEE Trans. on Robotics and Automation*, Vol.7, 1991,pp.859-865.

[DJMW2] Dudek, G., Jenkin, M., Milios, E., and Wilkes, D., "Map Validation and Self-location for a Robot with a Graph-like Map", *Robotics and Autonomous Systems*, Vol. 22(2), November 1997, pp. 159-178.

[DKP] X. Deng, T. Kameda and C. H. Papadimitriou, "How to Learn an Unknown Environment," *Journal of the ACM*, 45(2), 1998, pp. 215-245.

[DM] X. Deng, A. Mirzaian, "Competitive Robot Mapping with Homogeneous Markers," *IEEE Trans. on Robotics and Automation* 12(4), pp. 532-542, August 1996.

[DP] X. Deng, and C. H. Papadimitriou, "Exploring an Unknown Graph," *Proc. of the 31st Annual IEEE Symposium on Foundations of Computer Science* (FOCS'90), pp.355-361.

[Ed] J. Edmonds, "A combinatorial representation for polyhedral surfaces," Not. Am. Math. Soc. 7, page 646, 1960.

[GMR] L.J. Guibas and R. Motwani and P. Raghavan, "The robot localization problem," *SIAM Journal on Computing* 26(4):1120-1138, 1997.

[GT] J.L. Gross, and T.W. Tucker, "Topological Graph Theory," John Wiley and Sons, New York, 1987.

[HR] N. Hartsfield, G. Ringel, "Pearls in Graph Theory", Academic Press, 1990.

[KB] B. Kuipers, and Y. Byun, “A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representatinos,” *Robotics and Autonomous Systems* **8** (1991) 47-63.

[KL] B. Kuipers, and T. Levitt, “Navigation and mapping in large-scale space,” *AI Mag.*, Summer 1988, pp.61-74.

[Kw] S. Kwek, “On a Simple Depth-first Search Strategy for Exploring Unknown Graphs”, *Proc. of the 6th International Workshop on Algorithms and Data Structures* (WADS’97), Halifax, Nova Scotia, Canada, August 6-8, pp. 345-353. *Proc. of the 23rd Annual ACM Symposium on Theory of Computing* (STOC’91), pp. 278-288.

[LL] T.S. Levitt, and D. T. Lawton, “Qualitative Navigation for Mobile Robots,” *Artificial Intelligence* **44** (1990), 305-360.

[PP] Petrisor Panaite and Andrzej Pelc, “ Exploring Unknown Undirected Graphs”, *Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA’98), pp. 316-322.

[PY] C. Papadimitriou and M. Yannakakis, “Shortest paths without a map,” *Theoretical Comp. Science* 84, 1991, pp.127-150.

[RS] R. L. Rivest and R. E. Schapire, “Inference of Finite Automata Using Homing Sequences,” *Information and Computation* 103, 1993, pp.299–347.