BBN TECHNOLOGIES
A Verizon Company

BBN Report No. 8359

Movement Control Algorithms for Realization of Fault Tolerant *ad hoc* Robot Networks

Prithwish Basu and Jason Redi
Mobile Networking Systems Department
BBN Technologies
10 Moulton St., Cambridge MA 02138

August 30, 2002

Prepared For:
Deputy Commanding General, USASMDC
Attn: SMDC-TC-MT-A (Mr. Andre)
Huntsville, AL, 35807-3801
Contract No. DASG60-02-C-0060

and

Dr. Douglas Gage
Defense Advanced Research Projects Agency
Information Processing Techniques Office

Abstract

Autonomous and semi-autonomous mobile multi-robot systems require a wireless communication network in order to communicate with each other and collaboratively accomplish a given task. A multihop communications network that is self-forming, self-healing and self-organizing is ideally suited for such mobile robot systems which exist in unpredictable and constantly changing environments. However, since every node in a multihop (or ad hoc) network is responsible for forwarding packets to other nodes, the failure of a critical node can result in a network partition. Hence it is ideal to have an ad hoc network configuration that can tolerate temporary failures while allowing recovery. Since movement of the robot nodes is controllable, it is possible to achieve such *fault tolerant* configurations by moving a subset of robots to new locations. In this report, we propose a few simple algorithms for achieving the baseline metric of tolerance to node failures, namely, *biconnectivity*. Our algorithms which run in polynomial time transform a connected but non-biconnected network configuration to a biconnected one by hinting certain nodes to move to new positions. We compare the performance of the proposed algorithms with each other with respect to a "total distance moved" metric using simulations.

1 Introduction

Advances in electronics and mechanics have provided the basis technologies required for sophisticated robots. It is well recognized that robots have significant operational advantages over humans as they can perform tasks without requirements for rest, food, shelter or task heterogeneity. This makes them potentially useful in future military exercises on the battlefields (so much so that they may end up undertaking all the missions their human counterparts perform today), in several disaster relief situations (search and rescue), in cleaning cavities and surfaces that are otherwise cumbersome to clean, in collection of soil and samples on the surface of Mars (distributed sensing), and in undertaking routine tasks in flexible manufacturing environments and supermarkets, and in many more scenarios. Most of the aforementioned tasks need collaboration among different robot units for their timely and efficient completion.

Traditionally robotics researchers have proposed the use of *centralized* robot networks where all members of a team of robots communicate with a central controller (base station) over a wireless medium [9]. However, in most application scenarios described in the previous paragraph, it is difficult to guarantee the presence of a wireless base station which can coordinate the flow of information between any two robot units. Moreover, the movement of robots can be severely restricted in order to keep in communication range of the base station, and this can hamper the task that the robot team plans to execute. Hence we believe that self-forming, self-healing, and self-organizing multihop communications networks are ideal for autonomous and semi-autonomous robotic systems.

Although numerous ad hoc network protocols (also called packet radio, MANETs, or self-organizing networks) have been proposed and implemented, all of them were designed to be completely transparent to applications. One of the main reasons for adopting this approach is that the protocols are intended to be used with a wide variety of platforms and applications. The resulting extended applicability, however, comes at the cost of severe restrictions in the exchange of information between the application and the network, which makes it virtually impossible for them to anticipate each other's behavior and thus cooperate. In robotic systems such cooperation is highly desirable because robotic applications generally entail movement, which directly affects the communication network; conversely, the propagation of radio transmissions used for communication may be able to provide an additional means of sensing the environment. Such interaction is a feasible proposition since robots are unique in their integrated design in that the mission control, motion control, and networking protocols are typically all implemented within the same architecture.

Ad hoc networks consisting of robot nodes have a salient difference from standard MANETs: in the former the position and motion of nodes is controllable from other nodes in the network while in the latter, node motion is determined by the owner of the node and is not usually controllable. In this report, we focus on the specific problem of altering the positions of robots in order to achieve a desirable ad hoc network topology starting from an arbitrary initial spatial configuration[1]. In particular, we focus on fault tolerance: our goal is to move a subset of robot nodes from their initial locations to a new set of locations such that the new connectivity graph is more tolerant to node failures than the initial graph. We utilize a baseline property for fault tolerance from graph theory,

[1]Since we generally advocate a close cooperation between the ad hoc network subsystem in a robot and the subsystems controlling its mission and motion, dissemination of node locations is achieved without much overhead.

namely, biconnectivity, and propose a few simple algorithms which attempt to achieve that network property in a distributed manner. We compare the algorithms against each other with respect to a "total distance traveled" metric which should be minimized. We show by simulations that the "Block movement" algorithm completely outperforms the baseline "Contraction" algorithm with respect to that said metric.

The rest of the report is organized as follows: Section 2 introduces briefly the concepts from graph theory that are relevant to our solution approach. Section 3 gives a mathematical formulation of the problem of achieving biconnectivity by movement control of robots, and presents two algorithms. Section 4 presents the simulation results, and Section 5 concludes the report with pointers to future research directions.

2 Preliminaries

In this section we briefly introduce the concepts in graph theory that are relevant to the development of our problem. We then proceed to a mathematical formulation of the problem in both one dimensional and two dimensional settings.

