Proceedings of the 2003 IEEE/RSJ
Intl. Conference on Intelligent Robots and Systems
Las Vegas, Nevada · October 2003

Maintaining Line of Sight Communications Networks between Planetary Rovers

Stuart O. Anderson, Reid Simmons, Dani Goldberg
School of Computer Science, Carnegie Mellon University
soa@andrew.cmu.edu, {reids,danig}@cs.cmu.edu

Abstract

We present an algorithm designed to solve the problem of maintaining communications within a group of robotic explorers. The rovers we consider are equipped with communication hardware that is effective only over a limited range and requires direct line of sight to function. The paper presents the algorithm used to solve this problem and some details of our implementation. We also present the results of an experimental analysis of the algorithm's performance characteristics in a simulated multi-rover environment.

1. Introduction

The Federation of Intelligent Robotic Explorers (FIRE) project is exploring approaches to coordinating distributed heterogeneous rovers carrying out complex tasks during planetary exploration (Simmons et al, 2002). A key component to such coordination is the ability to maintain a communications network between the rovers as they carry out tasks on Martian terrain. The baseline communications hardware for the rovers is effective only over a limited range and requires a direct line of sight between communicating pairs of rovers. Because of these limitations, the task of maintaining communications becomes non-trivial as the rovers disperse over the terrain, especially since terrain features can cause occlusion. Separate areas may be out of range or have line-of-sight blocked by the terrain. In order to maintain communications between rovers working at separate sites, some of the rovers will need to act as communication relays for the rest of the group.

By formalizing this problem and reducing it to the repeated solution of a simpler problem, which is constructing a link between a pair of clusters of rovers, we have developed an algorithm that can be used to plan a network of relays that allows all desired communications to take place. The algorithm identifies pre-existing clusters of rovers, attempts to determine the least costly set of links between those clusters that will provide all requested communications, and then builds those edges sequentially, revising the original estimate of the least costly set of links as necessary.

The algorithm accepts requests for communication channels between pairs of rovers, and then produces movement commands for some of the rovers. These movements allow the communication requests to be satisfied by relaying communications between rovers. It is desirable that the planned movements minimize both the number of rovers moved and the difficulty of the movement undertaken. We require that a rough elevation map of the surrounding terrain be provided and that the location of each of the rovers be identified on this map. We assume that only some of the rovers are available to be relays and these are specified at the same time as the communications requests.

2. Related Work

A number of protocols have been developed to allow efficient construction of network topology in an environment with changing connectivity (Bellur and Ogier, 1999) (Broch et al, 1998). The mobile network routing research of (Alzoubi et al, 2003) uses a periodic model of node motion identical to our own, while (Haas, 1997) examines the effect of environmental obstacles on overall network connectivity. Neither of these studies considers the motion of network nodes to be controllable. These ad hoc networks are a common feature of mobile multi-robot coordination systems (Bhargav et al, 2002) (Fierro et al, 2003). The work of (Li and Rus, 2000), as well as (Arkin and Diaz, 2002) explicitly considers communication needs when planning movement. These projects are similar in spirit to our own and share aspects of their approach with our own. Many projects, however, choose not to explicitly treat communication when planning motion and rely instead on the robustness of the communications protocol they employ to relay messages when communications in a disjoint network are eventually restored. In the case of the FIRE scenario, this would not be a feasible solution, because the rovers are occupied with a number of different tasks that take place in widely separated locations. There is no guarantee that communications will be restored in a reasonable amount of time. To compensate for this we must actively manipulate the position of network nodes in order to ensure viable communications.

3. Algorithm

The algorithm attempts to satisfy the communication requests by augmenting the pre-existing communications

0-7803-7860-1/03/$17.00 © 2003 IEEE

network until there is a route between all pairs of rovers that have requested communication. We begin by identifying pre-existing clusters of rovers, and then connect pairs of clusters using a relay, or set of relays. The following pseudo code provides a rough outline of this algorithm.

```
1. Identify pre-existing clusters
2. Create groups of communicating clusters
3. For each group of clusters:
   1. Estimate least costly set of relays
      between pairs of clusters needed to
      provide communication.
   2. Attempt to build each relay in this
      set (Section 3.3)
   3. If cost dramatically exceeds
      expectation then regenerate the least
      costly set of edges.
```

Although the problem we are attempting to solve is expressed in terms of rovers and communication relays, we can formalize it using set notation. Given the set of all rovers ρ, we begin by defining a set **P** containing the sets C_1 through C_n each of which initially contains a single element r_n from the set ρ corresponding to the nth rover in the input. We define the function *Comm* from $(\rho \times \rho)$ to that is true whenever the pair of rovers corresponding to the given elements can communicate with each other. We *also define a similar function Requests from* $(\rho \times \rho)$ *to* that is true when either of the rovers specified has requested communication with the other.

