CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Autonomous discovery and repair of damage in Wireless Sensor Networks
Authors	Truong, Thuy T.;Brown, Kenneth N.;Sreenan, Cormac J.
Publication date	2013-10
Original Citation	Truong, T. T., Brown, K. N. and Sreenan, C. J. (2013) 'Autonomous discovery and repair of damage in Wireless Sensor Networks', 38th Annual IEEE Conference on Local Computer Networks, Sydney, NSW, Australia, 21-24 Oct. 2013, pp. 450-458. doi: 10.1109/LCN.2013.6761278
Type of publication	Article (peer-reviewed)
Link to publisher's version	10.1109/LCN.2013.6761278
Rights	© 2013 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Download date	2026-11-06 22:31:41
Item downloaded from	https://hdl.handle.net/10468/5100

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Autonomous Discovery and Repair of Damage in Wireless Sensor Networks

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

***Abstract*—Wireless Sensor Networks in volatile environments may suffer damage, and connectivity must be restored. The repairing agent must discover surviving nodes and damage to the physical and radio environment as it moves around the sensor field to execute the repair. We compare two approaches, one which re-generates a full plan whenever it discovers new knowledge, and a second which attempts to minimise the required number of new radio nodes. We apply each approach with two different heuristics, one which attempts to minimise the cost of new radio nodes, and one which aims to minimise the travel distance. We conduct extensive simulation-based experiments, varying key parameters, including the level of damage suffered, and comparing directly with the published state-of-the-art. We quantify the relative performance of the different algorithms in achieving their objectives, and also measure the execution times to assess the impact on being able to make autonomous decisions in reasonable time.**

I. Introduction

Many applications for wireless sensor networks will be in settings where network damage can be expected to occur, due to factors such as changes in the physical environment, node failures, vandalism, and power-source depletion. Damaged networks must be repaired, to re-establish connectivity for important data streams. The repairing agent (e.g. a robot or a pedestrian) must move through the sensor field, establishing the extent of the damage to the network and to the physical environment, and deploying new sensors or relays. The main goal of the agent is to minimise the cost of the repair, where the main cost factor may be the number of new nodes or the time taken to complete the repair, depending on circumstances.

We assume the agent starts with knowledge of the radio and physical environments before the damage occurred, obtained from earlier site surveys and network data. After damage, the agent is only aware of the sensor nodes (and the corresponding radio links) still connected to its current location. It must plan a deployment of nodes to restore connectivity for designated data streams, and a route through the environment to place those nodes. Each plan is a set of locations to be visited, and a detailed motion plan for reaching the first location. Since both subproblems (connectivity and multi-point path) are computationally hard, we use heuristic algorithms to generate the plans, with two different heuristics: prioritising the number of new nodes, and prioritising the path length. However, while executing the plan, the agent will encounter blocked paths and broken radio links, but also surviving components of the old network, and as it discovers new knowledge, it must revise its plans to complete the repair. We consider two approaches for revising the plans. The first conducts full replanning whenever it discovers new knowledge that changes the cost of the plan, or which renders the plan infeasible. The second attempts to repair the plan, by searching for a new motion plan to reach its current location target, and reverting to full replanning only after large changes.

We evaluate the approaches in simulation on randomly generated problems, where we vary the density of the connectivity problem and the level of damage sustained. We demonstrate that the full replanning approach produces better quality solutions, but has significantly higher runtime as damage and density increase. However, when we factor in the expected total time to execute the plan, full replanning becomes competitive. For slower moving agents, the reduction in path length with the path heuristic outweighs the increased runtime, while for faster agents, the repair actions are more significant, and the node heuristic with full replanning becomes more efficient at higher damage levels. The results illustrate the trade-offs between minimising node cost and minimising the speed of the repair, where the choice of approach will be dependent on the application.

In the remainder of the paper, we discuss some related work, and introduce the problem formulation, followed by the agent's abilities and knowledge structures. We then describe our heuristic approaches with the replanning strategies. We describe the experiments and results, and finish with conclusions.

II. Related Work

The problem of network repair is receiving significant attention in networking research. The ideal scenario would be to ensure the network is resilient to limited failures without requiring repair. In [9],[4] the goal is to deploy k-1 redundant nodes to achieve 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 can be prohibitive. [5] deploys a robot with unlimited nodes and drops nodes from time to time based on certain ordering rules. [27] controls the agent's motion to

explore the environment while dropping nodes while preserving the connectivity of the network. [16] assumes a mobile sensor network where nodes can use repel and attract forces to arrange the topology. These papers focus on topology control and deployment but do not consider repair/restoration after damage has occurred. To restore connectivity in damaged networks, some proposals use specialised nodes that can change position to restore connectivity, e.g. [3],[21], [1], [2]. While attractive, such solutions require significantly more expensive nodes, and thus may be impractical for many deployments. Many papers address the repair problem by placing relay nodes that re-connect partitions in the network, minimising the number of required nodes. For example, using centralised solutions, [17] uses a spider web approach while [11] forms a connectivity chain toward a centre of the network and [20] uses game theory to reconnect the network. However, these papers do not consider problems of limited mobility, assuming instead a free space model. Some papers [23], [25], [19], [18] consider more realistic terrain which has obstacles. However, they assume a static problem, in which all terrain and all network conditions are known in advance. None of them address the problem of restoring connectivity where exploration must be carried out during the repair. In [24] introduced the exploration problem, but without modelling the issues of planning and repair.

The problem of agent planning is a central topic in artificial intelligence and robotics. In particular, continual planning, in which the plan must be modified as knowledge is discovered, was first proposed in [14]. [6] uses iterative repair techniques to support a continuous planning process for autonomous spacecraft control. [7], for temporal planning, interleave decision and execution in a dynamic environment allow plan repair interleaved with execution. [15] uses a model-free approach which observes and classifies the actual behavior of the monitored systems into normal or faulty execution. [12] dynamically reasons about which goals to pursue in response to unexpected circumstances. [22] proposes a generic and reactive scheme for continous planning for complex problems.