2.1 Graph Theoretic Terminology

A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E such that $E \subseteq V \times V$. In our model, a vertex is a robot node with a location attribute *pos* and a transmission range attribute R. We assume that all nodes have omni-directional antennas and identical wireless propagation characteristics resulting in the same value of R; hence an edge $e = (u, v)$ exists between vertices u and v only if $\|u.pos, v.pos\| \leq R$. Such graphs are also known as *unit disk graphs*. G is called *r-partite* if V admits a partition into r classes such that every edge has its ends in different classes, i,e, vertices in the same partition class must not be adjacent. "2-partite" is usually referred to as *bipartite*.

A non-empty graph G is called *connected* if any two of its vertices are linked by a *path* in G. If $G' \subseteq G$ and G' contains all edges $xy \in E$ with $x, y \in V'$, then G' is a subgraph of G induced or spanned by V' in G. A maximal connected subgraph of G is called a *connected component* of G. If $A, B \subseteq V$ and $X \subseteq V \cup E$ are such that every $A--B$ path in G contains a vertex or an edge from X, we say that X separates the sets A and B in G. A vertex which separates two other vertices of the same connected component is called a *cutvertex* and an edge separating its endpoints is a *bridge*. Thus, the bridges in a graph are precisely those edges that do not lie on any cycle.

G is called $k - connected$ if $|G| > k$ and $G - X$ is connected for every set $X \subseteq V$ with $|X| < k$. In other words, no two vertices in G are separated by fewer than k other vertices. The greatest integer k such that G is $k - connected$ is the *connectivity* $\kappa(G)$ of G. If $\kappa(G) = 2$, then G is said to be *biconnected*, i.e., removal of no vertex of G causes a separation of the remaining vertices.

2.2 Achieving Biconnectivity

It can be easily seen from the discussion in Section 2.1 that *biconnectivity* is a desirable property for a network to have for fault tolerance. In a battle-bot scenario, if the ad hoc network formed by the robots is biconnected, even if any one robot fails or is shot down by the enemy, the network still remains connected. Hence robots should always attempt to form a biconnected network as long as that does not interfere with their current mission. This necessitates movement for some of them (assuming no power control in the radios) in order to *create* extra links such that the resultant topology is biconnected. Figure 1 illustrates the idea. The more general property of $k - connectivity$ is seemingly much harder to achieve, and we do not investigate that in this paper.

Achieving a biconnected configuration in a systematic fashion is difficult from purely local information since the identification of cutvertices of a network requires global knowledge of the network (explained in greater detail in Section 3.3). Hence proactive link state based MANET routing protocols such as OLSR [3] and HSLS [7] where each node keeps knowledge about the rest of the network, are more suitable for our endeavors. Reactive routing protocols on the other hand keep incomplete knowledge of the topology and this poses greater challenges for our schemes. We assume the presence of proactive link state routing protocols in this work.

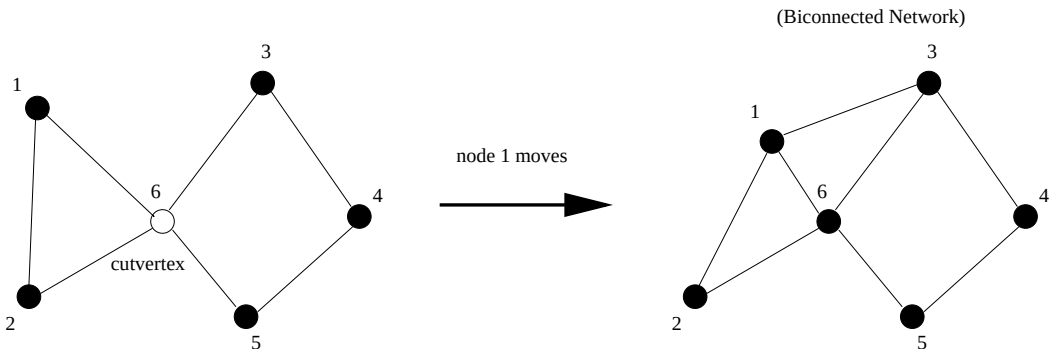


Figure 1: Achieving Biconnectivity by Node Movement

3 Movement Control Algorithms

In this section we present various algorithms for movement of robot nodes with the objective of making the network biconnected.

3.1 One Dimensional Case

We start our analysis with the 1-dimensional version of the problem where all nodes lie on the Real line, and have only one degree of freedom in movement: they can move either to the right or to the left.

Suppose, the initial positions of the nodes are given by $p_m, \forall_m \in [1, N], p_m \in \mathbb{R}$ where N is the total number of nodes. Without any loss of generality, we assume that the transmission range is 1.0. Suppose the positions of the nodes in a new configuration C_{new} are given by $p'_m, \forall_m \in [1, N], p'_m \in \mathbb{R}$. We want to come up with a movement schedule for the nodes such that C_{new} is a biconnected configuration. At the same time we want to minimize the total distance moved by all nodes in the 1D network (an isomorphic metric is average distance moved by a node). We can formulate the problem as follows:

$$\textbf{minimize} \quad D_{total} = \sum_{m=1}^{N} |p'_m - p_m|$$

subject to:

$$\begin{aligned}
p'_1 &\geq p_1; && (1)\\
p'_N &\leq p_N; && (2)\\
p'_m - p'_{m-1} &\geq 0, \quad \forall_m \in [2, N]; && (3)\\
p'_m - p'_{m-2} &\leq 1, \quad \forall_m \in [3, N]; && (4)
\end{aligned}$$

Constraints 1 and 2 are non-binding constraints which just illustrate the fact that the 1-dimensional network will compress in length after a biconnected configuration is reached. The $N-1$ linear ordering constraints in 3 restrict the search space as no node needs to move past its neighbors to achieve biconnectivity. Biconnectivity is ensured by the $N-2$ constraints which basically impose a condition on the nodes that every alternate pairs of nodes are within transmission range of each other. It is easy to see that these constraints are necessary and sufficient for ensuring biconnectivity.

