CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	An online approach for wireless network repair in partially-known environments
Authors	Truong, Thuy T.;Brown, Kenneth N.;Sreenan, Cormac J.
Publication date	2016-03-10
Original Citation	Truong, T. T., Brown, K. N. and Sreenan, C. J. (2016) 'An online approach for wireless network repair in partially-known environments', Ad Hoc Networks, 45(Supplement C), pp. 47-64. doi: 10.1016/j.adhoc.2016.02.021
Type of publication	Article (preprint)
Link to publisher's version	http://www.sciencedirect.com/science/article/pii/S1570870516300622 - 10.1016/j.adhoc.2016.02.021
Rights	© 2016 Elsevier B.V. This manuscript version is made available under the CC BY-NC-ND 4.0 license. - http://creativecommons.org/licenses/by-nc-nd/4.0/
Download date	2026-11-06 03:55:11
Item downloaded from	https://hdl.handle.net/10468/5095

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

An Online Approach for Wireless Network Repair in Partially-known Environments

Thuy T. Truong, Kenneth N. Brown, Cormac J. Sreenan

CTVR, Department of Computer Science, University College Cork, Ireland

Abstract

Wireless Sensor Networks in volatile environments may suffer damage which creates network partitioned, and connectivity must be restored. We present a model of the online problem, in which 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 focuses, one which locally focuses on a partition at a time and the other one globally focuses on all partitions at each step of the repairing progress. We apply each focus with two different planning priorities, one which attempts to minimise the cost of new radio nodes, and one which attempts to minimise the travel distance. For each priority, two strategies will be considered: full replanning re-generates a full plan when the agent discovers new knowledge, and the repairing tries to fix/repair the current plan in order to minimise the computation effort. We evaluate the approach in simulation, varying the density of the connectivity graph and the level of damage suffered. We demonstrate that the repairing method, while producing more expensive solutions, can require significantly less computation time, depending on the choice of heuristic. Finally, we evaluate the total time to repair the network for different speeds of agent, and we show the relative importance of the agent speeds on the two focuses. In particular, the algorithms which produce low mobility cost will be preferred with slow agents while with fast moving agents, the node based approaches will be preferred.

Keywords: Wireless Sensor Network, Node Deployment, Connectivity Repair.

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, attack or accidents involving wildlife and weather, and may 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 for important data streams.

There are four main challenges 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. In other words, the repairing agent, i.e. a robot or a human, 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.

Email addresses: t.truong@cs.ucc.ie (Thuy T. Truong), k.brown@cs.ucc.ie (Kenneth N. Brown), cjs@cs.ucc.ie (Cormac J. Sreenan)

We model the repairing task as an online approach in a domain specific problem formulation. The agent starts with knowledge of the radio and physical environments before the damage, and is aware of the connected sub-network after the damage, i.e. the set of nodes still successfully transmitting data to our sink. 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. However, while executing the plan, the agent will encounter blocked paths and broken radio links, but also surviving disconnected components of the old network, and as it discovers new knowledge, it must revise its plans to complete the repair. When it discovers knowledge that renders its current plan infeasible or changes its cost significantly, it will update its plan and continue.

Since both subproblems (connectivity and multi-point path) are computationally hard, we use heuristic algorithms to generate the plans, with two different priorities: prioritising the number of new nodes, and prioritising the path length. We have two focuses: a local focus, which first picks a terminal to connect, generates a plan, executes it, and then selects the next terminal, and a global focus, which generates a plan for reconnecting all terminals before it begins to execute the plan. For each focus, we consider multiple heuristics based on the two priorities: prioritising the node cost and prioritising the mobility cost. The first prefers plans that require few nodes and then finds cheap paths for visiting those locations while the second prefers cheap paths that would allow the agent to restore connectivity. We also consider two strategies: replanning and repairing. The first conducts full replanning whenever it discovers new knowledge that changes the cost of the plan significantly (e.g. the cost exceeds a threshold), or which renders the plan infeasible. The second attempts to repair the plan, by searching for a new motion path to reach its current location target, and reverting to full replanning only after significant changes. Summary of our approaches is in Table 1. The approaches we develop can be classified according to the focus (local or global), the priority (node cost or path cost), and the planning techniques (replan or repair). There are thus eight high level approaches, and we implement different heuristics for some of these approaches to provide different choices of algorithms.

