Distributed Map-Merging-Free Multi-Robot Positioning for Creating a Connected Network

Somchaya Liemhetcharat, *Student Member, IEEE,* Manuela Veloso, *Senior Member, IEEE,* Francisco Melo, *Member, IEEE,* and Daniel Borrajo

***Abstract*—We consider a set of static towers with communication capabilities, but not within range of each other, i.e., sparsely positioned in an environment with obstacles that degrade the communication signal, e.g., emergency teams in areas where connectivity has been lost. We address the problem of deploying mobile robots, initially not necessarily within range of each other or of the static towers, to be communication gateways among the towers. The robots do not know the environment, the towers positions, nor their own initial position in global coordinates. We first discuss the challenges of the domain. We then contribute our distributed algorithm, where robots share connectivity information without merging maps, acquire information through their navigation, and heuristically plan their exploration. The robots analyze their own accumulated knowledge, and determine if a positioning plan exists to achieve the joint connectivity goal. We introduce different navigation heuristics. We illustrate our algorithm in simulation and compare the efficiency of the proposed heuristics. We apply the most promising heuristic in a variety of realistic indoor scenarios, demonstrating its efficacy.**

***Index Terms*—distributed robot systems, networked robots, autonomous agents, no map-merging, mobile robots**

I. 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, interfere with the signal propagation, and pose challenges in terms of modeling the signal propagation. In open space, models of wireless signal decay allow the signal strength to provide a good distance estimate [1], while in the presence of poorly modeled obstacles, signal strength provides multiple distance hypotheses, preventing the use of the network signal strength for localization.

Manuscript received September 4, 2009.

S. Liemhetcharat and M. Veloso are with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 15213 USA (e-mails: som@ri.cmu.edu and veloso@cs.cmu.edu).

F. Melo is currently with the Intelligent Agents and Synthetic Characters Group at INESC-ID, Lisbon, Portugal. Most work described herein was developed while he was with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 (e-mail: fmelo@inesc-id.pt).

D. Borrajo is with the Departamento de Informtica, Universidad Carlos III de Madrid (e-mail: dborrajo@ia.uc3m.es).

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 communication towers or where the connectivity is lost because of a disaster, 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 connectivity goal, and could be extended to other multi-robot positioning needs with other objectives.

Previous work addressed the problem of network-based localization and navigation in either open space [2], [3] or given a prior representation of the environment [1], [4], [5]. In the presence of unknown obstacles, robots can start initially connected and reason about connectivity maintenance [6]. A sensor network can also be deployed after mapping an unknown environment [7]. We instead address the challenges of environments with obstacles, when a set of mobile robots needs to navigate to establish connectivity of existing static nodes from initially unknown and unconnected positions. Furthermore, we assume that our robots are limited robotic platforms, such as the Create robots by iRobot, and have no global access to a remote central planner.

We consider the deployment of autonomous mobile robots in the environment with the goal of acting as communication gateways. The robot navigation is ”driven” by the communication signals. The robots can identify each other and the towers from communicated identifiers.

Furthermore, our approach is targetted to be run on many small, low-cost robots indoors, where global positioning via GPS or wireless triangulation is unavailable. We do not use any assumptions about the nature of signal degradation in the environment. 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 all the robots. A robot will never ”instruct” another robot to head to a location that the latter has never visited, and so this ensures that the latter robot has the capabilities to reach its target.

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 hypergraph data structure for the connectivity state representation, which is incrementally recorded as the robots navigate. The robots share information when within range, throughout the environment. Our algorithm requires no prior knowledge of

the environment and no common map merging. The algorithm includes a navigation planner driven by different exploration heuristics, which we present and analyze. 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. II, we describe the problem, our assumptions, and a general overview of our approach and contributions. In Sec. III, we explain our algorithm and associated data structures in detail, as well as theoretical guarantees. We then discuss and analyze the planning heuristics used by the robots in Sec. IV. Sec. V illustrates the results of running our algorithm in different scenarios, and we summarize our contributions and discuss future work in Sec. VI.

II. Challenges, Assumptions, and Approach

In this section, we formally describe the problem and identify its technical challenges. We present our assumptions and an overview of the solution we contribute.

A. Problem Statement and Challenges

N autonomous robots are deployed in an unexplored environment containing M static (non-moving) communication towers. The goal is to find a configuration of robots such that all the towers are connected. In Fig. 1a, towers T1 and T2 are not within direct communication range, and mobile robots R1 and R2 have positioned themselves such that T1 and T2 are connected in the communication network, by using R1 and R2 to relay network packets.

The environment contains physical obstacles, such as walls, that impede the robots' movement as well as degrade signal propagation. As such, it is difficult for the robots to have an accurate model of signal propagation (due to signal degradation, reflection and interference) that will allow them to obtain an accurate estimate of the distance to the towers or other robots, as an accurate planning state. In Fig. 1b, R1 is connected to towers T1, T2, and T3, with equal signal strengths, due to the degradation of signal propagation in air and through the walls.

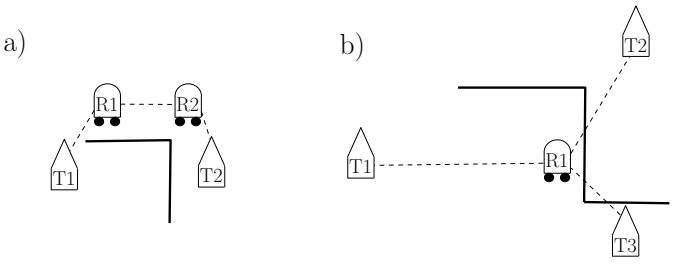

Fig. 1. a) Connectivity example with 2 towers (T1 and T2) that are not within communication range, and 2 mobile robots (R1 and R2). b) R1 is connected to 3 towers, with equal signal strengths. Bold lines indicate obstacles (walls) that degrade signal propagation, and dashed lines indicate connectivity.

The robots do not have a map of the world, nor do they possess any form of global positioning. Thus, there is no global coordinate system, and coordinates used by each robot are relative to its starting position and orientation. Hence, robots cannot share coordinates with other robots as they do not have a common meaning.

Robots can only communicate when in range. In addition, other than communicating via network packets, they are incapable of sharing information (e.g., by leaving physical markers in the world).

B. Assumptions

We list the assumptions of our approach, discuss the implications, and possible ways to overcome the assumptions:

Assumption 1. *The number and identification of the towers (M) are known.*

The identification of the towers can generally be retrieved via the network protocol. If M is unknown, we can use an iterative deepening approach combined with time-limited exploration at each iteration, to prevent the algorithm from running infinitely.

Assumption 2. *The environment is bounded.*

Assumption 3. *There exists at least one configuration for $k \leq N$ robots that connects all M towers.*

Assumptions 2 and 3 limit, in a very straightforward way, the amount of exploration that the robots need to entail in order to find a solution. More concretely, Assumption 2 ensures that the space that the robots need to explore in order to find the solution is somehow bounded. In practice, Assumption 2 is not limiting, as the space is limited by the finite number of towers. Assumption 3 is also quite reasonable in that it ensures that there is a solution. This, in turn, implies that the exploration algorithm does not go on forever and eventually terminates, if it is complete. Notice also that we do not require *all* robots to be part of the solution, which means that we do not need to know beforehand how many robots are necessary to attain a solution, as long as we have some upper bound on this number.

