Map-Merging-Free Connectivity Positioning for Distributed Robot Teams

Somchaya LIEMHETCHARAT [a,1], Manuela VELOSO [a], Francisco MELO [b], and Daniel BORRAJO [c]

[a] *School of Computer Science, Carnegie Mellon University, Pittsburgh, USA*
[b] *Instituto Superior Técnico, UTL, Porto Salvo, Portugal*
[c] *Departamento de Informatica, Universidad Carlos III de Madrid, Madrid, Spain*

Abstract. We consider a set of static towers with communication capabilities, but not within range of each other due to distance and obstacles. The goal is to achieve connectivity among the towers through a set of robots positioned in a way to act as gateways among the towers. The autonomous mobile robots are initially randomly deployed without necessarily being within range of each other, nor of the static towers, and without any common global coordinates. As the robots move, they may come within range of other robots or towers and can share information. We discuss the challenges of such a multi-robot positioning task without the common referential. We contribute a representation for the connectivity information that allows for the robots to share connectivity information without the need to merge the individual maps that they acquire while they navigate the environment. We further present several heuristics to guide the robot motion to explore the environment in search of towers and other robots. The robots analyze their own accumulated map and communicated information from other robots, and can determine if a complete positioning exists to achieve the joint connectivity goal. We further introduce different exploration heuristics, illustrate our algorithm in simulation, and compare the efficiency of the proposed exploration heuristics. We show that our representation is sufficient for the robot team to achieve a connected configuration with the static towers, without the need for merging their individual maps.

Keywords. multi-robot, connectivity, distributed, position label, network graph

Introduction

We are interested in planning for multiple distributed robots to achieve a common positioning goal, without the need for map-merging. Concretely, we address the problem of using a set of mobile robots to ensure connectivity between a number of static communication towers sparsely deployed in an unknown environment and not within range of each other. The robots are themselves communication nodes and can communicate with the static towers and with one another, when within range. We assume that the robots have no knowledge of the environment, both in terms of the obstacles and the positioning of the static towers. The obstacles, such as walls, impede the movement of the robots, and affect connectivity between robots and towers/other robots.

There are several real scenarios that are instances of the general problem we address. For example, emergency teams that need to assist in areas not fully covered with commu-

[1]Corresponding author: Somchaya Liemhetcharat, School of Computer Science, Carnegie Mellon University, Pittsburgh, USA; E-mail: som@ri.cmu.edu.

nication towers or where the connectivity is lost, can carry and drop small mobile robots to autonomously navigate and position themselves so that the connectivity is extended in the crisis area. More generally, this problem is not specific to the signal connectivity goal, and could be extended to other multi-robot positioning needs with other objectives.

Furthermore, our approach is targeted to be run on small, low-cost robots indoors, where global positioning via GPS or wireless triangulation is unavailable. Also, our approach does not require that the robots are homogeneous, or even know about the capabilities of the other robots — we find solutions to the problem readily without planning the full joint-actions of the robots.

The challenges of the multi-robot navigational planning include the fact that the state is initially completely unknown. In our algorithm, the robots plan their navigation as they incrementally gather connectivity information through plan execution. Our algorithm is fully distributed. It includes a network graph for the connectivity state representation, which is incrementally recorded as the robots navigate. The robots share information when within range. Our algorithm requires no prior knowledge of the environment and no common map merging. Given the information gathered, at each step each robot, individually and in a fully-distributed manner, checks for the existence of a solution configuration that achieves the desired connectivity goal. If such a configuration exists, the robots execute the corresponding navigation plan to position themselves in the previously visited locations that constitute the solution. Otherwise, they continue planning the state exploration, driven by heuristics to improve the efficiency of the solution finding.

The organization of our paper is as follows: in Sec. 1, we discuss related work and the differences with our problem and approach. In Sec. 2, we describe the problem, our assumptions, and a general overview of our approach and contributions. In Sec. 3, we explain our algorithm and associated data structures in detail, showing that our representation is sufficient to find a solution. We then discuss heuristics used for the robots' exploration in Sec. 4, and we summarize our contributions in Sec. 5.

