Repairing Wireless Sensor Network Connectivity with Mobility and Hop-Count Constraints

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
{tt11,k.brown,cjs}@cs.ucc.ie

Abstract. Wireless Sensor Networks can become partitioned due to node failure or damage, and must be repaired by deploying new sensors, relays or sink nodes to restore some quality of service. We formulate the task as a multi-objective problem over two graphs. The solution specifies additional nodes to reconnect a connectivity graph subject to network path-length constraints, and a path through a mobility graph to visit those locations. The objectives are to minimise both the cost of the additional nodes and the length of the mobility path. We propose two heuristic algorithms which prioritise the different objectives. We evaluate the two algorithms on randomly generated graphs, and compare their solutions to the optimal solutions for the individual objectives. Finally, we assess the total restoration time for different classes of agent, i.e. small robots and larger vehicles, which allows us to trade-off longer computation times for shorter mobility paths.

Keywords: Sensor Network, Connectivity Repair, Sink Placement.

1 Introduction

Wireless Sensor Networks 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.

In this work, we assume a survey has been completed, and so we know which nodes have failed, which radio links have been blocked, and which routes between positions can no longer be traversed. The tasks that remain are to decide on the positions for the new radio nodes, and to plan and follow a route through the environment to place those nodes. We assume possible locations for new radio

J. Cichoń, M. Gębala, and M. Klonowski (Eds.): ADHOC-NOW 2013, LNCS 7960, pp. 75–86, 2013.

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 so solutions which require fewer nodes are preferred. In addition, the users of the WSN may require data to be transmitted from the sensors quickly, to allow a timely response, and so there will be limits on the number of radio hops allowed between the sensors and the wider network. To achieve this, we may prefer to deploy some expensive sink nodes which provide their own network connection, in addition to relay nodes. 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 so 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.

We introduce the problem of simultaneous network repair with hop count limits and route planning with limited mobility. We assume a set of desired locations from which sensor data is required by the network, and we assume the agent knows the state of the network and accessibility. The objective is to connect as many as possible of these locations, placing extra sensors, relays and sinks as required, minimising the relay and sink costs and the mobility costs, while obeying the constraint on the number of allowed radio hops. We consider two different heuristic approaches for the multi-objective problem, each prioritising a different objective: minimising mobility costs, and minimising the relay and sink costs. We evaluate the two algorithms on randomly generated problems, and analyse their effectiveness under different assumptions. Finally, we consider the total estimated time to restore the network, for two different classes of agent (a small robot and a larger vehicle), and we show that the choice of priority should be dependent on the performance of the agent.

2 The Network Repair Problem

Given a damaged sensor network and set of terminal locations from which we require sensed data, our goal is to place new nodes to ensure that each terminal is connected to a sink within a given number of radio links, and to find a mobility path through the environment to place the nodes, while minimising both the cost of the radio nodes and the length of the path.

Let V be a set of possible radio locations. $E_c \subseteq V \times V$ is the set of possible radio links, $E_m \subseteq V \times V$ is a set of traversable edges, and $w{:}E_m \to \mathbb{N}$ specifies the length of each edge. A path p in graph $G{=}(V, E)$ is a sequence $[x_1, x_2, x_3, \ldots, x_{k-1}, x_k]$ where each $\{x_i, x_{i+1}\} \in E$. The *hop count* of a path in the *connectivity graph* $G_c{=}(V, E_c)$ is one less than the number of nodes in the path, while the length of a path p in the *mobility graph* $G_m{=}(V, E_m)$ is $\sum_{(x_i, x_{i+1}) \in p} w(\{x_i, x_{i+1}\})$. L_r is the set of locations with existing relay (and sensor) nodes, while L_s is the set of locations with existing sink (and sensor) nodes. T is the set of terminal nodes

which must be reconnected within the hop count limit k, and $\alpha \in V$ is the initial starting location of our agent. c_r is the cost of a relay node, while c_s is the cost of a sink node. The problem is to find two new subsets $R \subseteq V$ and $S \subseteq V$ of relay nodes and sink nodes, such that for each terminal $t \in T$ there is a sink $s \in S$ and a path through the connectivity graph $(L_r \cup L_s \cup R \cup S, E_c)$ to s with a hop count $\leq k$, and a tour p in the mobility graph G_m that starts and finishes at α and visits each element of $R \cup S$, which minimises the pair $(\text{length}(p), (|R| * c_r) + (|S| * c_s))$ of the mobility tour length and the node cost.

