Integration of Node Deployment and Path Planning in Restoring Network Connectivity

Thuy T. Truong, Kenneth N. Brown and Cormac J. Sreenan
Mobile & Internet Systems Laboratory and Cork Constraint Computation Centre,
Department of Computer Science, University College Cork, Ireland
Email: {tt11, k.brown, cjs}@cs.ucc.ie

Abstract

A wireless sensor network can become partitioned due to node failure, requiring the deployment of additional relay nodes in order to restore network connectivity. This introduces an optimisation problem involving a tradeoff between the number of additional nodes that are required and the costs of moving through the sensor field for the purpose of node placement. This tradeoff is application-dependent, influenced for example by the relative urgency of network restoration. We propose two heuristic algorithms which integrate network design with path planning, recognising the impact of obstacles on mobility and communication. We conduct an empirical evaluation of the two algorithms on random connectivity and mobility graphs, showing their relative performance in terms of node and path costs, and assessing their execution speeds. Finally, we examine how the relative importance of the two objectives influences the choice of algorithm.

Introduction

Wireless Sensor Networks consist of multiple sensing and relay units which communicate with each other using radios, exchanging information, making joint decisions on network and sensing configurations, and transmitting their data over multiple hops to one or more sink nodes which have access to the wider world. WSNs are becoming increasingly important for monitoring phenomena in remote or hazardous environments, including pollution monitoring, chemical process sensing, disaster response, and battlefield monitoring. As these environments are uncontrolled and may be volatile, the network may suffer damage, from hazards, direct attack or accidental damage from wildlife and weather. They may also degrade through battery depletion or hardware failure. The failure of an individual sensor node may mean the loss of particular data streams generated by that node; more significantly, node failure may partition the network, meaning that many data streams cannot be transmitted to the sink. This creates the network repair problem, in which we must place new radio nodes in the environment to restore connectivity to the sink for all sub-partitions.

There are four main subtasks in the problem: (i) determining what damage has occurred (i.e. which nodes have failed and what radio links have been blocked); (ii) determining what changes, if any, have happened to the accessibility of the environment (i.e. what positions can be reached, and what routes are possible between those positions); (iii) deciding on the positions for the new radio nodes; and (iv) planning a route through the environment to place those nodes. The problem thus involves both exploration and optimisation, and depending on circumstances may require the placement of nodes before the changes to connectivity and accessibility have been fully mapped. In this paper, we consider the simpler problem in which we assume the exploration tasks have already been completed, and so our aim is to optimise our use of resources in the static fully observed problem. We assume possible locations for new radio nodes are limited to a finite set of positions where a node can be securely placed and which can be accessed. Radio nodes are expensive, and thus solutions which require fewer nodes are preferred. Physically moving around the environment may be expensive in energy use, may take significant time, or may expose the agent placing the nodes to danger, and thus solutions which allow cheaper path plans are also preferred. Depending on the application, either one of the two objectives may be more important: placing expensive nodes in, for example, agricultural pollution monitoring favours solutions with fewer nodes, while restoring connectivity during disaster response favours solutions that can be deployed quickly even if they require more nodes. Thus the network repair problem is multi-objective.

Our contribution is the novel problem of simultaneous network connectivity restoration with constrained route planning, in the presence of obstacles, and the development and analysis of two heuristic algorithms. We assume a known connectivity graph which includes all possible new node locations and existing nodes, and where the edges indicate that two positions could communicate with each other. We also assume a mobility graph over the same set of positions, but where the edges represent a possible path between two positions. We propose two heuristic algorithms which integrate network design with path planning, and which trade-off the objectives of node cost and path cost. Our first algorithm, called Shortest Cheapest Path (SCP), prioritises node cost, and first finds the minimum number of nodes re-

quired to heal the network; it then finds the cheapest path which visits all the new node positions. The second algorithm Integrated Path (IP) integrates the two objectives by adding weights into the connectivity graph to approximate the mobility cost of establishing each link, and then searches for the cheapest tree that connects all existing nodes. We conduct an empirical evaluation of the two algorithms on random connectivity and mobility graphs. The SCP algorithm tends to find graphs with fewer nodes, while the IP algorithm finds slightly larger solutions but with cheaper mobility costs. The SCP algorithm is significantly faster, particularly on dense graphs. Finally, we examine how the relative importance of the two objectives influences the choice of algorithm.

