The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

Network-Guided Multi-Robot Path Planning in Discrete Representations

Ryan Luna and Kostas E. Bekris

Abstract— **This work deals with problems where multiple robots move on a roadmap guided by wireless nodes that form a communication network. The nodes compute paths for the robots within their communication range given information about robots only in their vicinity and communicating only with neighbors. The objective is to compute paths that are collision-free, minimize the occurrence of deadlocks, as well as the time it takes to reach the robots' goals. This paper formulates this challenge as a distributed constraint optimization problem. This formulation lends itself to a message-passing solution that guarantees collision-avoidance despite only local knowledge of the world by the network nodes. Simulations on benchmarks that cannot be solved with coupled or simple decoupled schemes are used to evaluate parameters and study the scalability, path quality and computational overhead of the approach.**

I. Introduction

This paper discusses an interesting variant of path coordination problems, where a static wireless network is responsible for the high-level path planning of multiple robots (see Figure 4). In this setup, the robots can divert their computation to tasks such as local obstacle avoidance and localization. This is beneficial in cases where robots are resource constrained and have a limited time to compute a path, such as planetary exploration [1] or warehouse management [2]. Moreover, the network may compute better quality paths for the robots because it has access to more information, as in the scenario of Figure 2(a). Similarly, static nodes can often take advantage of wired communication for coordination, which is more reliable compared to wireless communication. This could be the case in future cyber-physical systems for transportation, which can employ a networked infrastructure to guide automobiles in an urban environment to control traffic.

A. Background

Multi-robot path planning can be solved either with a coupled approach, where the robots are viewed as a single high degrees-of-freedom (DOF) system, or through a decoupled approach, where paths are computed individually for robots and then conflicts are resolved. The coupled approach is complete if combined with a complete planner but has exponential dependency on the number of robots. This has led to methodologies that reduce the size of the search space while offering completeness [3], [4], [5]. Prioritized schemes

This work is supported by NSF CNS 0932423 and a graduate fellowship to Mr. Ryan Luna by the Nevada NASA Space Grant Consortium. The computational experiments were executed on equipment obtained with internal funds of the University of Nevada, Reno.

The authors are with the Computer Science and Engineering Department at the University of Nevada, Reno, NV, USA (`rluna@cse.unr.edu`, `bekris@cse.unr.edu`).

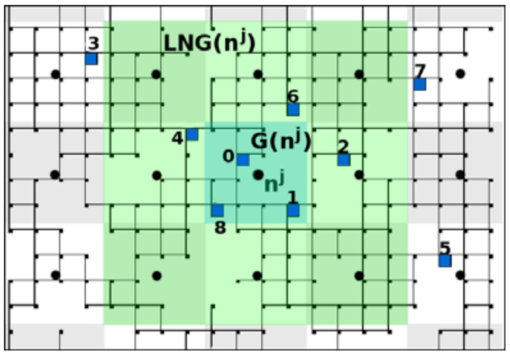

Fig. 1. Network-guided multi-robot path planning on a graph. Agents are squares. Network node n^j can directly observe agents within its local subgraph $G(n^j)$. Through communication with neighbors, n^j can also access information regarding its Local Neighborhood Graph $\mathcal{LNG}(n^j)$.

are a form of *decoupled planning* relevant in discrete representations. They compute paths sequentially for different robots in order of priority [6]. While prioritized schemes can solve problems faster than coupled alternatives, they are inherently incomplete [7]. Thus, there have been techniques which aim to increase the reliability of decoupled planners [8], [9], [10], [11].

Network-guided navigation has been used to guide robots without the need for a map or localization [12]. The focus is often on the distributed computation of paths over a network by encoding dangerous regions as obstacles and employing potential functions [13]. Flooding the network with messages is problematic and techniques try to minimize this effect by computing approximate shortest paths over a skeleton graph of the network [14].

B. Challenges in Distributing Multi-Robot Path Planners

Distributing the operation of coupled planners over a sensor network is not realistic, because they require the collection of global information from all the network nodes. Moreover, Table I shows the time it takes for a coupled, A*-based planner to solve the grid-based problem of Figure 2(b). The time becomes quickly prohibitive even for just 6 agents, (more than 4 hours). While existing work has shown how to optimally decouple multi-robot problems into sequential plans of smaller composite robots [5], the problem in Figure 2(b) cannot be decoupled to smaller problems. In the scenarios studied in this work, it is very often the case that within the vicinity of a sensor node there are more than 6 agents whose paths cannot be decoupled. Alternative coupled planners work only on certain types of graphs [4]. Thus, this paper focuses on decoupled solutions. Nevertheless, simple prioritized schemes are not a solution either. Figure 2(c) gives a simple case where a prioritized solution will

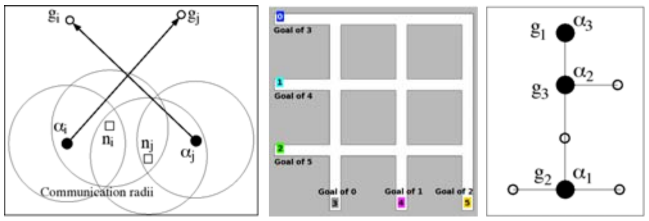

Fig. 2. (a) The paths of α_i and α_j towards g_i and g_j intersect. Nodes n_i and n_j can detect this earlier than the agents. (b) The environment where the coupled planner was tested. The proposed technique can solve this problem many orders of magnitude faster. (c) A prioritized technique considering only optimal paths for agents would fail in this case.

