Communication-Based Motion Planning

Lex Fridman*, Jay Modi*, Steven Weber† and Moshe Kam†*

*Department of Computer Science

†Department of Electrical and Computer Engineering

Drexel University, Philadelphia, Pennsylvania 19104

***Abstract*— The conventional philosophy in designing mobile networks is that network node movement should be independent of network state. However, there are practical situations where movement decisions may be modified to ensure connectivity. For example, emergency responders in a crisis region relying upon an ad hoc network may need constant reliable communications and therefore adjust their search plan to stay connected, though aspects of their mission may override their objective of staying connected. We present a discrete formulation for this problem and a method for solving it optimally. We propose a cooperative and a noncooperative algorithm, showing that the run-time of the latter is drastically more efficient with a minimal performance cost relative to optimality.**

I. Introduction

Path planning is used for a range of applications from optimal-path navigation [1], [2] to exploration [3], [4] to the movement of robotic arms in manufacturing plants [5], [6]. Many different constraints have been explored, including formations of robots, kinematic properties of vehicles, and robotic sensors and communication.

This paper focuses specifically on the movement of a team of nodes through an obstacle laden terrain with the objective of maintaining communication. The nodes can be vehicles, people, robots, or anything else that moves. The communication objective is defined as the number of strongly connected components in the network. The optimization problem seeks to minimize the average number of such components. The terrain contains obstacles which obstruct both movement and communication.

Both the terrain and the physical paths of the nodes are known *a priori*. We wish to move our nodes in such a way that minimizes the total time to scenario completion but also minimizes the time they spend disconnected.

In multi-agent systems there often appears a tradeoff between centralized optimality and distributed efficiency. This paper proposes two methods, cooperative and noncooperative, which embody the respective classes of optimization techniques. Furthermore, the distributed method is shown to be near-optimal while being low-cost computationally.

II. Background

The problem of constrained motion control has been thoroughly studied, most notably under the topic of formation control. Specifically, Bicho and Monteiro [7] use nonlinear attractor dynamics to model nonholonomic formations of robots. The key contribution of their work is the ability of robots to maintain robust formations in the face of unknown environmental perturbations. In earlier work in the field, Wang [8] studies the use of nearest neighbor tracking to maintain formation. Their method has the important qualities of being simple, computationally efficient, and distributed. However, it does not directly consider collision and obstacle avoidance. Desai, et al., [9] use methods of feedback linearization to stabilize the distance of a follower node to a leader node. They further investigate this problem in [10], proposing a graph theoretic framework for the coordination of transitions from one formation to another.

Formation control is a basis for the more recent efforts, most relevant for the proposed algorithms in this paper, on motion planning under communication and networking constraints. Beard and McLain [11] propose a dynamic programming method for the cooperation motion planning under communication constraints that is polynomial in the number of nodes but exponential in the depth of the lookahead window. The key characteristic of this method relative to our proposed methods is that the limited range connectivity is a hard constraint, while in our method it is an objective. The hard constraint is a prerequisite which makes their method inapplicable to the more general model considered in this paper. Pereira, et al., [12] present a decentralized method of motion planning under communication constraints. One of the defining features of their method is that it first seeks to achieve connectivity, and then move to the goal positions in such a way as to not break connectivity, which is essentially the problem of formation control. Spanos and Murray [13] define a connectivity robustness metric, which uses conservative connectivity constraints to attain good network performance despite the algorithm's distributed structure with minimal added limitations on physical reachability. The key difference of this work from our paper is that Spanos and Murray view connectivity as a constraint while we view it as an objective, which is more general in that it seeks to also optimize communication for cases when connectivity is sparse.

III. Problem Formulation

A. Path-Based Mobility Control

The discrete arena representation is an $s \times s$ kinematic constraint lattice $G = (V, E)$ with randomly positioned convex polygon obstacles. Each point $i \in V$ in the lattice represents a space in the arena that can be occupied by the node. Each edge $(i, j) \in E$ in the lattice represents the possible set of movements a node can make in a single time step. An example lattice is shown in Figure 1.