Note that the two objectives may conflict. As an example, Figure 1 shows (a) a connectivity graph and (b) a mobility graph for a set of terminals $T = \{t_1, t_2, t_3\}$ and a set of candidate locations {a,b,d,e,f,g,h,j}. Assuming $c_s = 3 * c_r$, $k = 2$ and the current location of the agent is at f, a minimal cost node deployment to reconnect all terminals within a hop count limit of 2 is $S = \{d, t_3\}, R = \{t_1\}$, with cost $7 * c_r$. The shortest path in the mobility graph is $[f, t_3, f, e, b, h, t_1, h, a, d, a, e, f]$ with length 45. For a deployment of $S = \{t_1, t_3, g\}, R = \{\}$, the node cost is $9 * c_r$, but there is a path $[f, g, f, t_3, f, e, b, h, t_1, h, b, e, f]$ with length 34. Which of these solutions should be selected will depend on the relative cost of the sink and radio nodes compared to the cost of traversing the path. High node costs and low mobility costs will prefer the first solution, while high mobility costs will prefer the second solution.

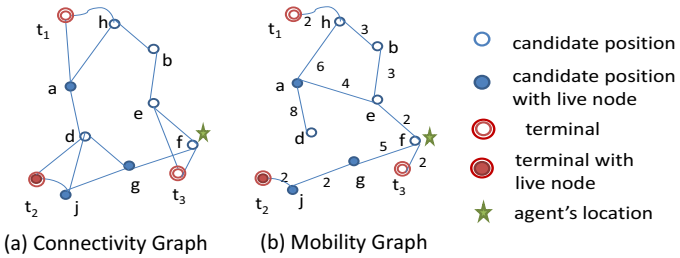

Fig. 1. Example Network Repair Problem

The objective of minimising the mobility path length has a travelling salesman path problem embedded inside it, and so we choose to investigate heuristic approaches. To address the two objectives, we consider two approaches, which each prioritise one of the objectives.

3 The Node Optimisation Heuristic Algorithm

Our first approach prioritises the node cost, by searching for a low cost set of relay and sink nodes which connect the terminals to sinks within the hop count limit (Algorithm 1). Given a set of nodes, we then search for the cheapest mobility path that visits those nodes (Algorithm 2). We start by finding, for each terminal j the set T_j of all possible sink locations (i.e. locations within k hops

of j in the connectivity graph). For each sink location s_i which is associated with two or more terminals, we determine the valuation (x_i, y_i), where x_i is the number of terminals which could be connected by s_i, and y_i is an upper bound on the number of new relay nodes that would be required to connect them. The set O=$\{s_i | x_i * c_s \geq c_s + y_i * c_r\}$ then contains all such sinks that could connect all of their terminals for less than the cost of placing a separate sink for each terminal. We order O by the expected total cost of deploying that sink, $(|T|-x_i)*c_s + y_i*c_r + c_s$, where $y_i*c_r + c_s$ is the cost of placing that sink with extra radio nodes y_i to reconnect the expected x_i terminals, and $(|T|-x_i)*c_s$ is an upper bound on the cost of placing sinks at the remaining terminals in T. We then select the first sink location in O, add it to S, the set of new sinks, compute relay node locations to connect the terminals and add them to R, the new relays. We then recompute the valuations and reorder the set O to reflect the changes in the unconnected terminals, and repeat. Once O is empty, for each remaining unconnected terminal j we place a sink at a random location in T_j. The most expensive operation is the calculation of (x_i, y_i) where we use a best-first search to find the shortest connectivity path from a terminal to each location $s_i \in O$, and we re-use those paths when we select the relay node locations.

For the problem of finding a short tour for the selected nodes (Algorithm 2), we create from the mobility graph G_m a metric closure graph for the new nodes in $S \cup R$. We then apply [1]'s Greedy-TSP heuristic - we sort the edges in increasing order of cost, and we 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 dominated by the time of building the metric closure graph, i.e. $O(|S \cup R| * |V|^2)$.