III. Problem Formulation

For the environment before the damage has occurred, we assume a rectilinear grid of locations G, in which a subset $V_b \subseteq G$ of grid squares are candidate locations for wireless nodes, with each square allowing at most one node, at a specified position within the square. A *connectivity* graph, (V_b, C_b), specifies potential radio links between the nodes. We assume symmetric links are required for the network operation, and so we ignore any asymmetric connections. $V_B \subseteq V_b$ is the set of locations with actual nodes. After damage, the connectivity graph is (V_a, C_a), where $V_a \subseteq V_b$ and $C_a \subseteq C_b$. $V_A \subseteq V_B \cap V_a$ is the set of locations with surviving nodes. The set $\tau \subseteq V_B$, of *terminals*, is the set of locations from which we require sensed data. $I_v \subseteq V_A$ is the set of nodes still successfully transmitting data to our sink, and I_c is the corresponding set of active links. The repairing agent can move from any square into one of its 4 rectilinear neighbours, unless that neighbour is blocked. The set of blocked squares before damage is B_b, while the set of blocked squares after damage is B_a, such that $B_b \subseteq B_a \subseteq G$. The agent can deploy a relay node or sensor node at any location x it visits if $x \in V_a$, and we will denote by V_n the set of newly added nodes. We assume the starting location of the agent is at $L \in I_v$.

The repair problem is to follow a path P through the grid, without visiting any location in B_a, deploying nodes at locations V_n in the path such that in the graph $(V_A \cup V_n, C_a)$, all elements of τ have a path to a node in I_v. The cost of a plan can be evaluated as (i) the number of nodes to be deployed ($|V_n|$), and (ii) the length of the path P. However, given the unknown damage, the initial plan is likely to be either infeasible or inefficient, and so while executing it, the agent must sense its environment to update its knowledge and then modify the plan. The agent can probe the accessibility of its neighbouring squares up to a distance of k, but cannot probe a square if there is a blocked square inbetween. The agent is able to test a radio link by listening for transmission from an active node, up to a distance of R, and can transmit to the same range. When the agent discovers a new live node, it will also be told all of that node's live connected subgraph. There is no cost for listening for transmissions. The total cost of the final executed repair can then be measured as (i) the number of deployed nodes, and (ii) the sum of the movement costs, the probe costs and the node costs.

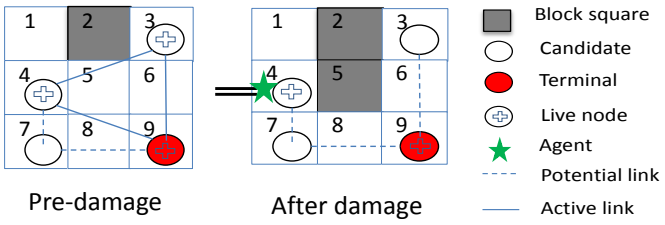


Fig. 1. Example of the network conditions.

IV. Representing the Agent

The agent has full knowledge of the environment before damage, obtained from earlier site surveys and network data, but must build its knowledge of what remains after damage as it executes its repair, and so it must distinguish between objects that are known to be active, those that are known to be damaged, and those that have not been verified.

The agent classifies grid locations for radio nodes into four classes: N_a, locations known to have an active radio; N_f, locations known to be feasible for placing a radio; N_i, locations known to be infeasible for placing radios; and N_u, locations whose condition is otherwise unknown. Initially, $N_a = I_v$, $N_f = I_v$, $N_i = G - V_b$, and $N_u = V_b - I_v$.

Pairs of locations are classified for radio links as follows: E_f, links known to be feasible for radio communication; E_i, links known to be impossible for radio communication; and E_u, links whose condition is otherwise unknown. Initially, $E_f = I_c$, $E_i = \{\{x, y\} : \{x, y\} \notin C_b\}$, and $E_u = C_b - I_c$.

Locations are classified for accessibility as follows: S_f, locations known to be accessible; S_b, locations known to be blocked; and S_u, locations whose condition is otherwise unknown. Initially, $S_f = L$ (the initial location of the agent), $S_b = B_b$, and $S_u = (G - B_b) - \{L\}$.

The world states consist of the post-damage conditions V_A, V_n, C_a, and B_a, and a single instance of the predicate $at(x)$, the location of the agent. As the agent executes a plan, it will update its knowledge, and should ensure that $N_f \subseteq V_a$, $N_a \subseteq V_A \cup V_n$, $E_f \subseteq C_a$, and $S_f \subseteq G - B_a$.

The agent has five possible actions, described below as rules with pre conditions and add and delete lists (if non empty), for use in the post-damage environment. We also include procedures for updating knowledge after each successful firing of an action.