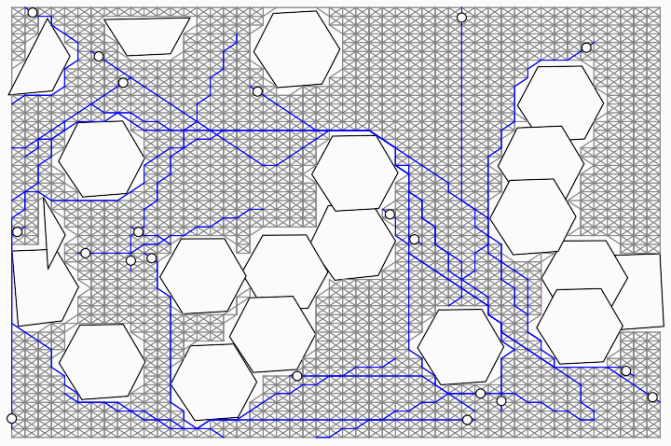

Fig. 1. 20 example paths in the lattice-discretized space with obstacles.

The mobile network consists of a set n nodes. Each node $i = 1, 2, ..., n$ has a corresponding path $\mathbf{P}_i$. One end of $\mathbf{P}_i$ is the *origin* of i, and is denoted as $o_i \in V$. The other is its *goal*, $g_i \in V$. The length $L_i > 0$ of the path $\mathbf{P}_i$ is the number of steps it requires a node to travel from the origin to the goal. Therefore, there are $L_i + 1$ positions on path $\mathbf{P}_i$ that can be occupied by node i. Specifically, $\mathbf{P}_i = \{P_i(0) = o_i, P_i(1), ..., P_i(L_i - 1), P_i(L_i) = g_i\}$.

Time is discrete and is advanced in steps. It is assumed that the path for each node as shown in Figure 1 is given a priori. Therefore, the motion planning problem involves a set of two choices for each node at each state of the network. The choices are (1) to step forward or (2) to stay in place. We denote $D_i(t)$ as the position of node i at time t, where $t = 0, 1, 2, ...$ is the elapsed time from the start of simulation.

The time it takes for a node i to travel along a path is denoted as T_i. For example, if the node marches down its prescribed path without stopping $T_i = L_i$. We refer to the sequence of choices (move forward or wait) in this paper as a *plan* or *policy*. Therefore, the length of a plan for node i is T_i, and the length of a path for node i is L_i. This is an important distinction in terminology, and will be used often, because the improvement from motion planning is attained when $T_i - L_i > 0$. $T_{\max} = \max\{T_i : i = 1, 2, ..., n\}$ designates the length of the longest plan, which is also the time to completion of the scenario. The term *scenario* is used to denote the set of n plans as a whole. After time $t = T_i$, node i remains in the same position until the completion of the scenario. In other words, node i remains at its goal state without the ability to move for a duration of $T_{\max} - T_i$. Nevertheless, it remains part of the global communication network during that time.

The number of possible movement combinations at each t is $\delta = 2^n - 1$, where δ is the upperbound branching factor for each state reachable from the initial state. It is assumed that at least one node must move forward at each time step. The state of the system at time t is the vector $\mathbf{D}(t) = \{D_1(t), D_2(t), \ldots, D_n(t)\}$ of positions of the nodes by time t. We use $\mathbf{\Pi}_i = \{D_i(0), D_i(1), \ldots, D_i(k-1)\}$ to denote the ordered set of positions which make up a plan. The cardinality of this set is k and, given the proposed model, the cardinality of $\mathbf{\Pi}_i$ is the same for all i. The function $K(\mathbf{\Pi}) = k$ returns this common cardinality. Furthermore, we use $\mathbf{\Pi}_i(t) = \mathbf{D}_i(t)$, $\mathbf{\Pi}(t) = \mathbf{D}(t)$, and $\mathbf{\Pi} = \{\mathbf{\Pi}_1, \mathbf{\Pi}_2, \ldots, \mathbf{\Pi}_n\}$. Both $\mathbf{\Pi}$ and $\mathbf{D}$ contain information about the topology of a network through time. However, $\mathbf{\Pi}$ is used to denote all potential plans, while $\mathbf{D}$ is used to denote actual topologies.