Figure 2 shows the NOH algorithm being applied to the example of Figure 1. First we find all sets T_j for each $j \in T$ (Figure 2(a)). We then find and order the set O (b). We select the first entry in O for a sink node, and it requires one additional relay node at t_1 (c). As t_1 and t_2 now have a connection to the sink at d within 2 hops, we remove them from the list T. Now a connects no terminals in T, while b and g only connect one terminal t_3. Therefore, we remove them all from the list O and terminate the *while* loop. We then select t_3 for a new sink (d) and finish the algorithm. Finally, we apply Greedy TSP to find a tour visiting those selected locations, giving $P = [f, t_3, f, e, b, h, t_1, h, a, d, a, e, f]$ which costs 45 units for 2 sinks and 1 relay node.

4 The Path Optimisation Heuristic Algorithm

Our second approach prioritises the mobility cost. First, for each terminal $j \in T$, we find the set C_j of all locations within the hop count limit from j but that require at most one extra node to connect j. Since we must guarantee connectivity within k hops for each terminal, we must place at least one node in each C_j. Placing a sink node in C_j ensures that no other node is required in C_j. Since the aim is to minimise mobility cost, we now search for a set of nodes that cover the C_j and which can be visited with the shortest possible tour.

Algorithm 1. Node Selection

Data: G_c=(V, E_c), G_m=(V, E_m), L_s, L_r, T, k
Result: S (new sinks), R (new relays)
begin
  foreach j *in* T **do**
    Find the set $T_j \subseteq V$ within k hops of j in G_c
  O={}
  foreach o_i *in more than one* T_j **do**
    Compute the values (x_i, y_i)
    if $c_s * x_i \geq c_r * y_i + c_s$ **then**
      add o_i to O in increasing order of $(|T|-x_i)*c_s + y_i*c_r + c_s$
  while O *and* T *are not empty* **do**
    Move first o_i from O into S
    foreach $j \in T$ *such that* $o_i \in T_j$ **do**
      Find connectivity path p for o_i to j
      foreach *location* l *in* p *not in* $L_s \cup L_r$ **do**
        add l into R
      Remove j from T
    Re-calculate (x_i, y_i) for all affected entries in O
    foreach $o_j \in O$ *with* x_j=1 **do**
      Remove o_j from O
    Re-order O
  foreach $j \in T$ **do**
    select a location l from T_j and add to S

Algorithm 2. GreedyTour

Data: A set of vertices V', a graph $G_M = (V, E_M)$
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] ;

We build a new graph G=(V', E') where V'=$\bigcup_{j \in T} C_j$, and E'={$\{u, v\}|u \in C_i$, $v \in C_j, i \neq j$}, i.e. the graph of locations in the cluster sets C_j with an edge between every pair of locations from different cluster sets. We associate a weight to each edge in E' equal to the shortest path length in G_m between the two endpoints. Note that any location which appears in two cluster sets will have a 0-weighted self-edge. The path optimisation problem can now be modeled as a Generalized Travelling Salesman Problem (GTSP) on G where the agent needs to visit exactly one node in each cluster C_i ([2]). We use the memetic algorithm for the GTSP proposed by Gutin and Karapetyan in [3]. The algorithm