1) MOVE(u,v): move from square u to square v;
   PRE: at(u) $\wedge$ neighbour(u,v) $\wedge$ $v \notin B_a$;
   ADD: at(v); DEL: at(u);
2) LISTEN(u): listen for radio signals at u;
   PRE: at(u);
   PROC:
   $(\mu, \epsilon) \leftarrow$listen(); //listen() reports live nodes and links
   $N_a \leftarrow N_a \cup \mu$; //remember reported nodes
   $N_f \leftarrow N_f \cup \mu$;
   $N_u \leftarrow N_u - \mu$;
   $E_f \leftarrow E_f \cup \epsilon$; //remember reported links
   $E_u \leftarrow E_u - \epsilon$;
   if $((x \in N_a || x = u)$ & &$y \in N_a$ & &$\{x, y\} \notin N_f)$
   then $E_i \leftarrow E_i \cup \{\{x, y\}\}$; //deduce blocked links
3) DROP(u): drop a node at u;
   PRE: at(u);
   ADD: if $(u \in V_a)$ then $V_n \leftarrow \{u\}$;
   PROC: $N_a \leftarrow N_a \cup \{u\}$;
4) PROBE(u,v): probe square v from u;
   PRE: at(u);
   PROC: if (p(u,v)==T) // T if v in range and free
   then $S_f \leftarrow S_f \cup \{v\}; S_u \leftarrow S_u - \{v\}$;
   else if (p(u,v)==F) // F if v in range but blocked
   then $S_b \leftarrow S_b \cup \{v\}; S_u \leftarrow S_u - \{v\}$;
   //p(u,v) reports ? if v not in range
5) INSPECT(u): check if u can take a radio node;
   PRE: at(u);
   PROC: if (insp(u)==T) // T if $u \in V_a$
   then $N_f \leftarrow N_f \cup \{u\}; N_u \leftarrow N_u - \{u\}$;
   else $N_i \leftarrow N_i \cup \{u\}; N_u \leftarrow N_u - \{u\}$;

For the simple world state of Figure 1, the initial knowledge structures are: $N_a = \{4\}, N_f = \{4\}, N_i = \{1, 2, 5, 6, 8\}, N_u = \{3, 7, 9\}, E_f = \{\}, E_i = \{\ldots\}, E_u = \{\{3, 4\}, \{3, 9\}, \{4, 7\}, \{4, 9\}, \{7, 9\}\}, S_f = \{4\}, S_i = \{2\}$, and $S_u = \{1, 3, 5, 6, 7, 8, 9\}$. The following sequence of actions will reconnect location 9: LISTEN(4); PROBE(4,7), with $S_f \leftarrow \{4, 7\}$; MOVE(4,7); LISTEN(7), with $N_f = N_a \leftarrow \{4, 9\}$ and $E_f \leftarrow \{\{4, 7\}, \{7, 9\}\}$; INSPECT(7), with $N_f \leftarrow \{4, 7, 9\}$; DROP(7), with $N_a \leftarrow \{4, 7, 9\}$.

V. Approach

There are two possible approaches for the agents. The first is conservative, and plans only with verified knowledge. The action representation must be extended to include explicit knowledge gathering, and would require uncertainty handling to determine the best action to take in each world state. In the second approach, the agent assumes that some elements of the unknown sets are available, and then replans when errors are discovered, and is the approach taken in this paper. We assume, until we discover otherwise, that all squares that were not blocked before damage remain unblocked and that all feasible radio links remain feasible, but that all previously existing radio nodes that are not reporting after damage have been lost. That means the agent will plan using grid squares in $S_f \cup S_u$, feasible radio locations $N_f \cup N_u$, feasible radio links $E_f \cup E_u$, and live radio nodes N_a. When executing a plan, the agent will insert LISTEN actions at each step, will PROBE immediately before trying to move to a new square, and will INSPECT immediately before dropping a node. When it discovers knowledge that renders its current plan infeasible or changes its cost significantly, it will update its plan and continue. A plan for the agent will be represented on two levels. At the higher level, we have an unordered set of locations that we intend to visit to drop a node. At the lower level, we have selected one of these locations, and we have a detailed path plan for moving there. Local repair will consist of generating a new path plan to the same selected location. Full re-planning will consist of generating a new set of locations at which to drop nodes, as well as a path plan to a newly selected node.

Note that the underlying problems, even when there is no damage, are computationally hard. The task of finding in a graph a minimal set of nodes which connect a terminal set is the minimal Steiner tree in graphs problem, and is NP-hard ([8]). Given a set of nodes in a mobility graph, the task of finding a minimal path through the graph that visits each selected node reduces to the TSP on a metric closure graph, built by finding all-pairs shortest paths for the selected nodes. Therefore, we consider heuristic approaches for generating the full plans.

A. Strategy 1: prioritising node cost

The aim of this strategy is to find a small set of nodes to reconnect all terminals, and then to find a short path to visit them. We first construct a directed weighted connectivity graph. Each candidate location ($N_f \cup N_u$) is a vertex, with connected components merged into supernodes. Each potential link is represented by two directed edges. An edge connecting a live node to a candidate location will have cost 1, while an edge in the other direction has cost 0. The agent then finds a Steiner node set N connecting all terminals using Steiner-MST on that directed weighted connectivity graph. The weights ensure that the heuristic prefers to bring existing live nodes into the tree rather than new candidate nodes. We use D* Lite to compute the cheapest mobility path [10] and then select the

Steiner node with the shortest path (the nearest Steiner node) to our current location.

Algorithm 1: Node Priority Heuristic

find D, the directed weighted connectivity graph;
find N, the Steiner nodes in D;
find P, the cheapest path to the nearest node in N;
return (N, P);

New knowledge that would change the estimated cost of the plan or render it infeasible are: (i) a blocked square on the path to the selected location, (ii) an expected radio link is not possible, (iii) a location is not suitable for dropping a node, or (iv) the existence of a surviving connected sub-network. Full replanning, however, is expensive, since we may have to recompute the directed weighted connectivity graph, and then re-run the Steiner MST heuristic. In particular, limited changes of type (iv) for a small surviving component, are unlikely to affect the cost significantly. Therefore, as well as full replanning, we also consider local repair, in which for case (iv) we use D* Lite to continue searching for the current target, until we discover α new surviving nodes, which triggers full replanning. Change of type (i) does not affect the node cost, therefore we do not need to recompute the Steiner nodes either in full replanning or local repair when we discover blocked squares, unless we have established that the node is not reachable. In all cases, options (ii) and (iii) trigger full replanning.