B. Communication Models

The number of connected components is used as the global network performance measure. This value is computed using Kosaraju's algorithm [14]. The range of values for this metric is in $\{1, 2, ..., n\}$.

Two nodes i and j are considered connected if the distance between them d_{ij} is less than or equal to the constant $d_{\max}$, which is proportional to a node's transmit power. In addition to the range requirement for connectivity, two nodes must have line-of-sight communication with each other. Therefore, the second condition of connectivity is that a straight line can be drawn which connects the two nodes without intersecting any obstacles.

C. Optimization Objectives, Constraints, and Search Space

The motion planning problem that we define involves maximizing a global network performance measure under the constraints of a fixed travel time. That is, a node i is given a path of length L_i, and a time to completion of T_i, the optimization seeks to find what the node should do with the extra $T_i - L_i \geq 0$ time units. In this paper, we ensure that for all i, $T_i - L_i$ is equal to a constant τ. Therefore, each node gets the same amount of extra time which can be used to improve the network performance throughout their plan.

The network performance is an average over the duration of the scenario. Therefore, the objective value is the measure of connectivity for the path divided by the global time to completion $T_{\max}$. We use the $C(\mathbf{D}(t))$ to denote the function which computes the number of connected components in the network given the position of the nodes $\mathbf{D}(t)$. The movement planning problem can be formulated as follows:

$$
\begin{aligned}
\min \quad & \frac{1}{T_{\max}+1} \sum_{u=0}^{T_{\max}} C(\mathbf{D}(u)) \\
\text{s.t.} \quad & T_i = L_i + \tau \qquad i = 1...n \\
& D_i(t) = g_i \qquad T_i \leq t \leq T_{\max}, \qquad i = 1...n \\
& D_i(0) = o_i \qquad i = 1...n \\
& (D_i(t), D_i(t+1)) \in E \qquad i = 1...n
\end{aligned}
\tag{1}
$$

Where E, as defined previously, is the set of all possible single-step movements allowed in the lattice.

Given the nodes, their paths, and the function $C(\mathbf{D})$, the search space of the optimization problem can be constructed in the form of an n-dimensional connectivity matrix. This matrix is a representation of the $C(\mathbf{D})$ function. An example of a 2-dimensional matrix is shown in Figure 2. The x-axis

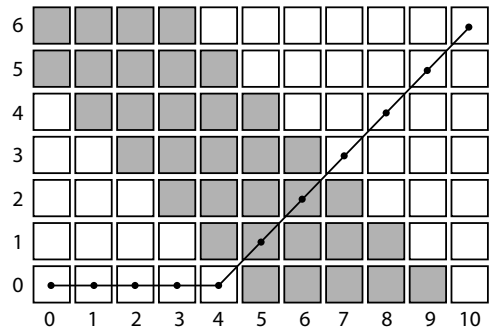

Fig. 2. Connectivity matrix for 2 nodes. The gray states have a $C(\mathbf{D})$ value of 2, while the white states have a value of 1. The path through the matrix defines the combined movement policy for the two nodes. The bottom left corner is the starting state, and the top right corner is the goal state.

of this matrix is the position of a_1 along its path from 0 to $L_1 = 10$. The y-axis of the matrix is the distance traveled by a_2 along its path from 0 to $L_2 = 6$. Each cell in the matrix contains the number of connected components in that specific network configuration . In Figure 2, the gray boxes designate the state with two connected components, which means that the network is completely disconnected. The white boxes designate one connected component, which means that the network is fully connected. Note that the dimension of the connectivity matrix is n.