Assumption 4. *The exploration algorithm for the robots are such that, at any time t,*

$$\mathbb{P}\left[\tau_C(t) < \infty\right] = 1,$$

where C denotes a general configuration of the robots in the environment and $\tau_C(T)$ is the first return time to C after a given time T.

This assumption states that each robot can revisit any configuration, in a finite amount of time, that may be relevant to find a solution. This assumption is used to guarantee that the relevant network information is passed between the robots and eventually propagates to all robots in the team. In practice, given the relatively large range within which the robots and towers can communicate, the solution configuration can be visited effectively. In general, the exploration algorithm can be designed so that each robot incrementally extends its area of exploration, cyclically returning to the areas already explored.

Assumption 5. *Communication is instantaneous, costless and error-free.*

In practice, communication is not instantaneous and is subject to error. Also, robots may come into and out of range of one another as messages are sent, causing messages to be lost. This assumption, however, only affects the efficiency of our algorithm. Our algorithm is completely asynchronous, and can use any communication protocol to make communication more reliable. Thus, we focus on achieving the goal, and abstract our problem from errors in communication.

C. Overview of Approach and Contributions

The robots explore the environment, and collect information on connectivity as they do so. When robots meet (get within communication range), they share their information, which allows them to readily find a solution configuration. Once a solution is found, the robots head to their solution positions and provide connectivity to all the towers in the environment.

We now describe some important features of our approach, outlining its contributions:

- Instead of sharing coordinates (which is impossible, since there is no global coordinate system and map-merging is not performed), the robots create position labels which they share with each other. A robot can reference another robot's position, without knowing where that position is in the actual environment (Sec. III-A).
- Instead of sharing and merging maps, the robots build a more effective representation of connectivity - a hyper-graph (Sec. III-B), and share this information whenever they meet (Sec. III-C). Merging a hyper-graph involves just the union of vertices and edges. Furthermore, the hyper-graph representation allows sharing of information that can be propagated across the team of robots efficiently.
- In this hyper-graph representation, determining whether the goal can be achieved with present knowledge equates to searching the graph for a connected solution, subject to certain constraints on the edges in the hyper-graph. Searching for such a solution is an NP-complete problem, but we contribute a method that reduces the search space and runs efficiently in practice (Sec. III-D).
- 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 propose multiple planning heuristics to perform the exploration (Sec. IV), and present extensive simulation results in representative scenarios (Sec. V).

III. Distributed Network Connectivity

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

Each robot Ri moves autonomously in the environment and the purpose of the robot team is to find a configuration C such that all the towers in $\mathcal{T}$ are connected, as illustrated in Fig. 1.[1] Notice that, 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. Position Labels

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 III.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.*

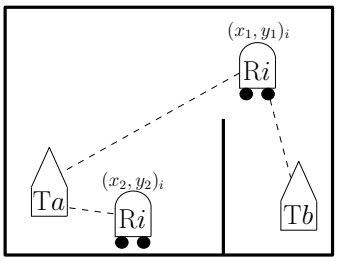

Fig. 2. Spatial representation of robot Ri in two distinct positions, $(x_1, y_1)_i$ and $(x_2, y_2)_i$. The coordinate system is with respect to Ri's initial position and orientation. The bold lines indicate obstacles (walls) in the environment, and dotted lines indicate connectivity between Ri and the towers (Ta or Tb).

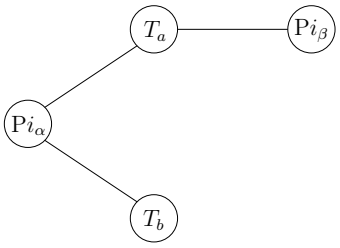

Fig. 3. Graphical representation of the same connections of robot Ri in Fig. 2 using position labels Pi_α and Pi_β. Note that the spatial position of the vertices in the graph have no meaning, while the edges between vertices indicate connectivity.

We illustrate the use of position labels through an example (see Figs. 2 and 3). Suppose that a given robot, Ri, at some time t_1, is at coordinates $(x_1, y_1)_i$, where the subscript i denotes the fact that the coordinates (x_1, y_1) 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 is at coordinates $(x_2, y_2)_i$, and is connected only to Ta. The spatial positions and connections of Ri at t_1 and t_2 are shown in Fig. 2.

The lack of a global coordinate system prevents robots other than Ri to assign any meaning to the coordinates $(x_1, y_1)_i$ and $(x_2, y_2)_i$ 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.,

$$\mathrm{P}i_\alpha = (x_1, y_1)_i; \qquad \mathrm{P}i_\beta = (x_2, y_2)_i$$

[1] A *configuration* is a vector of positions in the environment, one for each robot.

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. In particular, this connectivity information can be stored in the form of a graph, as shown in Fig. 3. Ri merely has to share the graph shown in Fig. 3 to allow Rj to store the new connectivity information.

B. Hyper-Graph Representation for Connectivity Information

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 III.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), and each edge $e \in E$ is a pair $\{v_1, v_2\}$, where $v_1, v_2 \in V$. Edges represent connections between vertices (robots/towers), and the weights of the edges represent the signal strengths of the connections.*

To illustrate the usage and benefits of a network graph, consider Figs. 4 and 5.

At time t_1 (see Fig. 4a), 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. 5a. Note that the robots synchronize and merge their graphs when connected, which is why R1 and R2 have identical graphs.

At time t_2 (see Fig. 4b), 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, 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. 5b, 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, the updating of graphs is asynchronous in the sense that not all robots need to have the same network representation at all times.

In addition, a configuration that ensures connectivity of all towers can be obtained directly from the graph. Formally, a solution s that connects all towers in a graph $G = (V, E)$ 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., $\forall i\ (\text{P}i_\alpha, \text{P}i_\beta \in V' \Leftrightarrow \alpha = \beta)$.

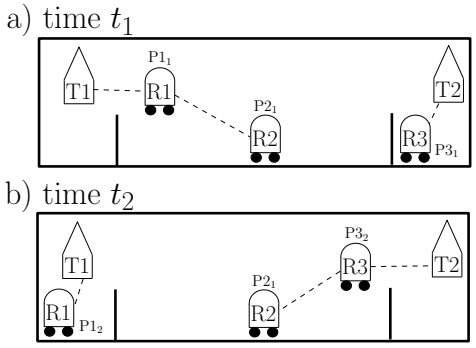

Fig. 4. Spatial representation of 3 mobile robots and 2 towers in an environment. Bold lines indicate obstacles (walls) and dashed lines indicate connectivity. a) At time t_1, R1, R2 and R3 are in positions P1_1, P2_1, and P3_1 respectively. b) At time t_2, R1 and R3 move to positions P1_2 and P3_2 respectively while R2 stays at P2_1.

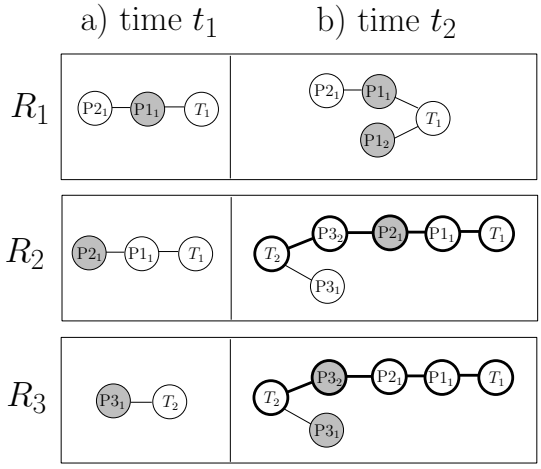

Fig. 5. Networks graphs for the 3 robots shown in Fig. 4. The shaded vertices indicate positions that the robots can convert into coordinates. Edges between vertices indicate connectivity between robots (at a certain position) and towers. a) At time t_1, R1 and R2 are connected and synchronize their graphs. b) At time t_2, R2 and R3 synchronize their graphs and discover a solution (outlined in bold).