3.1 Initial Clustering and Build Order

The first step in the computation is to compute the pre-existing clusters of rovers. To perform this step, we compute the transitive closure of the binary equivalence *relation* ϕ *on* **P** *defined such that it contains pairs* $\{(C_a, C_b) | Comm(r_a, r_b)=T, r_a \in C_a, r_b \in C_b\}$, essentially a relation between those clusters that have a direct communications link. We replace the contents of **P** with sets containing the equivalence classes of this relation. Each element of **P** now corresponds to a set of rovers that already have the ability to communicate internally. Furthermore, no two elements of **P** have any means of communication.

Finding these pre-existing clusters facilitates identifying the communication links that must be created in order to satisfy all communications requests. We define a new set Ω containing the equivalence classes of **P** under the transitive closure of the binary equivalence relation θ defined such that it contains pairs $\{(C_a, C_b) | Requests(r_a, r_b)=T, r_a \in C_a, r_b \in C_b\}$. This divides the clusters of rovers into groups for which desired communications would be entirely internal. We wish to construct a separate communication network for each of these groups.

This further narrows our selection of communications links that must be constructed since we need only ensure that all the clusters in each element of Ω form a single equivalence class under the relation ϕ. To do this, we need to build links only between clusters in the same communications network. Consider the graph **G**, the nodes of which correspond to the elements of a single communication network. An edge (C_a, C_b) exists if $C_a \phi C_b$. Initially this graph will have no edges, since initial clusters are not in communications with one another. Since the transitive closure of a spanning tree is a fully connected graph, constructing a spanning tree on this graph will satisfy all communications requests in the corresponding communication group since there will be some route between any pair of clusters.

The remaining task is to construct a spanning tree on each communications group at minimal cost. We create an estimated construction cost for each potential edge (C_a, C_b) using a heuristic based on the estimated cost of moving between the nearest pair of rovers, $\{(r_a, r_b) | r_a \in C_a, r_b \in C_b\}$, and the minimal number of rovers that could possibly create this edge based on the distance between this pair. Using these estimates we construct a minimum spanning tree on **G**. If we can place relays to build all the edges in this minimum spanning tree then all communications requests will be satisfied.

We attempt to build each of these edges starting with the largest estimated cost. The decision to proceed from largest to smallest is motivated by two considerations. First, the edges with the highest cost will be the most difficult to build. If we build them first, when there are more available rovers and thus more diversity in the resources that can be allocated to solve the problem, then we expect more efficient solutions for the more expensive portions of the problem. This should reduce overall cost. Second, if an edge cannot be built, or overruns its cost by a preset threshold, the algorithm revises its estimate for the cost of that edge, regenerates the minimum spanning tree, and begins building edges for the given communications group from scratch. These failures are more likely to occur for longer edges, and thus we expect to lose less work, in general, if we build these edges first.

3.2 Heuristics

In this section, we discuss the six heuristics that are used to estimate the costs of potential plans. Each heuristic function takes as input one or two points on the terrain and computes a scalar value as output. When more than one heuristic function with output bounded

between zero and one is to be combined into a composite value, we calculate that composite using the formula

$$\frac{\sum_i C_i H_i^{E_i}}{\sum_i C_i}$$

where H_i is the value of the i^{th} heuristic to be combined, C_i is a weighting coefficient for that heuristic, and E_i is an exponent which can be used to modify the response curve of a given coefficient. A radix sort is used to compare unbounded heuristics.

An important heuristic used both in the edge cost estimation that takes place during the initial clustering and in the edge-building routine is movement cost. This heuristic attempts to approximate the difficulty of moving between any two points on the map. Starting from either point, we compute the shortest path to the other point, assuming that the more sloped the terrain is the more costly it is to cross.