FOCUS	Global	Local
PRIORITY	Node cost	Path cost
PLANNING TECHNIQUE	Replanning	Repairing

Table 1: Summary of the approaches

We evaluate the approaches in simulation on randomly generated problems, assessing the impact of increasing damage, increasing number of nodes to be connected, and increasing locations for radio nodes. We have found that the costs all increase when we increase the disconnected terminals (i.e. the desired locations where we want to get data report from but we could not due to the partitioned network) and the damage levels but do not always increase with the number of locations. In some cases, the full replanning methods (with global focus) in both node and path approaches lose to few locally focused heuristics. This is due to the unknown environments which make the initial global decision poorer. In addition, we show that different movement speeds of the repairing agent have a significant impact on performance, and must be taken into account when selecting the algorithm. With slow agents, the time to move through the sensor field outweighs the time to place nodes and the computation time. Therefore, the algorithms which produce low mobility cost should be prioritised. With medium speed, the gap between the path and the node approaches in total restoration time is smaller. However, with a fast moving agent, the speed means that the higher mobility costs are less significant, and thus the time to place nodes and the runtime become more important. Therefore, the node based approaches should be preferred in this situation.

In the remainder of the paper, we discuss related work, then we 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 the conclusions.

2. Related Work

The subject of network restoration for wireless networks is an active area of research. The different approaches can be classified as (i) deploying redundant nodes 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, (iv) the deployment of additional nodes to restore connectivity after failures have occurred, and (v) sensor relocation by mobile robots.

Deploying redundant nodes to achieve a level of connectivity

In [1],[2], [3], [4], [5], the goal is to deploy redundant nodes with the intention of achieving k-connectivity. The main idea of these papers is to place redundant nodes at some calculated locations to create a k-connected graph. Those redundant nodes start in sleeping mode and only wake up to offer new paths if a node fails. These approaches tend to require many redundant nodes, which makes them expensive. Finally, the main focus is not on repairing the damaged network, but on achieving fault tolerance.

Repairing connectivity in Wireless Sensor and Actor Networks (WSAN)

Wireless Sensor and Actor Networks (WSANs) are networks of sensors and actors that communicate via a wireless medium to perform distributed sensing and actuation tasks. Actors usually take decisions and perform suitable actions upon the information collected from the sensors. The actors collect data from the nearby sensors and can exchange information with other actors to make the right decisions. Several papers consider the use of mobile actor nodes in network restoration, e.g. [6],[7], [8], [9], [10], [11], [12], [13], [14]. The papers assume that all sensors are connected and propose different strategies to choose the moving actors, for example, based on estimating the shortest moving distance and/or degree of connectivity to achieve goals of connectivity or coverage for the actor network (an overlay network of all actors).

The repair methods discussed above are for restoring the connectivity for a single node failure at a time only. The work is extended in [15] to deal with multiple failures. It proactively pre-computes cut-nodes and the Connected Dominating Set (CDS), designates the appropriate neighbours to cover them if they fail and then applies cascaded movement (i.e. block movement where the movement of each node depends on that of all previous moving nodes in order to maintain the network connectivity) for replacement. Therefore, this work involves all dominatees (i.e. the nodes whose absence do not lead to any partitioning of the network) to the cut-node. The work is different from our work in which they assume that mobility is unimpeded by obstacles in free space.

Dispatching mobile nodes to avoid disconnection

[16] 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.

[17] deploys mobile robotic helper nodes to physically carry the data to the base station. The approace designates the speed for those mobile helpers in order to carry data with delay-tolerance. However, this work is only suitable for applications with delay-tolerance and where those helper nodes can move in free space to bring the data back.

Deploying additional nodes to repair the connectivity

[18], [19], [20], [21], [22], [23], [24], and [25] 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. [18] uses a spider web approach to reconnect the segments. In [20], the authors propose a Distributed algorithm for Optimized Relay node placement using Minimum Steiner tree (DORMS). This approach forms a connectivity chain from each segment toward a centre point and then seeks to optimize the number of additional nodes that are needed. [22] and [23] also propose algorithms using minimum Steiner trees where [22] finds the best subsets of three segments and forms a triangular Steiner minimum tree with minimum Steiner points while [23] uses a minimum Steiner tree on the Convex hull and places relay nodes inwards towards the center of the damage area.

Also [19] and [21] model the area as a grid of $R\sqrt{2}$ size squares where R is the transmission range of a node and then map the problem of finding the optimal number and position of relay nodes into the problem of finding the cell-based least-cost paths that connect all partitions in the network and also meet the QoS requirement.

In [25], the authors also consider different aspects of a segmented network such as the sizes and shapes of segments, and possible holes in segments. Besides the node cost, the paper also minimises the average path length from a centre point of each segment to the sink. The paper assumes a static segmented network and a uniform distribution of mobile/relay nodes (MNs), and proposes centralised and distributed connectivity restoration. The centralised approach uses a genetic algorithm which applies a heuristic to reduce the search space. The distributed approach establishes the connection between two adjacent segments without considering all the segments in a network. Therefore, the distributed approach has lower overhead but it is more costly (longer path length to the sink, more MNs used) than the centralized approach.