Number of Agents	2	3	4	5	6
Time (s)	0.09	0.21	307.557	1010.94	15085.6

TABLE I
COMPUTATION TIME FOR A COUPLED SOLUTION

fail. Thus, it is necessary to consider more sophisticated decoupled planners, where coordination arises naturally from the constraints imposed on the robots, provides collision avoidance and minimizes the occurrence of deadlocks.

C. Proposed Approach and Contribution

The proposed approach is an online, distributed solution for network-guided, multi-robot path planning where no priorities are used and the sensors have only local information. The sensors first compute alternative local paths for robots in their vicinity and then coordinate to find an assignment of paths to robots. This is achieved through a Distributed Constraint Optimization (DCO) formulation. Coordination Graphs [15], [16], [10], [11] have been used to describe the interaction of agents in DCO, which lead to message-passing solutions with asynchronous belief propagation [16], [17]. This paper follows this formulation but there are also alternative schemes for DCO, such as ADOPT [18], [19] and action-based solutions [20]. The technique also provides collision avoidance guarantees. Experimental results confirm that the proposed approach achieves collision-free multi-robot path planning for instances where the coupled solution is infeasible and the prioritized schemes quickly result in deadlocks.

To the best of the author's knowledge, the paper's problem has not been studied in the past. Network-based navigation [12], [13], [14] focuses on a single agent and does not consider future interactions of multiple agents. Similarly, multi-robot path planning does not consider the network-related constraints [3], [4], [5], [7], [8], [9]. In contrast to prioritized scheme, the DCO formulation allows the assignment of paths to robots that are locally suboptimal but which allow the global existence of a solution to other robots. The algorithm also lends itself to a message-passing protocol, which is appropriate for a network-based problem.

This work studies an abstract version of the problem on a discrete representation, without parameters such as robot dynamics, bandwidth limitations and localization errors. This abstraction allows this paper to focus on path planning, to compare against optimal single-agent paths and formulate the requirements for collision avoidance.

II. Problem Definition

Consider a graph $G(V, E)$ embedded in the obstacle-free part of a workspace $\mathcal{R}$. In this paper $\mathcal{R} \subset \mathbb{R}^2$, but the treatment extends to higher-dimensional problems. G has vertices that are collision-free positions in $\mathcal{R}$ and edges that correspond to collision-free paths between vertices. There are also agents $\mathcal{A} = \{\alpha_1, \ldots, \alpha_{|\mathcal{A}|}\}$, which occupy vertices of the roadmap and move along the roadmap's edges. The vertex occupied by agent α_i at time t will be denoted as $v_i(t)$. The time it takes to traverse edges is the same along the graph. Two agents cannot occupy the same vertex or the same edge simultaneously. Each agent has a goal $g_i \in V$.

$\mathcal{R}$ contains static wireless nodes $\mathcal{N} = \{n^1, \ldots, n^{|\mathcal{N}|}\}$. The nodes can communicate among themselves and with the agents as long as they are within a predefined radius. The nodes are placed in such a way so that they form a connected network. Moreover, an agent can communicate with at least one node from every vertex of G. Nodes are aware of the graph's structure but can detect only the location of agents within their communication radius. Agents are also referred here as robots, while nodes are also called sensors.

An agent does not compute its own path but receives commands from the nodes in order to reach its goal. A path π_i for agent α_i is a sequence of vertices: $\pi_i(0 : t_m) = (v_i(t_0), \ldots, v_i(t_m))$. Given a path, the edge $e_i(t_j)$ is the edge that has to be traversed in order to go from $v_i(t_j)$ to $v_i(t_{j+1})$. Two paths for two agents α_i and α_j are *compatible*: $\pi_i(0 : t_m) \asymp \pi_j(0 : t_m)$, as long as:

$$\forall\ 0 \le t \le t_m : v_i(t) \neq v_j(t) \wedge$$
$$\forall\ 0 \le t \le t_m - 1 : e_i(t) \neq e_j(t)$$

If one path is shorter than the other, then it can be expanded by repeating its last vertex. A *solution path* $\pi_i(0 : t_m)$ for α_i is one where $v_i(t_m) = g_i$.