B. Strategy 2: prioritising mobility cost

This strategy aims to find a set of locations which can be visited by a short path, and for which nodes would reconnect the terminals. We first build a weighted connectivity graph, augmenting each link in $E_f \cup E_u$ with the cost of the cheapest mobility path between the two locations. Again, we use D* Lite to compute the cheapest mobility path, since we expect to have to compute these paths many times as we discover blocked locations. In the weighted connectivity graph, we then search for a low-cost Steiner tree using the Steiner-MST heuristic ([26]) to find a set of nodes which connects all unconnected terminals to the network. We then select the closest node to our current location, using D* Lite.

Algorithm 2: Path Priority Heuristic

find W, the weighted connectivity graph, using D* Lite;
find N, the Steiner nodes in W, using Steiner-MST;
find P, the cheapest path to the nearest node in N;
return (N, P);

Again, when we discover knowledge that changes the cost of the plan, we revise the plan. We consider the same triggers for full replanning and local repair as above, except that change of type (i) would change the path cost. Therefore, in this case of (i) the full replanning will restart the plan with new knowledge while the local repair uses D* Lite to continue searching for the current target, until the expected total path length exceeds the original path length by β steps, which triggers full replanning. The overall approach, applicable to both heuristic approaches and to both full replanning and local repair, is shown in Figure 2.

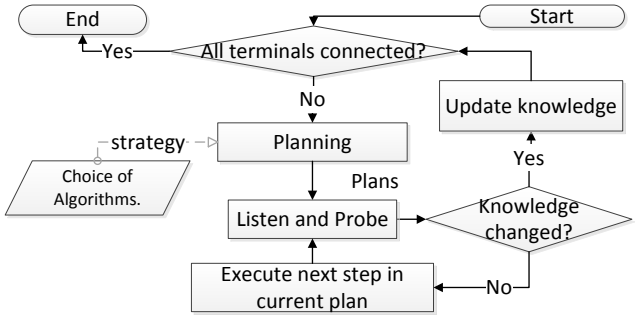


Fig. 2. Flow chart for the agent planning.

C. The DORMS approach

As a comparison, we adapt the DORMS approach [11] which was designed to tackle a similar problem. DORMS tries to re-connect network partitions to a central point and then perturbs the solution to reduce the number of additional nodes needed. However, DORMS assumes free space for mobility, and so the mobility paths are simply straight lines. In our adaptation, we select a location which is closest to the centre of the area. We use D* Lite to find the shortest path from each terminal to that centre location in the connectivity graph. Then for each pair of adjacent terminals, we find a graph which contains all nodes and edges in the current map which are in the smallest area bounded by the two connectivity paths to the centre. Then we find a steiner minimal tree in that graph which spans the two terminals and the centre location. After finding all steiner minimal trees for all pair of adjacent terminals, each terminal is now part of two separate trees formed with its neighbours, and the algorithm chooses the trees which require the fewest additional nodes. We then select the closest steiner node to our current location, using D* Lite. As before, when the agent discovers new information that would change the cost, it recomputes, and continues from its current location.

VI. Experiments

We evaluate our algorithms empirically on randomly generated maps, to compare the quality of their solutions for the two different measures, and to compare their runtimes. We assume a pre-damage grid map consisting of $n \times n$ squares representing a $300m \times 300m$ area. We randomly select c grid squares to be candidate locations, assigning a random location within the square, and g squares to be blocked. For each pair of candidate locations separated by less than 60 metres[1], we allow a potential radio link with probability 0.85. For the map after

[1]Chosen to lie well within the maximum range of the popular TmoteSky sensor node [13]

Algorithm 3: Adaptive DORMS Algorithm

Result: N: node plan; P: path plan.
while *there are unconnected terminals* **do**
  centre = Find_Centre_Location();
  for *each* $t \in \tau$ **do**
    $p_t =$
    $Shortest_Connectivity_Path(t, centre, G_C)$;
  for *each* $t \in \tau$ **do**
    $ta = Find_adjacent_terminal(t)$;
    $G' = bounded_graph(p_t, p_t a, centre, G_C)$;
    $T_t = Steiner_M inimal_T ree(t, ta, centre)$;
  while *any* $t \in \tau$ *is not in* L **do**
    $T = Find_Smallest_tree_for_t()$;
    Add T into L;
  find P, the cheapest path to the nearest node in N;

damage, we randomly select a of the candidate locations to be live nodes, and select t candidate locations to be terminals (the locations for which we require sensor data). We randomly pick an additional $b\%$ of the total squares to be blocked due to obstacles, and remove $r\%$ of the radio links. For this paper, we ensure the problems are feasible - i.e. that there is a set of reachable locations for which nodes would reconnect all terminals. In each case, the algorithms only probe a square that the agent intends to move into. For the local repair, the cost threshold is $\alpha = 4$, and the live node threshold is $\beta = 3$ for all experiments. In all experiments, we set $a = 15$ live nodes and the results are the average of 50 runs at each data point. We study the effects of different numbers of candidate locations and different levels of damage.

First, we consider the effect of varying the number of candidate locations for nodes. We fix the size of the grid at 45×45, vary c, the number of candidates, from 50 to 150, and fix the number of terminals, t, to 5 and the damage level to $(b = 10\%, r = 10\%)$, thus creating problems with increasing connectivity graph density. The results are shown in Figure 3 and Table I.

The repair methods incur approximately 13% extra node cost, for both path priority and node priority heuristics. The path priority approach is suprisingly close in node costs to the node priority approach, and full replanning for path priority results in a lower node cost than node priority with local repair. For mobility costs, the local repair method is worse than the full replanning method for path priority. Both methods for path priority are better than the methods for node priority. As the graphs become more dense, there is little impact on node costs, but the path costs drop. This appears to be because the increased density does not lower the transmission range, and so similar length multi-hop radio paths will be required, and thus a similar number of nodes, but there will be more choice in where to place them, and so the path costs can be reduced. The DORMS-AD approach is consistently poorer on both measures, with up to 50% higher costs compared to the best heuristic in each case.