The problem formulated above can be solved optimally. Since the objective function has a non-linear term (absolute value), the problem can be solved using ℓ_1 norm minimization techniques [1, 2], more specifically, using an iteratively re-weighted least squares technique. We do not investigate the 1-D example further in this report and move on to discuss the more interesting two dimensional scenario of robots on flat plane ground.

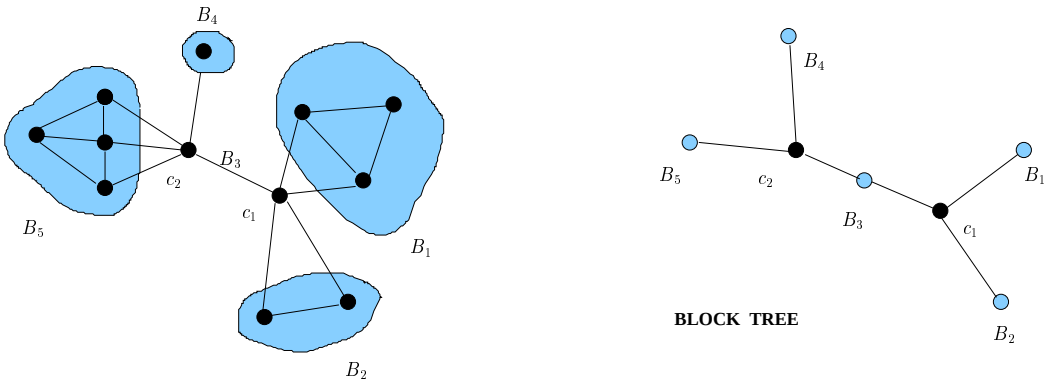


Figure 2: Decomposition into Biconnected Components and the Corresponding Block Tree

3.2 Two Dimensional Case: A Contraction Algorithm

In a higher dimensional setting, "contraction" is a very simple scheme that can be easily implemented in a distributed fashion. Every robot node includes its location information (GPS coordinates or indoor relative location information) whenever it floods an LSU (link state update) to the rest of the network. When all LSUs have arrived at a node X and no more LSUs are arriving from new nodes, X extracts the location information for all the other nodes in the network and calculates the geographic center C for the entire network using the following formula:

$$C = \frac{1}{N} \sum_{m=0}^{N} p_m, \tag{5}$$

where p_m is the position vector of node m. In 1-dimensional space, $p_m \in \mathbb{R}$, whereas in 2-dimensional space, $p_m = (x_m, y_m), \quad x_m, y_m \in \mathbb{R}$. After calculating the geographic center C of the network, all nodes move towards C by a weighted distance calculated as follows: if the contraction parameter is α, a node m with current position p_m moves radially inward towards the center by distance $(1-\alpha)\,\|\vec{C} - \vec{p_m}\|$.

As a result of following this algorithm, a nodes near the periphery of the network move greater distance than the nodes in the interior of the network. In fact the nodes very near the center move very little distance. The rationale behind moving the robot nodes towards the center is that as they move inward, the topology will become richer and richer, and because of the introduction of more edges, the cutvertices will be removed and sooner or later the network will become biconnected.

The choice of parameter α is somewhat important here. If $\lim \alpha \to 1.0$, every nodes moves only a small distance. On the other hand if $\lim \alpha \to 0$, everything collapses to the center of the network eventually. Hence choosing a large α results in unnecessarily dense networks (albeit with higher connectivity than 2!), whereas choosing a small α results in little change in network topology. In the latter case, the contraction algorithm has to be repeatedly applied until the network is biconnected.

Note that each node m travels on the same straight line joining its starting position p_m and C even when multiple iterations are needed to make the network biconnected. Hence when the network reaches a final biconnected configuration with node positions p'_m, the total distance traveled will be given by:

$$D_{total} = \sum_{m=0}^{N} \|\vec{p'_m} - \vec{p_m}\| \tag{6}$$

3.3 Two Dimensional Case: A Block Movement Algorithm

Now we describe more systematic mechanisms for achieving a biconnected configuration which we believe will reduce D_{total} while allowing efficient execution in low order polynomial time.

As mentioned in Section 2.1, removal of a cutvertex breaks a connected graph G into more than one connected components. Hence, the basic rationale behind the algorithm presented in this section will be to consciously remove *all* the cutvertices from the network by moving robot nodes appropriately to new locations. Note that the contraction algorithm presented in Section 3.2 does not attempt to remove the cutvertices systematically.

In Figure 2(left), the biconnected components of a graph have been identified along with its cutvertices. On the right side of the same figure, a corresponding graph whose vertices are biconnected components (or *blocks*) and cutvertices of the original graph has been depicted. Such a graph is referred to as a Block graph [4]. A block graph has the following properties:

P1 A block can have between 0 and N nodes (both inclusive). If two cutvertices are connected by a *bridge*, then the corresponding block contains no nodes. Block B_3 in Figure 2 illustrates this point. If the original graph has no cutvertices, then it is already biconnected and its block graph consists of only one node which contains all N vertices.