Problem: *Given the above described nodes $\mathcal{N}$ and agents $\mathcal{A}$, an initial placement of agents $\{v_1(0), \ldots, v_{|\mathcal{A}|}\}$ and goals $\{g_1, \ldots, g_{|\mathcal{A}|}\}$ on the graph $G(V, E)$, compute over the network paths $\{\pi_1(0 : t_m), \ldots, \pi_{|\mathcal{A}|}(0 : t_m)\}$ so that*:

- *Each path is a solution path for the corresponding agent.*
- *Each pair of paths is compatible:*
  $\pi_i(0 : t_m) \asymp \pi_j(0 : t_m),\ \forall\ i, j \in [0, |\mathcal{A}|], i \neq j.$
- *And t_m is minimized.*

III. High-Level Description of The Approach

The approach first partitions the graph into subgraphs $G(n^j)$ for each node n^j during an offline phase. A point along G is assigned to subgraph $G(n^j)$ as long as n^j is its closest sensor according to Euclidean distance. This roadmap partition is available to the sensors before the online operation of the algorithm. It is similarly possible to partition the agents at each time step into subsets $\mathcal{A}(n^j)(t)$ based on the vertex they occupy: $\alpha_i \in \mathcal{A}(n^j)(t)$ iff $v_i(t) \in G(n^j)$.

During the online operation and as multiple agents enter and exit a node's subgraph $G(n^j)$, previously computed paths become invalid and they have to be recomputed. This suggests a replanning, partial solution, where nodes compute

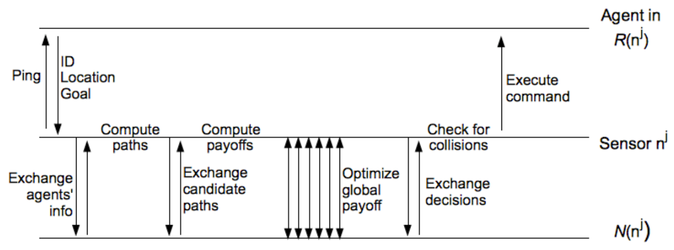

Fig. 3. The communication between sensor n^j, agents in $\mathcal{A}(n^j)$ and neighboring sensors $\mathcal{N}(n^j)$. The sensors first exchange information regarding the agents in their local neighborhood graphs. Then they compute paths for these agents and exchange them. For every two paths of different agents, the sensors compute the pairwise payoff expressing whether the two paths will lead to collisions or not. Once payoffs have been computed, the sensors enter an asynchronous and distributed optimization protocol to make assignments of paths to agents that minimize conflicts. There is a final check in the algorithm that guarantees the avoidance of collisions.

paths for agents at periodical intervals, within the time it takes agents to traverse an edge. One such interval will be referred to here as a cycle. During cycle $(t-1:t)$, nodes compute paths that the agents are going to execute during cycle $(t:t+1)$. Figure 3 explains the protocol for agent-sensor communication during a cycle:

- Nodes ping agents within their communication range.
- Agents respond with their ids, coordinates and goals.
- Nodes coordinate to assign paths.
- Each n^j transmits to $\mathcal{A}(n^j)$ the action for the next cycle.

For the coordination step the sensors are limited to exchange messages only with their neighboring nodes. Each sensor n^j generates a set of candidate paths for all the agents $\mathcal{A}(n^j)$ in its subgraph and exchanges such information with its neighbors. The proposed approach views these candidate paths as discrete action sets in a Distributed Constraint Optimization (DCO) problem, so as to optimize a global objective function. The global objective function is defined in such a way so that when optimized, the agents do not collide and follow short paths towards their goal. Given the available amount of time, the sensors select the action that is returned as the best by the optimization protocol. The sensors implement a final check to guarantee that no collisions arise from the current path assignment and then they communicate the resulting action to the agents.

IV. Implementation Specifics

This section details how the sensors compute the candidate paths for the robots, the formulation of the DCO protocol and how to guarantee collision avoidance.

A. DCO Coordination Graph Formulation

Given the agents $\mathcal{A} = \{\alpha_1, \ldots, \alpha_{|\mathcal{A}|}\}$ at states $\{v_1, \ldots, v_{|\mathcal{A}|}\}$, the objective is to select an optimal joint assignment of paths $\{\pi_1, \ldots, \pi_{|\mathcal{A}|}\}$ that maximizes a global utility function Q decomposed into local utility functions:

$$Q(\mathcal{A}) = \sum_i Q_i(V(\mathcal{A}), \Pi(\mathcal{A}))$$

Q_i expresses the individual utility of α_i and depends upon α_i's interactions with other agents. Often, however, an agent depends only on a small subset of $\mathcal{A}$. An approach that exploits such dependencies involves a coordination graph $CG(V^C, E^C)$. In CG a vertex represents an agent and an edge (i,j) represents the fact that α_i and α_j are interacting. Then the global utility function can be decomposed as follows:

$$Q(\mathcal{A}) = \sum_{\forall i \in [0,|\mathcal{A}|]} f_i(\pi_i) + \sum_{(i,j)\in E^C} f_{ij}(\pi_i, \pi_j) \qquad (1)$$

where f_i is a unary payoff vector based for the paths α_i has available and f_{ij} is a pairwise payoff matrix that expresses the interactions between the paths of α_i and α_j.