However, given that the size of the network graph increases as the robots explore the environment, searching this graph can become computationally expensive. In order to cope with this growth, we consider a *hyper-graph representation*, in which all nodes corresponding to each robot are collapsed into a single hyper-node. A network graph and its corresponding hyper-graph are depicted in Fig. 6.

Definition III.3. *A **hyper-node** $\mathbf{v}$ is an equivalence class defined over the set of vertices of the original network graph $G = (V, E)$, that corresponds to a single robot or tower, e.g., the hyper-node R$i = \{\text{P}i_\alpha : \text{P}i_\alpha \in V, \forall\alpha\}$.*

Definition III.4. *A **hyper-edge** $\mathbf{e} = \{\mathbf{v}_1, \mathbf{v}_2\}$ is an equivalence class defined over the set of edges in the original network graph $G = (V, E)$, that corresponds to all connections between the 2 hyper-nodes (i.e., $\mathbf{v}_1$ and $\mathbf{v}_2$), e.g., the hyper-edge $\{\text{R}i, \text{R}j\} = \{\{\text{P}i_\alpha, \text{P}j_\beta\} : \{\text{P}i_\alpha, \text{P}j_\beta\} \in E, \forall\alpha, \beta\}$.*

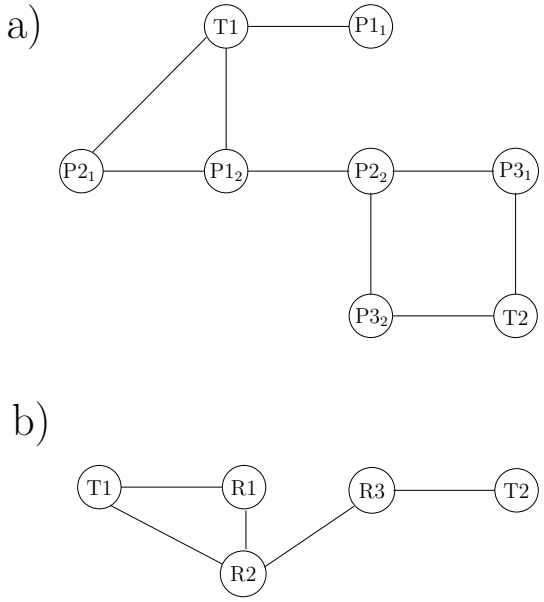

Fig. 6. a) A network graph of 3 robots and 2 towers. b) The corresponding network hyper-graph of the same 3 robots and 2 towers. The position labels are not shown in the hyper-graph, even though the information is embedded in the hyper-nodes.

Definition III.5. *A **network hyper-graph** is an undirected graph $H = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of all hyper-nodes, and $\mathcal{E}$ is the set of all hyper-edges.*

A network graph can be represented as a network hyper-graph, and vice versa. In a hyper-graph,[2] $|\mathcal{V}| \leq M + N$, where M and N are the number of towers and robots respectively, and $|\mathcal{E}| \leq \binom{M+N}{2}$. Any particular hyper-edge $\{\mathbf{v}_1, \mathbf{v}_2\} \in \mathcal{E}$ means that, in the original network graph, the robots/towers corresponding to nodes $\mathbf{v}_1$ and $\mathbf{v}_2$ share at least one connection. Each hyper-edge can also be seen as a set of *constraints* on the robots' positions. These constraints limit the possible robot positions in order to have the connection described by the hyper-edge. Referring back to our example, the constraints for the edge $\{\mathrm{R1}, \mathrm{R2}\}$ in the hyper-graph of Fig. 6 would be $E_{\{\mathrm{R1},\mathrm{R2}\}} = \{(\{\mathrm{P1}_2, \mathrm{P2}_1\}, \sigma), (\{\mathrm{P1}_2, \mathrm{P2}_2\}, \sigma')\}$, where σ, σ' denote the signal strengths of the corresponding connections. So, if R1 is in position $\mathrm{P1}_2$, then R2 is constrained to positions $\mathrm{P2}_1$ and $\mathrm{P2}_2$ if the connection $\{\mathrm{R1}, \mathrm{R2}\}$ is desired.

Each hyper-edge $\mathbf{e} \in \mathcal{E}$ is associated with the corresponding equivalence class or constraint set that must store all edges in the original network graph and corresponding signal strength information. This means, in particular, that the hyper-graph representation is equivalent to the original network graph representation in terms of space-efficiency. However, the hyper-graph representation provides significant advantages when searching for a solution, which we will soon show.

We conclude by observing that a solution is a connected subgraph of H that includes all hyper-nodes corresponding to towers in $\mathcal{T}$, and each robot Ri can be in a position Pi_α such that all constraints are met in the solution subgraph. Further details are provided in Sec. III-D.

[2]We henceforth drop the usage of the word *network* for brevity.

C. Communication Phase

As the robots explore the world, they individually maintain a hyper-graph which they use to store connectivity information. Upon coming within communication range, robots share their corresponding hyper-graphs and update them to include the information coming from the other robots. This process can be decomposed into several steps which we now describe. We focus on the situation where two robots meet. The extension to more than two robots follows directly.

Suppose that at some time t_1, Ri comes within range of Rj. Ri communicates its current position label and receives Rj's position label. If necessary, Ri adds Rj's position label to the network graph, using the signal strength as the edge weight. Rj performs the same operation with Ri's position label.

Next, both robots share their hyper-graphs. To do so, they share all new constraints in the hyper-graph discovered since they last met (say t_0). Note that Ri does not need to include any constraints involving Rj, since the Rj's hyper-graph should already contain those constraints. Thus, only a subset of vertices and edges of the Ri's hyper-graph (associated with the constraints discovered since t_0 that do not involve Rj) has to be sent to Rj.

Upon receiving the new set of constraints from Rj, Ri can update its hyper-graph and its resulting hyper-graph will match that of Rj (after Rj receives Ri's message and merges the information). This communication protocol ensures the synchronization of the hyper-graphs of the two robots when they connect. When there are more than 2 robots, this protocol similarly ensures that all connected robots will synchronize their hyper-graphs, through serveral rounds of communication.

Theorem III.6. *Suppose R1, ..., Rn are connected, with hyper-graphs $H_1, \ldots, H_n$ respectively before the communication phase. After the communication phase, R1, ..., Rn will have the same hyper-graph H, such that $H \supseteq \bigcup_{i \in \{1,\ldots,n\}} H_i$).*

Proof: Let the current positions of the R1, ..., Rn be $\mathrm{P1}_{\alpha_1}, \ldots, \mathrm{P}n_{\alpha_n}$.

Let the robots directly connected to robot Ri be $\mathcal{R}_i \subset \mathcal{R}$.

Each robot Ri first adds constraints to its hyper-graph H_i (if the constraints do not already exist) regarding its direct connections to all robots in $\mathcal{R}_i$, i.e., $\left\{\mathrm{P}i_{\alpha_i}, \mathrm{P}j_{\alpha_j}\right\}, \forall \mathrm{R}j \in \mathcal{R}_i$. After this step, R$i$'s new hyper-graph is $H_i' \supseteq H_i$.