P2 A block graph is a *bipartite* graph. The two classes of the bipartite graph are Cutvertices and Blocks. No two cutvertices can be adjacent in the block graph, neither can be two blocks.

P3 A block graph is a *tree*. This can be proved easily. Since the block graph is bipartite, it cannot have an odd-cycle. The presence of an even cycle would mean that some two blocks are connected via two different cutvertices; in that case one of the two cutvertices can be safely removed without disconnecting the graph – thus we arrive at a contradiction and a block graph is a tree.

P4 A block tree of a graph G can be computed in linear ($O(|V| + |E|)$) time. This can be achieved during a depth first traversal (DFS) of G in the same pass [8].

While executing DFS on an undirected graph, we start at an arbitrarily chosen node which becomes the root. We keep traversing fresh edges and mark nodes as "visited"; on the way we keep pushing nodes into a stack data structure. This process is continued until we reach a node which is only connected to already visited nodes. At this point we keep backtracking upto a vertex which has edges connecting them to nodes which have hitherto not been visited. With a little thought it can be seen that such a node will always be a cutvertex of the graph. Alongside the identification of the cutvertex, it is easy to pop the downstream nodes from the stack into a set which corresponds to a biconnected component or a block. Since the above steps can be executed during DFS in the same pass, identification of cutvertices and blocks takes only linear time.

3.3.1 A Heuristic Algorithm for Translation of Blocks

After a brief introduction to algorithms for the identification of blocks and cutvertices in a graph G, we present an algorithm for computing new positions for certain nodes which results in making G biconnected and thus collapses all blocks into a single one.

As described in Section 3.2, every node receives LSU updates from other nodes in the network and extracts their position information from the LSUs. Additionally neighbor information of a node is also extracted from an LSU in order to construct a view of the current network topology. Although in a perfect world, knowing the locations and the transmission range of each radio is enough to construct a view of the network topology, in the real world, one actually needs neighbor information from every node to construct a view. Since LSU packets contain that information anyway, no extra overhead is registered. After constructing a full view of the topology, each node independently computes the block tree BT of the topology graph G. Note that every node should have the same view of the topology hence internal representation of the graph should be consistent so that DFS results in the same block tree at all nodes. This can be easily achieved by a systematic insertion of nodes and edges ordered by node IDs into G at every node.

A salient property of a block is that it is a connected subgraph of G. Hence if all nodes in a block are *translated* together using the same translation vector, distances between all pairs of nodes in that block will remain the same, and hence there will not be any change in the connectivity inside that block. On the other hand, if *some* nodes in a block are translated, it may result in a change of connectivity within the block. Because of this fact we advocate collective translation of all nodes in a block whenever needed so that the connectivity within the block is preserved while progress is made in increasing the connectivity of the network by moving the block itself. The sub-optimality of our scheme stems from this fact as it may not be necessary to move all nodes in a block, but it does result in a faster algorithm.

Algorithm 1 MAKEBICONNECTED(G)

```
1: Given: G
2: G_orig ← G;
3: BT ← COMPUTE_BICONNECTED_COMPONENTS(G);
4: while (NUMBER_OF_NODES(BT) ≠ 1) do
5:    MARKROOTBLOCK(BT);                    /* Select ROOT block with maximum number of nodes */
6:    MARKOTHERBLOCKS(BT);                  /* Mark LEAF, INTERMEDIATE blocks and parents */
7:    MOVE_LEAF_BLOCKS(G, BT);              /* Algorithm 2 for translating leaf blocks */
8:    BT ← nil;
9:    RECALCULATE_EDGES(G);
10:   BT ← COMPUTE_BICONNECTED_COMPONENTS(G);
11: end while
12: G is biconnected now;
13: D_total ← CALCULATE_DISTANCE_MOVED(G_orig, G);
```

Algorithm 2 MOVE_LEAF_BLOCKS(G, BT)

```
1: Given: G, BT
2: for all nodes blk ∈ BT do
3:    if (blk is a BLOCK node and a LEAF) then
4:       par_cv ← BT.parent[blk];                          /* parent cutvertex */
5:       par_blk ← BT.parent[par_cv];                      /* parent BLOCK */
6:       if (NUMBER_OF_NODES(BT[par_blk]) ≠ 0) then
7:          nearest ← FIND_NEAREST_NODE(G, blk, par_blk);  /* find a node in par_blk nearest from blk */
8:          TRANSLATE_BLOCK(BT, blk, nearest);             /* translate all nodes in blk towards nearest */
9:       else
10:         pcv ← parent cutvertex of par_blk;
11:         TRANSLATE_BLOCK(BT, blk, pcv);                 /* translate all nodes in blk towards pcv */
12:      end if
13:   end if
14: end for
```

Which blocks to move and where? Suppose block B_k has edges with two cutvertices c_u and c_v in BT. Let B_m and B_n be two blocks connected to c_u and c_v, respectively. Now, since we want to minimize the D_{total} metric, we should move nodes as little as possible. If we translate B_k towards B_m, c_u may cease to be a cutvertex but some other node in B_n may become one as the link between B_k and c_v may be broken. Hence we may not have made any progress towards reducing the number of cutvertices in G after this translation step. To prevent this from happening, we only translate blocks in BT which have degree 1. In order to heuristically minimize the total distance moved, we choose the block which has the maximum number of nodes as the root of BT, and identify all other blocks with degree 1 as leaf nodes.