Another heuristic, used only in the initial estimation, is the minimum number of rovers needed to connect two points. This is the distance between the points in the x-y plane divided by the range of the rover communications equipment, rounded up. This heuristic provides useful discrete steps in its cost function that allow us to distinguish more strongly between cases with similar movement costs but in which a smaller number of rovers could be used in one of the cases.

The third heuristic is the normalized height of any given location on the map. This heuristic, which we refer to as *height cost*, is useful when we wish to encourage placing rovers at, or near, the tops of hills, since these positions are less obstructed by nearby terrain, given them a greater visible area, and thus tend to lend themselves to reuse in other links.

The fourth heuristic is used to represent the cost of allocating additional rovers. Its value is one whenever moving a particular rover to a particular cell would increase the number of rovers used overall. The overall rover count is not increased when the rover has already been allocated or is being considered as a relay in its initial position.

The fifth heuristic is the normalized move distance. Given two points on the map we calculate the length of the shortest path between the points as a percentage of the length of a straight line between them, accounting for variation in terrain height. This provides an estimate of the difficulty of the terrain between the points.

The sixth and final heuristic is the progress value. Given a proposed move, the progress value indicates the movement towards the destination cluster that this move makes relative to the maximum possible movement, while remaining in communication range of the source cluster. The progress value encourages placement of rovers along a straight line connecting the source and destination clusters, while most of the other heuristics encourage placing rovers in favorable positions relative to the terrain.

We have found that these heuristics are useful in estimating which edges are harder to build, as discussed in 3.1, and in guiding the recursion of the edge building procedure discussed in 3.3. The experiments section (5) describes how changes to these weights can affect performance.

3.3 Building Edges

This section describes the algorithm used to connect two clusters by moving rovers into positions where they act as relays. These chains are built using a recursive method that places one rover at a time. The recursion proceeds until a certain confidence threshold has been reached indicating that the present solution, if any, is expected to be nearly optimal.

The key to efficiently building lines of communication between clusters is the way constraints on the positions of rovers can be naturally and efficiently expressed through their inclusion in multiple clusters. We maintain the invariant that all rovers in a given cluster have some means of communicating with each other using as relays only rovers in that cluster. When a rover is a member of two clusters this invariant ensures that these two clusters are in communication. We can constrain two clusters to be in contact by placing the same rover in both clusters. Later, we can determine if a proposed rover movement violates these constraints by checking only the internal communications consistency of each cluster to which the rover in question belongs. Thus, the algorithm can consider moving rovers that are already part of a relay chain to support a different relay, as long as this connectivity constraint is not violated.

The edge building procedure takes as arguments the current leading cluster in the relay chain, the destination cluster, a partial solution, and the current best solution. The procedure iterates through each map cell the leading cluster can communicate with. Each of these positions is rated for each available rover according to a combination of all the heuristics (except the minimum rover count). If a rover's existing constraints prevent it from moving to the cell in question, it is not evaluated.

If any of the evaluated positions permit communication with the destination cluster then this move is a possible solution. The solution is constructed by adding that rover to both the destination cluster and the current leading cluster, then removing it from its initial cluster, if it had not been moved already. If the cost of

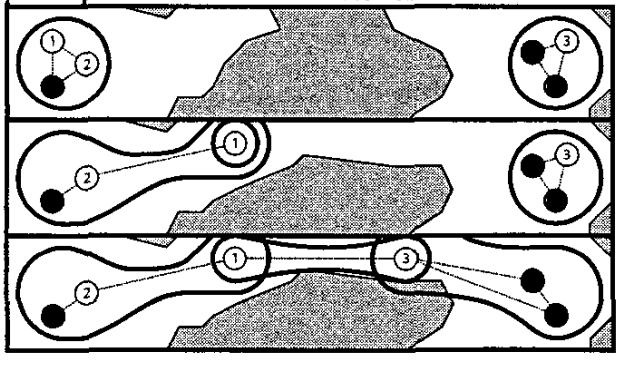
this proposed solution is less than the cost of the best-known solution, the proposed solution is returned immediately. This is what occurs in the third step of the figure at the bottom of this page where rover 3 is added to the leading edge cluster (which previously contained only rover 2) and to the destination cluster.

