2012 IEEE International Conference on Robotics and Automation
RiverCentre, Saint Paul, Minnesota, USA
May 14-18, 2012

Multi-Agent Deterministic Graph Mapping via Robot Rendezvous

Chaohui Gong, Stephen Tully, George Kantor, and Howie Choset

Abstract— **In this paper, we present a novel algorithm for deterministically mapping an undirected graph-like world with multiple synchronized agents. The application of this algorithm is the collective mapping of an indoor environment with multiple mobile robots while leveraging an embedded topological decomposition of the environment. Our algorithm relies on a group of agents that all depart from the same initial vertex in the graph and spread out to explore the graph. A centralized tree of graph hypotheses is maintained to consider loop-closure, which is deterministically verified when agents observe each other at a common vertex. To achieve efficient mapping, we introduce an active exploration method in which agents dynamically request rendezvous tasks from other available agents to validate graph hypotheses.**

I. INTRODUCTION

For mobile robots to effectively navigate unknown environments, a map of the surrounding environment must be incrementally built during exploration. In the majority of existing literature, this task is referred to as simultaneous localization and mapping (SLAM). The goal of SLAM is to concurrently estimate the most likely pose of the mobile robot while efficiently constructing an environment map.

The typical approach to SLAM is to, in a metric sense, maintain a probability distribution over the joint space of maps and poses [1]–[3]. But an alternative approach is to represent the environment topologically [4]–[7]. Topological maps represent the environment as a graph whose vertices define interesting places in the map and whose edges define feasible paths between places [5]. The goal of topological SLAM is to automatically detect the interesting places in the map and determine the connectivity of the graph through perception and graph inference.

For topological SLAM, when a single robot is mapping a graph-like world, ambiguity can present itself when a robot closes a loop (returns to a previously visited vertex). A common way to solve this problem is to maintain multiple hypotheses when ambiguity occurs (via a hypothesis tree expansion algorithm) with the hope that eventually all but the true graph will be eliminated from the set of generated hypotheses upon receiving distinguishable measurements. This technique is adopted in [8]–[11].

In general, topological SLAM is solved probabilistically, with the likelihood of map hypotheses dependent upon the measurements that are obtained when the robot travels through the environment [8]. But for a more guaranteed approach, there is a body of work that instead explores a deterministic solution to graph mapping with a single mobile robot [12], [13]. For this approach, the way that loop-closing ambiguity is avoided is with a somewhat theoretical but arguably impractical solution: the robot is given a pebble or marker that can be left behind at graph vertices so that upon revisiting a vertex, the robot will recognize the marker and loop-closure can be deterministically verified.

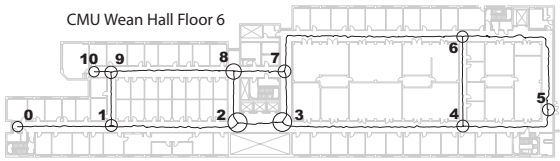

Fig. 1. This is an example of a topological graph that can be deterministically mapped using our exploration algorithm.

This begs the question: what if a second robot could take the place of the marker that is used in the single robot case, thus in a sense, acting as an independently movable marker? In this case, what was once an impractical solution with a single robot, is now a practical opportunity to leverage the cooperation between two or more mobile agents mapping a graph-like environment. This approach has been discussed previously in [14]–[17]. The algorithms presented thus far focus on developing strategies for mobile agents to meet at vertices in the graph (robot rendezvous [15]) to share map and sensory information in a way that leads to deterministic graph inference.

Most of the existing topological mapping algorithms for multiple robots rely on either the usage of markers or simple rendezvous. Das et. al. in [14] assume the environment has whiteboards associated with each vertex that can be marked by agents passing through the vertices to eliminate ambiguity. Other similar algorithms require the existence of markers in some form to verify the true graph structure [16]. Other algorithms require robot rendezvous to share information and to clarify ambiguities [15], [17]. For example, the algorithm proposed by Dudek et. al. [16] has multiple robots explore an environment for an agreed-upon interval, and then the robots return to a common location to merge their individual partial world model.