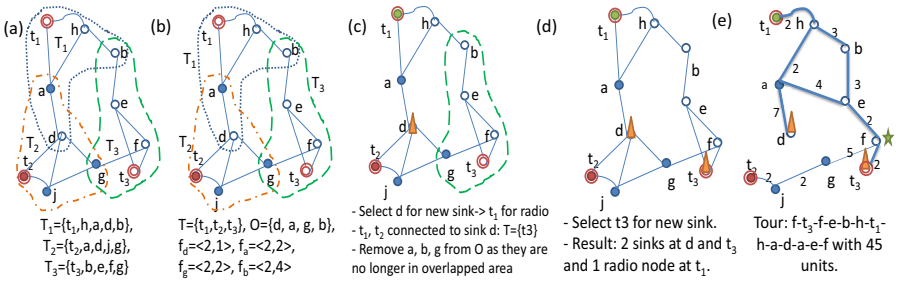

Fig. 2. A sample execution of NOH

creates a first generation of $2*|T|$ tours by creating random permutations of the clusters and then finds the best vertex in each cluster using the Cluster Optimisation Heuristic (CO). The CO heuristic uses the shortest (s,t)-path for acyclic digraphs to find the best vertex for each cluster when the order of clusters is fixed (see [4]). It then runs a local improvement procedure on each solution. The procedure runs several local search heuristics sequentially including *Swaps* (swap every non-neighboring pair of vertices), *k-Neighbor Swap* (try all the permutations which are not covered by any of i-Neighbor Swap, $i = 2, 3, ..., k-1$), *2-opt* (try to replace every non-adjacent pair of edges (s_i, s_{i+1}) and (s_j, s_{j+1}) by the edges (s_i, s_j) and (s_{i+1}, s_{j+1})), *Direct 2-opt* (a modification of 2-opt, which only selects some of the longest edges in the solution), and *Insert* (remove a vertex from the solution and insert it in different position). The procedure applies all those local search heuristics in a loop, removing any heuristic that fails to improve the solution. Once the loop terminates, it applies the CO heuristic again, and stops. The next generation is created by reproduction, crossover, and mutation operators applied in parallel to the previous generation. Reproduction simply copies the best solutions from the previous generation. The crossover operator is a 2-point crossover producing a single child, by selecting a fragment from the first parent, and then completes the tour by copying the order of the 2nd parent's nodes starting with the node at the end of the selected fragment, and deleting any repeated nodes. The parents are selected randomly from the top 33% of individuals. The mutation operator modifies each selected parent (selected from the top 75%) by removing a random fragment and inserting it randomly in a new position. Reproduction, crossover and mutation generate new children in the ratio $(1 : 8 : 2)$. The local improvement moves are then applied to each individual. The algorithm repeats until a time limit is reached. When running the algorithm in the experiments, we use the same parameter values given in [3]. Note that the overall running time is dominated by the creation of the initial weighted graph.

The memetic algorithm results in a sequence of locations to be visited, and we obtain a feasible solution if we place a sink at each location. We now improve that solution by replacing as many sinks with relay nodes as possible (Algorithm 4). We start by assuming all locations in the set are occupied by relay nodes. We order the set in decreasing order of the number of terminals within k hops

Algorithm 3. Memetic Algorithm

Data: $G = (V', E')$ (graph), a set of clusters $C_i, i \in T$
Result: a tour visiting exactly one node in each C_i
begin
  Initialize, construct first generation of solutions;
  Improve the first generation by local search, eliminate duplicate solutions;
  while *not termination condition* **do**
    Produce next generation by genetic operators (reproduction, crossover, mutation);
    Improve the next generation by local search, eliminate duplicate solutions;

Algorithm 4. Node Selection Algorithm

Data: N (set of locations), $G_c = (L_s \cup L_r \cup N, E_c)$ (graph), T (set of terminals)
Result: S (locations for sinks), R (locations for relays)
begin
  S$\leftarrow${};
  while T *is not empty* **do**
    Sort N in decreasing order of number of $j \in T$ within k hops in G_c;
    Move first element n_0 from N to S;
    Remove all j from T where j is connected to n_0 within k hops in G_c;
  R$\leftarrow$N

of each location. We select the first location, convert it into a sink, and remove from T all terminals connected to that sink in $\leq k$ hops. We repeat until T is empty. Any locations remaining in N are left as relay nodes.

Figure 3 shows the POH algorithm being applied to the example of Figure 1. First, we find a set of clusters for each terminal in G_c (Figure 3(a)). We then calculate the costs of moving from each node in each cluster to other nodes in different clusters (b). We apply the memetic algorithm to this new graph and it produces the tour $[f, t_3, g, t_1, f]$ (c) which is then mapped into G_m as P=$[f, t_3, f, g, f, e, b, h, t_1, h, b, e, f]$ with a mobility cost of 34 (d). Finally, we apply the Node Selection algorithm in G_c for those visited locations $\{t_1, t_3, g\}$ which results in 3 new sinks.