To define the coordination graph at each cycle, this approach takes into account the communication and information constraints imposed by the network. While n^j has information regarding only its communication radius, there are also interactions between agents in different subgraphs. But agents that are far away one from the other have limited interaction. Moreover, each sensor should communicate only with neighboring sensors, denoted as $\mathcal{N}(n^j)$. Thus, two agents share an edge in CG only if they are both located in neighboring subgraphs. This gives rise to the definition of the Local Neighborhood Graph $\mathcal{LNG}(n^j)$ of a sensor node:

$$\mathcal{LNG}(n^j) = G(n^j) \cup \Big(\bigcup_{\forall n \in \mathcal{N}(n^j)} G(n) \Big)$$

which corresponds to the union of $G(n^j)$ and neighboring subgraphs, and is the subset of the original graph that each sensor node has access to. This means that neighboring sensors must exchange information regarding the agents within their subgraphs. Given the above definitions, a sensor n^j implements the following procedure:

1. Exchange "ids, current locations, and goals" with $N(n^h)$ for all agents in $\mathcal{LNG}(n^j)$.
2. Generate the set of candidate paths $\pi_i, \forall\ \alpha_i \in \mathcal{A}(n^j)$.
3. Exchange the paths (π_i's) with $N(n^h)$ for all agents in $\mathcal{LNG}(n^j)$.
4. Participate in the distributed optimization of Q
5. Exchange with $N(n^h)$ the paths p_i^* assigned by the optimization to agents within the $\mathcal{LNG}(n^j)$.
6. Check if paths p_i^* lead to collision. If they do, enforce collision avoidance in the final paths transmitted to $\mathcal{A}(n^j)$.

B. Generating Candidate Paths for the Robots

In order to construct a complete set of paths π_i for each agent, it would be necessary during each cycle to compute all the paths from its current location to its goal by applying the Dual Dijkstra algorithm [21]. Unfortunately, this is prohibitively expensive even for relatively small graphs. Thus, the proposed approach selects a set of plans that locally provide a variety of choices for each agent. These plans are computed by running the A* algorithm and they correspond to the shortest path to the goal via all the leaves of the subgraph $\mathcal{LNG}(n^j)$. If the goal g_i is contained in $\mathcal{LNG}(n^j)$, then the shortest path to g_i is also added to this set. Otherwise the shortest path to g_i goes through one of the leaves of the subgraph $\mathcal{LNG}(n^j)$ and is already in π_i. The "zero" path, which corresponds to the agent remaining in the same position indefinitely, is also added to the set. While undesirable, this path should be included so as to provide alternatives to other agents.

C. Message-Passing Protocol for Optimization

The *unary payoff function* $f_i(\pi_i)$ stores the utilities of paths for agent α_i. The shorter the path to the goal, the higher its utility. For example, the utility can be computed as follows: $C - dt$, where C is an estimate of the maximum path length in $\mathcal{R}$ and dt is the path's length. The "zero" path, as an undesirable solution, gets a payoff of 0. As is typically done in implementations of belief propagation, a small noise value is added to all payoffs to avoid the creation of multiple local maxima in Q. Furthermore, to avoid paths that are forcing the agent to backtrack along its current path, the payoffs of all such paths are further penalized. Experiments show that this penalization is beneficial to the algorithm's performance.

The *pairwise payoff function* $f_{ij}(\pi_i, \pi_j)$ expresses the pair-wise interactions between the paths of different agents. For a pair of paths $\pi_i(dt_i)$ and $\pi_j(dt_j)$ there are two cases:

- If the paths are compatible $\pi_i(dt_i) \asymp \pi_j(dt_j)$ (i.e., not colliding), the utility is the sum of the unary payoffs.
- If the paths collide during the next cycle, the utility is the maximum negative number.

Given the above payoff functions, every path assignment to robots that contains even one imminent collision corresponds to global objective function Q getting the maximum negative value. The assignment where all agents do not move corresponds to $Q = 0$. If at least one agent moves towards the goal, then the objective function will be positive. The optimization of Q will promote the selection of short paths.

The optimization of the global objective function can be achieved by a protocol that is analogous to the belief propagation algorithm in Bayesian networks, and which operates by iteratively sending messages $\mu_{ij}(\pi_j)$, between neighboring agents i and j in CG. Each sensor n^j produces the following messages: For all agents $\alpha_i \in \mathcal{A}(n^j)$ and for all neighboring agents α_j of α_i in the $\mathcal{LNG}(n^j)$, it has to compute the message $\mu_{ij}(\pi_j)$:

$$max_{\pi_i}\{f_i(\pi_i) + f_{ij}(\pi_i, \pi_j) + \sum_{\alpha_k \in \mathcal{LNG}(n^j)/\ j} \mu_{ki}(\pi(\alpha_i))\}$$

The message $\mu_{ij}(\pi_j)$ is an anytime estimate of the maximum payoff that α_i can achieve for path π_j of agent α_j. It is computed by maximizing (over all paths of α_i) the sum of f_i and f_{ij} and all messages for α_i except that from α_j. If α_i and α_j are both in $\mathcal{A}(n^j)$, then the sensor can internally update this message without communication.