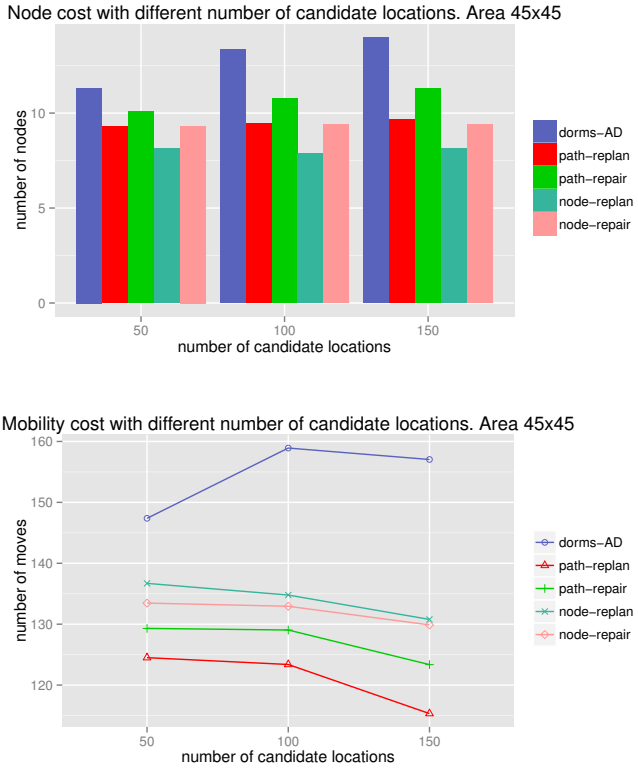


Fig. 3. varying number of candidate locations

	50	100	150
dorms-AD	0.202	0.991	10.922
path-replan	0.267	3.345	41.541
path-repair	0.116	0.863	9.935
node-replan	0.186	1.741	24.504
node-repair	0.107	0.897	12.578

TABLE I
RUNTIME (SEC) VS NUMBER OF CANDIDATE LOCATIONS.

Since the application requires real-time repair, the runtime of the different methods is important, and results are shown in I. The local repair methods in both approaches are faster than the full replanning methods. The path approach is slower than the node approach respectively. In the node priority approach, the full replanning takes closer to 100% more time to complete, compared to the local repair, while in the path priority approach, the full replanning takes approximately 100% to 400% more time to complete, compared to the local repair. Further, for all methods, we see a significant impact from the increased density of the candidate locations, with approximately an order of magnitude increase for each additional 50 nodes. DORMS is competitive on runtime, becoming the second fastest algorithm for the most dense problems.

Second, we vary the number of terminals, from 5 to 15, and fix the number of candidate locations, c, to 100 and the damage level to $(b = 10\%, r = 10\%)$. The results are shown

in Figure 4 and Table II. As expected, all the costs increase with the increasing number of terminals to be connected.

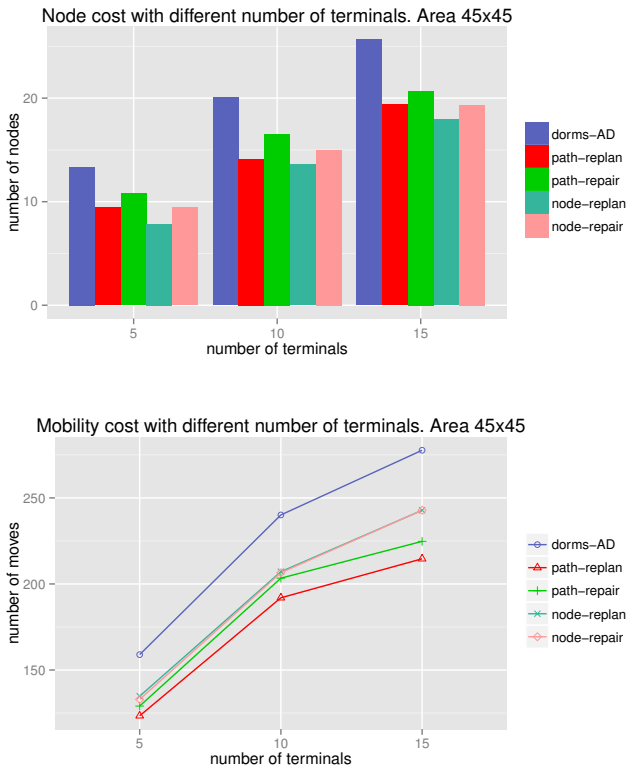


Fig. 4. varying number of terminals

	5	10	15
dorms-AD	0.917	1.73	2.985
path-replan	3.473	5.985	10.313
path-repair	0.853	1.410	2.206
node-replan	1.727	2.938	4.263
node-repair	0.832	1.411	2.251

TABLE II
RUNTIME (SEC) VS NUMBER OF TERMINALS.

We now vary the damage level from $(10\%, 10\%)$ to $(30\%, 30\%)$, fixing the grid at 45×45, candidate locations at 100 and number of terminals at 5, creating problems in which the agent is expected to revise its plans more often. The results are shown in Figure 5 and Table III.

	10%,10%	20%,20%	30%,30%
dorms-AD	0.889	1.055	1.607
path-replan	3.404	5.142	8.602
path-repair	0.885	0.916	1.127
node-replan	1.708	2.061	2.431
node-repair	0.928	1.016	1.249

TABLE III
RUNTIME (SEC) VS DAMAGE LEVELS.