In this paper, we present a novel algorithm for deterministically mapping an undirected graph-like world with multiple synchronized agents. The algorithm relies on a group of agents that all depart from the same initial vertex in the graph and spread out to explore the graph. A centralized tree of graph hypotheses is maintained to consider loop-closure, which is deterministically verified when agents observe each

S. Tully is with the Electrical and Computer Engineering Department at Carnegie Mellon University, Pittsburgh, PA 15213, USA. C. Gong, G. Kantor, and H. Choset are with the Robotics Institute at Carnegie Mellon University, Pittsburgh, PA 15213, USA, {stully@ece, chaohuig@andrew, kantor@ri, choset@cs}.cmu.edu.

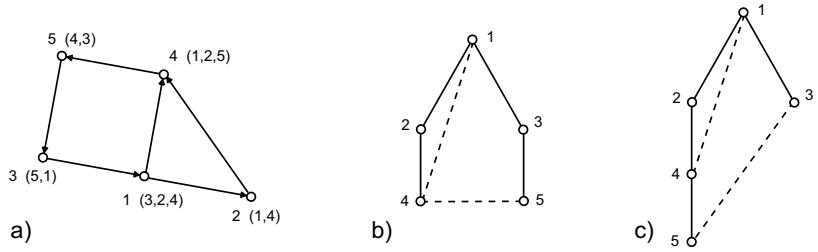

Fig. 2. In (a), an undirected graph is depicted with labeled vertices. In (b) and (c), the same graph is depicted in two different ways as a tree with extra $crossing$ edges.

other at a common vertex. To achieve efficient mapping, we introduce an active exploration method in which agents request rendezvous tasks from other available agents to validate a graph hypothesis. We validate our approach with experiments in an office-like environment, see Fig. 1.

II. Graph Representations and Hypothesis Expansion

With careful exploration, multi-robot graph mapping can be formulated as a deterministic exploration process. The reasoning is that via robot rendezvous, loop-closure can be deterministically verified. In this section, we describe the graph representation we use for the exploration algorithm and how regions of the graph are labeled according to whether they have been verified as a subgraph of the true map. We will also discuss a hypothesis expansion method for proposing loop-closure hypotheses.

A. Graph Representation

As in [8], the topology of an environment can be represented by an edge-ordered undirected graph, which can be explicitly defined by the number of vertices in the graph N and by a set of circular neighbor lists L (one list per vertex), thus a graph $G = (N, L)$. This representation is similarly used in [18]. A neighbor list, such as $L(v)$ stores the vertices in the graph that are neighbors of vertex v in the order that they occur (counter-clockwise from the first mapped edge). In Fig. 2-(a), we show an example graph. The vertices are labeled according to the index of each vertex and each vertex stores a neighbor list.

B. Representing an Undirected Graph as a Tree

Our algorithm assumes that all agents begin an experiment at a common vertex in the graph. In Sec. III, we will discuss the exploration strategy that we use to have the multiple agents deterministically map the graph. But to support that discussion, we first must introduce a way to represent an undirected graph as a tree.

In general, the difference between a graph and a tree is that there is no loop in a tree. In Fig. 2-(b), though, we show that the undirected graph depicted in Fig. 2-(a) can be completely represented by a tree despite having loops, as long as we insert additional edges (shown as dashed lines) to account for the loops. We call these additional edges *crossing edges* because they typically cross over the tree from one branch to another. In Fig. 2-(c), it is demonstrated that the same graph can have two different tree representations. The goal of our multi-agent mapping algorithm is to find a tree representation (with crossing edges) that is isomorphic to the true environment map.

C. Hypothesis Expansion and Loop-Closure

When mapping a graph, our algorithm must maintain multiple hypotheses of the graph to account for the ambiguity of loop-closure. For this reason, hypothesis tree expansion, such as in [8], plays an important role in our algorithm. During mapping, we incrementally expand and prune the tree according to the measurements that are obtained. At each time step, hypotheses that are inconsistent with the observations obtained by the robots will be eliminated. Specifically, for a rendezvous situation, if robot $R1$ can observe robot $R2$ at its current vertex location, then all of the hypotheses in which robots $R1$ and $R2$ are not colocated will be eliminated.

An example is given in Fig. 3. Fig. 3-(a) shows a hypothesis that might be stored in the tree of possible graphs maintained by the algorithm. When the robots move, there are two possible hypotheses that are spawned, shown in Fig. 3-(b) and Fig. 3-(c). The first possibility, shown in Fig. 3-(b), accounts for the possibility that robot $R2$ upon traversing a new edge has closed a loop to another vertex that has an unexplored edge. Fig. 3-(c), on the other hand, accounts for the possibility that robot $R2$ did not close a loop and is instead located at a newly explored vertex in the graph. We note that if, after an additional edge traversal, the two robots were to observe each other at a common vertex, the loop-closure would be verified and the hypothesis in Fig. 3-(c) would be discarded.