Preliminaries

An undirected graph G is a pair (V,E), where V is a set of vertices, and E is a set of edges = $\{(x, y) : x \in E, y \in E\}$. We can augment a graph with a cost function, which is normally either a vertex-weight $w : V \rightarrow N$ assigning a cost to each vertex, or an edge-cost $c : E \rightarrow N$ assigning a cost to each edge. A subgraph S of G is a graph $S = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$. The vertex-weight of S is then $\Sigma_{v \in V'} w(v)$, while the edge-cost of S is $\Sigma_{e \in E'} c(e)$. A path P from vertex x_0 to x_n in a graph G is a sequence of edges $< (x_0, x_1), (x_1, x_2), ..., (x_{n-1}, x_n) >$, where each edge $(x_i, x_{i+1}) \in E$. The edge cost of a path P is $\Sigma_{e \in P} c(e)$. A circuit is a path in which $x_n = x_0$. A cycle is a circuit in which $x_0 (= x_n)$ is the only vertex that appears twice. A connected graph is one in which there is a path between every pair of vertices. The problem of finding a minimal cost circuit in an edge-weighted connected graph is NP-hard. A tree is a connected graph with no cycles, and a subtree of a graph is simply a subgraph which is also a tree. For a graph G=(V,E), a spanning tree is a subtree $T = (V, E')$. If we select a subset τ of V (the terminals), then a Steiner tree for τ in G is a subtree $T = (V', E')$ in which $\tau \subseteq V'$. In other words, a spanning tree is a subtree that spans all vertices of a given graph while a Steiner tree is a subtree that spans a given subset of vertices. The problem of finding a minimal Steiner tree in an edge-weighted graph is NP-hard, and remains NP-hard even if all edge-costs are equal. If the edge costs are all the same, then the problem is equivalent to that of finding a minimal vertex-weighted Steiner tree with equal costs.

A wireless network can be represented by an undirected graph G=(V,E), where the vertices represent radio nodes and the edges represent viable radio links between the nodes. We assume all radio links are symmetric, and thus the graph is undirected with at most one edge between any pair of vertices.

The Network Repair Problem

We assume all possible accessible positions for the radio nodes are known, as are the potential radio links between them. Some connected components of the original network will still exist, and our aim is to select enough new positions for radio nodes to create a connected graph. We represent this problem as a set τ, where each $v \in \tau$ is a connected component, and a connectivity graph $G_C = (V, E_C)$, where $\tau \subseteq V$ and each vertex $v \in V - \tau$ is a possible location, and each edge $(v_i, v_j) \in E_C$ represents a potential radio link between the two positions. If $v_i \in \tau$ or $v_j \in \tau$ then the edge represents a potential link from the new position to any radio in the connected component. We assume all radios have the same cost, and so we associate a unit vertex weight function w with G_C, such that $w(v) = 1$ for each $v \in V$, representing the cost of positioning a radio at that position. Our first aim is then to find a Steiner tree S in G_C for τ; that is, a connected set of vertices that includes all components in τ. If we find a minimal Steiner tree (minimising $w(S)$) then we ensure as few radios are used as possible.

We assume accessibility paths are known between the different candidate radio positions, and that there is a known cost of moving between any pair of positions, where the cost may represent time, energy, distance or hazard. We represent this as a graph $G_M = (V, E_M)$ with an associated edge cost function c. For any set $V' \subseteq V$, a circuit P in G_M that visits all vertices in V' represents a tour for the agent, and we can compute the associated path cost. For any given Steiner tree $S = (V', E'_C)$, a circuit P in G_M that visits every vertex $v \in V' - \tau$ then represents a possible tour in which the agent can place all necessary radio nodes to reconnect the network. Minimising $c(P)$ ensures that the cheapest circuit is selected. Note that the two objectives may conflict: a larger Steiner tree may allow a cheaper path, and more expensive path may be required for a smaller Steiner tree.

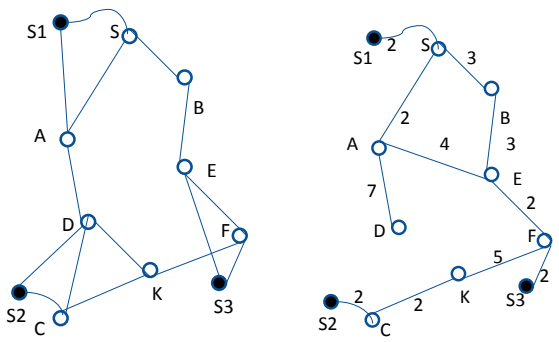
We can now state the formal problem:

- PROBLEM: Network Repair.
- INSTANCE: A graph $G_C = (V, E_C)$ with a unit vertex weight function ($w(v) = 1$, for all $v \in V$), a graph $G_M = (V, E_M)$ with an edge cost function c, and a terminal set $\tau \subseteq V$.
- OBJECTIVE: Find a Steiner tree S for G_C, where $S = (V', E'_C)$, and a circuit P in G_M, such that each $v \in (V' - \tau)$ appears in the path P, which minimises the pair of objectives ($w(S)$,$c(P)$).