The node costs rise as it requires more nodes to compensate

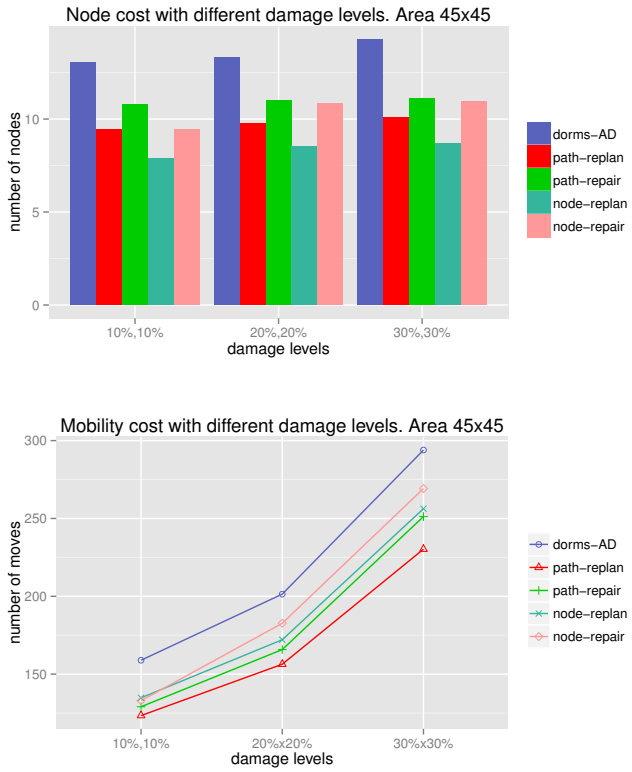


Fig. 5. varying damage levels

for heavier damage. The repair methods incur approximately 20% extra node cost, for both path priority and node priority heuristics. For the mobility costs, the node priority heuristics now produce path plans that are competitive with path priority local repair, and within 10% of path priority with full replanning. The full replanning for path priority remains the best for all cases. The mobility costs are rising as the damage increases - the number of candidate locations decreases, and there are more obstacles blocking the route - and the lack of knowledge of the true mobility problem counteracts the benefits of the path priority approach. For runtime, the difference is clear, and the path priority with full replanning requires the longest runtime, due to the repeated requirement to recompute the metric closure graph.

We note that the representation of the physical area may have an impact on the results - large grid squares may obscure details of the obstacles and thus available paths, but are easier for computation. To assess the impact of this, we vary the granularity of the grid. First, we generate a problem for a 300mx300m area with a 45x45 grid (square size 6.66m), 100 candidate locations, 5 terminals, and damage level (20%, 20%). We then fix the physical locations for candidates, terminals, blocked squares, we also fix the radio links before and after damage for those candidate locations. We model the area into finer grids of 100x100, 150x150, 200x200 and 300x300 with the square size of 3m, 2m, 1.5m and 1m respectively. The results are shown in Figure 6, where we

know plot mobility cost as the path distance. As expected, the number of required nodes show only minor variation. In this experiment, the mobility costs by number of moves (from square to square) obviously increases with the finer grids. However, we are interested of the distance travelled by the agent when we reduce the size of the grid square. The mobility costs w.r.t distance travelled are significantly reduced as the grid granularity increases. This is because we are able to find paths between obstacles which would have been blocked with the larger squares. The runtimes IV increase for all algorithms as the finer grids mean more options to explore. The node priority approach appears to be most heavily impacted by the increase granularity. We notice the runtime of local repair for path priority is longer than that of full replanning. We believe this phenomenon is because the local repair keeps the current plan unless it discovers significant changes, and conservatively keeping the same target even when the environment is changed will force the agent to always looks for different paths to same targets and this might get worse due to damage.

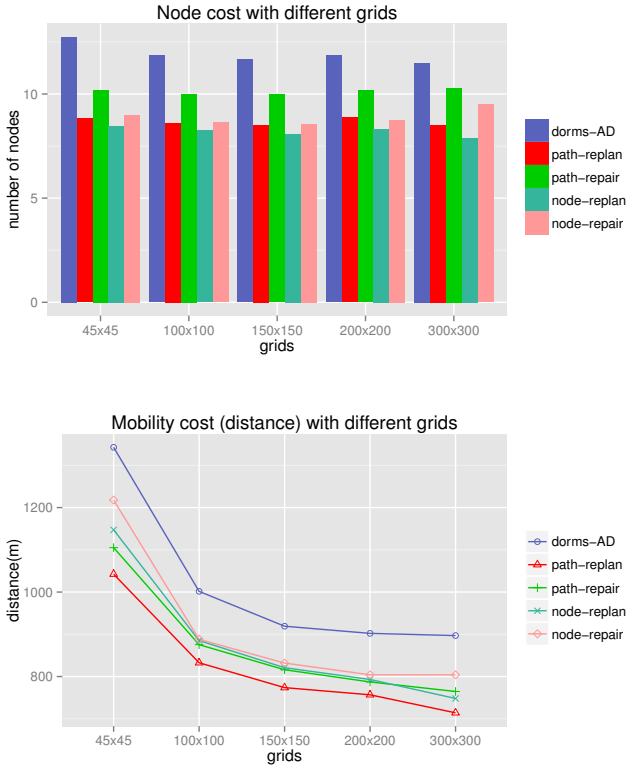


Fig. 6. varying the granularity of the mobility grid

Finally, the mobility costs are associated only with the distance travelled. For the application, one of the most important objectives is the time required to complete the repair. Elements of the process that contribute to the total time include movement along the path, probing, placing each node, and also the time to plan and replan during execution. We assume that it takes the agent 30s to position a new node. We then consider two scenarios, one representing a small robot which moves at 0.1ms^{-1}, and the second representing a larger vehicle moving over rough terrain at 4ms^{-1}. For the 45×45 grid in the $300m \times 300m$ area, the individual squares are of size $6.66m \times 6.66m$. The results are shown in Tables V and VI and Figure 7.