Based on this hypothesis tree expansion procedure, we also propose a simple way to decide which part of the map is reliable. During the expansion of the hypothesis tree, we always spawn the connected hypothesis as the left child and the non-connected hypothesis to the right side. After several rounds of expansions and eliminations, the least connected hypothesis would be in the bottom right of the entire hypothesis tree. This organization is important because when a robot needs to navigate to a desired vertex, this is the map that will be trusted for use with a graph search algorithm. The reason for using this hypothesis for navigation is that it will not be affected by a false loop-closure, and so

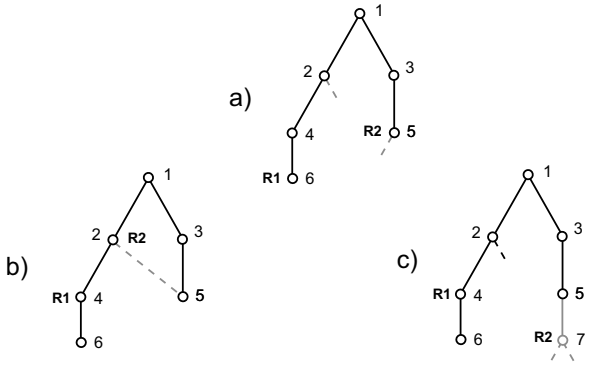

Fig. 3. This is an example of the hypothesis expansion algorithm.

a robot can safely backtrack along verified edges.

D. Maintaining the Verified Subgraph

During the execution of our hypothesis expansion algorithm, we maintain, for each hypothesis, the part of the graph that has been deterministically verified to be a subgraph of the true map. We then label the remaining nodes and edges *unverified*. In Fig. 3-(a), we show the verified component of the hypothesis with dark (black) lines and the unverified part of the map with light (gray) lines. Essentially, an unverified component of the map is converted to *verified* when two robots meet at a common vertex and verify a loop-closure hypothesis.

III. Robot State and Task Assignment

A major difference between single robot mapping and multi-robot mapping is that for multiple robots, cooperation is required. We are introducing here an exploration algorithm that is based on robots *requesting rendezvous tasks* from other available robots to help verify loop-closure hypotheses. The problem of assigning a task to the appropriate robot is the most challenging aspect of this collective approach. Additionally, it is possible to have one robot that is assigned several tasks, and thus needs to plan a route to optimally complete the assigned tasks. We note that some tasks are more important than others, and thus we are presenting a method that prioritizes the robot state according to their current task during the mapping process.

A. Robot State

For our exploration algorithm, each robot has four possible states at any given point in time: *wait*, *busy*, *returning*, and *exploring*. The priority decreases from *wait* to *exploring*.

1) ***Exploring State****:* By default, a robot without any tasks assigned to it will be in the *exploring* state which means that the robot will travel to unexplored vertices in the map. If a new task comes in, the robot's *exploring* state will be interrupted and changed to the *busy* state.

2) ***Wait State****:* When a robot $R1$ arrives at a new vertex which is similar to a previously visited vertex in the verified map, robot $R1$ will *wait* at its current vertex and generate new tasks requesting other robots go to the vertex at which the robot believes it is located (a rendezvous request). When a robot arrives at the requested vertex according to the verified map, there are two possibilities. First, the two robots might observe each other. This would validate a loop closure hypothesis. The other option is that the robots do not observe each other and therefore the loop closure hypothesis is invalid. In the former case, robot $R1$ will cross back to its own branch and continue exploring. In the latter case, robot $R1$ will add a new node to the verified map and continue down that branch.

3) ***Busy State****:* When a robot is assigned a new task (for a rendezvous request) and switches from *exploring* to *busy*, it will instantly record its current position as its *return position* for future use. Only when all of the assigned tasks are finished (the robot reaches all of the vertices it is assigned to), it will change from *busy* to *returning*.

4) ***Returning State****:* If a robot is in the *returning* state, it will navigate to the position stored as *return position* that was recorded when the robot switched from *exploring* to *busy* state. Once at this position, the robot can then continue its normal exploration procedure. Of course, if a new task comes in at this time, the robot will instantly change to the *busy* state once again.