The sensors exchange messages until they converge. If the coordination graph CG happens to be a tree, then convergence to a fixed point is guaranteed in a finite number of steps. In graphs with cycles, there are no guarantees that the algorithm converges. In practice, however, it has been shown that the algorithm operates effectively even in graphs with cycles [16]. Here, it is sufficient if every sensor has an internal clock that marks the exhaustion of the available time for optimization at which point the sensor selects the best path: $\pi_i^* = argmax(g_i(\pi_i))$, where $g_i(\pi_i) = f_i(\alpha_i) + \sum_{\alpha_k \in \mathcal{LNG}(n^j)} \mu_{ik}(\pi_i)$. The function $g_i(\pi_i)$ can be computed internally without communication.

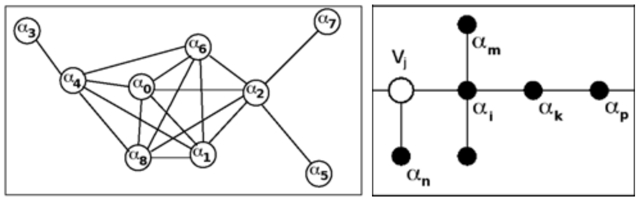

Fig. 4. (left) The coordination graph CG that corresponds to the problem of Figure 1. Agents a_0, a_8 and a_1 belong to the subgraph of node n^j but they may also interact with other agents in the Local Neighborhood Graph of n^j. (a_4, a_6, a_2) (right) Consider this roadmap where a_i, a_k, a_m, a_n, a_p are agents. If both agents α_i and α_n decide to move to node v_j, there will be a collision. The node guiding α_i can decide to keep the agent in its current location. But then this will effect the current choices of α_m, α_k and α_p.

D. Guaranteeing Collision Avoidance

The final path assignment $\{\pi_1^*, \ldots, \pi_{|\mathcal{A}|}^*\}$ may still contain incompatible paths, since the message-passing protocol is not guaranteed convergence. To detect such an issue, neighboring nodes must exchange the selected paths π_i^*. To resolve incompatibilities, it might appear that forcing one, or all, of the agents involved in the event to stop is sufficient but this is not true. If an agent α_i stops, then another agent α_k, one not involved in the incompatibility, may have been assigned an action that takes it through v_i. Therefore, if α_i has to stop, then α_k must also stop and this can have a chain reaction over all the agents and throughout the network. This means that a sensor which detects an incompatibility has to inform its neighbors which can lead to a flooding of the network.

Thus, collision avoidance has to be guaranteed through local decisions and without any need for communication. This can be achieved, as long as forcing an agent to stop is guaranteed to raise no conflicts. This means that other agents should not plan through the current position of an agent. In particular, if agent α_i occupies vertex v_i and agent α_k occupies neighboring vertex v_k, then no path in the set π_k is allowed to have as its first vertex v_i. Now, each agent α_i can safely be stopped without its guiding sensor having to inform any neighbor. If the best path of α_k goes through v_i, then a stop action should be added in the beginning of this plan.

This solution, however, will often cause agents to stop. To reduce this undesirable effect the following steps can be executed. Node n^j identifies agents α_k where their first step is to stop and the second step is to move to a vertex where another agent α_i is currently located. Such paths can arise from the introduction of the safety rule described above. If agent α_i actually departs vertex v_j, then α_k does not have to stop and can accelerate the execution of its path by moving directly to v_i. However, multiple agents might be in the same situation as α_k, waiting for α_i to move out of v_i. In this case a priority has to be used to decide which agent gets v_i. This improvement can also have a chaining effect if multiple agents are in consecutive vertices. The associated implementation in this paper ignores this chaining effect, as this is introduced only for efficiency and is not needed for collision avoidance.

V. Experiments

To show the feasibility of network-guided multi-robot path planning, a series of experiments is conducted to evaluate different parameters for the approach, solution quality, computational efficiency, and scalability.

A. Setup

- Hardware: Parallel computing cluster made of Sun Fire X4100 M2 Nodes. Each node has two quad-core 2.6 GHz CPUs and 8GB of RAM.
- Software: The simulation is implemented using C++ for the sensor network and robot processes. All interprocess communication utilizes the Message Passing Interface standard (MPI), which guarantees lossless and in-order delivery of messages. Results are simulated using a Java based visualization to ensure the proper behavior of sensors and robots in the environment.
- Each node in the sensor network is simulated with its own process space and dedicated processing core in the parallel cluster. Because of the relatively low computational overhead for robots, all robots are simulated using a single process and core in the cluster.

B. Implementation Specifics

To evaluate the approach, a series of experiments was performed using three different environments, shown in Figures 1 and 5.

i) Sparse, Urban Roadmap (Figure 5(a))
- 111 vertices and 113 directed edges

ii) Dense grid roadmap (Figure 1(a))
- 329 vertices and 1026 undirected edges

iii) Sparse, random roadmap (Figure 5(b))
- 217 vertices and 442 undirected edges

All simulations utilize an equally spaced static sensor network that results in uniform coverage of the space. The subgraph $G(n^j)$ of each sensor node n^j is computed as the set of vertices that are closest in terms of Euclidean distance to the node. For each of the three roadmaps, a set of entry and exit vertices is defined for all agents. In order to make the problem more challenging, constraints are placed on the selection of these points so that robot interactions are likely to occur during the simulation. In the sparse urban roadmap and the dense grid roadmap, start vertices are chosen along the boundary of the roadmap. The exit vertex for each robot is chosen so that it lies along a different bounding edge of the roadmap than the start vertex. For the sparse random roadmap, there are 12 predefined vertices for robot entry and exit; 6 on the west side and 6 on the east side. Each robot is assigned one start vertex, with its corresponding goal vertex lying on the opposite side of the environment. With these conflicting start and goal assignments, robots have to coordinate their paths, otherwise deadlocks will occur.