As an example, Figure 1 shows a connectivity graph and a mobility graph for a set of terminals $\tau = \{S_1, S_2, S_3\}$ and a set of candidate locations {A,B,C,D,E,F,K,S}. The minimal Steiner tree in the connectivity graph has the vertex set $\{S_1, S_2, S_3, A, D, K, F\}$. However, this is not an easy tree for the agent to create, since there is no short path between D and K. The Steiner tree $\{S_1, S_2, S_3, S, B, E, F, K, C\}$ requires two extra radios, but allows a shorter circuit. Which of these solutions should be selected will depend on the relative cost of the radio nodes compared to the path cost. High radio costs and low mobility costs will prefer the first tree, while high mobility costs will prefer the second tree.

The Shortest Cheapest Path SCP Algorithm

Our first approach assigns an ordering to the objectives. We first try to minimise the number of nodes required to connect all terminals. Given a minimal set of nodes, we then try to



(a) Connectivity Graph (b) Mobility Graph

Figure 1: Example connectivity graph and mobility graph.

find the cheapest circuit for the agent that visits all of those nodes.

Data: A graph $G_C = (V, E_C)$, a set of terminals $\tau \subset V$.
Result: A Steiner Tree $T = (V', E'_C)$ for τ in G_C.
begin
  $G_\tau(\tau, E_\tau, w_\tau) = metric\_closure(\tau, G_C)$;
  $T_\tau = Minimum\_Spanning\_Tree(G_\tau)$;
  $T \leftarrow 0$;
  for *each edge* $e_{uv} \in E(T_\tau)$ **do**
    if *node u and node v are not connected in T* **then**
      $P = shortest\_path\_replace(e_{uv})$;
      if *P contains less than two vertices in T* **then**
        Add P *to* T;
      end
      else
        Add to T all edges in P which are not already in the tree T, and do not create a cycle in T;
      end
      if *T is connected and includes all terminals* **then**
        break;
      end
    end
  end
  return T;
end

Algorithm 1: Steiner-MST Algorithm

Since the Minimum Steiner Tree in Graphs problem is NP-Hard, we use a heuristic algorithm, adapted from (Wu and Chao 2004) (Algorithm 1). First, we create a metric closure graph for the terminals, which is a complete graph over the terminal nodes, where each edge has a weight equal to the shortest path between the two endpoints in the original graph. We obtain the metric closure graph by repeated application of Dijkstra's algorithm. We then find a minimum spanning tree in that graph, using Kruskal's algorithm. We then consider each edge in that tree in turn and, if the end points are not yet connected in the new tree, select the corresponding shortest path from the original graph. If this shortest path contains no more than 1 vertex already added, we add all edges into the tree, with their required vertices; if it contains more than 1 vertex already added, then we add those edges that are not already in the graph, and their associated vertices if needed. The most expensive operation is the creation of the metric closure graph, and thus the entire algorithm has complexity $O(|\tau||V|^2)$.

For the problem of finding the shortest circuit (Algorithm 2), we take all the non-terminal nodes in the Steiner tree created above, and then for those vertices create a metric closure graph from the mobility graph G_M. We then apply (Lawler et al. 1985)'s Greedy-TSP heuristic which is based on Kruskal's algorithm - we sort the edges in increasing order of cost, and we then iteratively add the lowest cost edge which does not increase any vertex's degree to 3, and which does not create a cycle unless it completes the tour. The runtime is again dominated by the time of building the metric closure graph, i.e. $O(|V'||V|^2)$.

Data: A set of vertices V', a graph $G_M = (V, E_M, c)$
Result: a tour in G_M visiting all nodes in V'
begin
  $G'' = (V'', E'', w'') = metric\_closure(V', G_M)$;
  $P = Greedy\_TSP(G'')$;
  return [V'',P] ;
end

Algorithm 2: GreedyTour

Figure 2 shows the SCP algorithm being applied to the example of Figure 1. First we create the metric closure on the terminal nodes, then we create a spanning tree from that (Figure 2(a)). We then extract the corresponding Steiner tree from the connectivity graph (b). We extract the new locations (c), construct the metric closure for those vertices (d), and then create the tour (e), which in this case requires the agent to backtrack out from vertices D and K. This requires 4 new nodes in total, with a mobility cost of 36.