Algorithm 1 shows the steps of making a robot network biconnected in a systematic fashion. In every iteration of our algorithm, we attempt to remove a number of cutvertices from BT. In the context of Figure 1, block B_5 will be the root block in BT and B_1, B_2, B_4 will be leaf blocks. All nodes in B_4 will translate towards B_5 since the latter is the parent of the former block. B_1 and B_2 on the other hand have en empty parent block B_3, and hence all nodes in the respective blocks will translate towards the parent cutvertex of the parent block, that is c_2. Every block is translated towards the nearest node in the parent block, whenever applicable, by enough distance such that exactly one new edge appears between the current and the parent block. The appearance of this new edge causes the cutvertex between the two blocks to vanish. Hence in one iteration of the algorithm, several cutvertices are removed. The time complexity of finding the nearest node as a target edge partner requires B^2 comparisons if B is the average number of nodes in a block. Since $B = O(|V|)$ in the worst case, the FIND_NEAREST_NODE function has a worst case time complexity of $O(|V|^2)$.

For large networks several iterations may be needed to remove layers of cutvertices before only one block remains. Also, after every iteration as the number of blocks increases, the blocks grow in size. Hence a small

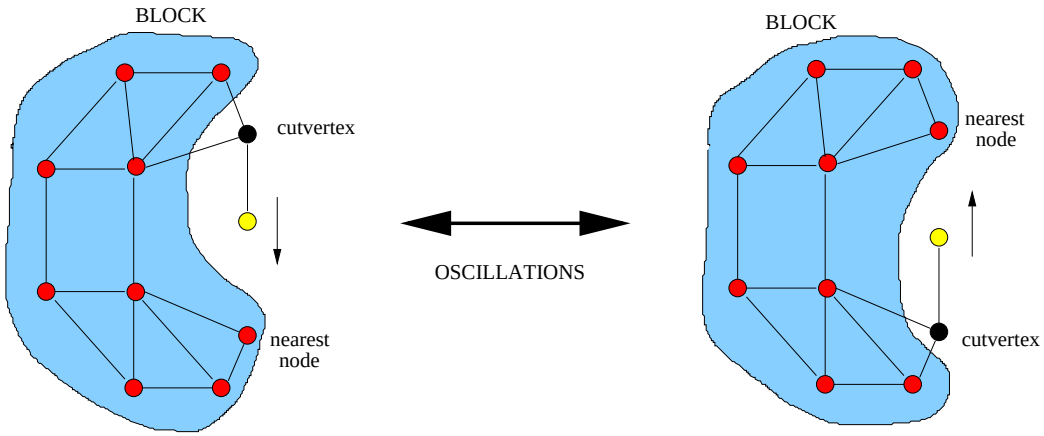


Figure 3: Exception to the Block Movement Scheme

translation by a large block may contribute a significant amount to D_{total}.

In the worst case, we can have $O(|V|)$ iterations of the while loop in Algorithm 1 before achieving a biconnected configuration (e.g. case of a line graph). However since the number of iterations is bounded, convergence is guaranteed in almost all situation except very special cases as depicted in Figure 3. This special case can be solved by translating the block towards the nearest node which is a direct parent of the cutvertex. Although doing this repeatedly also guarantees convergence, we follow this only when translation toward a nearest node in the parent block does not remove a cutvertex. This is because, the former is likely to take many more iterations before achieving biconnectivity for the whole network.

Note that there can be two different schemes for moving the robots: (1) robots start moving as soon as a single iteration is over, or (2) no robot node actually starts moving until the final positions of robots have been determined, i.e., after convergence of Algorithm MakeBiconnected(G). Since the convergence occurs rapidly even for large networks, we adopt the latter scheme which is better since it may result in a much lower value of D_{total} due to the vector addition of translation vectors for every node over all iterations of the algorithm.