B. Task Array and Task Assignment

When one robot $R1$ arrives at a new vertex $V1$ that is perceptually similar to another vertex $V2$, a new task is immediately created. The reason that a new task is required is that $R1$ will need to arrange a rendezvous with another robot to verify loop-closure. So, $V2$ is added into a task array. This task array will continually send the rendezvous request to all robots for one of them to visit that vertex. In our algorithm, we use a breadth-first-search implementation to find the closest robot from vertex $V2$. When the closest robot is found, we will first check this robot's state. Except for the case that this robot is in a *wait* state, the task will immediately be claimed by this robot and the task will be eliminated from the task array. However, if no robots are in a non-wait state, this task will be kept in the task array and will wait for the next round of task assignments.

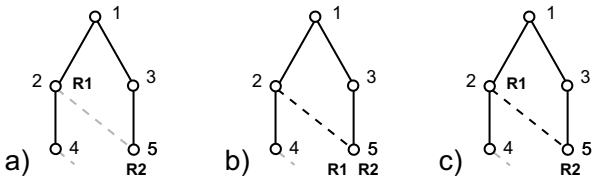

Fig. 4. This is an example of when an *exploring* robot reaches the end of its own branch.

1) Path for Exploring Robot: We require *exploring* robots to move on their own branch of the tree structure (until the branch is complete), hence our use of a tree representation for general graphs with added *crossing edges*. At each time step, an *exploring* robot keeps going down its current branch until reaching the end of a branch. Actually, the end of a branch is typically a vertex which connects to another visited

vertex. Because we do not allow an *exploring* robot to cross to another branch before its own branch is complete, the robot reaching the end of a branch will backtrack to its parent vertex looking for another unvisited edge in the branch. This is illustrated in Fig. 4. In Fig. 4-(a), robot $R1$ moves from vertex $V2$ to vertex $V5$, and the *crossing edge* $V2$-$V5$ is identified. Because robot $R1$ has not finished exploring its own branch, it backtracks to vertex $V2$ during the next time step, as showed in Fig. 4-(c).

2) Path for Busy Robot: A busy robot is different than an *exploring* robot in that it can traverse *crossing edges* as long as they have been verified. The right most graph in the hypothesis expansion tree does not contain false crossing edges, and as a result the path generated based on this graph must be reliable.

3) Rule for Wait Robot: When one *exploring* robot arrives at a vertex which is similar to a vertex that has already been visited, the robot will change into the *wait* state and send a task to the task array in order to arrange a meeting with another robot to verify a loop-closure. Only when another robot claims the task and arrives at the task vertex, the *wait* state of the first robot can be removed. However, there are two different cases. If these two robots finally meet each other at the vertex, a loop is detected and the waiting robot has to backtrack to its own branch. If the second robot arrives at the desired vertex but the robots cannot see each other, we can conclude that the first robot just happens to be visiting an ambiguous vertex which still belongs to its own sub-tree. In this case, the first robot will just return to the *exploring* state and keep exploring the unverified part of the map.

C. BFS Using the Verified Map

Prior topological SLAM algorithms generally do not keep track of the verified part of the map because most of them are non-deterministic. As a result, they are not able to use the knowledge about the environment at hand to accelerate the mapping process. However, because a deterministic mapping approach is adopted here, we can identify the reliable part of the graph and take full advantage of exploration. Specifically, breadth-first-search (BFS) is used in our algorithm to generate the shortest paths for the robots navigating to their task locations.

D. Algorithm Description

Our algorithm is outlined in Alg. 1. It will continue to execute until every vertex v in the verified map is complete. During the exploration, the motion (next moving edge) e_r for robot r is generated according to r's current state. For a $wait$ robot, it will not move until the $wait$ state is removed. An $exploring$ robot always follows a branch leading to an unvisited vertex. However, when this robot r arrives at a vertex v' which is similar to a visited vertex $v_{similar}$, its state s_r will immediately change to $wait$ and $v_{similar}$ will be added into the $TaskArray$. Once the $TaskArray$ is updated, it will try to assign the tasks to other robots. Then, all of the robots states are updated according to task assignment. BFS is used to search for the first edge