The Integrated Path IP Algorithm

The SCP algorithm prioritises the number of nodes over the mobility cost. Our second approach attempts to combine the two objectives, adding approximate mobility costs into the connectivity graph, and then searching for a minimal Steiner tree on this modified graph (Algorithm 3). First, for each edge in the connectivity graph we compute the shortest path between the vertices in the mobility graph using Dijkstra's algorithm, and add that as an edge cost to the connectivity graph. We then apply the SCP algorithm on this new problem, with the difference being in the step where we create the Steiner tree from the minimum spanning tree of the metric closure, since the edges in the connectivity graph now have non-uniform costs. The runtime is dominated by

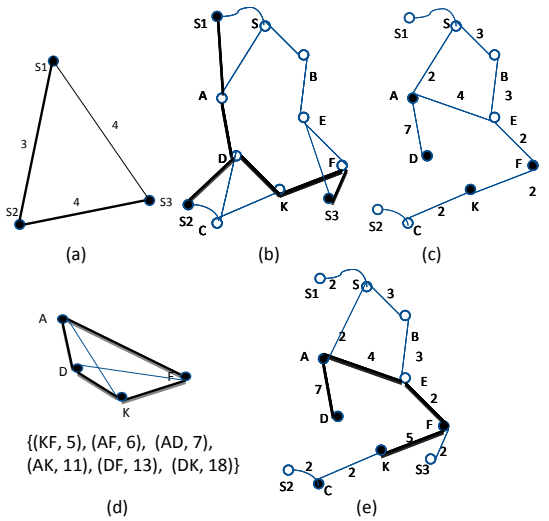


Figure 2: A sample execution of Shortest Cheapest Path

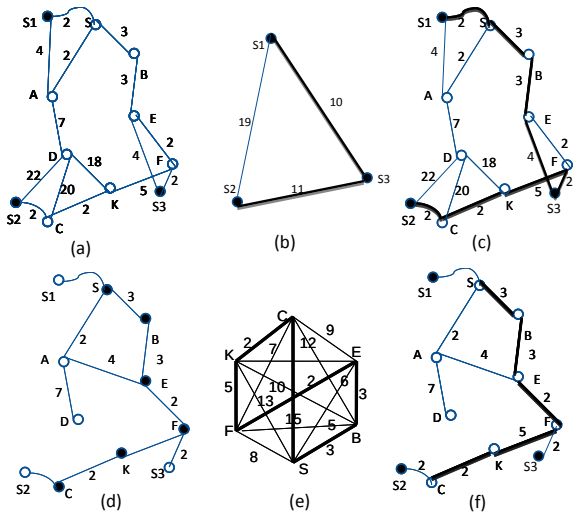


Figure 3: A sample execution of Integrated Path

the cost of building the initial weighted connectivity graph, $O(|E_c||V|^2)$.

Data: A graph $G_C = (V, E_C)$, graph $G_M = (V, E_M)$, an edge cost function c for G_M, a set of terminals $\tau \subset V$.
Result: Number of nodes placed and a tour in G_M visiting those nodes.
begin
  $G_C* = (V, E_C, w*) = w\_conn\_graph(G_C, G_M)$;
  $T = (V', E') = Steiner\_MST(G_C*)$;
  $P = Greedy\_TSP(V' - \tau, G_M)$;
  return [$V' - \tau$,P] ;
end

Algorithm 3: Integrated Path Algorithm

Figure 3 shows the IP algorithm being applied to the example of Figure 1. First we create the weighted connectivity graph (Figure 3(a)), then metric closure on the terminal nodes, followed by its spanning tree from (b). We then extract the corresponding Steiner tree from the weighted connectivity graph (c). We extract the new locations (d), construct the metric closure for those vertices (e), and then create the tour (f). This requires 6 new nodes in total, with a mobility cost of 30.

Experiments

Both algorithms presented above are heuristic, and take different approaches to the multi-objective problem. Therefore, we evaluate them empirically on randomly generated graphs, to compare the quality of their solutions on both objectives, and also on their runtime. For the graphs, our aim is to represent a physical area rather than abstract random graphs, and so we overlay the graphs on a rectangular grid. Connectivity is based on the distance between two locations. To represent a landscape or a building interior, we add obstacles into the grid, which may hinder or forbid access. The mobility graph is then based on line-of-sight, with some limited ability to cross the obstacles.