If no immediate solutions are found, then all the cell-rover pairs that were considered are inserted into a priority queue based on the heuristic cost of those pairs. Those cells whose progress value is less than a given threshold are not enqueued. This is to ensure that the algorithm makes progress towards a solution. A random shift is applied to each cost to increase the spatial diversity of cells near the top of the queue (Cicirello and Smith, 2002).

The second phase of the edge building procedure is a loop that continues until the confidence threshold is reached. To begin, the confidence value is set to zero. Then, during each iteration an element is removed from the priority queue and used as the basis for the next level of recursion. Depending on the results of this recursion and the current recursion depth the confidence value is increased by a certain amount.

Once a rover-cell pair is removed from the queue, the rover's location is temporarily set to the location of the cell it is paired with. Additionally, the rover is removed from its initial cluster if it is still a member of that cluster, added to the leading cluster, and made the sole member of a newly created cluster. This is the transition that takes place between the first and second steps in the figure below, in which rover 2 is removed from its initial cluster, moved to a new location, added to it own new cluster and to the leading cluster, which in this case is the initial cluster it was removed from. The edge-building procedure is then invoked with this new cluster as the leading cluster. Once the procedure finishes, these changes are undone.

This proceeds until the queue is empty, the confidence value reaches the threshold, or the marginal cost of any of the remaining cell-rover pairs in the queue added to the cumulative cost of all moves in the current partial plan would exceed the cost of the best known solution. In any case, the best known solution is always returned.



Once the entire recursive procedure has completed, we have a solution in the form of a number of rover movement commands and new cluster memberships for the rovers. If the cost of building the edge was low enough, relative to the estimated cost, we proceed to the next edge in the minimum spanning tree of the current communications group. However, if the cost of the solution exceeds a constant multiple of the estimated cost, then the minimum spanning tree on **G** is re-evaluated using the cost found by the edge-building algorithm. We continue to attempt to build edges until the entire minimum spanning tree is built or we have restarted the construction of a given minimum spanning tree a set number of times, in which case the situation is considered unsolvable.

4. Implementation

The algorithm, its user interface, and its interface to the FIRE architecture are all implemented in C++ for a target system running UNIX and X11R6. Some aspects of the implementation of the algorithm are worth noting alongside a discussion of the algorithm itself, as they are vital to its useful operation. We also describe the interface to the rest of the FIRE architecture as it provides a framework for using the algorithm in a system with changing communication needs and rover configurations.

4.1 Caching Scheme

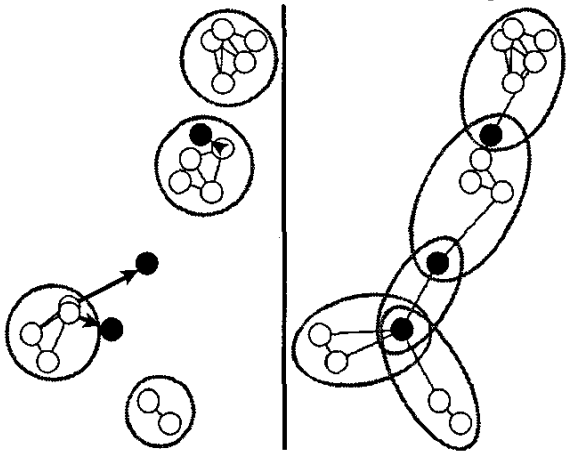
During execution we make heavy use of visibility data and movement cost data, which is time consuming to compute on the fly. A caching scheme makes it possible for the algorithm to have a reasonable running time. The algorithm typically makes requests for the movement or visibility data from a small set of points corresponding to the previous cluster in the recursion or the current positions of each of the rovers to a much larger set of points corresponding to the possible moves that could be connected to those clusters. Thus, it made sense to implement a FIFO caching scheme to exploit the regularity in these requests. Because visibility and movement cost data can be computed more efficiently in larger chunks and because the same locations tend to show up in many visibility requests, we compute this data between a point and every other point on the map and store that as a single entry in the cache. Currently, we consistently achieve over a 99.5% hit rate with a 100 element cache.