1. Related Work

Previous work addressed the problem of dispersing a robotic swarm to provide coverage, using wireless signal intensity as a measure of distance between robots, assuming open space between the robots [3]. Schwager, McLurkin, and Rus presented how robots can position themselves to optimize sensor readings from the environment, using Voronoi graphs [7]. Our goal is to provide connectivity between static towers, using the robots as gateways, and not to maximize coverage of an environment.

By deploying RFID tags as coordination points, robots build a joint map and can coordinate to explore an environment [8]. Our approach does not require any form of map-merging or common global reference frame, or leaving markers in the environment. Instead, the robots use position labels to refer to other robots' positions, without knowing where these positions are in the environment. Also, our goal is not the exploration of an unknown space, but to establish connectivity.

Reich et al. showed that in an environment with unknown obstacles, a robot team that is initially connected can reason about connectivity maintenance, and constrain their mobility in order to avoid disconnecting the network [6] . Similarly, Michael et al. showed that connected robot teams can reason about which links to delete while maintaining connectivity, through distributed consensus and market based auctions [4]. From an initial connected network, robots can achieve biconnectivity, i.e., every robot is connected to at

least 2 other robots, to enable robustness in the network if any robot fails [1]. We address the case when mobile robots start from unknown and unconnected positions.

Poduri and Sukhatme achieved connectivity of mobile robots through coalescence, where robots that are connected coalesce into a cluster and stay connected as they explore the space together [5]. Our goal is to connect static towers using robots as gateways. These robots are capable of exploring the environment, while the towers remain fixed in their locations. In addition, our approach does not require that robots stay connected together; robots can disconnect from other robots and explore independently.

2. Problem Statement and Assumptions

In this section, we formally describe the problem, identify its technical challenges, and present our assumptions.

N autonomous robots are deployed in an unexplored environment containing M static (non-moving) towers. The goal is to find a configuration of robots such that all the towers are connected. Connectivity could be defined as line-of-sight. Thus, connectivity is a general concept that is used to specify the goal, i.e., all towers must be connected via the robots, and can be customized to fit the domain.

The robots do not have a map of the world, nor do they possess any form of global positioning or perform map-merging. Thus, there is no global coordinate system, and robots cannot share coordinates with other robots as there is no common reference frame.

Robots can only communicate when in range. Besides communicating via network packets, they are incapable of sharing information (e.g., by leaving physical markers). The static towers relay packets between robots, and do not perform any computation.

We now list the assumptions of our approach and discuss their implications:

1. The number of towers (M) is known.
2. Each tower has a unique identifier.
3. A robot can approximately revisit any position it previously visited.
4. Connectivity between any 2 robots/towers is unaffected by the locations of other robots/towers.

The 1st assumption allows the robots to know the scope of the problem, while the 2nd allows them to distinguish between the towers.

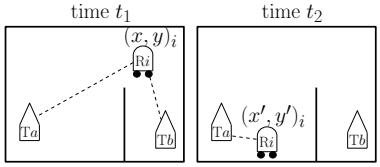
The 3rd assumption implies that the robots are capable of accurately travelling from one location to another in its own reference frame, e.g., by using odometry, and other sensor feedback to adjust for errors in the motion model.

The 4th assumption guarantees that when robots return to previously-visited locations, connectivity that was observed at that location will be restored (assuming the other robot/tower is also in position), regardless of the movements of the other robots.

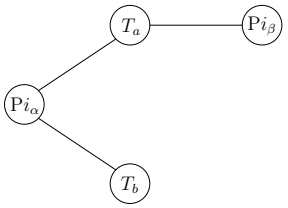
3. Representation of Network Connectivity

Let $\mathcal{R} = \{\mathrm{R}1, \ldots, \mathrm{R}N\}$ be the robots in the environment, and $\mathcal{T} = \{\mathrm{T}1, \ldots, \mathrm{T}M\}$ be the static towers.

Each robot moves autonomously in the environment and the goal is to find a configuration such that all the towers in $\mathcal{T}$ are connected. In any given environment, multiple such configurations may exist, and we make no requirements as to which one the robots should adopt. The goal is to find *any* such configuration.