5 Evaluation

Both proposed algorithms (NOH and POH) 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 use a grid to generate the graphs. Connectivity is based on the distance between two

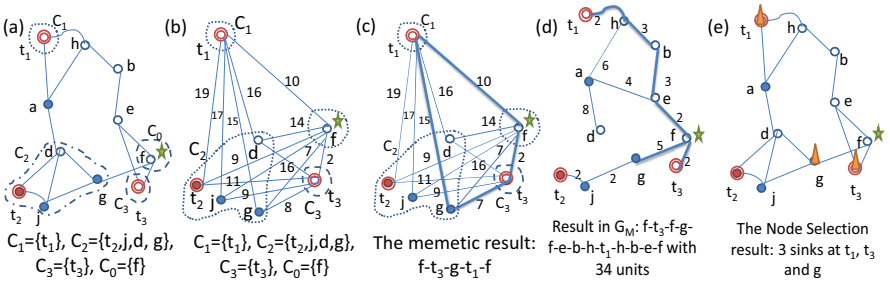

Fig. 3. A sample execution of POH

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 o 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 not able to be traversed.

We then create the connectivity graph by adding edges indicating that two candidate locations are within transmission range. For each pair of locations, we add with probability 0.85 an edge if they are within 10 units apart; we add with probability 0.2 an edge for each pair of locations which is between 10 and 20 units apart. This is to simulate the radio obstacles where we don't have uniform communication ranges. 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. For any pair of locations separated by a distance of less than 25 and which has a straight line that traverses all obstacle with a weight less than or equal to 0.2, we add those edges into the mobility graph. The cost of the edge is the distance plus 10*weight for each obstacle it crosses.

We consider the problem size: (i) a 10 × 10 grid, and thus a maximum of 100 candidate locations, and 20 possible obstacles[1]. We perform two sets of experiments: varying the number of terminals and varying the hop count limit k. 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 or live nodes. Finally, we calculate the total time to restore the network for different agent speeds, by combining the runtime with the estimated travel time.

[1] We also experimented with the problem size 5 × 10 grid, and thus a maximum of 50 candidate locations, and 10 possible obstacles, and got similar results.

To assess the quality of the solutions, we compare the results against the optimal solutions for each individual objective, generated by an exhaustive search. That is, OPT-N first finds the set of sinks and relay nodes with lowest cost that connect all the terminals with the hop count. For that set, a tour is then generated using Algorithm 2. OPT-P first generates the optimal mobility tour that could connect all terminals within the hop count (by selecting the optimal tour that visits each cluster set). For the locations on the tour, we then select the locations to be used as sinks using Algorithm 4.

First, we vary the number of terminals and fix the hop count limit as $k = 2$. Figure 4 shows the number of nodes placed by each algorithm. As the number of terminals to be connected increases, the node cost rises as expected. The two algorithms that prioritise mobility incur 25% higher node costs than their node equivalents. The mobility costs rise as we increase the number of terminals. The heuristic POH is within 30% of the exact OPT-P. As above, the algorithms that prioritise node cost create approximately 40% higher mobility costs.

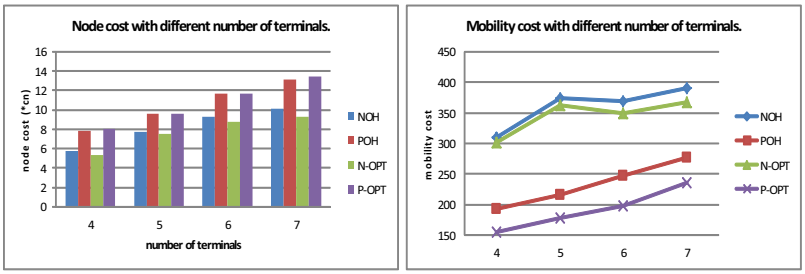

Fig. 4. Varying number of terminals

In the second set of the experiments, we fix the number of terminals at 5, and vary the hop count limit k. As the hop limit increases, the node costs decrease, as the connectivity problem becomes easier (Figure 5). The performance gap between the node-based algorithms and the path-based algorithms increases. The mobility costs also decrease as the hop limit increases. We believe this is because the reduction in the cost is due to placing a small number of sinks in the centre of the map, thus requiring a shorter path to visit those locations.