Next, Ri shares its updated hyper-graph H_i' with all the other robots (both directly and indirectly connected) in the following way: Ri sends H_i' to its direct neighbors, who merge H_i' with their hyper-graphs. The neighbors then share their updated hyper-graphs with their neighbors, and so on. This synchronization can take multiple rounds of communication until no new information is available to all the connected robots. Thus, Ri receives hyper-graph information from the other robots, and Ri incorporates the shared information. After this operation, Ri's new hyper-graph is:

$$\begin{aligned} H_i'' &= H_i' \cup (H_1' \cup \ldots H_{i-1}' \cup H_{i+1}' \cup \ldots \cup H_n') \\ &= \bigcup_{j \in \{1,\ldots,n\}} H_j' \end{aligned}$$

Therefore, after the communication phase, every connected robot has the same hyper-graph H, where:

$$H = \bigcup_{i\in\{1,\dots,n\}} H'_i$$
$$H \supseteq \bigcup_{i\in\{1,\dots,n\}} H_i \text{ (since } H'_i \supseteq H_i, \forall i \in \{1,\dots,n\}\text{)}$$

■

D. Finding a Solution

The robots' goal is to find a configuration that connects all the towers in $\mathcal{T}$, using the robots to pass messages as necessary. Such a configuration is known as a solution.

Definition III.7. *A hyper-edge $\mathbf{e} = \{Ri, \mathbf{v}\}$ is **applicable** to a position label Pi_α if:*

$\mathbf{v} = Rj$ *and* $\{Pi_\alpha, Pj_\beta\}$ *is in the equivalence class of* $\mathbf{e}$ *for some* β

or

$\mathbf{v} = Ta$ *and* $\{Pi_\alpha, Ta\}$ *is in the equivalence class of* $\mathbf{e}$

Definition III.8. *A **solution** s of a network hyper-graph H is a sub-graph $H_s = (\mathcal{V}_s, \mathcal{E}_s) \subseteq H$ such that all the towers in $\mathcal{T}$ are connected, and each robot can be at a single position, i.e., $\forall\ Ri \in \mathcal{V}_s, \exists\ Pi_\alpha$ such that $\forall \mathbf{e} \in \mathcal{E}_s$, $Ri \in \mathbf{e} \Rightarrow Pi_\alpha$ is applicable to $\mathbf{e}$.*

So, in a solution, each robot has a single position to go to such that when all the robots are in their solution positions, the towers are connected wirelessly, using the robots as network relays as necessary.

To reduce the search space for s, or equivalently H_s, the restriction that H_s is acyclic is added - if H_s contains cycles, then hyper-edges from H_s can be removed (eliminating the cycles) while still ensuring that all the towers are connected.

In order to find such a solution H_s, the search begins at one of the towers (e.g., Ta), by inserting all hyper-edges connected to Ta into a queue (i.e., $Q = \{\mathbf{e} \in \mathcal{E}' : \mathbf{e} = \{\mathrm{R}i, \mathrm{T}a\}, \mathrm{R}i \in \mathcal{V}\}$), and running the function *find_solution* recursively (see Alg. 1 for the pseudocode).

The *find_solution* algorithm proceeds as follows: P contains the possible positions that the robots can be in - initially, all robots can be in all positions. Given a queue Q of hyper-edges and the first hyper-edge $\mathbf{e}$ in the queue, the algorithm attempts to use $\mathbf{e}$ and recurse, as well as not use $\mathbf{e}$ and recurse. This ensures that all combinations of using and not using hyper-edges are tested. In addition, as the algorithm recurses, Q, the queue of hyper-edges to consider, $\mathcal{V}_c$, the set of vertices (robots/towers) already connected, $\mathcal{E}_{used}$, the set of hyper-edges used, and $\mathcal{E}_{skip}$, the set of hyper-edges that were skipped, are updated.

Although it may seem that this search performs a complete search of the hyper-graph, the search tree is pruned quickly, due to the constraints in each of the hyper-edges. Thus, P becomes more and more constrained, limiting the number of hyper-edges still available for use, and reducing the search space considerably.

1) Updating Constraints: Given a set of possible positions $P = \{P_1, \dots, P_N\}$, where P_i refers to possible positions of Ri, and a hyper-edge $\mathbf{e} = \{v_1, v_2\}$, we update P such that the constraints of the hyper-edge $\mathbf{e}$ are satisfied. In order to do so, we take the intersection of the constraints of $\mathbf{e}$ and the relevant elements of P. Next, we iterate through all $\mathbf{e}' \in \mathcal{V}_s$ and further constrain P (since the reduced positions of Ri may further constrain positions of Rj through a previously-used hyper-edge $\mathbf{e}' = \{\mathrm{R}i, \mathrm{R}j\}$). After all the propagations have completed, if $\exists i$ s.t. $|P_i| = 0$, then P becomes invalid. We have the following result:

Theorem III.9. *For a problem with M towers and N robots verifying Assumptions 1 through 5, all robots will find at least one solution w.p.1.*

Proof: From Assumption 2, there is at least one configuration in which all towers are connected (a solution). The fact that the environment is bounded in the sense of Assumption 3 and that the exploration algorithm is thorough in the sense of Assumption 4 guarantees that at least one robot eventually determines a solution (the probability of this not happening goes to 0 as $t \to \infty$).

From Thm. III.6, robots synchronize their hyper-graphs when they meet. Using Assumptions 4 and 5, this implies that the network structure eventually propagates to all robots. Thus, if a solution to the connectivity problem is found by some robot Ri, then all the robots will find this solution w.p.1 in the limit, as $t \to \infty$.

As a result, the solution propagates to all robots in the limit, thus establishing the desired result. ■

Algorithm 1 find_solution($Q, P, \mathcal{V}_c, \mathcal{E}_{used}, \mathcal{E}_{skip}$)

if Q is empty **then**
  return false
end if
$\mathbf{e} =$ popQueue(Q)
$P' =$ updateConstraints($\mathbf{e}$, P)
if P' is valid **then**
  $Q' =$ updateQueue($\mathbf{e}$, Q)
  $\mathcal{V}'_c =$ addVertex($\mathbf{e}$, V_c)
  $\mathcal{E}'_{used} \leftarrow \mathcal{E}_{used} \cup \{\mathbf{e}\}$
  if $\mathcal{T} \subseteq \mathcal{V}'_c$ **then**
    $solution \leftarrow (\mathcal{V}'_c, \mathcal{E}'_{used})$
    return true
  end if
  if find_solution($Q', P', \mathcal{V}'_c, \mathcal{E}'_{used}, \mathcal{E}_{skip}$) = true **then**
    return true
  end if
end if
$\mathcal{E}'_{skip} \leftarrow \mathcal{E}_{skip} \cup \{\mathbf{e}\}$
if find_solution($Q, P, \mathcal{V}_c, \mathcal{E}_{used}, \mathcal{E}'_{skip}$) = true **then**
  return true
end if
return false

E. Converging to a Solution

In Sec. III-D, we have established that, with enough exploration, all robots eventually determine a solution. However, in environments where multiple solutions exist, it is possible that at any time step not all robots determine the same solution. Therefore, it is necessary to ensure that, in the presence of multiple solutions, all robots adopt and move to the same solution.