A solution to this problem is a path through this connectivity matrix from the starting point (where all nodes are at their origins) to the goal point (where each node has reached its goal). In the matrix in Figure 2, three different movements are possible: moving up corresponds to moving node 2 along its path, moving to the right corresponds to moving node 1 along its path, moving diagonally corresponds to moving node 1 and node 2 along their respective paths. The path shown in Figure 2 has a value of 1 in the each of the first 5 positions, then a value of 2 in each of the the next two positions, and a value of 1 in each of the last 4 positions. Therefore, the value for this policy as defined in the objective function of (1) is $\frac{9\times1+2\times2}{11} = \frac{13}{11}$.

For a given instance of a movement planning problem, the increase of τ is likely to result in an increased improvement in performance in the average case. Therefore, the optimization problem defined in (1) is solved for $\tau = 0, 1, 2, \ldots$. The solutions to each of these problems form a Pareto-optimal front [15]. τ is a secondary objective in that we may wish to minimize the time to completion $T_{\max}$. Therefore, the Pareto front allows one to choose between faster travel time or better network performance.

IV. Search Methodology

The search space for the problem defined by (1) is all the possible paths from the origin to the destination in the connectivity matrix described in the previous section. We propose two search methods: cooperative and noncooperative. The cooperative method finds the optimal solution but at the cost of greater running time. The noncooperative method is much faster, but it does not guarantee finding the optimal. However, as we show in the results, it performs very well in the average case.

A. Cooperative Optimization

Uniform-cost search [16] is used to traverse the search space of the optimization problem defined in (1). This method is labeled as *cooperative* because it considers at each time point all possible combinations of movements by the nodes. In other words, the movement choice of one node directly affects the movement choice of all other nodes.

Algorithm 1 `UniformCostSearch`

Require: n nodes with determined paths
  searchNode $\leftarrow \mathbf{D}(0)$
  openlist.add(searchNode)
  loop
    searchNode $\leftarrow$ **`MinCost`**(openlist)
    if searchNode = goalState **then**
      return **`Plans`**(searchNode)
    end if
    openlist.add(**`Successors`**(searchNode))
  end loop

Algorithm 1 initiates its search at the state where all the nodes are at their respective origins. A state in this context is defined by $\mathbf{D}(t)$, that is, the position of all the nodes at a specific time. The successor states from each current state are determined using Algorithm 2.

Algorithm 2 **`Successors`**(state)

Require: n nodes with determined paths
  for $i = 1$ to δ **do**
    change $\leftarrow$ **`IntToBinaryString`**(i)
    next $\leftarrow$ state + change;
    if **`Valid`**(next) **then**
      list.add(next)
    end if
  end for
  return list

`Successors`(state) takes a state as input and returns a list of states which are its valid successors. A valid state is one that is within the bounds of the path. **`IntToBinaryString`**(i) converts the integer into a binary string that is then used to move each node either 0 or 1 steps forward. For a mobility model where backwards mobility or multiple-step mobility is allowed, the integer would instead be converted to a higher base. **`Plans`**(searchNode) traces the searchNode state back up the search tree to generate a set of plans that was used to reach that state.

Algorithm 1 also defines a **`MinCost`**(list) method which takes a set of states and returns a single state which minimizes a connectivity cost function **`Cost`**$(\mathbf{\Pi}(t))$:

$$\texttt{Cost}\,(\mathbf{\Pi}) = \sum_{u=0}^{K(\mathbf{\Pi})} C(\mathbf{\Pi}(u)) + \frac{K(\mathbf{\Pi})}{T_{\max}+1} \tag{2}$$

The second term is the travel time cost and is used as a tie breaker for when the values of the first term are equal for two or more states. Its maximum value is smaller than the minimum non-zero incremental change of $C(\mathbf{D}(t))$. Therefore, it cannot influence the cost to a point where a state with worse communication be chosen over another with better communication. In other words, the second term is a preference for shorter paths.

B. Noncooperative Optimization

The noncooperative method does not consider the nodes in the network in a centralized way when planning their movement. Instead, it plans each node's movement individually assuming fixed movements for other nodes. It repeats this procedure for each node until convergence.