	45x45	100x100	150x150	200x200	300x300
dorms-AD	1.116	1.864	8.076	25.297	140.144
path-replan	5.157	6.147	8.327	21.775	97.639
path-repair	0.890	1.728	6.540	19.560	101.346
node-replan	1.872	4.335	15.505	46.725	247.647
node-repair	0.999	2.380	9.544	26.594	156.865

TABLE IV
RUNTIME (SEC) VS GRID GRANULARITY LEVELS.

(a) varying number of candidate locations

	50	100	150
dorms-AD	10165.135	10995.857	10899.655
path-replan	8578.667	8511.878	8019.941
path-repair	8923.116	8926.930	8572.335
node-replan	9357.120	9222.874	8987.971
node-repair	9177.040	9146.163	8953.244

(b) varying number of terminals

	5	10	15
dorms-AD	10995.784	16610.796	19284.185
path-replan	8512.006	13226.919	14899.113
path-repair	8926.919	14054.944	15605.873
node-replan	9222.860	14220.271	16729.129
node-repair	9146.099	14219.411	16771.184

(c) varying damage levels

	10%,10%	20%,20%	30%,30%
dorms-AD	10986.755	13826.722	20026.007
path-replan	8511.937	10720.676	15661.069
path-repair	8926.952	11381.427	17081.394
node-replan	9222.842	11730.484	17348.764
node-repair	9146.195	12503.394	18277.449

TABLE V
TOTAL RESTORING TIME (SEC) WITH SPEED V=0.1MS^{-1}

For the slow moving agent (Table V), the path priority approach is faster, since the extra runtime is recovered by shorter paths for the agent. In all cases, path priority with full replanning is the fastest algorithm, despite having almost an order of magnitude higher runtime. DORMS is consistently slower. For the fast agent (Table VI), the path priority with full replanning is unable to compensate for the longer runtime and extra nodes in some cases. Node priority becomes the fastest algorithm. As the movement speed of the agent increases, the time taken to place the nodes becomes more significant, and the node priority approaches benefit from their lower node costs and faster runtime.

Figure 7 shows the results for the granularity of the mobility grids. Surprisingly, in almost all cases, full replanning is faster than or competitive with local repair, even in the high damage cases. With the slow moving agent, the total restoring time for all methods almost consistently reduces with the finer grids,

(a) varying number of candidate locations

	50	100	150
dorms-AD	585.435	666.057	692.055
path-replan	486.167	492.178	524.141
path-repair	518.616	539.330	553.935
node-replan	471.620	462.174	487.271
node-repair	502.140	505.063	511.044

(b) varying number of terminals

	5	10	15
dorms-AD	665.984	1004.296	1234.985
path-replan	492.306	749.519	948.813
path-repair	539.319	836.544	997.773
node-replan	462.160	756.171	947.129
node-repair	504.999	795.611	986.584

(c) varying damage levels

	10%,10%	20%,20%	30%,30%
dorms-AD	656.955	735.722	919.907
path-replan	492.237	557.926	694.169
path-repair	539.352	606.594	753.394
node-replan	462.142	545.067	690.564
node-repair	505.095	630.060	778.149

TABLE VI
TOTAL RESTORING TIME (SEC) WITH SPEED V=4MS^{-1}

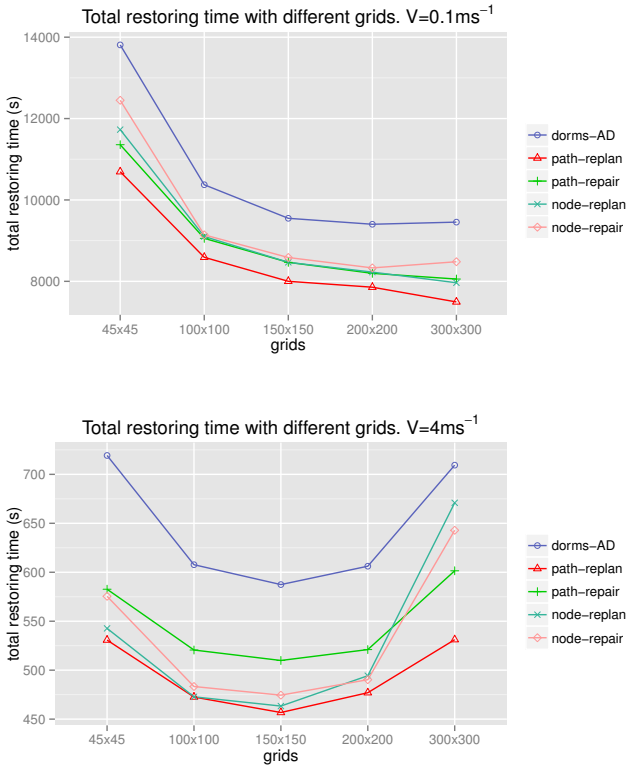


Fig. 7. Total restoring time with different mobility grids

and the path-replan always remains the fastest. For the fast moving the agent, the restoration time starts to drop with finer grids, but then starts to rise again. This is because the time taken to place nodes is more significant, and so the shorter path length cannot compensate for the increased runtime. We note that for the 300x300 grid, there is no benefit in restoration time over the 45x45 grid, although the total distance travelled will be reduced.

The experiments quantify the trade-off between the costs of using additional nodes and the total restoring time, and thus offer guidance to network repair operators in selecting a specific strategy. In all cases, the node-replan heuristic always offers the lowest node costs and this algorithm would be a good choice for applications where cost dominates, for example where nodes may be expensive. In other situations the time to restore the network may be a greater consideration, in which case the path-replan is most suitable.