(a) Spatial representation of robot Ri in two distinct positions, $(x, y)_i$ and $(x', y')_i$. The bold lines indicate obstacles, and dashed lines represent connectivity.



(b) Graphical representation of the same connections of robot Ri using position labels Pi_α and Pi_β.

Figure 1. Position labels and graphical representation of connectivity

In this section, we ignore how the robots move, and focus on the information the robots collect in order to find a solution configuration. We show that through the use of position labels, the network graph representation is sufficient to solve the problem, without any global coordinate system.

3.1. Position Labels

The robots do not perform map-merging, and do not have a shared or global coordinate system. Thus, in order to refer to different positions, they are unable to use a coordinate system and instead use position labels.

Definition 1. Let R$i \in \mathcal{R}$ be a robot. A **position label** Pi_α is a name that refers to a position (indexed by α) of Ri.

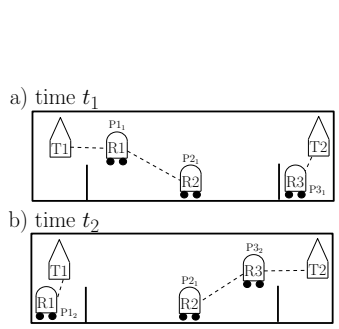
We illustrate the use of position labels through an example. Suppose that at some time t_1, a robot, Ri, is at coordinates $(x, y)_i$, where the subscript i denotes the fact that the coordinates (x, y) are expressed in terms of Ri's reference frame. Let Ri be connected to towers Ta and Tb in this position. At some other time t_2, Ri moves to $(x', y')_i$, and is connected only to Ta. Fig. 1a shows the spatial positions and connections of Ri.

The lack of a global coordinate system prevents robots other than Ri to assign any meaning to the coordinates $(x, y)_i$ and $(x', y')_i$ and as such, Ri assigns a label to each of the two positions, and stores a mapping of the position labels to the coordinates, e.g.,

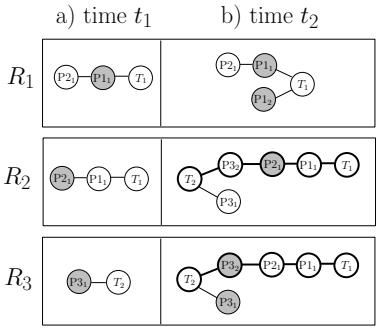
$$\text{P}i_\alpha = (x, y)_i; \qquad \text{P}i_\beta = (x', y')_i$$

Each robot can convert position labels of its own positions into coordinates in its own reference frame, and these position labels can be shared readily among all the robots. For example, when Ri meets another robot Rj, it can share that it (Ri) is connected to Ta and Tb when at position Pi_α, and is connected to Ta when at position Pi_β. Rj can update its information, without knowing the exact coordinates of Ri. All Rj needs to know is that Ri is capable of connecting to Ta and/or Tb at those positions, and that Ri can travel to the positions (since Ri has the mapping of its position labels to coordinates) if need be.

Thus, through position labels, the robots can share connectivity information, without having a global reference frame. Referring back to the example, Rj can share Ri's connectivity information (e.g., Ri is connected to Ta when Ri is at position Pi_β) with other robots, and none of the robots (except Ri) know where the positions actually are.



(a) Spatial representation of 3 mobile robots and 2 towers in an environment.



(b) Network graphs for the 3 robots shown in Fig. 2a. The shaded vertices indicate positions that the robots can convert into coordinates. Edges between vertices indicate connectivity. The solution found is outlined in bold.

Figure 2. Graph representation shared between robots to find a solution

In particular, this connectivity information can be stored in the form of a graph (Fig. 1b). Ri's connectivity information can thus be shared with the team, and they can then update their information, only knowing the position labels of Ri.

3.2. Graph Representation for Connectivity

Position labels allow each robot to refer to other robots' positions in the environment, without knowing the exact coordinates that they refer to. We developed a data representation, that we call a network graph, which allows robots to store, share and merge connectivity information readily.