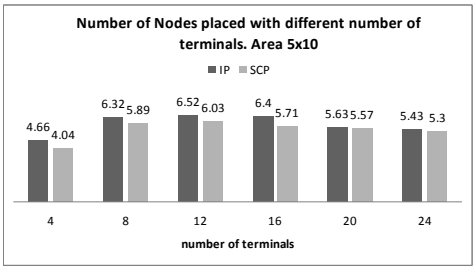
We generate graphs within a rectangular area consisting of $n \times m$ squares of size 10 units. Within this space, we place k mobility obstacles, where each obstacle is a random polygon contained within a randomly selected pair of neighbouring cells. For each square, we generate a random position within it; if that position is inside an obstacle, we discard it, otherwise we designate it as a candidate location. Each obstacle is given a random weight w between 0 and 1, representing the difficulty it creates for the agent to traverse it, and such that any obstacle with a weight greater than 0.2 is assumed to be not able to be traversed. We consider two different problem sizes: (i) a 5×10 grid, and thus a maximum of 50 candidate locations, and 15 obstacles, and (ii) a 10×10 grid, and thus a maximum of 100 candidate locations, and 28 obstacles.

We then create the connectivity graph by adding edges indicating that two candidate locations are within transmission range. For each pair of locations, if they are within 10 units apart, we add an edge with probability 0.85; if they are between 10 and 20 units apart, we add an edge with probability 0.2. For the mobility graph, we add an edge between any pair of locations which are less than 25 units apart and which can be connected by a straight line that does not cross an obstacle. The weight of the edge is simply the length of the connecting line. In order to create more dense graphs with varied mobility costs, we then add extra edges. For each pair of nodes separated by a distance of less than 45 we select it with a probability of 0.2; if the straight line between them traverses any obstacle with a weight greater than 0.2, we discard it, otherwise the cost of the edge is the distance plus 10*weight for each obstacle it crosses.

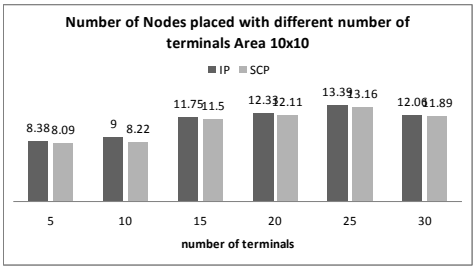
For each of the two problem sizes, for each data point, we generate 50 instances, and present the average solution cost (mobility cost, number of nodes needed) and runtime.

For each instance, we randomly select candidate locations as terminals.

Figure 4 shows the number of nodes placed by each algorithm, as we vary the number of terminals. As the number of terminals to be connected increases, the number of required nodes initially rises to a peak when 25% of the locations are terminals, and then starts to decline: as we add more terminals we require more nodes until the graph becomes sufficiently dense that we can start to re-use the nodes to connect multiple terminals. The IP algorithm requires more nodes, although on average no more than one extra node.



(a) Area 5x10

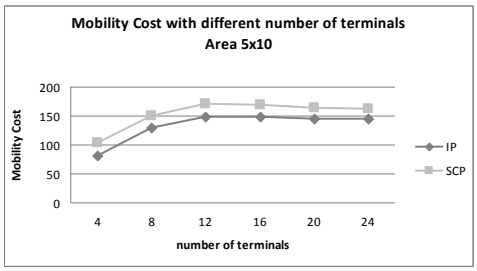


(b) Area 10x10.

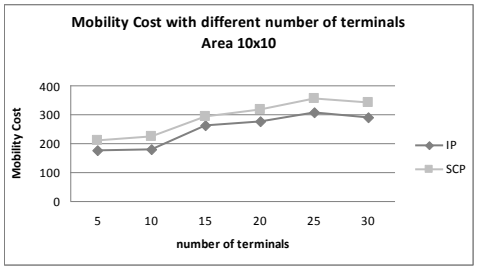
Figure 4: Number of nodes placed with varied number of terminals.

Figure 5 shows the mobility costs as we vary the number of terminals. Again, the costs rise as we increase the number of terminals to approximately 25% of the locations, and then start to decline. IP always produces solutions with lower mobility cost than SCP, ranging from a 10% to a 21% improvement. Figure 6 shows the runtime for the two algorithms. The SCP algorithm is significantly faster, with a speedup factor between 10 and 30.