Algorithm 1 Deterministic Mapping Algorithm

```
1:  while Incomplete(G) do
2:    for each each robot r do
3:      if s_r=wait then
4:        continue;
5:      else if s_r=exploring then
6:        e_r = FindBranch(r)
7:        v' = execute(e_r);
8:        if v_similar = IsSimilar(v') then
9:          TaskArray.push(v_similar);
10:         s_r ← wait;
11:         AssignTask(TaskArray);
12:         UpdateStates( );
13:       end if
14:     else if s_r=returning then
15:       e_r=BFS(v^r_returning);
16:       v' = execute(e_r);
17:       UpdateStates( );
18:     else
19:       e_r=BFS(v^r_task);
20:       v' = execute(e_r);
21:       UpdateStates( );
22:     end if
23:   end for
24: end while
```

e_r of the shortest path leading a $returning$ robot to the $return\ position$. After executing e_r, if the $return\ position$ is reached, s_r changes to $exploring$. While, for a $busy$ robot, e_r returned from BFS will lead the robot to its closest task location. As long as v^r_{task} is empty, s_r will be updated to $returning$ state. Additionally, if v' eliminates the ambiguity rooted from a $wait$ robot, that robot's $wait$ state can also be removed.

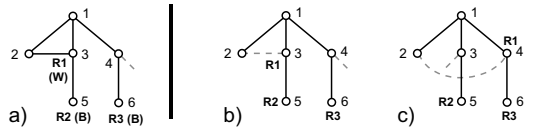

Fig. 5. The robot R1 moves from V2 to V3. And two tasks V3, V4 are generated and distributed to robot R2, R3 respectively. Letters in parentheses denote states of the robots. W indicates *waiting*, B indicates *busy* and E means *exploring*.

E. Task Generation Example

Task generation plays a central role in our algorithm. To successfully map an environment or identify a loop, the correct tasks need to be assigned to the appropriate robots. Additionally, task assignments will determine the efficiency of this mapping procedure. During the mapping process, robots are playing different roles to cooperatively fulfill different tasks. As introduced in Sec. III, robots dynamically change their states according to their current tasks with different priorities. A task is generated when one robot arrives at a vertex via an edge which is suspected to be a

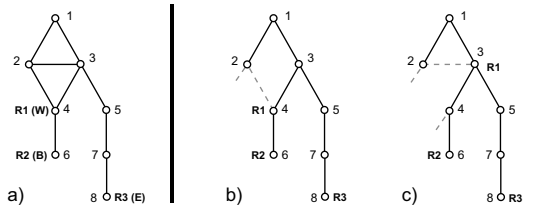

Fig. 6. R1 moves from V2 to V4 and, again, two tasks, V3 and V4 are generated. As a result of BFS, both tasks are claimed by robot R2. Letters in parentheses denote states of the robots. W indicates *waiting*, B indicates *busy* and E means *exploring*.

crossing edge. This robot has to call other robots coming to all of the suspected vertices to clarify the ambiguity. These suspected locations will be stored in a task array for future allocation to robots.

To efficiently finish all tasks, our algorithm again uses BFS to determine the closest available robot to each task in the task array. Thus, different tasks can either be assigned to different robots or to just one single robot depending on their distances to the tasks. For example, robot $R1$ in Fig. 5 is now arriving at a suspected vertex which might close a loop. Two tasks ($V3$ and $V4$) are generated and distributed to $R2, R3$ respectively, as a result of BFS. Fig. 6 shows another example in which all of the tasks are claimed by just one robot. Robot $R1$ moves from $V2$ to $V4$ and two suspected loops are formed. The generated tasks $V3$ and $V4$ are both claimed by $R2$, keeping $R3$ in the *exploring* state.

IV. Algorithm Evaluation

To demonstrate the effectiveness of our algorithm, we have compared the algorithm's performance with a mapping result derived from a purely random exploration strategy. We collected experiment data with a wheeled mobile robot on the 6th floor of Wean Hall University at Carnegie Mellon University in Pittsburgh, Pennsylvania, USA, as shown in Fig. 1. We post-processed our deterministic mapping algorithm using the collected data in such a way that it simulated 5 robots.

When we ran our deterministic mapping algorithm on this topology, the maximum number of generated hypotheses at any point in time during the experiment was 2, while the result from a random motion exploration strategy produced a maximum of 11 hypotheses (and we note that the random exploration was not deterministic). For this map, our algorithm required only 28 time steps to completely map the graph. However, the random motion strategy varied between trials and averaged more than 100 time steps to completely map the graph. The random exploration strategy spawns many more hypotheses because it does not manage the robots to actively eliminate hypotheses, and the number of hypotheses grows exponentially, as is the property of tree expansion.