`NoncoopSearch` as shown in Algorithm 4 loops through each node on which it performs a procedure similar to **`UniformCostSearch`**, except that instead of adding δ successors at each iteration it adds at most 2. As input, the algorithm is given a set of plans $\mathbf{\Pi}$. For each node i, it searches for a way to adjust $\mathbf{\Pi}_i$ such that the global network performance is improved. The cost of a state is determined based on the current node's choice of movement and the prespecified choices of the other nodes. **`Plan`**(searchNode) traces the searchNode state back up the search tree to generate a plan that was used to reach that state. Note that "searchNode+1" indicates a state that would be reached by moving one step forward from searchNode.

`ConvergePlan` as shown in Algorithm 3 first initializes a set of plans to the march-ahead policies using **`GenerateMarchAheadPlans`**. What this function does is for each node i, it sets $\mathbf{\Pi}_i$ to $\mathbf{P}_i$ plus $T_{\max} - L_i$ copies of the goal state $P_i(L_i)$. In other words, the initial plan for each node is to step forward at each time step and once the goal state is reached to wait there until the end of the scenario. After the initialization, **`ConvergePlan`** repeatedly runs **`NoncoopSearch`** until the first sign of convergence, that is, when the cost of the policies from adjacent iterations are the same. Note that in the actual implementation other signs of convergence are monitored for such as repeated patterns in cost values. The set of policies returned by this algorithm is not the last one found, but the best one found before the detection of convergence.

C. Iterative Depth Search

The solution that is returned both by the cooperative and noncooperative methods is a policy for a single value of τ. In order to obtain the entire Pareto front, the search algorithm is run repeatedly while iterating τ from zero up by one. The resulting set of solutions is Pareto optimal for the two objectives of communication and travel time.

Algorithm 3 **`ConvergePlan`**

Require: n nodes with determined paths
  costPrevious $\leftarrow 0$
  costBest $\leftarrow 0$
  $\mathbf{\Pi} \leftarrow$ **`GenerateMarchAheadPlans`**($\mathbf{P}$)
  loop
    $\mathbf{\Pi} \leftarrow$ **`NoncoopSearch`**($\mathbf{\Pi}$)
    costCurrent $\leftarrow$ **`Cost`**($\mathbf{\Pi}$)
    if costCurrent = costPrevious $\cap$ costPrevious $\neq 0$ **then**
      return $\mathbf{\Pi}_{\text{best}}$
    else if costCurrent $<$ costBest **then**
      costBest $\leftarrow$ costCurrent
      $\mathbf{\Pi}_{\text{best}} \leftarrow \mathbf{\Pi}$
    end if
    costPrevious = costCurrent
  end loop

Algorithm 4 **`NoncoopSearch`**

Require: n nodes with determined plans $\mathbf{\Pi} = \{\mathbf{\Pi}_1, ..., \mathbf{\Pi}_n\}$
  for $i = 1$ to n **do**
    searchNode $\leftarrow o_i$
    done $\leftarrow$ false
    openlist.add(searchNode)
    while done = false **do**
      searchNode $\leftarrow$ **`MinCost`**(openlist)
      if searchNode = goalState **then**
        Π_i = **`Plan`**(searchNode)
        done $\leftarrow$ true
      else
        openlist.add({searchNode, searchNode+1})
      end if
    end while
  end for
  return $\mathbf{\Pi}$

D. Complexity Analysis

Algorithm 1 is an exhaustive search. In the majority of cases, it does not consider all the possible states, but in the worst case it must consider $O(\delta^{T_{\max}})$ states, because δ is the branching factor of the search tree and $T_{\max}$ is its depth. For $\delta = 2^n - 1$, the asymptotic size of the search space is $O(2^{nT_{\max}})$.

For each state, **`Cost`** is executed. This function is implemented using Kosaraju's algorithm [14], which is $O(n)$. For each link in the network, we check if it intersects any of the obstacles in the arena. There are $O(n^2)$ such links, and the intersection algorithm's running time is $O(\beta)$ for each link, where β is the number of obstacles in the arena. For example, in Figure 1, $\beta = 20$. The time complexity of **`Cost`** is $O(n + \beta n^2) = O(\beta n^2)$. Note that each computation of $C(\mathbf{\Pi}(t))$ is cached and therefore the running time of **`Cost`** only includes computing the cost of the latest addition to the plan, assuming it hasn't already been encountered.