The process of ensuring consensus in a common solution arises from a common and deterministic solution selection mechanism. As seen before, the algorithm to find a solution (Alg. 1) is deterministic. Furthermore, since robots synchronize their hyper-graphs when they meet, connected robots will find the same solution.

Once a robot Ri has found a solution, it attempts to find its neighbors in the solution and synchronize its hyper-graph with them. After all its neighbors have synchronized their hyper-graphs, Ri heads to its solution position. Finally, after arriving at the solution position, Ri will stop moving. If at any time, a better solution is found (e.g., by receiving new information from other robots), Ri will restart its convergence process and attempt to find its neighbors again.

If a robot Ri finishes sharing its solution with its neighbors and moves to its final position while other robots are still negotiating, 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 robot Ri. At this point, they synchronize their hyper-graphs and adopt the same solution. If the solution found is different, Ri restarts its sharing process. This means that, since the number of robots is finite, they all eventually settle in one solution and move to the corresponding position. This conclusion is stated in the following result:

Theorem III.10. *For a problem with M towers and N robots verifying Assumptions 1 through 5, all robots converge to the same solution w.p.1.*

F. Algorithm for Distributed Network Connectivity

The robots do not perform map-merging, and run a fully-distributed algorithm which allows them to find and converge to a solution. A flow-chart of the algorithm is shown in Fig. 7. The fully-distributed algorithm shown in Alg. 2, which includes a state heuristic function. The robot can be in one of four states, namely *Explore*, *Share_Solution*, *Goto_Solution*, and *Stop*. Each robot Ri starts in the *Explore* state, with an empty hyper-graph. Fig. 8 shows the state transition diagram for the robots.

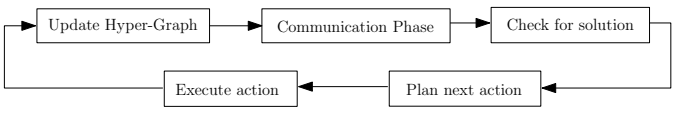

Fig. 7. Flowchart of the fully-distributed algorithm

At each time step, the robot Ri generates a list of towers $\mathcal{T}_i \subseteq \mathcal{T}$, and a list of robots $\mathcal{R}_i \subseteq \mathcal{R}$ that are in range.

Algorithm 2 Distributed Network Connectivity Algorithm

$H \leftarrow \{\}$ // H is the Hyper-Graph
$state \leftarrow$ Explore
loop
  $(\mathcal{T}_i, \mathcal{R}_i) =$ getConnections()
  // Create position label
  P$i_\alpha =$ getPositionLabel(*current coordinates*)
  // Update Hyper-Graph H with connections to towers
  for all T$a \in \mathcal{T}_i$ **do**
    addConstraint({Ta, Pi_α}, H)
  end for
  // Update Hyper-Graph H with connections to robots
  // Synchronize Hyper-Graphs
  performCommunicationPhase($\mathcal{R}_i$, H)
  // Check for solution and update state
  if graph_was_updated **then**
    $s =$ checkForSolution(H)
    if s is valid and $s \neq current\_solution$ **then**
      $current\_solution \leftarrow s$
      $sol\_posn =$ getSolutionPosition(s)
      $neighbors =$ getNeighborsToInform(s)
      $state \leftarrow$ Share_Solution
    end if
  end if
  if $state =$ Share_Solution & informed($neighbors$) **then**
    $state \leftarrow$ Goto_Solution
  else if $state =$ Goto_Solution & P$i_\alpha = sol_posn$ **then**
    $state \leftarrow$ Stop
  end if
  // Plan the next action
  $action =$ getNextAction($state$)
  // Execute the action
  executeAction($action$)
end loop

For each tower T$a \in \mathcal{T}_i$, robot Ri adds a constraint into its hyper-graph as Ri's current position label (i.e., Pi_α), the tower's ID (i.e., Ta), and the signal strength of the connection.

Ri then enters a communication phase, where it adds constraints into its hyper-graph as Ri's current position label, the position labels of the robots Ri is directly connected to ($\mathcal{R}_i$), and the signal strengths of the connections. Ri then shares and synchronizes its hyper-graph with all the robots it is directly connected to ($\mathcal{R}_i$). After this phase, all robots that are connected will have the same hyper-graph.

Following the communication phase, Ri now searches the hyper-graph for a solution if the hyper-graph was updated.

Lastly, Ri switches its internal states if necessary, based on whether a solution has been found.

IV. Planning an Action

As mentioned above, the robot can be in one of 4 states: *Explore*, *Share_Solution*, *Goto_Solution*, and *Stop* (see Fig. 8).

In the *Explore* state, the goal of the planner is to traverse the world such that the robots will eventually find a

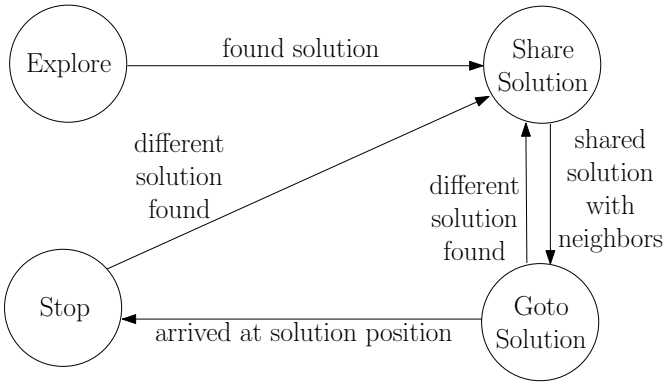

Fig. 8. State transition diagram for each robot. The robots start in the *Explore* state. When all robots are in the *Stopped* state, the solution configuration has been achieved and all towers are connected.

solution configuration s that connects all the towers. In the *Share_Solution* state, a solution s has been found, and the goal is to communicate this solution s to neighbor robots in s. In the *Goto_Solution* state, the planner has to find the shortest path from the current location to the robot's position in the solution s. Lastly, in the *Stop* state, the planner has no work to do and merely stops the robot in place.

A. Exploring the Environment

In order to find a solution s, each robot Ri has to traverse the world in such a way that the solution algorithm can find a solution as quickly as possible. We explored a number of different heuristics for this purpose:

1) 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 if n actions were available, they would each 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.

2) Coverage of the Space: The next heuristic we considered was a coverage algorithm. We adapted the node-counting algorithm described in [8]. Each robot kept a counter C_c of how many times it visited a cell c. Then, when choosing an action, it picks the adjacent cell c' such that $C_{c'}$ is the minimum among all adjacent cells. In the case where more than one cell has the minimum value, it picks randomly among the minimum cells. All cells are initialized to have a counter of 0, so unexplored cells always have priority over explored cells.

3) Weighted Exploration: This heuristic was similar to the above coverage algorithm, except that the robot uses a weighted dice to decide among its adjacent cells, instead of choosing the least-visited cell (i.e., with the minimum value). We defined a ratio γ, that represents the exploration vs exploitation probabilities. Given k_1 unexplored adjacent cells, and k_2 explored adjacent cells, it chooses to explore with $\frac{k_1\gamma}{k_1\gamma+k_2(1-\gamma)}$ probability, and exploit otherwise. If it chooses to explore, it picks one of the unexplored cells randomly. Otherwise, if it chooses to exploit, it picks an explored cell, weighted on how many times it has previously visited that cell. For each explored cell c_k and corresponding counter C_{c_k}, we define $p_k = 1 - \frac{C_{c_k} - \min_j C_{c_j}}{\max_j C_{c_j} - \min_j C_{c_j}} + \alpha$, where α is a weighting term such that the cell with the maximum counter will not have a 0 probability of being chosen. Given the p_k of the adjacent cells, the robot picks a cell c_k with a probability of $\frac{p_k}{\sum_j p_j}$.