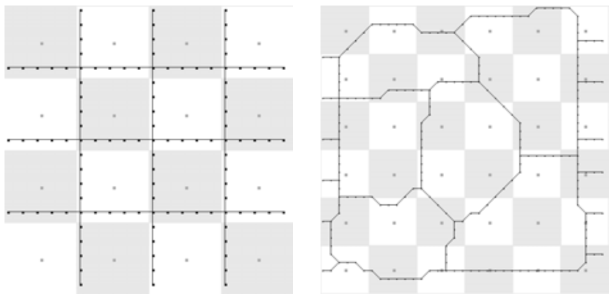

Fig. 5. (a) The sparse, urban roadmap with a graphical representation of the coverage of a 16 sensor configuration. (b) The sparse random roadmap with a graphical representation of the coverage of a 36 sensor configuration.

When a robot enters the environment, it must ensure that its start vertex is unoccupied at entry time. If the start vertex for a robot is occupied when a robot is to enter, a queue is formed for that vertex. As more robots attempt to enter the environment at the same point, they will enter the same queue. Robots are released from this queue as soon as the entry point becomes free.

C. Evaluation of Parameters and Important Metrics

1. *Path quality:* This metric evaluates the quality of paths computed through the following parameters:
   - the steps each agent takes to reach its destination
   - the stops in an agent's path
   - the number of times an agent backtracks
   - the number of times the optimization procedure results in a collision and collision avoidance has to be enforced
   - the number of times the set of robots reached a deadlock (all robots stop moving)

   During each experiment the simulation reports the average value for the above parameters, as well as its standard deviation. One comparison point for the number of steps each robot takes to its goal is the ratio of the length of the path taken versus the length of the shortest path if interactions with other robots are ignored. The duration of the single-robot shortest path is a highly optimistic estimate, with ratios near 1.0 yielding a very optimal solution.
2. *Computational Efficiency:* The duration of a planning cycle can impact the quality of the results and is an important parameter of the evaluation procedure.
3. *Scalability:* The number of sensors and robots in an experiment effect the algorithm's performance. The objective is to be able to scale these numbers while minimizing the degradation in path quality or computational efficiency.

Penalizing backtracking paths: One possible way to improve path quality is to penalize paths that move the robots along the previously traveled edge. By penalizing these backtracking paths, robots will be discouraged from reversing direction which can significantly improve the quality of the solution. On the other hand, selecting such paths may be necessary in order to avoid collisions or

		Penalty	No Penalty
Random Roadmap	Solution Steps	211.02	228.3
36 Sensors	Path Ratio	2.52	3.16
100 Robots	Deadlocks	2	30
Dense Grid	Solution Steps	78.79	83.6
25 Sensors	Path Ratio	1.24	1.32
75 Robots	Deadlocks	2	10

TABLE II
THE EXPERIMENTAL VALUES SHOWING THE IMPROVEMENTS WHEN PENALIZING BACKTRACKING PATHS. ALL VALUES ARE AVERAGED OVER 50 SIMULATIONS.

deadlocks. Therefore, the question arises whether the benefit from such a penalty is higher than any potential drawback. Experiments are conducted in the two roadmaps that allow backtracking. If a candidate path is computed that reverses the direction of the robot, the unary payoff corresponding to that path is divided by 2. The experimental results, shown in Table II, show that penalizing backtracking paths significantly improves the overall quality of the solution. The total number of steps for the solution, the ratio of the path's length to the optimal value, and the number of deadlocks is significantly reduced.

Computational Efficiency: Because this technique is to be used in an online fashion, it is important that each planning step take a relatively small amount of time. This, however, can limit the quality of the coordinated paths between robots because the message passing protocol may not converge. The coordination portion of the approach is capped at 100 iterations of the message passing protocol, or 500ms in the sparse roadmaps and 1500ms in the dense environment, whichever comes first. More time is allotted for the dense roadmap because a much larger number of candidate paths are generated for each robot during each time step. These deadlines prove very effective in the three roadmaps. The sparse urban roadmap has an average cycle duration of 0.68 seconds in the 25 sensor, 250 robot case. The dense grid environment has an average planning time of 2.07 seconds in the 25 sensor, 100 robot case. Lastly, the sparse random roadmap has an average planning time of 0.75 seconds in the 36 sensor, 100 robot case. These three configurations can be seen in the video submitted in conjunction with this work.