`MinCost` is executed for all non-leaf states, of which

there are $O(2^{nT_{\max}-1})$. However, because the *openlist* never contains duplicate states, the running time complexity of **MinCost** is constant in the number of states. In summary, the expensive processes involved in Algorithm 1 are the generation of the states and the computation of the cost for each state. Therefore, the total worst-case running time of the cooperative algorithm is:

$$O(\beta n^2 \cdot 2^{nT_{\max}}) \tag{3}$$

The *openlist* may potentially contain half of the search space. Therefore, the worst-case space complexity of the uniform-cost search algorithm is $O(2^{nT_{\max}-1})$.

The noncooperative method in Algorithm 3 is more efficient than Algorithm 1 because of its distributed structure. The outer loop performs a small number c of iterations for convergence. The planning is done for each of the n nodes. The computation of the cost is still $O(\beta n^2)$. Since the branching factor is 2, and the search tree depth is again $T_{\max}$, the maximum number of states that is considered is $2^{T_{\max}}$. Therefore, the total worst-case running time of the noncooperative algorithm is:

$$O(c\beta n^3 \cdot 2^{T_{\max}}) \tag{4}$$

This method is not exponential in the number of nodes and therefore is scalable to any size network. However, it can only plan a limited number of steps ahead ($T_{\max}$) before becoming computationally burdensome.

V. Simulation and Results

A. Simulation Setup

A simulation arena of 100×100 sq. meters with a variable number of nodes is used. The paths for the nodes are generated randomly using a goal-based method. First, the origin and goal of each path are randomly and independently placed in the arena by a 2D Poisson process. The actual path between these two points is generated by performing a simulated random-waypoint walk. The initial direction of the velocity vector for the node at each waypoint is toward the goal. A uniformly-distributed random turn of limited-magnitude is then performed. The path construction completes when the current waypoint is sufficiently close to the goal or an obstacle obstructs the path-construction process to the point where the naive avoidance algorithm is not able to find a path around the obstacle. Figure 1 shows a typical realization of this process.

The range of communication requirement $d_{\max}$ is chosen using a parameter $m \geq 0$ that is scalable in the number of nodes and the size of the world:

$$d_{\max} = \frac{m}{\pi}\sqrt{\frac{A \log n}{n}} \tag{5}$$

Where A is the area of the arena and n is the number of nodes. When $m = 1$ the network is expected to be asymptotically connected [17] under similar conditions. When $m \gg 1$ the network is well connected and motion planning is unlikely to improve the network performance, unless most of the arena is covered with obstacles. When $m \approx 1$, the network is likely to be partially disconnected and therefore motion planning is expected to show improvement. When $m \ll 1$, the granularity of the movement graph is greater than $d_{\max}$ and therefore motion planning is likely to be unable to improve connectivity.

B. Algorithm Performance

We analyzed the scalability of the noncooperative and cooperative methods in relation to the number of nodes and to the search depth. The complexity bounds presented in Section IV-D showed that cooperative methods should be exponentially faster in n and significantly faster in the search depth. This was also observed in simulation. The running time for a scenario with 5 nodes and search depth of 3 was 98 seconds for the cooperative method and 0.04 seconds for the noncooperative method. The average was taken from 200 simulation runs of the former and 10000 runs from latter.

C. Cooperative vs. Noncooperative Search

Noncooperative search is scalable and computationally efficient, however, in order for it to be a viable alternative for the optimal cooperative method it must perform well relative to the cooperative case. We ran a simulation for each method with $n = 3$ nodes and $\tau = 0, 1, ..., 4$. A small-scale network example was used because of the costly running-time complexity of cooperative method. The connectivity parameter m was kept constant at 0.4. Figure 3 shows the results of the simulation. As τ increases, the improvement in average connectivity also increases to over 80% for $\tau = 4$. Most importantly, the plot shows that the average performance of the more efficient noncooperative method is near-optimal, therefore, validating it as good approximation for the policies generated by the cooperative method. The concave shape of curve and the small change from $\tau = 3$ to $\tau = 4$ suggests a convergence in τ. This was investigated further for the noncooperative method for $\tau = 5, 6, ..., 12$, and in fact for this method the improvement does converge to about 87%.