Definition 2. A **network graph** G is an undirected graph $G = (V, E)$, where each vertex (or node) $v \in V$ is a position label (e.g., Pi_α) or a tower (e.g., Ta). Each edge $e \in E$ is a pair $\{v_1, v_2\}$, where $v_1, v_2 \in V$, and represent connections between the vertices (robots/towers).

To illustrate the usage and benefits of a network graph, consider Figs. 2a and 2b.

Fig. 2a shows that at time t_1, the robots R1, R2 and R3 are at positions P1_1, P2_1, and P3_1 respectively. R1 is connected to R2 and T1, while R3 is connected to T2. The network graphs of the robots are shown in Fig. 2b. The robots synchronize and merge their graphs when connected, which is why R1 and R2 have identical graphs.

At time t_2, R1 and R3 move to positions P1_2 and P3_2 respectively; R2 stays in position P2_1. R2 and R3 are now connected, and R3 is connected to T2. At this time, through the use of position labels, R2 can share information regarding R1 with R3, even though R1 and R3 have never met. This allows both R2 and R3 to discover a solution where R1, R2 and R3 are at positions P1_1, P2_1, and P3_2 respectively. The network graphs of the robots are shown in Fig. 2b, and the solution found is outlined in bold.

The network graph representation offers multiple benefits. First of all, robots can readily share information. When two robots Ri and Rj meet, they can update their individual network graphs and unify their knowledge in all parts of the graph, independently of their current position. Furthermore, they share connectivity information about other robots, without knowing the positions of the robots (only the position labels). This allows

connectivity information to be readily propagated across the robot team, i.e., robots can share connectivity information not only about themselves, but about the entire team.

In addition, a configuration that ensures connectivity of all towers can be obtained directly from the graph. Formally, a solution that connects all towers in a graph G exists iff a sub-graph $G' = (V', E') \subseteq G$ exists such that all towers T$a \in \mathcal{T}$ are connected, and each robot Ri is in at most one position, i.e., $\forall i \ (\text{P}i_\alpha, \text{P}i_\beta \in V' \Leftrightarrow \alpha = \beta)$.

Searching a network graph for a solution can be computationally expensive, since the number of edges grow for each new connection between robots and towers. As such, we developed a representation known as a macro network graph, which is isomorphic to the original network graph, but allows the search to be performed more efficiently.

In a macro network graph, all vertices corresponding to a single robot are collapsed into a single macro node, and macro edges represent all connections between 2 macro nodes (robots/towers). We use this macro network graph representation in the experiments described in the following sections, but since this representation is not the focus of our paper, we do not elaborate on it.

3.3. Updating Connectivity Information

Each robot individually maintains information about network connectivity, using a network graph, that does not include spatial information about robots' positions, but allows connectivity information to be readily shared among the robots. This system of using connectivity information, instead of spatial information, allows the robots to refer to positions, without knowing the actual coordinates of the positions. Thus, no map-merging is required, and robots can even have different map representations.

As each robot moves in the environment, it updates its information in the network graph, adding direct connections to towers and other robots, as well as indirect connections via other robots. When two or more robots are connected, they share and combine their information, so that all connected robots have identical network graphs. Synchronizing graphs involves the union of vertices and edges, and can be performed quickly.

Besides storing connectivity information, the network graph allows robots to check for solutions. The algorithm to check for a solution is deterministic, and so robots with identical network graphs will discover the same solution.

3.4. Converging to a Solution

The goal is to find a solution configuration such that the static towers are connected, using the robots as gateways. In order to do so, the robots run an algorithm, where they can be in one of four states, namely *Explore, Share_Solution, Goto_Solution*, and *Stop*. Each robot starts in the *Explore* state. Fig. 3 shows the state transition diagram.

When a solution is found (in the robot's network graph), the robot transitions from the *Explore* state to *Share_Solution*. In this state, the robot shares its network graph with its neighbors (other robots it is connected to) in the solution. For example, if a robot R1 is connected to robot R2 in the solution, then R1 will search for R2.

Once all neighbors of the solution have been informed, the robot transitions to the *Goto_Solution* state. In this state, the robot heads to its position in the solution found.