To decide which algorithm should be selected will require greater knowledge of the relative cost of the new radio nodes versus the mobility costs. We should also take into account the runtime of the algorithm, since as the graphs grow in size, the runtime will become more significant: waiting for the algorithm to complete may remove any benefit gained from a shorter path. If we regard the mobility cost as the time to traverse the circuit, we can then consider the total time for restoring the network as being the runtime of the algorithm plus travelling time of the agent, plus any time



(a) Area 5x10

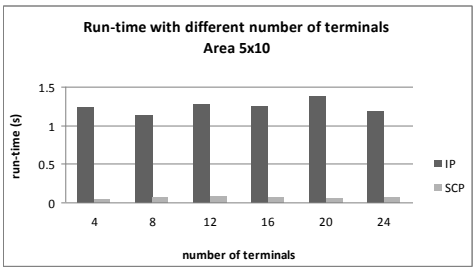


(b) Area 10x10.

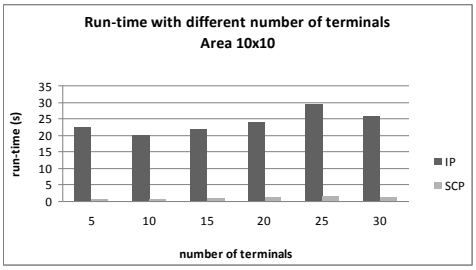
Figure 5: Mobility Costs with varied number of terminals.

required to place the nodes in position. If we assume that the agent is a small robot, then a typical speed might be 0.1ms^{-1}, while the time to place a node might be 30 seconds. Based on the results above, for a 25 terminal problem in the 10×10 grid, where 1 unit is 1m, the total time to repair the network will be 300/0.1 + 13.4*30 +21 = 3423 seconds for the IP algorithm, while it will be 350/0.1 + 13.16*30 + 1 = 3896 seconds for the SCP algorithm. In this case, the IP algorithm will reduce the total repair time by 473 seconds. This reduced time may be significant for network repair during an emergency. For faster moving agents (e.g. a motor vehicle over rough terrain), if we assume an average speed of 4ms^{-1}, the total repair time for IP is 300/4 + 13.4*30 + 21 = 498, while for SCP the total repair time is 350/4 + 3.16*30 + 1 = 483, and thus SCP is slightly faster. For a given problem size, as the speed of the agent increases, we are more likely to prefer SCP.

We also note that the number of terminals for a fixed size problem does not have a significant impact on the runtime, but that there is a significant difference as we increase the network size. Therefore, we perform another experiment in which we vary the density of nodes and edges. More specifically, we vary the maximum number of nodes able to be placed in a single grid square from 1 to 4. During generation, for each square, we randomly select the number of nodes to be placed, and then select their positions. As we increase this parameter, the number of edges increases significantly, and particularly so for the mobility graph. The results are shown in Figure 7, where the x-axis again shows different number

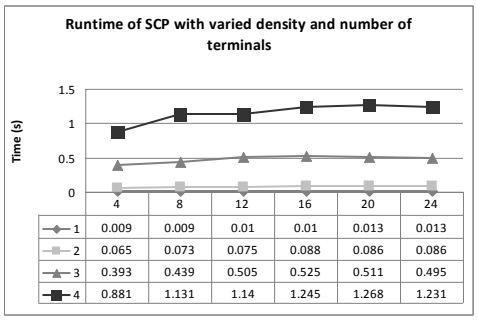


(a) Area 5x10

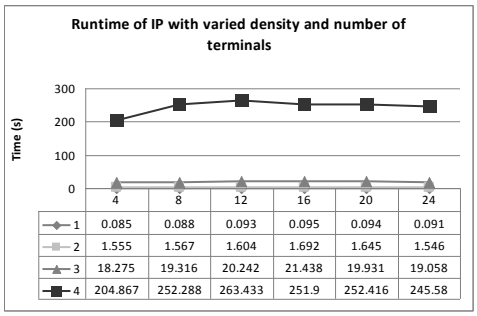


(b) Area 10x10.

Figure 6: Runtime with varied number of terminals.