D. Improvement vs. Communication Range

The magnitude of network performance benefit that motion planning can provide depends on the communication range of the nodes in the network. If each node has a long range, then the network is mostly connected and therefore allow for only limited improvement. On the other hand when the range parameter m is small, and thus the communication range is short, the network is mostly disconnected. In this case, it can benefit significantly from motion planning. The improvement in connectivity, denoted as N, is computed by taking the difference between the average connectivity for the march-ahead without stopping policy (when $\tau = 0$) and the communication-optimal policy (when $\tau > 0$), and dividing that value by the average connectivity of the $\tau = 0$ policy.

In Figure 4, three different communication ranges are considered: $m = 0.2, 0.4, 0.6$. Under each value of the range parameter, a simulation similar to the one in Section V-C is run. However, the scale of the network is larger. The simulation

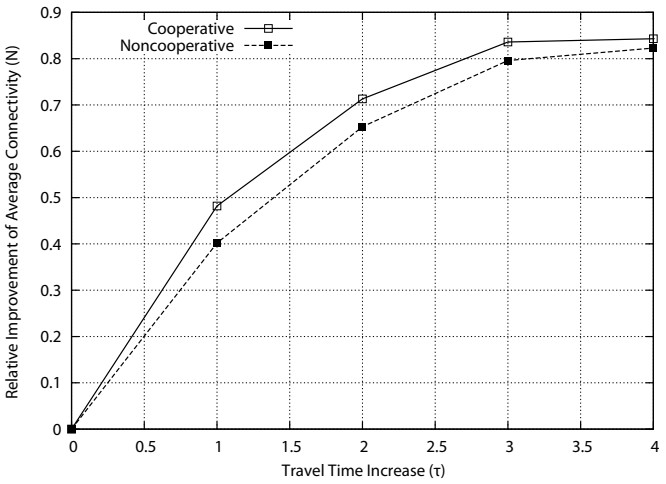

Fig. 3. Comparison of connectivity improvement for cooperative and noncooperative planning method.

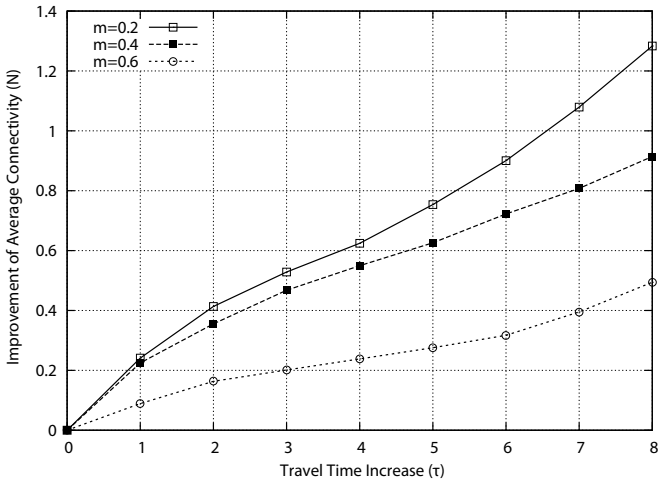

Fig. 4. Performance of noncooperative planning algorithm under varying communication capabilities of network nodes.

is of $n = 20$ nodes, and for $\tau = 0, 1, ..., 8$. The noncooperative search was run 10000 times and averaged for each data point on the plot.

There are three key insights that this scenario provides. First, as more slack is added to the travel time requirement, movement planning improves the average network performance. Moreover, the improvement increases approximately linearly with the magnitude of τ. The plot shows an improvement as high as 128%. Second, the curve for the values of τ tested is not concave and does not appear to converge even when tested up to $\tau = 22$. Therefore, the more time that can be allowed for the nodes in the network to "wander", the better the average performance of the network becomes. Third, as the connectivity parameter m increases the benefit that motion planning provides is lessened. Therefore, the classes of scenarios in which motion planning is most applicable as those which have poor connectivity.