Finally, after a robot arrives at its solution position, it transitions to the *Stop* state. The robot remains stationary in this state, acting as a gateway for the towers.

If a different solution is found while the robot is in the *Goto_Solution* or *Stop* state, it returns to the *Share_Solution* state, and looks for its neighbors in the new solution.

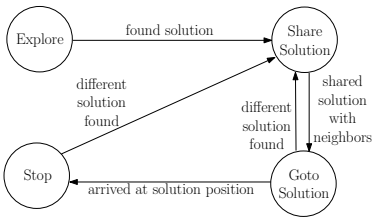


Figure 3. State transition diagram for each robot. The robots start in the *Explore* state. When all robots are in the *Stopped* state, the solution configuration has been achieved and all towers are connected.

Similarly, if a robot is in the *Share_Solution* state and discovers a new solution (by discovering a new connection, or from information shared by another robot), it restarts its sharing process and looks for the neighbors in the new solution.

When all the robots reach the *Stop* state, the solution configuration has been achieved. If a robot Ri is in the *Stop* state while other robots are in other states, either the other robots settle in their positions corresponding to the solution adopted by robot Ri or some robot (that adopted a different solution) will not stop until it connects to Ri. At this point, they synchronize their information and adopt the same solution. If a different solution is found, Ri returns to the *Share_Solution* state. This means that, since there is a finite number of robots, they eventually settle in one solution.

An example of the robot states can be seen using Fig. 2a. At time t_1, robots R1, R2, and R3, are at positions $P1_1$, $P2_1$ and $P3_1$ respectively. R1 and R2 share information since they are in range. At this time, all of the robots are in the *Explore* state.

At time t_2, R1 and R3 move to positions $P1_2$ and $P3_2$ respectively. R2 and R3 are in range and share information. Thus, both R2 and R3 find a solution where R1, R2 and R3 are at positions $P1_1$, $P2_1$, and $P3_2$ respectively. R2 enters the *Share_Solution* state, since it has to inform R1 of the solution. R3 also enters the *Share_Solution* state, but immediately transitions to *Goto_Solution* since its only neighbor in the solution, R2, has been informed. Then, it transitions to *Stop*, since it is already at its solution position.

Once R2 comes in range of R1, they share information, and R1 adopts the same solution. Both R1 and R2 now enter the *Goto_Solution* state (since their neighbors have been informed). They then head to their solution positions and transition to the *Stop* state. At this point, all robots are in their final positions and the towers are connected.

4. Effectively Exploring the Environment

In the previous section, we ignored how the robots moved, and showed that position labels and the network graph representation is sufficient for the robots to find a solution configuration. However, the efficiency in achieving the goal largely depends on the exploration strategy used. In this section, we explore different heuristics used in the robots' exploration, and describe some experimental results.

4.1. Exploration Heuristics

In order to find a solution configuration, the robots have to traverse the world in such a way that the algorithm finds a solution in the network graph as quickly as possible. We explored a number of different heuristics for this purpose.

Random Movement
The simplest heuristic was random movement, where a robot would choose an action randomly from the list of possible actions. There was no weighting of the actions, so with n actions, each would have a $\frac{1}{n}$ probability of being selected. This heuristic provides a baseline for comparison, since it is arguably the most naive form of exploration.

Coverage of the Space
The next heuristic we considered was a coverage algorithm. We adapted the node-counting algorithm described in [2]. Each robot kept a counter of how many times it visited a cell. Then, when choosing an action, it picks the adjacent cell such that its counter is the minimum among all adjacent cells. Unexplored cells always have priority (having a value of 0), and in the case where more than one cell has the minimum value, it picks randomly among the minimum cells.

Weighted Exploration
This heuristic was similar to the coverage algorithm, except that the robot uses a weighted dice to decide among its adjacent cells. Also, we defined an exploration-exploitation ratio to decide between exploring new cells or revisiting cells. If explore was chosen, then the adjacent unexplored cells were chosen with equal probability. If exploit was chosen, then cells that were visited less had a proportionally higher chance of being revisited.