Table 1 shows the runtimes for the algorithms in both experiments. All increase with the number of terminals and the hop count k. The running time of NOH is significantly faster that of POH due to POH takes time to compute the clusters and path cost between clusters.

Finally, we note that the mobility costs are associated only with the distance travelled. For real scenarios, there is a tradeoff between the cost of the extra nodes and the speed at which connectivity is restored, and so we should consider the combination of runtime and the estimated time to execute the solution. We assume that it takes an agent 30s to position a new node. We then consider two scenarios, the first representing a small robot which moves at 0.1ms^{-1}, the second representing a larger vehicle moving over rough terrain at 4ms^{-1}.

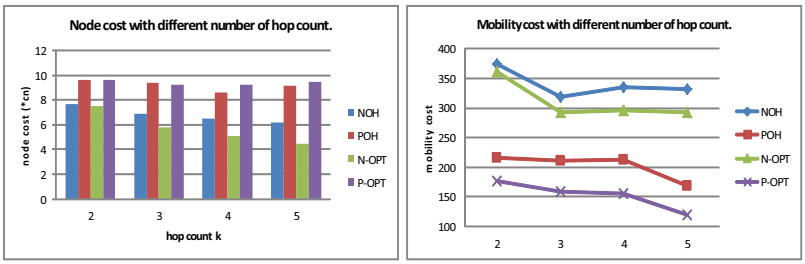

Fig. 5. Varying hop count limit k

Table 1. Runtime

(a) Varying number of terminals

	4	5	6	7
NOH	1.1	1.7	2.2	2.8
POH	17.9	23.2	43.1	57.7

(b) Varying hop count limit

	2	3	4	5
NOH	1.7	1.8	2.6	3.2
POH	23.2	28.8	88.6	91.4

The total time to restore the network is thus the initial computation time, the time to move along the path, plus the time to place the new nodes. We assume each unit distance is 1 meter. The results are shown in Tables 2. For the slow small robot, prioritising the mobility cost results in a faster restoration time for all parameter settings, as the mobility costs outweigh the time to place nodes and the increased runtime. For the vehicle, prioritising the node cost becomes more important, since the reduction in mobility cost by the path-based algorithms has difficulty compensating for the increased runtime and the increased node-placement cost. Thus the WSN restoration problem is subtle, with the choice of approach clearly dependent on the details of the specific problem. Solution methods must take into account the main objectives (minimising infrastructure and minimise time), but also consider the capabilities of the agent that will implement the eventual solution.

6 Related Work

Wireless sensor networks are prone to failures, which can lead to a loss of connectivity when the network becomes partitioned and/or nodes become isolated. Several researchers have addressed this problem by devising appropriate planning methodologies such that a deployed network will be resilient in the face of limited failures, or can be altered to avoid anticipated failures, for example due to node power exhaustion. In contrast, our work seeks to repair a network after failures have occurred. Some other papers have addressed this problem by using specialised nodes that can be moved into position and restore connectivity, e.g. [5], [6], [7] but such solutions are not attractive because mobile nodes are expensive and in large networks many such nodes may be required.

Table 2. Total Restoring Time

(a) V=0.1ms^{-1}, vs number of terminals

	4	5	6	7
NOH	3275.1	3978.3	3969.8	4212.8
POH	2187.9	2478.5	2873.7	3225.8

(b) V=0.1ms^{-1}, vs hop count limit

	2	3	4	5
NOH	3978.3	3391.8	3549.9	3505.1
POH	2478.5	2419.1	2478.6	2055.2

(c) V=4ms^{-1}, vs number of terminals

	4	5	6	7
NOH	252.6	326.3	373.4	404.7
POH	300.3	365.3	456.0	521.3

(d) V=4ms^{-1}, vs hop count limit

	2	3	4	5
NOH	326.3	288.3	281.4	270.5
POH	365.3	363.5	399.9	407.9

Many papers address the repair problem by placing relay nodes that reconnect partitions in the network, minimising number of required nodes. For example, using centralised solutions, [8] uses a spider web approach while [9] forms a connectivity chain toward a centre point of the network. In contrast, we optimise both the number of additional nodes and the mobility path length needed for their deployment. We also explicitly take into account the impact of obstacles that can impede both the available paths and the ability of nodes to communicate directly, and we bound the number of network hops by judiciously deploying additional sink nodes in the interest of quality of service constraints.