Scalability: Fig. 6 shows the performance of the technique in terms of total time steps for the random sparse roadmap. The best solutions appear from a 36 sensor configuration. For network configurations of 25, 36, 49, and 64 sensors, the technique is able to solve scenarios involving up to 75 robots with no reported deadlocks. It isn't until the 100 robot scenarios when deadlocks appear for this roadmap. Similar results appear in the dense grid roadmap. For a network configuration of 25 sensors, the 25 and 50 robot simulations do very well, with deadlocks beginning to appear in the 75 and 100 robot cases. The sparse urban roadmap proved to be the easiest of the three scenarios. The environment allows no backtracking because the edges are all directed. Because of this, a robot never encounters a scenario where another robot

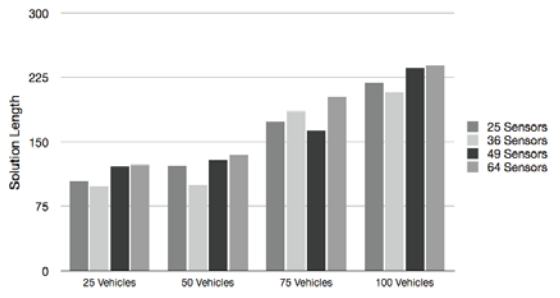

Fig. 6. The average solution length for varying numbers of robots and sensors in the random, sparse roadmap.

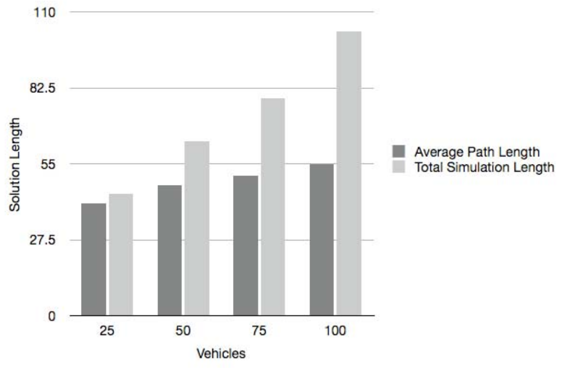

Fig. 7. Path quality in the dense grid roadmap with 25 sensors. The average path length and total simulation time steps are shown for varying numbers of robots.

is moving towards it from the opposite direction, easing the constraints between any two robots. Experiments with up to 250 robots moving through this roadmap were consistently successful without any deadlocks.

From Figures 6, 7, and 8, it can be seen that as the number of robots increases, the total solution length for each roadmap scales linearly. Experiments in each of the three roadmaps stress the total number of robots that the environment can support at any one time. These limits appear implicitly with the robot queues at the predefined entry points of each roadmap. In the sparse urban roadmap, the number of robots present inside of the roadmap peaks at 65. This equates to nearly 60% of the vertices of the roadmap occupied. For the dense roadmap, the number of robots peaks at 62 in the graph of 329 vertices. Although this roadmap has nearly three times the number of vertices as the sparse urban roadmap, the robots reach their goal positions much quicker in the dense grid because of the large number of candidate paths available to each robot. The sparse random roadmap is the most constrained roadmap of the three tested, and is also the largest in terms of area. In this roadmap, solutions are consistently computed with up to 100 robots present in the roadmap, which equates to about 50% occupancy of the underlying graph.

The number of sensors in each environment plays an important role in the solution quality for each roadmap. The number and length of the candidate paths generated by each robot is dependent on the size of each sensor node's coverage

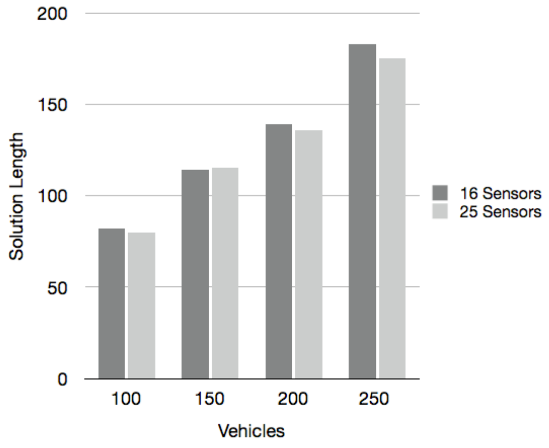

Fig. 8. The average solution length for varying numbers of robots and sensors in the random roadmap.

area. If there are very few sensors in an environment, the area covered by each node will be large, resulting in many long candidate paths for each robot. This introduces coordination between robots that may otherwise be unaffected by the action of the other because of their large distance from each other. The opposite situation can also be detrimental. In an environment that is over saturated with sensors, the coverage area for each node becomes small, resulting in a small number of short candidate paths for each robot. The small number of paths, coupled with the fact that robots will not coordinate until they are in very close proximity to each other can lead to sub optimal choices in path coordination. Figure 6 shows that network configurations with small or large numbers of sensors result in worse solutions for the same roadmap.

VI. Discussion

This paper introduces the problem of multi-robot path planning through a set of network nodes that guide agents moving on a graph. This work proposes a distributed, online algorithm for this problem, where each node of the network has information about robots only in a local neighborhood and exchanges information only with 1-hop neighboring nodes. The algorithm casts the challenge as a Distributed Constraint Optimization problem, models the interactions of agents through a coordination graph, and applies a message passing protocol for its solution. The method guarantees collision avoidance without causing the network to flood with messages. Simulated experiments showed that deadlocks occur rarely on benchmarks where prioritized schemes failed very quickly and the coupled approach is infeasible.