We also tested our algorithm in a more complex virtual environment which contains 45 vertices. For this map, our probability-based single-robot algorithm [8] required more than 1000 edge traversals to find the true hypothesis. Additionally, 10 thousand hypotheses were spawned during

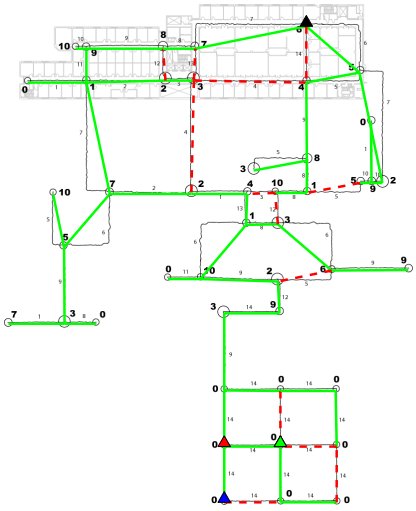

Fig. 7. To further validate our algorithm, we ran our algorithm in a much more complex imagined map. This figure shows the final result of the simulation. The graph is decomposed into a tree and several *crossing edges*. All of the green edges belong to the tree structure and the red dashed lines are *crossing edges*.

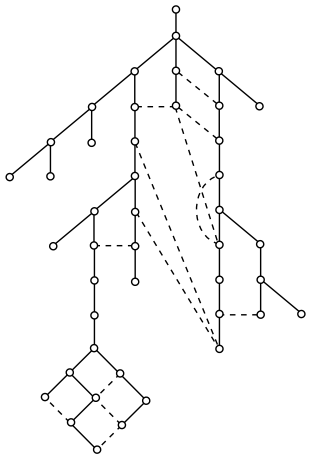

Fig. 8. The result of the the experiment can be displayed as a tree with crossing edges.

the simulation. As a comparison, for the same map, the maximum number of hypotheses that were required at any given point in time by our deterministic algorithm was only 10 and it required only 352 edge traversals to complete the mapping process. In addition to more efficient mapping, this deterministic algorithm guarantees that the correct map is found, which is not true for a probabilistic approach. Fig. 7 shows the final result for this experiment. The tree representation for this result can be seen in Fig. 8.

Because the same graph can be represented as different trees with additional *crossing edges*, the tree structure belonging to the decomposed graph directly affects the required number of hypotheses and edge traversals. Also, the tree's structure is affected by the order of edges added into the tree.

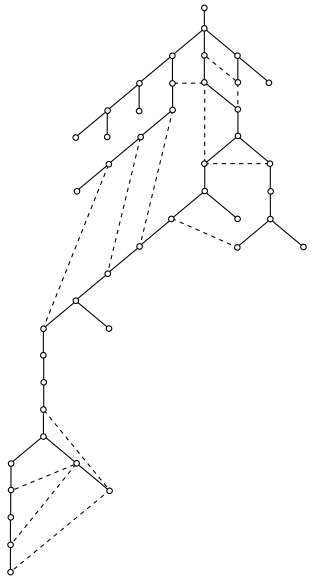

Fig. 9. By varying the ordering of edge choice, a different tree will merge from the deterministic algorithm.

Specifically, when an $exploring$ robot is at the boundary of a verified portion of the map, there are different edges that it can choose to explore the map. This choice will result in different trees. We ran the afforementioned experiment in the same environment with a different edge ordering. This time, only 289 edge traversals were needed and the maximum number of hypotheses was reduced to 6. Fig. 9 shows the tree representation for this result. The difference in performance between these two experiments indicates a potential improvement based on intelligent edge traversal choices for the cooperating robots.

V. Conclusion

The algorithm we present in this paper is a novel multi-agent deterministic mapping strategy. The application of this algorithm is the collective mapping of an indoor environment with multiple mobile robots. Our algorithm relies upon the idea that two agents meeting at a rendezvous location is a powerful source of information that can deterministically confirm loop closure hypotheses.