4.2 The FIRE Architecture

We have built an interface between the FIRE executive layer and the communications planner that allows the operation of the algorithm to be tested in a fully simulated environment. The planner operates in a loop beginning with a situation in which all rovers are in communication. When rover agents broadcast their new desired positions, the communications planning agent generates a plan using these positions as input that will result in a new connected configuration. The agent then sends this plan to its executive layer, which distributes the movements to individual rover agents for execution.

This cycle repeats once the rovers reach the new configuration. It should be noted rovers can leave the network by not requesting any communication. A stationary base provides a known relay point for such rovers to reconnect to once they have completed roaming.

The figure below illustrates the results of an actual run of the algorithm. The left side of the figure corresponds to the initial configuration of the rovers, with their eventual movements indicated by the black circles. The right side of the figure corresponds to the final configuration reached by the algorithm. Grey ellipses indicate the clusters into which the rovers are grouped,



and dotted lines indicate lines of communication.

5. Results

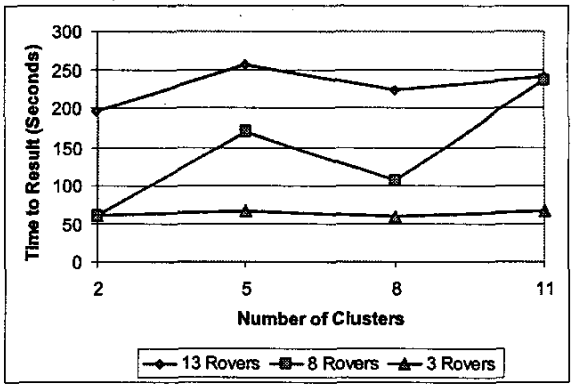
We performed a series of experiments to determine the performance of the algorithm in a variety of situations and to assess the sensitivity of the algorithm to the tuning of the heuristic coefficients. Section 5.1 describes the changes in the performance of our algorithm in response to changes in the input to the algorithm. This suggests how well the algorithm can be expected to perform in different scenarios. The results in section 5.2 illustrate the sensitivity of the algorithm to changes in its internal parameters and some of the tradeoffs that can be made by altering these variables.

5.1 Input Variation

We varied four properties of the input. These were (1) the random seed used to generate rover position, cluster position, and map elevation data; (2) the number of rovers present on the map; (3) the number of clumps those rovers were placed in; and (4) the difficulty of the terrain. The rover configuration is based on the random placement of rovers into a predefined number of small clusters, meant to approximate the distribution of rovers working on a number of different tasks. Terrain difficulty ranges from perfectly flat to very rough terrain with many obstacles and obstructions to line of sight.

There were many cases where our algorithm did not find a solution for a given input. However, when a subset of these inputs was examined by hand they appeared to be unsolvable in each case. Since a brute force solution to the rover communication problem is intractable, it cannot be confirmed that the algorithm succeeds in nearly all solvable cases, but that is our best conjecture at this point.

We found that increasing either the number of rovers or the number of clusters of rovers tends to increase the time the algorithm takes to produce a result (a declaration of "no solution" is considered to be a result). The figure below illustrates the confounded relationship between these two variables. We see that for high (13) and low (3) numbers of rovers the number of clusters has relatively little effect on the time to solution. However, for intermediate numbers of rovers the time increases dramatically with the number of clusters.



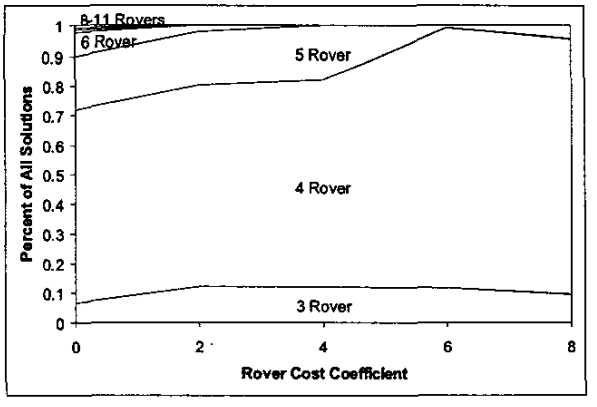
This behavior makes sense when we consider that the number of clusters cannot possibly exceed the number of rovers. Thus we expect the 3-rover case to take nearly constant time regardless of the number of clusters we attempt to form. Nor is it unreasonable for the solution time to remain relatively constant for a case with many rovers. Here there is a tradeoff between building a single

edge that requires many rovers to be evaluated in the two-cluster case and building multiple edges that will generally be shorter since the rovers are more evenly distributed. The increase in time for an intermediate number of rovers is also reasonable if we assume that the effect of the increased number of edges to build outweighs the benefits from the more evenly spread initial distribution for this number of rovers. We would expect that for even greater numbers of rovers there would be a negative correlation between the number of clusters and time to solution, since the even distribution of the rovers would create decreasing numbers of initial clusters until at some point we would expect the rovers to form a single cluster based on their initial locations.