There are many exciting extensions for this line of work: (1) Integrating the proposed algorithm with coupled planners so as to guarantee deadlocks avoidance, while keeping the computational cost as low as possible. (2) Extending this work to continuous space planning and considering dynamic motion constraints. Similarly communication constraints, such as bandwidth and throughput, or location uncertainty can also be introduced. (3) Load balancing schemes where heavy-loaded nodes outsource computation to less loaded neighbors can be helpful. Nodes could also direct robots to different regions of the workspace so as to avoid congestion. (4) It is interesting to investigate the case where certain vehicles do not cooperate, but can be detected by the network, as well as the case of malicious agents. (5) What happens when there are gaps in the coverage of the workspace or how should the network adapt when a node fails.

References

[1] C. R. Weisbin, P. S. Schenker, R. Easter, and G. Rodrigue, "Robotic space colonies," in *6th Inter. Symp. on Artificial Intelligence, Robotics and Automation in Space (ISAIRAS-01)*, Montreal, Canada, 2001.

[2] P. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative autonomous vehicles in warehouses," *AI Magazine*, 2008.

[3] C. Clark, S. Rock, and J.-C. Latombe, "Motion planning for multiple robot systems using dynamic networks," in *Proc. IEEE Int. Conf. on Rob. and Autom. (ICRA)*, 2003, pp. 4222–4227.

[4] M. Peasgood, C. Clark, and J. McPhee, "A complete and scalable strategy for coordinating multiple robots within roadmaps," *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 282–292, 2008.

[5] J. van den Berg, J. Snoeyink, M. Lin, and D. Manocha, "Centralized path planning for multiple robots: Optimal decoupling into sequential plans," in *Robotics: Science and Systems V*, 2009.

[6] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," in *IEEE ICRA*, 1986, pp. 1419–1424.

[7] G. Sanchez and J.-C. Latombe, "Using a prm planner to compare centralized and decoupled planning for multi-robot systems," in *IEEE Int. Conf. Robotics and Automation (ICRA)*, 2002, pp. 2112–2119.

[8] M. Bennewitz, W. Burgard, and S. Thrun, "Finding and optimizing solvable priority schemes for decoupled path planning for teams of mobile robots," *Robotics and Autonomous Systems*, vol. 41, no. 2, pp. 89–99, 2002.

[9] M. Saha and P. Isto, "Multi-robot motion planning by incremental coordination," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006, pp. 5960–5963.

[10] Y. Li, K. Gupta, and S. Payandeh, "Motion planning of multiple agents in virtual environments using coordination graphs," in *IEEE Int. Conf. Robotics and Automation (ICRA)*, 2005, pp. 378–383.

[11] K. E. Bekris, K. I. Tsianos, and L. E. Kavraki, "A decentralized planner that guarantees the safety of communicating vehicles with complex dynamics that replan online," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007, pp. 3784–3790.

[12] M. Batalin, G. S. Sukhatme, and M. Hattig, "Mobile robot navigation using a sensor network," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2004, pp. 636–642.

[13] P. Corke, R. Peterson, and D. Rus, "Localization and navigation assisted by cooperating networked sensors and robots," *Intern. Journal of Robotics Research (IJRR)*, vol. 24, no. 9, pp. 771–786, 2005.

[14] C. Buragohain, D. Agrawal, and S. Suri, "Distributed navigation algorithms for sensor networks," in *IEEE Int. Conf. on Computer Communications (INFOCOM)*, April 2006, pp. 1–10.

[15] C. E. Guestrin, D. Koller, and R. Parr, "Multiagent planning with factored mdps," in *Proc. 14th Neural Information Processing Systems (NIPS-14)*, 2001, pp. 1523–1530.

[16] N. Vlassis, R. Elhorst, and J. Kok, "Anytime algorithms for multiagent decision making using coordination graphs," in *IEEE Transactions on systems, Man and Cybernetics*, The Hague, Netherlands, 2004.

[17] J. Kok and N. Vlassis, "Collaborative multiagent reinforcement learning by payoff propagation." *Journal of Machine Learning Research*, vol. 7, pp. 1789–1828, 2006.

[18] P. Modi, W. Shen, M. Tambe, and M. Yokoo, "Adopt: Asynchronous distributed costraint optimization with quality guarantees," *Artificial Intelligence*, vol. 161, no. 1-2, pp. 149–180, 2005.

[19] W. Yeoh, F. A., and S. Koenig, "BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm." in *AAMAS*, 2008, pp. 591–598.

[20] M. Lagoudakis, V. Markakis, D. Kempe, P. Keskinocak, S. Koenig, A. Kleywegt, C. Tovey, M. A., and S. Jain, "Auction-based multi-robot routing," in *Robotics: Science and Systems I*, 2005, pp. 343–350.

[21] Y. Fujita, Y. Nakamura, and Z. Shiller, "Dual dijkstra search for paths with different topologies," in *IEEE Int. Conf. on Robotics and Automation, (ICRA)*, vol. 3, 2003, pp. 3359–3364.