VII. CONCLUSION

We have proposed a solution for the problem of connectivity repair for damaged wireless sensor networks, in which the repairing agent must discover the damage autonomously. We consider heuristic approaches, and compare the use of full replanning when damage is discovered to local repair of the immediate path plan. The full replanning approach produces better quality plans, but at the cost of sometimes significantly higher runtime, particularly where the connectivity graphs are dense or the damage is extensive. We also assess the total time to repair, and we demonstrate that the full replanning can be more efficient, particularly when the agent is slow moving, or where it can reduce the high costs of repair actions.

For future work, we are developing local greedy heuristics, to further explore the tradeoff between cost and repair time. We will also consider distributed algorithms, allowing collaboration between multiple agents and sensor nodes. Finally, we aim to tackle the situation where the network may suffer from ongoing damage during the repair process.

ACKNOWLEDGMENT

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

REFERENCES

[1] A. A. Abbasi, M. F. Younis, and U. A. Baroudi, "A least-movement topology repair algorithm for partitioned wireless sensor-actor networks," *International Journal of Sensor Networks*, vol. 11, no. 4, pp. 250–262, 2012.

[2] ——, "Recovering from a node failure in wireless sensor-actor networks with minimal topology changes," *IEEE Transaction on Vehicular Technology*, vol. 62, no. 1, pp. 256–271, 2013.

[3] K. Akkaya, F. Senel, A. Thimmapuram, and S. Uludag, "Distributed recovery from network partitioning in movable sensor/actor networks via controlled mobility," *IEEE Transaction on Computers*, vol. 59, pp. 258–271, 2010.

[4] H. M. Almasaeid and A. E. Kamal, "On the minimum k-connectivity repair in wireless sensor networks." in *ICC*, 2009, pp. 1–5.

[5] M. Batalin and G. S. Sukhatme, "The design and analysis of an efficient local algorithm for coverage and exploration based on sensor network deployment," *IEEE Transactions on Robotics*, vol. 23, no. 4, pp. 661–675, 2007.

[6] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, "Using iterative repair to improve responsiveness of planning and scheduling," in *AIPS*, 2000, pp. 300–307.

[7] P. Doherty, J. Kvarnström, and F. Heintz, "A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems," *Journal of Autonomous Agents and Multi-Agent Systems*.

[8] M. R. Garey, R. L. Graham, and D. S. Johnson, "The Complexity of Computing Steiner Minimal Trees," *Journal of Applied Mathematics*, vol. 32, no. 4, pp. 835–859, 1977.

[9] B. Khelifa, H. Haffaf, M. Merabti, and D. Llewellyn-Jones, "Monitoring connectivity in wireless sensor networks." in *ISCC*, 2009, pp. 507–512.

[10] S. Koenig and M. Likhachev, "D*lite," in *AAAI*, 2002, pp. 476–483.

[11] S. Lee and M. Younis, "Recovery from multiple simultaneous failures in wireless sensor networks using minimum steiner tree," *Journal of Parallel Distributed Computing*, vol. 70, pp. 525–536, 2010.

[12] M. Molineaux, M. Klenk, and D. W. Aha, "Goal-driven autonomy in a navy strategy simulation." in *in Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[13] *Tmote Sky Datasheet*, Moteiv Corporation, http://www.eecs.harvard.edu/ konrad/projects/shimmer/references/tmote-sky-datasheet.pdf, 2006.

[14] K. L. Myers, "Cpef: A continuous planning and execution framework." *AI Magazine*, vol. 20, no. 4, pp. 63–69, 1999.

[15] O. Pettersson, L. Karlsson, and A. Saffiotti, "Model-free execution monitoring in behavior-based robotics," *IEEE Transaction on Systems, Man and Cybernetics, Part B*, vol. 37, no. 4, pp. 890–901, 2007.

[16] S. Poduri, S. Pattem, B. Krishnamachari, and G. S. Sukhatme, "Using local geometry for tunable topology control in sensor networks," *IEEE Transactions on Mobile Computing*, vol. 8, pp. 218–230, 2009.

[17] F. Senel, M. Younis, and K. Akkaya, "A robust relay node placement heuristic for structurally damaged wireless sensor networks," in *LCN*, 2009, pp. 633 –640.

[18] I. F. Senturk and K. Akkaya, "Energy and terrain aware connectivity restoration in disjoint mobile sensor networks," in *12th IEEE International Workshop on Wireless Local Networks at LCN 2012*, 2012, pp. 767–774.

[19] ——, "On the performance of sensor node repositioning under realistic terrain constraints," in *LCN*, 2012, pp. 336–339.

[20] I. F. Senturk, S. Yilmaz, and K. Akkaya, "Connectivity restoration in delay-tolerant sensor networks using game theory," *Journal of Ad Hoc and Ubiquitous Computing*, vol. 11, no. 2/3, pp. 109–124, 2012.

[21] M. Sir, I. Senturk, E. Sisikoglu, and K. Akkaya, "An optimization-based approach for connecting partitioned mobile sensor/actuator networks," in *3rd International Workshop on Wireless Sensor, Actuator and Robot Networks, at INFOCOM 2011*, 2011, pp. 525 –530.

[22] F. Teichteil-Knigsbuch, C. Lesire, and G. Infantes, "A generic framework for anytime execution-driven planning in robotics," in *ICRA*, 2011, pp. 299–304.

[23] T. T. Truong, K. N. Brown, and C. J. Sreenan, "Integration of node deployment and path planning in restoring network connectivity," in *The 29th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSig)*, 2011.

[24] ——, "Restoring wireless sensor network connectivity in damaged environments." in *International Workshop on Cooperative Robots and Sensor Networks (RoboSense)*, 2012.

[25] ——, "Repairing wireless sensor network connectivity with mobility and hop-count constraints," in *ADHOC-NOW, 2013*, 2013, pp. 75–86.

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

[27] M. M. Zavlanos and G. J. Pappas, "Distributed connectivity control of mobile networks," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1416–1428, 2008.