VI. Conclusion

When a set of assets that are part of a communication network have movement policies that are not constructed with network performance as the objective, the adjustment of their movement along those paths based on such an objective can improve communication. We propose an optimal cooperative method and an efficient near-optimal noncooperative method which shows, in simulation, significant improvements in connectivity of the moving network. Both algorithms provides a Pareto-optimal set of solutions which give a range of choices between the conflicting objectives of quick travel time and good communication.

References

[1] A. W. Ho and G. C. Fox, "Learning to plan near-optimal collision-free paths," in *Proceedings of the Fifth Distributed Memory Computing Conference*, vol. I, Applications. Charleston, SC: IEEE, Apr. 1990, pp. 131–139.

[2] A. Mei and Y. Igarashi, "An efficient strategy for robot navigation in unknown environment," *INFO-PROC-LETT*, vol. 52, no. 1, pp. 51–56, Oct. 1994.

[3] M. J. Mataric, "Parallel, decentralized spatial mapping for robot navigation and path planning," in *PPSN*, 1990, pp. 381–386.

[4] Y. Fukazawa, T. Chomchana, J. Ota, H. Yuasa, T. Arai, and H. Asama, "Region exploration path planning for a mobile robot expressing working environment by grid points," in *ICRA*, 2003, pp. 2448–2454.

[5] M. Jabee, "Robot arm modelling and simulation in a 3d factory environment." WSEAS, Sept. 14-17 2003, p. 9. [Online]. Available: http://www.worldses.org/online/

[6] V. Lumelsky and K. Sun, "A unified methodology for motion planning with uncertainty for 2D and 3D two-link robot arm manipulators," Yale U., Tech. Rep. Systems and Informations Sciences, TR 8805, Department of Engineering and Applied Science, Yale University, New Haven Connecticut, January, 1988.

[7] E. Bicho and S. Monteiro, "Formation control for multiple mobile robots: a non-linear attractor dynamics approach," in *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, NV, October 2003, pp. 2016–2022.

[8] P. K. C. Wang, "Navigation strategies for multiple autonomous mobile robots moving in formation," in *Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems*, Tsukuba, Japan, September 1989, pp. 486–493.

[9] J. P. Desai, J. Ostrowski, and V. Kumar, "Controlling formations of multiple mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Leuven, Belgium, May 1998, pp. 2864–2869.

[10] J. P. Desai, P. Ostrowski, and V. Kumar, "Modeling and control of formations of nonholonomic mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 905–908, December 2001.

[11] R. W. Beard and T. W. McLain, "Multiple UAV cooperative search under collision avoidance and limited range communication constraints," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, Maui, Hawaii, December 2003, pp. 25–30.

[12] G. A. S. Pereira, A. K. Das, R. V. Kumar, and M. F. M. Campos, "Decentralized motion planning for multiple robots subject to sensing and communication constraints," in *Proceedings of the 2003 International Workshop on Multi-Robot Systems*, 2003, pp. 267–278.

[13] D. P. Spanos and R. M. Murray, "Motion planning with wireless network constraints," in *Proceedings of the 2005 American Control Conference*, Portland, OR, June 2005, pp. 87–92.

[14] R. Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM Journal on Computing*, vol. 1, p. 146, 1972.

[15] Y. Censor, "Pareto optimality in multiobjective problems," *Applied Mathematics and Optimization*, vol. 4, no. 1, pp. 41–59, 1977.

[16] K. R. E., "Artificial intelligence search algorithms," University of California, Los Angeles, Computer Science Department, Technical Report 960029, June 30, 1996.

[17] P. Gupta and P. Kumar, "Critical power for asymptotic connectivity in wireless networks," *Stochastic Analysis, Control, Optimization and Applications: A Volume in Honor of WH Fleming*, pp. 547–566, 1998.