(a) SCP



(b) IP

Figure 7: Runtime with varied density and number of terminals.

of terminals. For the SCP algorithm the runtime appears to scale with the square of the density parameter. For the IP algorithm, the runtime appears to grow exponentially. Note that the density parameter relates to the density of the nodes packed into the same geographic area, and so a higher value involves more nodes as well has a higher average degree for each node in the graph. The SCP algorithm complexity is the product of the square of the number of nodes and the size of the Steiner tree. As more nodes are added, the graph density increases, and we expect the Steiner tree to grow sublinearly. However, the complexity of the IP algorithm is the product of the number of edges and the square of the number of nodes, and so increasing the density parameter should have a more significant effect, as indicated by the increasing runtime. In Figure 8 we examine the results of density parameter = 4 in more detail. Although the IP algorithm saves about 20% of the mobility costs, it is two orders of magnitude slower in runtime, and requires slightly more nodes on average.

Table 1 gives a summary of the conditions under which we would prefer one algorithm over the other. For high node costs, dense graphs, or fast moving agents, we expect to prefer the SCP algorithm, while for cases where energy costs are significant, or where agents are relatively slow, then the IP algorithm will be preferred.

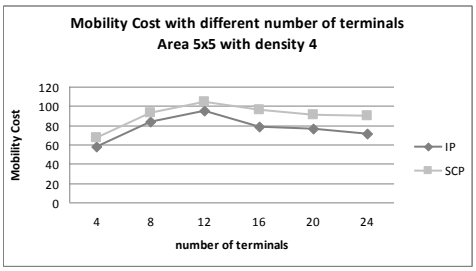
Algorithm	IP	SCP
Energy (Movement)	y	n
Total restoring time with low-medium speed	y	n
Total restoring time with high speed	y	y
Expensive node	n	y
Very dense network ($d \approx 4$)	n	y

Table 1: IP or SCP.

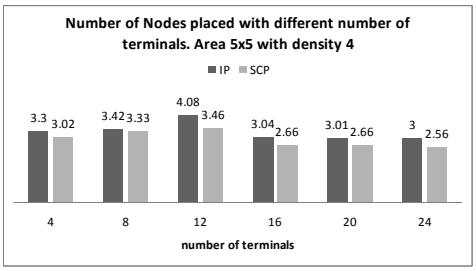
Related Work

The subject of network restoration for wireless sensor networks has been an active area of research. The different approaches can be classified as (i) deploying redundant nodes so as to be able to cope with a pre-determined number of failures, (ii) use of mobile (actor) nodes that can be moved into position in order to restore connectivity, (iii) dispatching mobile nodes in a pre-emptive manner to avoid failures in connectivity, and (iv) the deployment of additional nodes to restore connectivity after failures have occurred. The latter work is closest to our research but is different in two respects, firstly in that we optimise both the number of additional nodes as well as the path length needed for their deployment, and secondly in that we explicitly take into account the impact of obstacles that can alter both the available paths and the ability of nodes to communicate directly. We now provide a summary of the related papers.

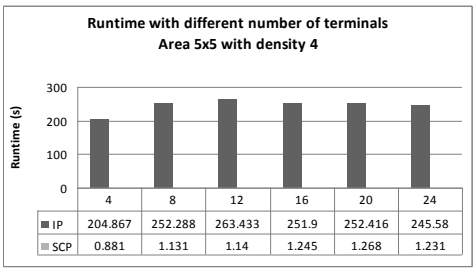
In (Khelifa et al. 2009),(Almasaeid and Kamal 2009) the goal is to deploy k-1 redundant nodes with the intention of



(a) Mobility Cost



(b) Placing Nodes



(c) Runtime

Figure 8: Runtime with density 4 and varied number of terminals.

achieving k-connectivity, for example by placing nodes at the intersection between the communication range of each pair of nodes. The number of additional nodes required by these approaches is prohibitive.

Several papers consider the use of mobile actor nodes in network restoration, e.g. (Abbasi, Akkaya, and Younis 2007),(Akkaya and Senel 2009),(Abbasi, Younis, and Baroudi 2010),(Akkaya et al. 2010),(Sir et al. 2011). The papers propose different strategies to choose the moving actors, for example, based on estimating the shortest moving distance and/or degree of connectivity. The solution space is quite limited, focused on dealing with a single failure and re-connecting just two networks at a time, and with an assumption that mobility is unimpeded by obstacles.

(Dai and Chan 2007) proactively deploys additional helper mobile nodes, controlling their trajectories in response to predicted network disconnection events. The work assumes that the mobile nodes are always fast enough to reach the desired destination in case of a predicted disconnection event, and that a full map of the physical terrain and radio environment is available. Details of how to determine the number of mobile nodes that are needed and the related path planning are not provided.