5.2 Internal Parameter Variation

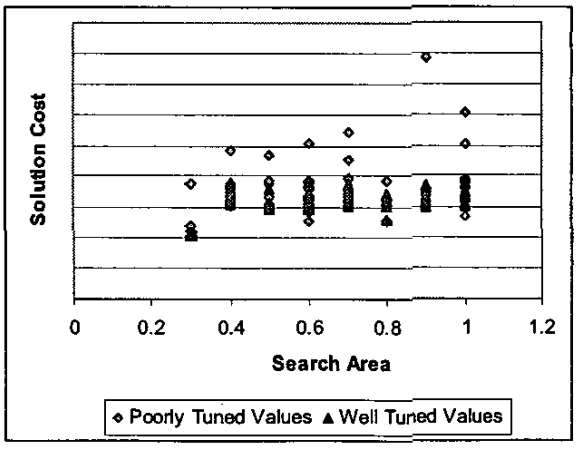
This experiment involved varying four internal parameters of the algorithm and observing the effect these changes had on performance for a single input. The parameters we varied were the height value coefficient, the progress value threshold, the rover cost coefficient, and the confidence depth coefficient. Higher values of the height value coefficient correspond to lower cost for placing rovers at high elevations during edge planning. Increasing the progress value threshold expands the subset of the viewable area that the edge building procedure considers searching recursively. The rover cost coefficient increases the cost of adding a rover to a solution. Finally, the confidence depth coefficient controls the rate at which the confidence threshold value shrinks as the recursion depth increases. Higher values here mean the algorithm will truncate a given search branch earlier for each search depth.

We observed that increasing the rover cost value narrows the distribution of the number of rovers used in the solution and helps eliminate the generation of solutions with exceptionally large numbers of rovers used. It does not, however, seem to have a significant effect on the expected number of rovers used in the solution. The graph below shows the change in the percent of all solutions with a given number of rovers in response to changes in the rover cost coefficient. While we might expect to find a significant decrease in the mean number of rovers used and a possible decrease in the mode, we found neither. It appears that using a moderate rover coefficient is sufficient for generating solutions as good as those with higher rover costs, while still eliminating the outliers associated with exceptionally low values for the coefficient.



The result that yields the most compelling insight into the operation of our algorithm is the relation between search area, which is controlled by progress value threshold, and solution cost. It seems reasonable to expect that increasing the search area, in effect the breadth of our search, should always result in less costly solutions. However, when we look at the data relating the combined rover allocation and movement costs to the search area we see a positive correlation. Overall, the solutions we generate actually tend to get worse as the search area increases.

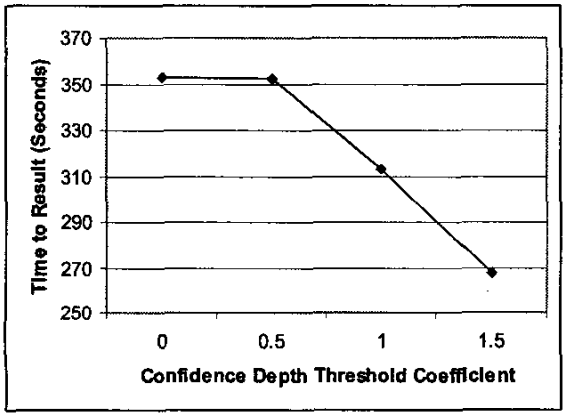
The explanation turns out to be that the algorithm is sensitive to poorly tuned heuristic coefficients. The graph below shows the relationship between search area and solution cost for two sets of data. The "poorly tuned" set is composed of runs for which the height value was set higher then the rover cost. The "well tuned" set consists of runs for which height value was significantly less than rover cost. What is evident is that the data points showing the greatest positive correlation to the search area are all members of the "poorly tuned" set. Meanwhile the well-tuned set shows no clear correlation between solution cost and search area.