Stay-at-Towers
In this heuristic, the robots had one of 2 roles: stay at an assigned tower, or avoid towers. A robot is assigned the role of staying at tower if it has the most connections to the tower.

In the *stay at tower* role, if the robot is not currently connected to its assigned tower, then it plans the shortest path that connects it to the tower. If the robot is already connected to the tower, then it decides to explore or exploit, similar to the weighted exploration heuristic above. However, it ignores all adjacent explored cells that do not have a connection to the tower, and so the robot stays close to its assigned tower and may lose connection only if it goes to an unexplored cell that is out of the tower's range.

In the *avoid towers* role, the robot chooses between explore, exploit, and visiting a tower. The robot chooses between these 3 options based on a exploration-exploitation-visit ratio, weighted by how many cells match the categories.

By using the heuristic, robots that do not have assigned towers tend to visit areas that have no connections to any tower, and unexplored regions. This allows new towers to be found quickly, and connections to be found between towers.

4.2. Experiments and Results

We created a simulator that models a discrete 2D world, with horizontal and vertical walls placed in between cells. The simulator calculates signal strength between any two cells, based on degradation from distance and obstacles. We modeled 3 scenarios, an office, a corridor, and a lobby (see Fig. 4), with 20×24, 40×24, and 50×50 cells respectively. The robots had 4 possible actions, moving North, East, South and West.

We ran the different heuristics in the 3 scenarios, with 100 trials per heuristic per scenario. In each trial, 5 robots were placed in the environment. The *Stay-at-Towers* heuristic performed best in the 3 scenarios, and Fig. 5 shows the heuristics' performance.

We performed further experiments on the *Stay-at-Towers* heuristic, since it had the best performance. We varied the number of robots from 5 to 50, and observed the percentage of trials that found a solution in a given amount of time. Increasing from 5 to 15

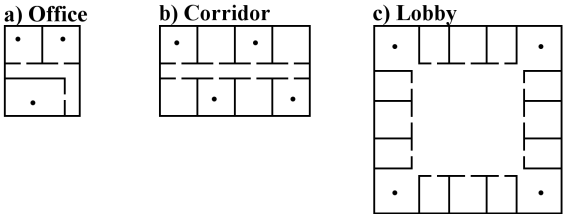
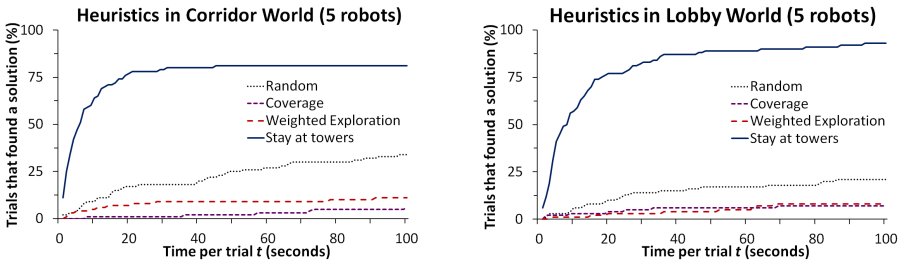
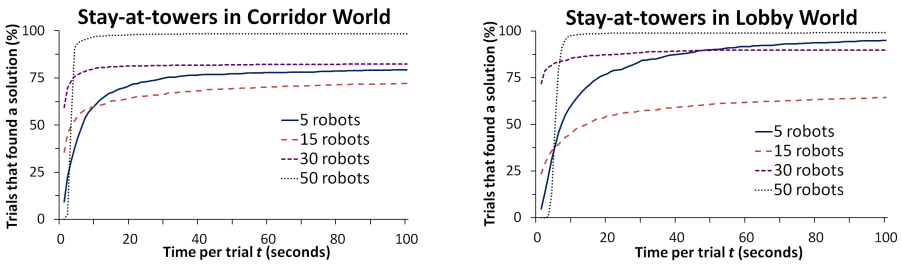


Figure 4. Representative scenarios that were experimented on: a) Office b) Corridor, c) Lobby.



(a) Corridor World (b) Lobby World

Figure 5. Percentage of trials that found a solution in t seconds or less with 5 robots.