(Senel, Younis, and Akkaya 2009) and (Lee and Younis 2010) assume multiple simultaneous failures involving many failed nodes and a network that is partitioned into many segments. The approach is to re-connect those segments in a centralized manner with the main objective of using the smallest number of additional nodes. (Senel, Younis, and Akkaya 2009) uses a spider web approach to reconnect the segments while (Lee and Younis 2010) forms a connectivity chain from each segment toward a centre point and then seeks to optimize the number of additional nodes that are needed.

Conclusion

We have defined the new problem of simultaneous network connectivity restoration with constrained route planning, in the presence of obstacles. We formalise the problem as a multi-objective problem of minimising a Steiner tree in a connectivity graph and minimising a tour of the nodes in that tree in a mobility graph. We present two heuristic algorithms, Shortest Cheapest Path and Integrated Path. Shortest Cheapest Path prioritises node cost, and first finds the minimum number of nodes required to heal the network; it then finds the cheapest path which visits all the new node positions. Integrated Path combines the two objectives by adding weights into the connectivity graph to approximate the mobility cost of establishing each link, and then searches for the cheapest tree that connects all existing nodes. We conducted an empirical evaluation of the two algorithms on random connectivity and mobility graphs. The SCP algorithm tends to find graphs with fewer nodes, while the IP algorithm finds slightly larger solutions but with cheaper mobility costs. The SCP algorithm is significantly faster, particularly on dense graphs.

Immediate future work will involve developing a hierarchical routing approach, which will allow us to handle dense mobility graphs, by initially merging adjacent location nodes into a supernode to reduce the complexity. Longer term work will focus on expanding the problem to include the simultaneous exploration of network connectivity and mobility constraints as we optimise the node selection and placement. This will require heuristic algorithms that tradeoff the cost of further exploration against the cost of placing nodes early. We will also consider distributed algorithms, allowing multiple agents and sensor nodes to collaborate to determine the damage to the network. Finally, we aim to tackle problems where the damage is continually changing, and thus the solutions need to be continually regenerated.

Acknowledgment

This work is part of the NEMBES project, which is funded by the Irish Higher Education Authority PRTLIIV programme.

References

Abbasi, A. A.; Akkaya, K.; and Younis, M. 2007. A distributed connectivity restoration algorithm in wireless sensor and actor networks. In *Proceedings of the 32nd IEEE Conference on Local Computer Networks*, 496–503. Washington, DC, USA: IEEE Computer Society.

Abbasi, A.; Younis, M.; and Baroudi, U. 2010. Restoring connectivity in wireless sensor-actor networks with minimal topology changes. In *Communications (ICC), 2010 IEEE International Conference on*, 1 –5.

Akkaya, K., and Senel, F. 2009. Detecting and connecting disjoint sub-networks in wireless sensor and actor networks. *Ad Hoc Netw.* 7:1330–1346.

Akkaya, K.; Senel, F.; Thimmapuram, A.; and Uludag, S. 2010. Distributed recovery from network partitioning in movable sensor/actor networks via controlled mobility. *IEEE Trans. Comput.* 59:258–271.

Almasaeid, H. M., and Kamal, A. E. 2009. On the minimum k-connectivity repair in wireless sensor networks. In *ICC*, 1–5. IEEE.

Dai, L., and Chan, V. 2007. Helper node trajectory control for connection assurance in proactive mobile wireless networks. In *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, 882 –887.

Khelifa, B.; Haffaf, H.; Merabti, M.; and Llewellyn-Jones, D. 2009. Monitoring connectivity in wireless sensor networks. In *ISCC*, 507–512. IEEE.

Lawler, E. L.; Lenstra, J. K.; Rinnooy Kan, A. H. G.; and Shmoys, D. B. 1985. The traveling salesman problem.

Lee, S., and Younis, M. 2010. Recovery from multiple simultaneous failures in wireless sensor networks using minimum steiner tree. *J. Parallel Distrib. Comput.* 70:525–536.

Senel, F.; Younis, M.; and Akkaya, K. 2009. A robust relay node placement heuristic for structurally damaged wireless sensor networks. In *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*, 633 –640.

Sir, M.; Senturk, I.; Sisikoglu, E.; and Akkaya, K. 2011. An optimization-based approach for connecting partitioned mobile sensor/actuator networks. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, 525 –530.

Wu, B. Y., and Chao, K.-M. 2004. *Spanning Trees and Optimization Problems*. USA: Chapman & Hall / CRC Press.