From the experimental results, we can see that our exploration algorithm provides an efficient approach for multi-robot deterministic graph mapping. The challenging problem of loop closure is solved efficiently by an exploration strategy in which robots request rendezvous tasks to verify loop hypotheses. Also, by using a graph decomposition approach, the graph coverage task can be dynamically divided and assigned to each robot according to their states. Also, by keeping the order of the expanded hypotheses in our expanded hypothesis tree, we can always extract a reliable map on which to plan paths for the robots. This ensures that a robot that has been given a rendezvous task can find a path to the meeting point.

For future work, some modifications can be made to improve the performance of the presented algorithm. For example, if one robot has traveled deeply along a single branch in the tree representation of the topological graph and is requested by another robot to return to the top of the tree to verify loop closure, that robot has to travel a great distance. However, if we carefully restrict the depth of robot exploration, this situation can be avoided.

Also, as discussed at the end of the last section, the order that the robots explore unvisited edges can also impact the performance of the algorithm. This is because the resulting tree structure will vary depending on the exploration, which can highly affect the distance that the robots need to travel upon receiving new rendezvous tasks.

References

[1] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," *Autonomous Robot Vehicles, Springer*, 1990.

[2] M. Dissanayake, P. Newman, H. Durrant-Whyte, S. Clark, and M. Csorba, "A solution to the simultaneous localisation and map building (SLAM) problem," *IEEE Transactions of Robotics and Automation*, vol. 17, no. 3, pp. 229–241, June 2001.

[3] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proceedings of the 2002 AAAI National Conf. Artificial Intelligence*, 2002.

[4] H. Choset and K. Nagatani, "Topological simultaneous localization and mapping (slam): toward exact localization without explicit localization," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 2, pp. 125–137, April 2001.

[5] B. Kuipers and Y.-T. Byun, "A robot exploration and mapping strategy based on a semantic hierachy of spatial representations," *Robotics and Autonomous Systems*, vol. 8, pp. 46–63, 1991.

[6] B. Lisien, D. Morales, D. Silver, G. Kantor, I. Rekleitis, and H. Choset, "The hierarchical atlas," *Robotics, IEEE Transactions on*, vol. 21, no. 3, pp. 473–481, June 2005.

[7] E. Remolina and B. Kuipers, "Towards a general theory of topological maps," *Artificial Intelligence*, vol. 152, no. 1, pp. 47–104, 2004.

[8] S. Tully, G. Kantor, H. Choset, and F. Werner, "A multi-hypothesis topological slam approach for loop closing on edge-ordered graphs," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, October 2009, pp. 4943–4948.

[9] F. Savelli and B. Kuipers, "Loop-closing and planarity in topological map-building," *Intelligent Robots and Systems, 2004. IROS 2004. IEEE/RSJ International Conference on*, pp. 1511–1517, 2004.

[10] G. Dudek, P. Freedman, and S. Hadjres, "Using local information in a non-local way for mapping graph-like worlds," *Proc. the 3rd International Conference on Artificial Intelligence*, pp. 1639–1645, 1993.

[11] ——, "Using multiple models for environmental mapping," *Journal of Robotic Systems*, vol. 13, no. 8, pp. 539–559, 1996.

[12] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "Robotic exploration as graph construction," *Robotics and Automation, IEEE transactions on*, vol. 7, no. 6, pp. 859–865, 1991.

[13] G. Dudek, M. Jenkin, and D. Wilkes, "Map validation and robot self-location in a graph-like world," *Robotics and Autonomous Systems*, vol. 22, no. 2, pp. 159–178, 1997.

[14] S. Das, P. Flocchini, A. Nayak, and N. Santoro, "Distributed exploration of an unknown graph," *Structural Information and Communication Complexity*, vol. 3449, 2005.

[15] N. Roy and G. Dudek, "Collaborative robot exploration and rendezvous: Algorithms, performance bounds and observations," *Autonomous Robots*, vol. 11, pp. 117–136, 2001.

[16] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "Topological exploration with multiple robots," *7th International Symposium on Robotics with Application (ISORA)*, vol. 3, no. 3, 1998.

[17] H. Wang, M. Jenkin, and P. Dymond, "Enhancing exploration in graph-like worlds," in *Computer and Robot Vision, 2008. CRV '08. Canadian Conference on*, May 2008, pp. 53–60.

[18] G. Vijayan and A. Wigderson, "Planarity of edge ordered graphs," *Technical Report 307, Princeton University*, vol. TR307, December 1982.