(a) Corridor World (b) Lobby World

Figure 6. Performance of *Stay-at-Towers* heuristic with varying number of robots.

robots decreases the performance of the algorithm, since the size of the network graph grows with the number of robots, and increases the amount of time taken to check the graph for a solution configuration. However, increasing the number of robots from 15 to 50 increases the performance, since the number of goal configurations increases dramatically, and the robot team has a larger probability of starting close to a goal configuration. Fig. 6 shows the performance of the heuristic with varying numbers of robots.

5. Conclusion

The goal of the robot team is to achieve connectivity between static towers, by positioning themselves as gateways. The robots explore the environment, and collect information on connectivity as they do so. When robots meet, they share their information in their network graphs, which allows them to readily find a solution configuration.

The robots create position labels which they share with each other. These position labels do not contain coordinate information, since there is no global coordinate sys-

tem and map-merging is not performed. A robot can reference another robot's position, without knowing where that position is in the actual environment.

Instead of sharing and merging maps, the robots build a more effective representation of connectivity — a network graph, and share this information whenever they meet. Merging a network graph involves just the union of vertices and edges, which can be efficiently performed. Furthermore, the network graph representation allows sharing of information that can be propagated across the team effectively, since robots can share connectivity information about all their positions, as well as positions of other robots.

Once a solution is found, each robot simply has to travel to its solution position. Thus, the difficulty of the overall planning problem consists of effectively exploring the space for configurations that can be useful for the connectivity goal. We introduced several exploration heuristics, and showed that the *Stay-at-Towers* heuristic had the best performance, and is effective in finding solutions in representative scenarios.

The connectivity goal is not limited to communications. It can be generalized to any sort of goal involving a binary relation between objects in the world that are affected by spatial positions. For example, in a surveillance scenario, line-of-sight could be used as a measure of connectivity. The robots would position themselves such that the towers would be within line-of-sight of some robot, and each robot would be in sight of another.

Acknowledgments

The first and second authors were partially supported by the Lockheed Martin, Inc. under subcontract 8100001629/1041062. The third author was partially supported by the Portuguese Fundação para a Ciência e a Tecnologia (INESC-ID multiannual funding) through the PIDDAC Program funds and by the CMU-Portugal Program. The fourth author acknowledges the funding source that supported his sabbatical leave at Carnegie Mellon University. The views and conclusions contained in this document are those of the authors only.

References

[1] Butterfield, J.; Dantu, K.; Gerkey, B.; Jenkins, O.; and Sukhatme, G. 2008. Autonomous biconnected networks of mobile robots. In *Wireless Multihop Communications in Networked Robotics*, 640–646.

[2] Koenig, S., and Szymanski, B. 1999. Value-Update Rules for Real-Time Search. In *Proc. 16th Int. Conf. Artifical Intelligence*, 718–724.

[3] Ludwig, L., and Gini, M. 2006. Robotic swarm dispersion using wireless intensity signals. In *Proc. Int. Symp. Distributed Autonomous Robotic Systems*, 135–144.

[4] Michael, N.; Zavlanos, M.; Kumar, V.; and Pappas, G. 2008. Maintaining Connectivity in Mobile Robot Networks. In *Proc. Int. Symp. on Experimental Robotics*.

[5] Poduri, S., and Sukhatme, G. 2007. Achieving Connectivity through Coalescence in Mobile Robot Networks. In *Int. Conf. on Robot Communication and Coordination*.

[6] Reich, J.; Misra, V.; Rubenstein, D.; and Zussman, G. 2008. Spreadable connected autonomic networks (SCAN). Technical Report TR CUCS-016-08, CS Department, Columbia University.

[7] Schwager, M.; McLurkin, J.; and Rus, D. 2006. Distributed coverage control with sensory feedback for networked robots. In *Proc. Robotics: Science and Systems*.

[8] Ziparo, V.; Kleiner, A.; Nebel, B.; and Nardi, D. 2007. RFID-based exploration for large robot teams. In *Proc. IEEE Int. Conf. Robotics and Automation*, 4606–4613.