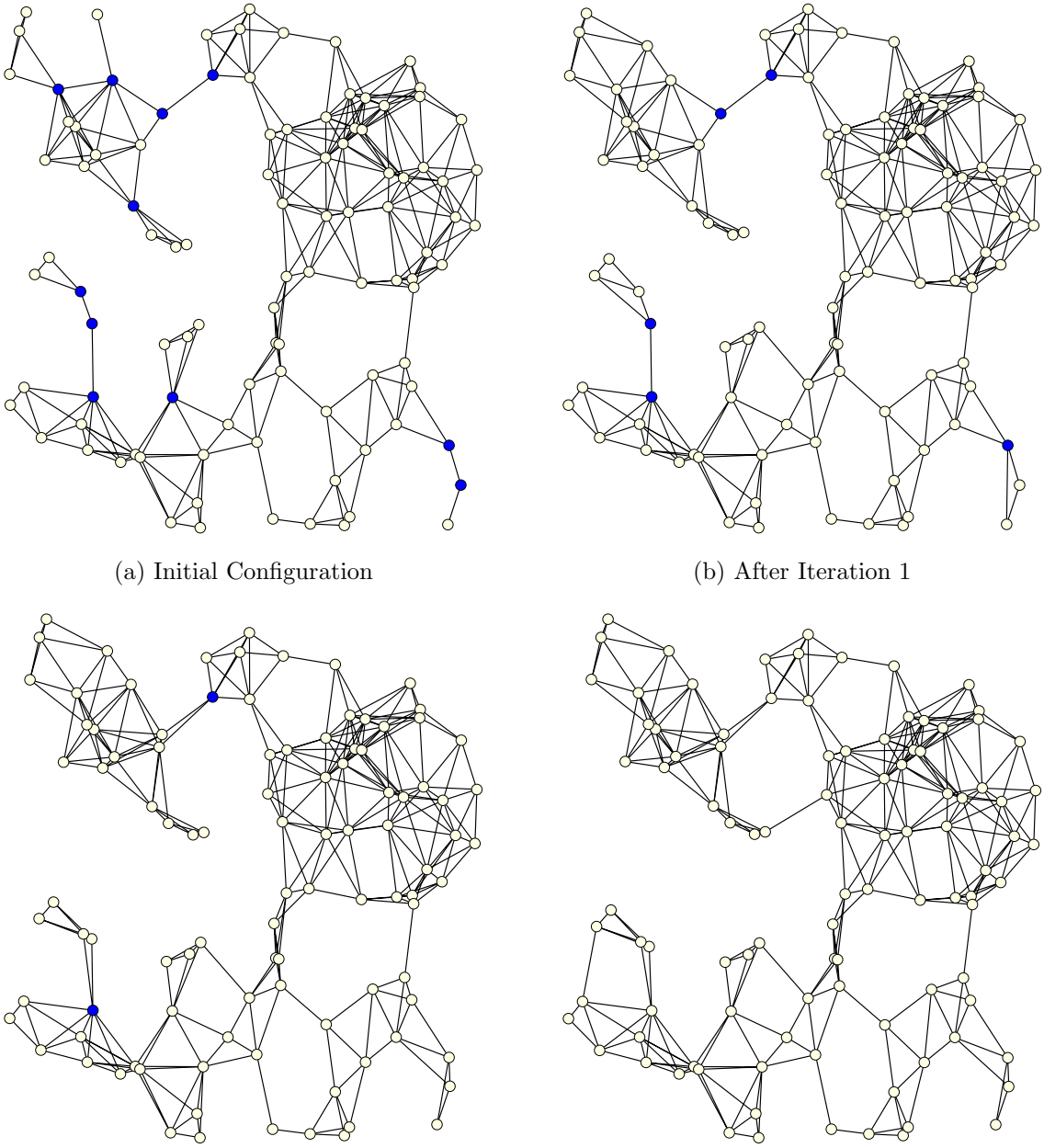
Figure 4 depicts a complete execution of the Make_Biconnected algorithm on a randomly selected initial topology. The dark points represent cutvertices in the network. We can observe the several steps of the algorithm in action as the graph becomes biconnected after only two iterations.

The algorithm described in the previous section attempt to translate leaf blocks only towards their parent blocks or cutvertices in order to remove cutvertices. However, it is easy to conceive of situations where a slight movement towards a non-parent block may cause removal of several cutvertices in a single iteration. Hence a more intelligent block movement scheme can reduce D_{total} as well as the number of iterations in suitable scenarios. At the time of writing this memo, we have only envisaged a scheme which can yield better results than the algorithms proposed in this report, and it is work in progress.

3.4 Related Work

To the best of our knowledge this is the first approach to efficient fault tolerant network design using node movement as a primitive. Ramanathan et al. have proposed optimal schemes for topology control in ad hoc networks using variation of transmission power as a primitive [6]. They monotonically increase transmit power locally at every node and attempt to satisfy properties of connectivity and biconnectivity. Our problem is very different from theirs since the movement of a node aimed at the removal of a cutvertex by creating a new edge can instead result in the creation of a cutvertex at another location near the moved node in the network. Although we do take care during block movement so as not to create new cutvertices, the Euclidean nature of the D_{total} metric makes it is extremely difficult to characterize the optimal solution.

Li and Rus have mentioned the use of node motion in order to relay messages between nodes in [5] but they have not considered the problem of efficient network design. Winfield has proposed the use of MANETs for networking between robots [10] but he too has not considered the problem of efficiently moving robots to achieve a desirable network configuration.