For example, suppose the adjacent cells are $(c_1, 0), (c_2, 0), (c_3, 3), (c_4, 5), (c_5, 2)$, where each tuple represents an adjacent cell and its corresponding counter (where 0 means unexplored). The robot will choose to explore with a $\frac{2\gamma}{2\gamma+3(1-\gamma)}$ probability. If it decides to explore, it will pick either c_1 or c_2 with equal probability. If it decides to exploit, it will pick c_3 with a probability of $\frac{\frac{1}{2}+\alpha}{(\frac{1}{2}+\alpha)+(\alpha)+(1+\alpha)}$, c_4 with a probability of $\frac{\alpha}{(\frac{1}{2}+\alpha)+(\alpha)+(1+\alpha)}$ and c_5 with $\frac{1+\alpha}{(\frac{1}{2}+\alpha)+(\alpha)+(1+\alpha)}$.

4) Stay at Towers: In this heuristic, the robots had one of 2 roles: stay at an assigned tower, or avoid towers. A robot Ri is assigned the role of staying at tower Ta if in its hyper-graph, it has the most connections to Ta. Otherwise, the robot Ri assumes the avoid towers role.

In the *stay at tower* role, if the robot Ri is not currently connected to its assigned tower Ta, then it plans the shortest path (using breadth-first search) to the nearest cell that connects it to Ta (based on the map it built while exploring the world). If the robot is already connected to Ta, then it decides to explore or exploit using a weighted dice, similar to the weighted exploration heuristic above. However, in this case, it ignores all adjacent explored cells that do not have a connection to Ta. Thus, the weightage only occurs for unexplored cells, and explored cells that are known to have a connection to Ta. In this way, Ri stays close to Ta and may lose connection only if it goes to an unexplored cell that is out of Ta's range.

In the *avoid towers* role, instead of choosing between explore and exploit, the robot chooses between explore, exploit, and visiting a tower, with probabilities α, β, and $1-\alpha-\beta$ respectively. The robot chooses between these 3 options based on the number of adjacent cells that match the requirement: i.e. if there are k_1 unexplored cells, k_2 explored cells that do not have a connection to a tower, and k_3 explored cells that have a connection to a tower, then the robot chooses to explore with $\frac{k_1\alpha}{k_1\alpha+k_2\beta+k_3(1-\alpha-\beta)}$ probability, exploit with $\frac{k_2\beta}{k_1\alpha+k_2\beta+k_3(1-\alpha-\beta)}$, and visits a tower cell otherwise. If it chooses to exploit or visit a tower, then the relevant cells are selected probabilistically based on their counter values, similar to the weighted exploration heuristic.

By using the heuristic, the robots that do not have an assigned tower tend to visit areas that have no connections to any tower, and explore new regions. This allows new towers to be discovered quickly, as well as connections to be found between towers. We experiment on this heuristic in detail in Sec. V.

V. Experimental Results

A. Setup

We created a simulator in Java that models a discrete 2D world, which allows horizontal and vertical walls (in between the discrete cells) to be placed anywhere in the world. The

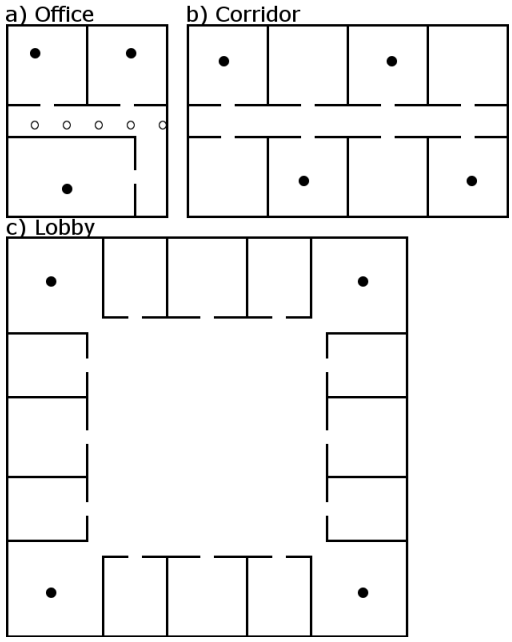

Fig. 9. Representative worlds that were experimented on: a) Office World b) Corridor World, c) Lobby World. Black lines indicate walls/obstacles, filled circles represent towers, and hollow circles (only in Office World) show the fixed initial configuration of robots. Corridor and Lobby Worlds had random initial configurations of robots.

simulator calculates signal strength between any two cells in the world, based on an exponential decay rate, as well as degradation from the walls. We did not simulate inteference between robots, or reflection of signals from the walls; the underlying algorithm would not be significantly affected even if the signal strength calculations were different. In this 2D world, each robot had 4 possible actions, namely to move north, south, east, or west. In the event that an action would cause the robot to hit a wall, the action would fail and the robot would stay where it originally was; otherwise the robot would move in the direction specified.

We implemented our algorithm as described in Sec. III, as well as the different heuristics described in Sec. IV-A. In addition, we created 3 different scenarios: an Office World, a Corridor World, and a Lobby World (see Fig. 9).

1) Office World: The Office World was 20×24 cells in size, and contained 2 small offices on the top, and a larger office at the bottom left (see Fig. 9a). An L-shaped corridor ran in between the top offices and the bottom. A tower was placed inside each office, at a distance such that they could not communicate directly.

Due to the small size of the Office World, we fixed the number of robots to 5 and selected initial starting positions of the robots. This world was designed as a proof-of-concept of the algorithm, as well as to compare the performance of the different heuristics given a fixed initial state.

2) Corridor World: The Corridor World was 40×24 cells in size, and contained 8 offices. 4 offices were arranged horizontally on the top of the world, and the other 4 were arranged horizontally at the bottom (see Fig. 9b). A long corridor ran in between the top row of offices and the bottom, and 4 towers were placed in a zigzag fashion in the offices.

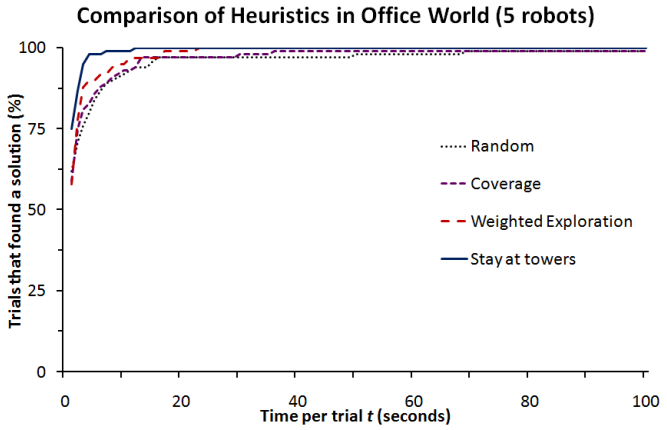

Fig. 10. Percentage of trials that found a solution in t seconds or less in the Office World with 5 robots.

The Corridor World provided a realistic depiction of many corridors in university hallways which are flanked by offices on both sides.