In regard to the problem of multiple sink deployment, some other authors have addressed this for the purpose of improving network performance. There are two main strategies: to reduce energy consumption so as to maximise the network lifetime, e.g. [10], [11] and to minimise the average data latency or maximum hop distance between a node and its nearest sink, e.g. [12]. These papers do not deploy sink nodes in the act of network restoration, and thus do not consider the mobility cost to deploy sink nodes, and the implications of obstacles on the choice of possible sink locations.

7 Conclusion

In this paper we address the problem of restoring connectivity in a wireless sensor network, using a mobile agent to place relay and sink nodes, avoiding obstacles and respecting bounds on quality of service (defined in terms of network hop-count). We formulate the problem as one of searching over two linked graphs which share the vertex set. The multi-objective problem requires us to minimise both the cost of additional nodes and the path to be taken by the mobile agent. We propose two heuristic algorithms which prioritise the different objectives. We evaluate them on randomly generated networks and compare their solutions to the optimal solutions. Finally, we also evaluate the total restoration time for each solution as a function of the mobile agent's speed, quantifying a trade-off with computation time. In future work, we will consider different algorithms for the two different priorities, and we will investigate the Pareto frontier. We will also

consider the more general problem, in which we must discover the damage to the network as we repair it. Finally, we will consider the problem of continually spreading damage, and the use of teams of agents cooperating to repair the network.

Acknowledgment. This work was funded by the HEA PRTLI4 project NEMBES, and by the SFI centre CTVR (10/CE/I1853).

References

1. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: The Traveling Salesman Problem. John Wiley & Sons (1985)
2. Fischetti, M., Salazar-Gonzalez, J.J., Toth, P.: The generalized traveling salesman and orienteering problems. In: Gutin, G., Punnen, A.P. (eds.) The Traveling Salesman Problem and Its Variations. Combinatorial Optimization, vol. 12, pp. 609–662. Springer (2004)
3. Gutin, G., Karapetyan, D.: A memetic algorithm for the generalized traveling salesman problem. Journal of Natural Computing 9(1), 47–60 (2010)
4. Fischetti, M., Gonzlez, J.J.S., Toth, P.: A branch-and-cut algorithm for the symmetric generalized travelling salesman problem. Operations Research 45, 378–394 (1997)
5. Akkaya, K., Senel, F., Thimmapuram, A., Uludag, S.: Distributed recovery from network partitioning in movable sensor/actor networks via controlled mobility. IEEE Transaction on Computing 59, 258–271 (2010)
6. Abbasi, A.A., Younis, M.F., Baroudi, U.A.: A least-movement topology repair algorithm for partitioned wireless sensor-actor networks. International Journal of Sensor Networks 11(4), 250–262 (2012)
7. Abbasi, A.A., Younis, M.F., Baroudi, U.A.: Recovering from a node failure in wireless sensor-actor networks with minimal topology changes. IEEE Transaction on Vehicular Technology 62(1), 256–271 (2013)
8. Senel, F., Younis, M., Akkaya, K.: A robust relay node placement heuristic for structurally damaged wireless sensor networks. In: LCN 2009, pp. 633–640 (2009)
9. Lee, S., Younis, M.: Recovery from multiple simultaneous failures in wireless sensor networks using minimum steiner tree. Journal of Parallel Distributed Computing 70, 525–536 (2010)
10. Kim, H., Kwon, T., Mah, P.S.: Multiple sink positioning and routing to maximize the lifetime of sensor networks. IEICE Transactions on Communications 91-B(11), 3499–3506 (2008)
11. Gu, Y., Ji, Y., Li, J., Chen, H., Zhao, B., Liu, F.: Towards an optimal sink placement in wireless sensor networks. In: ICC 2010, pp. 1–5 (2010)
12. Kim, D., Wang, W., Sohaee, N., Ma, C., Wu, W., Lee, W., Du, D.Z.: Minimum data-latency-bound k-sink placement problem in wireless sensor networks. IEEE/ACM Transaction on Networking 19(5), 1344–1353 (2011)