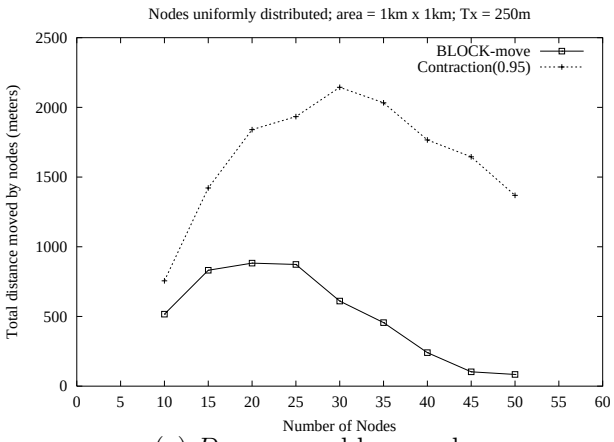
(a) Initial Configuration

(b) After Iteration 1

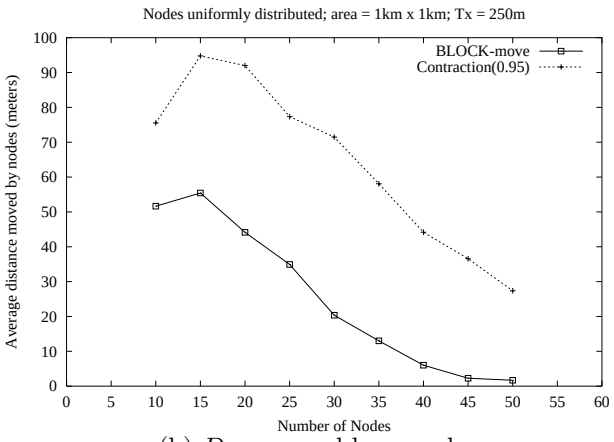
(c) After Iteration 2

(d) Final (Biconnected) Configuration

Figure 4: Execution of the Block Movement Algorithm



(a) D_{total} moved by a node



(b) D_{avg} moved by a node

Figure 5: Distance Moved by Nodes

4 Simulation Results

In this section, we report results of simulation of the execution of our biconnectivity algorithms on random initial configurations. We simulated a $1km \times 1km$ square area with upto 50 robots randomly distributed therein. All robots were assumed to have omni-directional radios with transmission ranges of 250 meters each. The ground is assumed to be flat and devoid of obstacles and trenches thus allowing the robots to move anywhere they want. Harder versions of the problem which include obstacles and imperfect radio propagation are beyond the current scope of this work and are left for future investigation. The initial random configuration of robots obeys the uniform probability distribution while keeping the network connected – we keep generating topologies until a connected topology is found. We simulated 100 runs for every data point with the same parameters.

Figure 5 compares the performance of the block movement algorithm against the baseline contraction algorithm with respect to the distance moved metric while varying the number of nodes (and hence the density) in the network. We can clearly observe that the contraction algorithm is completely outperformed by the block movement algorithm for all values of N considered. This is obviously because contraction is an ad hoc approach which unnecessarily moves every node which adds up to the value of D_{total}.

We observe that the total distance moved increases and then decreases for both algorithms as N is increased from 10 to 50. The reason behind this is that for low values of N, there are only a few nodes that can move and also since the topology is connected, the nodes are not very far from each other. This results in a low value of D_{total}. As N increases, more nodes have to move to make the network biconnected, and this increases D_{total}. However, as N is increased beyond a certain threshold, D_{total} begins to drop significantly. This is because higher values of N result in richer, denser topologies which do not have a large number of biconnected components. In other words, a little movement causes the topology to be biconnected.

We also plot the average distance moved metric in Figure 5(b) against N. The curves look similar to their counterparts for the D_{total} metric apart from the fact that the peaks are shifted slightly to the left owing to division by N. For low values of N there are only a few blocks (see Figure 6) and the initial connectivity is sparse hence a large fraction of blocks have to move in order to make the network biconnected. This results in a high value of D_{avg}. For large values of N, however, the number of blocks reduces as the network is richly connected at many places, and only a small fraction of blocks needs to be moved to make the network biconnected. This results in low values of D_{avg}. In fact for $N = 50$, a node has to move less than 5 meters on average while following the block movement algorithm whereas the contraction algorithm makes nodes move about 30 meters each.

Figure 7(a) shows the number of iterations needed to achieve biconnectivity. We observe that the block movement scheme requires lesser number of iterations than the contraction scheme. To be fair to the latter, if the parameter α is lowered to the 0.7–0.8 range, then a lesser number of iterations should be required. However, in that case, there is a possibility of contracting the robot network more than necessary. Hence we chose a high value for α.

Figure 7(b) illustrates the impact of the block movement algorithm on the diameter of the network which

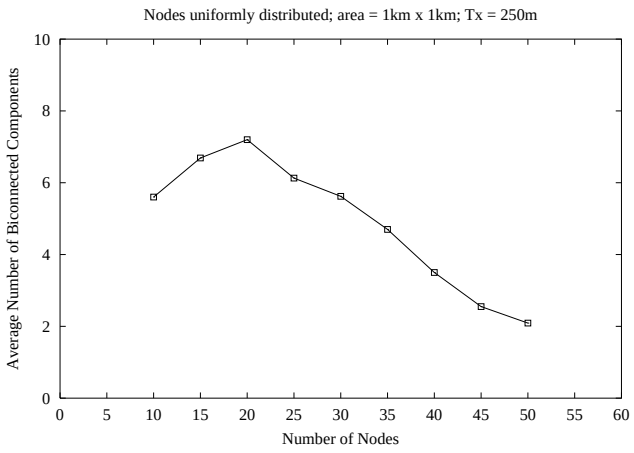


Figure 6: Average Number of Blocks in G

is defined as the maximum length of a shortest path over all source-destination node pairs. As expected, the diameter shrinks for all values of N as it is a monotonic property, in the sense that it can only decrease when edges are added to the network.

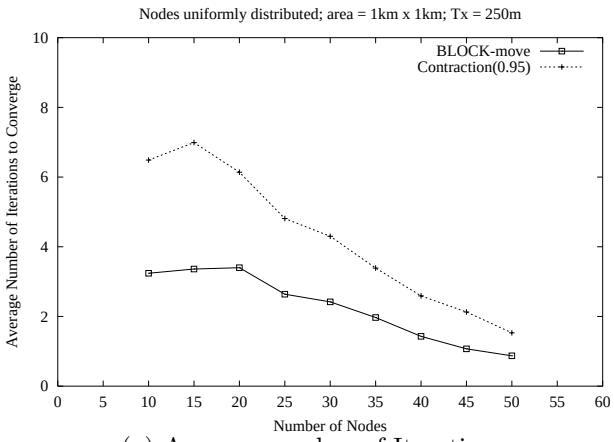
While this simulation study is by no means complete or representative of the real world, it definitely gives us an insight into how block movement algorithms work and how they fare against simple contraction schemes. Other metrics that we have not investigated in this report but feel are interesting are: (1) the number of nodes moved by the algorithm, and (2) fairness in node movement. There can be application scenarios where these metrics are as important as the total/average distance moved metric. We plan to investigate these in future.