3) Lobby World: The Lobby World was 50×50 cells in size, and contained a large lobby in the middle (30×30). Around the lobby were 12 small offices - 3 on each side, as well as 4 slightly larger offices located at each corner of the lobby (see Fig. 9c). We placed 4 towers in this world, 1 in each of the corner offices.

The Lobby World provided a depiction of a large lobby area, that is connected to many small rooms. This world was designed not only to simulate real-world situations, but also to provide a worst-case scenario for our algorithm. By having a large open area, the robots would be able to move around freely and create a dense hyper-graph, where each robot vertex would have every other robot vertex as a neighbor. This could cause the search for solution to take extremely long, and so we wanted to test the effectiveness of our algorithm in such a scenario.

B. Comparing the Heuristics

We ran the different heuristics in the Office World scenario (which had 5 robots in a fixed initial configuration), with 100 trials per heuristic. In each trial, we measured how much time the algorithm took in order to find a solution configuration. We then compared the percentage of trials that found a solution in t seconds on less; Fig. 10 shows the comparison of the different heuristics in the Office World. All the heuristics performed well in this scenario, with the *stay at towers* heuristic performing slightly better than the others. It found 100% of the solutions within 12s of execution, compared to 23s of the *weighted exploration* heuristic. The *coverage* and *random* heuristics found 99% of the solutions within 36s and 69s respectively.

We believed that all the heuristics performed relatively well in the Office World scenario due to the fact that many solution configurations existed, and that the robots began in a configuration that was close to many solutions. Therefore, we performed similar experiments on the Corridor and Lobby

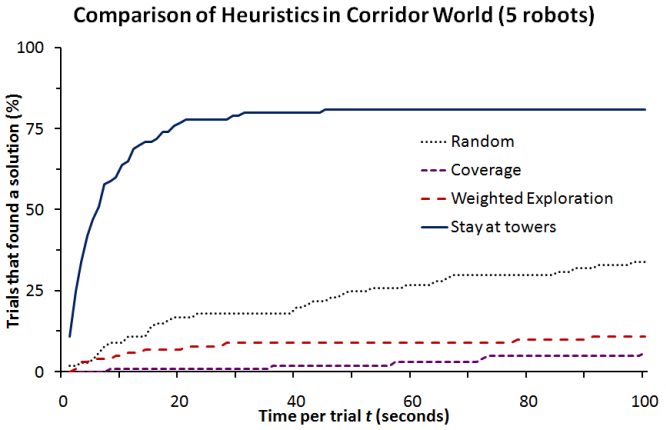

Fig. 11. Percentage of trials that found a solution in t seconds or less in the Corridor World with 5 robots.

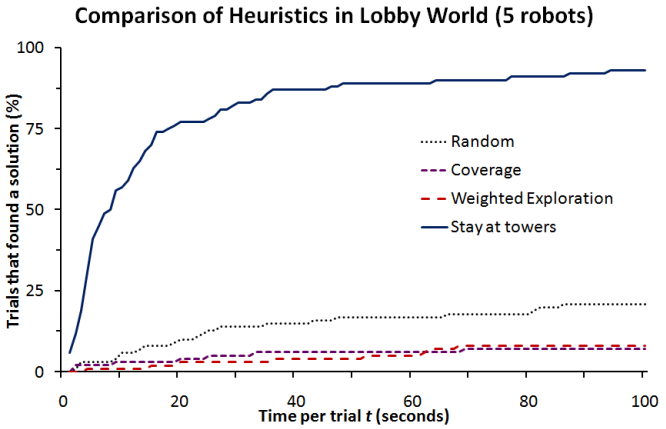

Fig. 12. Percentage of trials that found a solution in t seconds or less in the Lobby World with 5 robots.

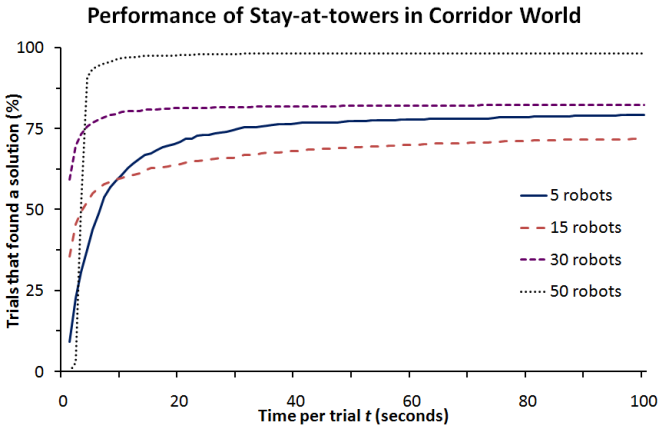

Fig. 13. Percentage of trials that found a solution in t seconds or less in Corridor World with varying number of robots running *stay at towers* heuristic.

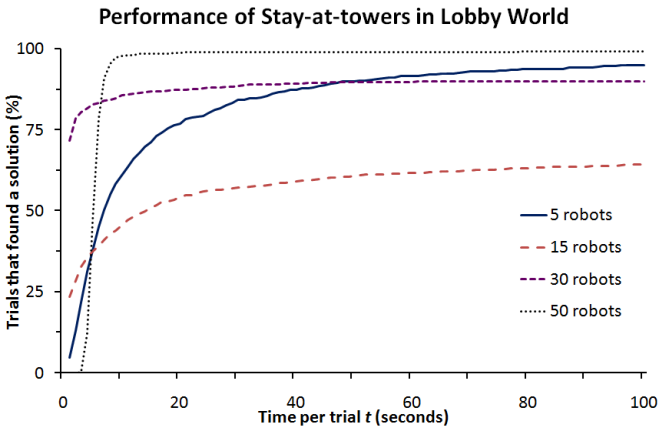

Fig. 14. Percentage of trials that found a solution in t seconds or less in Lobby World with varying number of robots running *stay at towers* heuristic.

Worlds, with 5 robots in each case, and a random initial configuration for the robots in each trial.

As shown in Figs. 11 and 12, the *stay at towers* heuristic outperformed all other heuristics by a large margin. It is interesting to note that the *coverage* and *weighted exploration* heuristics performed more poorly than the *random* heuristic. This is because in a large world, a large emphasis on exploration reduces the chance that robots will meet in a "useful" configuration (where useful refers to a partial configuration that can later be used as part of a solution), since they rarely return to previously visited cells. As such, it is difficult for the robot team to find a solution configuration, even after they have individually explored the entire environment.

After these experiments, we concluded that the *stay at tower* heuristic was the most promising, as it performed well in all the scenarios. We then tested this heuristic extensively in the Corridor and Lobby Worlds, as described below.

C. Further Experiments for Corridor and Lobby

We used the *stay at towers* heuristic exclusively for all the experiments described below, as it was the most promising heuristic (see Sec. V-B). In each experiment, we fixed the number of robots and ran 1000 trials. The initial configuration of the robots was randomly selected in each trial, and we measured how long the algorithm took to find a solution. We then compared the percentage of trials that found a solution in a limited amount of time t or less.

In Fig. 13, we show the results in the Corridor World, with the number of robots varying from 5 to 50. Similarly, in Fig. 14, we show the results in the Lobby World, also with the number of robots varying from 5 to 50. In Fig. 15, we show some random initial configurations of 5 robots and the corresponding solutions found in the Corridor World; in Fig. 16, we do the same for the Lobby World.