This suggests that when the search area is small the edge building routine is essentially forced to pay attention to only a small set of terrain elements that are generally good choices for recursion. However, when the

area is large the algorithm is much more susceptible to poorly tuned heuristics, which cause it to recurse on terrain elements that are poor choices.

The figure below illustrated the relationship between increasing confidence depth threshold coefficient and the time the algorithm requires to complete. Increasing depth threshold confidence coefficient does not change the time that the algorithm takes to reach a solution for low values of the coefficient. For values of 1 and above we observed a nearly linear decrease in the amount of time needed to find a solution. This suggests that decreasing the coefficient below a certain point is changing the timing limit to a dependence on the search area rather than the confidence threshold. This would occur if instead of confidence reaching threshold we found that the movement priority queue was being



emptied in most cases.

6. Summary and Future Work

We have presented an algorithm that can be used to plan the movements needed to maintain limited range line of sight communications within a team of rovers. We have shown how the performance of this algorithm responds to a range of possible inputs, and that the algorithm continues to find plans throughout this range. Finally we illustrated some of the trade offs that can be made in tuning the algorithm if a certain type of performance is desired.

We plan to investigate a number of approaches to further improve the performance and robustness of the algorithm including using selectively down sampled terrain maps, reducing the area of the map in each cache element, and taking into account signal degradation at longer ranges.

Of particular interest is the possibility of reducing the amount of data cached in the visibility maps. If this were limited to a constant amount of data corresponding to a local patch of the terrain the algorithmic dependence on the size of the terrain would be removed entirely. That is, the time to find a solution for a rover distribution on a small patch of terrain would remain constant if additional terrain were added outside this local area.

1. R. Simmons, T. Smith, M.B. Dias, D. Goldberg, D. Hershberger, A. Stentz, and R.M. Zlot, "*A Layered Architecture for Coordination of Mobile Robots*," Multi-Robot Systems: From Swarms to Intelligent Automata, Proceedings from the 2002 NRL Workshop on Multi-Robot Systems, Kluwer Academic Publishers, May, 2002.

2. V. Cicirello and S. Smith, *Amplification of Search Performance through Randomization of Heuristics*, Principles and Practice of Constraint Programming: 8th International Conference, Proceedings, Springer-Verlag, Vol. LNCS 2470 of Lecture Notes in Computer Science, September, 2002, pp. 124-138

3. Ronald C. Arkin, Jonathan Diaz, "*Line-of-sight constrained exploration for reactive multiagent robotic teams*," AMC 7th International Workshop on Advanced Motion Control, 2002

4. Bhargav R. Bellur, Mark G. Lewis and Fred L. Templin, "*An Ad-hoc Network for Teams of Autonomous Vehicles*," Procedings of IEEE 2002 AINS (Symposium on Autonomous Intelligence Networks and Systems) , 2002

5. R. Fierro, A. Das, J. Spletzer, R. Alur, J. Esposito, Y. Hur, G. Grudic, V. Kumar, I. Lee, J. P. Ostrowski, G. Pappas, J. Southall and C. J. Taylor, "*A framework and architecture for multirobot coordination*," accepted to International Journal of Robotics Research (IJRR), July 2002.,

6. B. Bellur and R. Ogier. "*A reliable, efficient topology broadcast protocol for dynamic networks*," Proceeedings of IEEE INFOCOM, March 1999

7. Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, Jorjeta Jetcheva, "*A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols*," Mobile Computing and Networking

8. Qun Li, Daniela Rus, "*Sending Messages to Mobile Users in Disconnected Ad-hoc Wireless Networks*," Mobile Computing and Networking (2000)

9. K. M. Alzoubi, X.-Y. Li, Y. Wang, P.-J. Wan, and O. Frieder, "*Geometric Spanners for Wireless Ad-Hoc Networks*," IEEE Transactions on Parallel and Distributed Systems, 14(5), May 2003.

10. Z. J. Haas, "*The Relaying Capability of the Reconfigurable Wireless Networks*," VTC'97, Phoenix, AZ, May 4-7, 1997.