4.1 Simulations of Distributed Algorithms

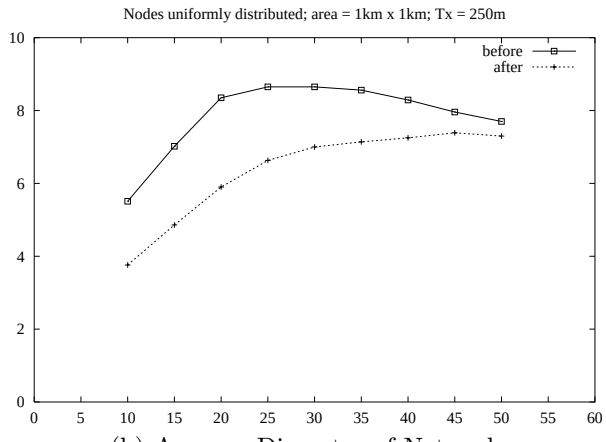
We simulated our algorithms in the OPNET Network Simulator too where all the real world protocols for neighbor discovery, link formation and link state routing were simulated. The distributed version of the algorithm is trivial to construct from a centralized version if we assume that all nodes in the network use proactive link state routing to get information about the entire MANET topology. Each node executes the same algorithm and moves to a new location if the algorithm prescribes it to move there. Nodes which are kept static by the algorithms do not move. We also assume that all nodes start execution of the block movement algorithm at the same time when the LSU floods have died down and the topological views are the same at every node. This precludes the use of the algorithm, as it is, in highly mobile scenarios. We argue that our algorithm will be invoked only under the directive of the mission control subsystem, and the latter will do so only when nodes are not highly mobile. Achieving biconnectivity while nodes are moving to achieve their mission is a harder problem and is beyond the current scope of this report.

5 Conclusions

Fault tolerance is an extremely desirable property in network design, and biconnectivity is a baseline feature in that domain. Since the position and movement of nodes in an ad hoc network of robots are controllable, greater fault tolerance can be achieved by means of moving nodes to locations which results in richer topologies. At the same time nodes should move as little distance as possible insofar the desired topological property is achieved. In this paper, we proposed simple algorithms for moving nodes to new locations such that the resulting network becomes biconnected. We then compared the more sophisticated and systematic algorithm MAKE_BICONNECTED against a baseline heuristic which although very simple and efficient is quite sub-optimal as it cause many nodes to move much greater distances than necessary. MAKE_BICONNECTED on the other hand attempts to remove cutvertices from the graph in a systematic iterative manner. We recognize that finding a polynomial time optimal algorithm is extremely hard and plan to actively pursue this search for optimal algorithms in the future.



(a) Average number of Iterations



(b) Average Diameter of Network

Figure 7: Number of Iterations and Diameter of Network

Acknowledgments

We thank DARPA for sponsoring this work. (Project ERNI is supported under Contract No. DASG60-02-C-0060.) Additionally, we also thank Ram Ramanathan for sharing with us his insights into the problem domain.

References

[1] R. L. Branham Jr., "Alternatives to least-squares," *Astrophysics Journal*, **87**, 1982, pp. 928–937.

[2] T. M. Cavalier and B. J. Melloy, "An Iterative Linear Programming Solution to the Euclidean Regression model," *Computers Ops Res.*, **18**, 1991, pp. 655–661.

[3] T. Clausen, P. Jacquet, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, L. Viennot, "Optimized Link State Routing Protocol," Internet Draft, September 2001. Work in progress.

[4] R. Diestel, "Graph Theory," *Graduate Texts in Mathematics,* **173**, Springer, 1997.

[5] Q. Li and D. Rus, "Sending Messages to Mobile Users in Disconnected Ad-hoc Wireless Networks," *Proc. ACM MobiCom 2000*, Boston MA, August 2000.

[6] R. Ramanathan and R. Rosales-Hain, "Topology Control of Multihop Wireless Networks using Transmit Power Adjustment," *Proc. IEEE Infocom 2000*, Tel Aviv, Israel, March 2000, pp. 404–413.

[7] C. Santivanez and R. Ramanathan, "Hazy Sighted Link State (HSLS) Routing: A Scalable Link State Algorithm," *BBN Technical Memo. 1301.* 2001.

[8] R. Sedgewick, "Algorithms," *Addison Wesley*, 1984.

[9] A. F. T. Winfield and O. E. Holland, "The Application of Wireless Local Area Technology to the Control of Mobile Robots," *Microprocessors and Microsystems*, **23/10**, 2000, pp. 597–607.

[10] A. F. T. Winfield, "Distributed Sensing and Data Collection via Broken Ad Hoc Wireless Connected Networks of Mobile Robots," *Distributed Autonomous Robotic Systems 4, Eds. L. E. Parker, G. Bekey, and J. Barhen, Springer*, 2000, pp. 273–282.