In both the Corridor World (Fig. 13) and Lobby World (Fig. 14), increasing the number of robots from 5 to 15 robots actually decreases the percentage of trials that find solutions. This decrease in performance is because the graph grows polynomially with the increase in the number of robots, but searching for a solution in the graph can take exponentially longer.

However, it can be seen in both figures that increasing the number of robots beyond 15 tends to increase the percentage of solutions found, because the number of goal states increases with the number of robots. Thus, when there are many robots in the world, the initial configuration of the robots has a large

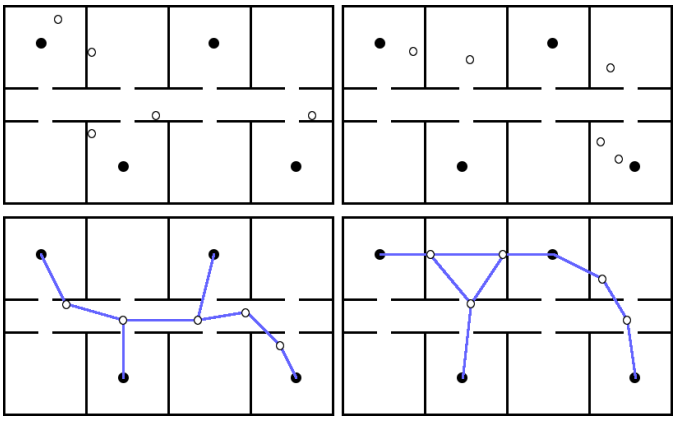

Fig. 15. The top row shows random initial configurations of 5 robots (hollow circles) and 4 towers (filled circles) in Corridor World; the bottom row shows the corresponding solutions found. The black lines represent walls/obstacles, and the blue lines indicate connections between robots and towers.

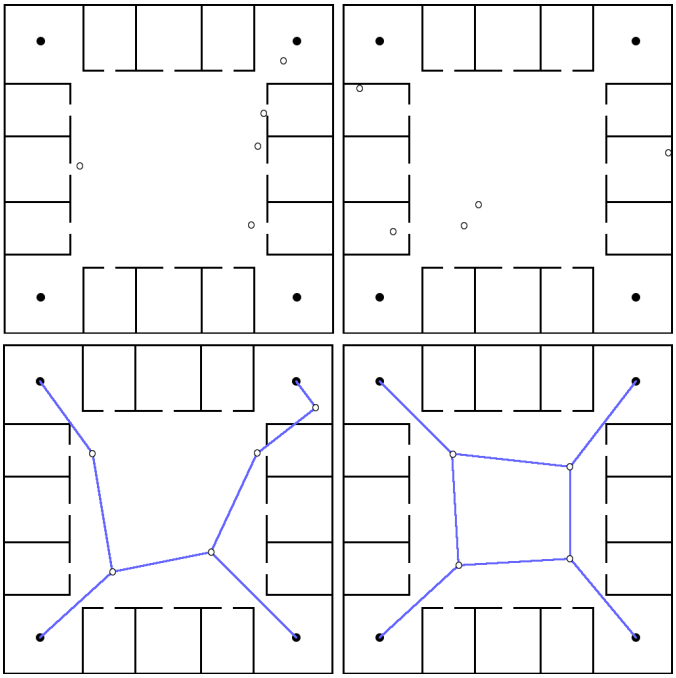

Fig. 16. The top row shows random initial configurations of 5 robots (hollow circles) and 4 towers (filled circles) in Lobby World; the bottom row shows the corresponding solutions found. The black lines represent walls/obstacles, and the blue lines indicate connections between robots and towers. Notice that the solution shown in the bottom-right image only uses 4 robots.

probability of being a goal state (or close to one), so it takes very little time to find a solution configuration.

VI. Conclusion

We addressed the challenging problem of coordinating multiple robots to position themselves in an unknown environment to enable connectivity of a static set of communication nodes. One of the main technical contributions of the algorithm is its fully distributed nature. Robots explore the environment simultaneously and separately, each maintaining a record of the connectivity information associated with its own visited positions in its own frame of reference. When within range, robots share the connectivity information corresponding to their positions with no need for map merging. The robots can separately determine that there is a solution at some visited configuration. Another technical contribution is the hyper-graph data structure that is used by the algorithm to share information and represent solutions. The hyper-graph abstracts the physical details of the world, while keeping enough information such that a solution can be found efficiently without map-merging.

The fact that our algorithm is fully distributed and requires no map-merging is, in our view, very interesting. It allows the robots to use different granularity and even different representations of the environment, i.e., one robot may use a grid-based representation, while another may use some topological map. Any additional information shared among the robots, if common references are available, can easily be accomodated by the algorithm and could be used in the distributed exploration. It is worth mentioning that the process by which the network structure is tracked and maintained allows the robots to completely separate the problem of navigation from the problem of tracking the network. Therefore the existence of unknown obstacles in the environment has no impact on the performance guarantees of the algorithm.

The empirical results with our algorithm also point out several interesting directions for future work. We are currently exploring other heuristics, as well as methods to determine the number of robots to deploy in a scenario.

References

[1] A. Howard, S. Siddiqi, and G. Sukhatme, "An experimental study of localization using wireless ethernet," in *Proc. 4th Int. Conf. Field and Service Robotics*, 2003.

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

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

[4] N. Priyantha, A. Chakraborty, and H. Balakrishnan, "The CRICKET location-support system," in *Proc. 6th ACM Conf. Mobile Computing and Networking*, 2000, pp. 32–43.

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

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

[7] A. Howard, L. Parker, and G. Sukhatme, "Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection," *International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 431–447, 2006.

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

Somchaya Liemhetcharat is currently pursuing his Ph.D. in robotics at Carnegie Mellon University (Pittsburgh, PA, USA). He received his B.S in computer science at Carnegie Mellon University in 2007. He is interested in multi-robot coordination, and has participated in robotics competitions such as RoboCup.

Manuela M. Veloso is Herbert A. Simon Professor of Computer Science at Carnegie Mellon University. She researches in artificial intelligence and robotics, in the areas of planning and learning for single and multirobot teams in uncertain, dynamic, and adversarial environments. She directs the CORAL research laboratory (www.cs.cmu.edu/ coral) for research with her students on agents that Collaborate, Observe, Reason, Act, and Learn. Prof. Veloso is a Fellow of the American Association of Artificial Intelligence, and the President of the RoboCup Federation. She was recently awarded the 2009 ACM/SIGART Autonomous Agents Research Award. Prof. Veloso is the author of a book on ”Planning by Analogical Reasoning” and editor of several other books. She is also an author of over 200 journal articles and conference papers.

Francisco S. Melo received his Ph.D. in Electrical and Computer Engineering at Instituto Superior Tcnico, in Lisbon, Portugal. He held previous appointments at the Computer Vision Lab, Institute for Systems and Robotics (Lisbon, Portugal) and at the Computer Science Department, Carnegie Mellon University (Pittsburgh, PA, USA). From June 2009 he is an Assistant Researcher at INESC-ID (Lisbon, Portugal). His research addresses problems within developmental robotics, reinforcement learning, planning under uncertainty, multiagent and multi-robot systems, and sensor networks.

Daniel Borrajo is a Professor of Computer Science at Universidad Carlos III de Madrid. He received his Ph.D. in Computer Science in 1990 and B.S. in Computer Science both at Universidad Politcnica de Madrid. He has published over 140 journal and conference papers mainly in the fields of problem solving methods (search, planning and game playing) and machine learning.