All the above work assumes a free space where nodes can move freely to achieve minimum travel distance or other goals (minimum number of nodes, etc). The work also assumes uniform transmission range modelled as a disk centred at the node, and thus ignores radio propagation obstacles as well as obstacles to free movement. The last approach above is closest to our research but different in three respects, firstly in that we optimise both the number of additional nodes as well as the path length needed for their deployment, 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, and thirdly we model the problem as continual planning task where the agent has to discover the environments in order to perform tasks.

Sensor relocation by mobile robots

Random deployment or sensor failures in Wireless Sensor Network may cause sensing holes and redundant sensors. The work in this category deploys a team of robots to relocate sensors and improve the area coverage. Existing work focuses on two main approaches: centralized approaches ([26], [27], [28]) where one or more robots are located at a base station which has full knowledge of the network and the algorithms to find robot trajectory to repair the network coverage are run globally; and localized solutions ([29], [30], [31]) where each robot may carry at most one sensor and makes decisions that depend only on locally detected information. The work focuses on rearranging the redundant sensors to cover any sensing holes and it aims to maximize the network coverage.

Other related work in WSNs

[24] uses game theory to reconnect the network. Again, the work assumes free space where the mobile nodes can move freely and new nodes can be placed in any positions. [32], [33] and [34] consider more realistic terrain with obstacles, assuming all terrain and all network conditions are known in advance.The methods focus on networks where the nodes themselves are mobile, and consider single instances of moving a node to re-establish a link without breaking other links.

[35] and [36] consider on distributed mobile nodes for re-establishing network connectivity. [35] relocates some of the sensors to the locations of the failed sensors to re-establish the routes with the sink node based on local information. [36] proposes two distributed relay node positioning approaches which use virtual force-based movements of relays and Game Theory respectively to guarantee network recovery for partitioned WSNs.

There is also research on topology control using mobile agents. [37] deploys a robot with unlimited nodes and drops nodes from time to time based on certain ordering rules. [38] controls the agent's motion to explore the environment while dropping nodes and preserving the connectivity of the network. [39] assumes a mobile sensor network where nodes can use repulsion and attraction forces to arrange the topology. These papers focus on topology control and deployment but do not consider repair/restoration after damage has occurred.

Our work in this paper extends from [40], [41], and [42]. In [40] and [42], we introduce the network repair problem in the presence of obstacles in a static environment, and propose different heuristics to solve the problem. [41] models the network repair as a continual planning where an agent must discover surviving nodes and damage to the physical and radio environment as it moves around the sensor field to execute the repair. The paper focuses on 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. For each

approach, there are two different heuristics, one which attempts to minimise the cost of new radio nodes, and one which aims to minimise the travel distance. In addition to the findings in [41], there are a number of other solutions that have been explored in this journal. We also provides more results in the simulation to fully compare all the proposed solutions.

Continual Planning 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 [43]. The paper proposes a framework (called CPEF) for a continuous planning and execution system. It is based on a central Plan Manager responsible for the overall control of system operation: plan generation, monitoring, and execution. [44] uses iterative repair techniques to support a continuous planning process for autonomous spacecraft control. [45] and [46], for temporal planning, interleave decision and execution in a dynamic environment to allow plan repair interleaved with execution. [47] uses a model-free approach which observes and classifies the actual behavior of the monitored systems into normal or faulty execution. [48] dynamically reasons about which goals to pursue in response to unexpected circumstances. [49] proposes a generic and reactive scheme for continuous planning for complex problems. Finally, [50] describes a CSP-based continual planner for web service composition.

Most of the work in replanning focuses on two models: replanning as restarting (planning again) ([51], [52], [53], [54]), and replanning as repair (repairing the current plan locally) ([55], [56]).

[51] describes a framework that can use most heuristic planner/search for replanning. [52] combines ideas from the artificial intelligence and the algorithms literature. It repeatedly finds shortest paths from a given start vertex to a given goal vertex while the edge costs of a graph change or vertices are added or deleted while reusing information from previous searches. Given an optimal plan, the objective in [53] is to monitor its continued optimality, electing to replan only in those case where continued execution of the plan will either not achieve a goal, or will do so sub-optimally. [54] uses rewards and penalties to reason about adding or changing goals. Newly arriving goals are modelled with rewards, while commitments made by the existing plan are modelled with penalties. For example, a robot is on its way to rescue an injured person and discovers a group of children stuck in a burning home. The replanner must resolve the problem, exploit the opportunities but also respect the commitments inherent in the current plan.

[55] describes the Least Cost Flaw Repair strategy where it defines a repair cost for any flaw (threat or open conditions) and selects the flaw with minimal cost to repair. [56] describes a planning system which interleaves plan generation, execution, and repair in a dynamic environment. It generates a solution plan for a problem, and then executes it. If an action fails while the plan is not finished, it attempts to repair only the related part of the failing action. This approach keeps track of a dependency graph which contains a derivation tree for the generated plan and the causal links between the nodes of that tree. The causal links show the relationship between the effects of an action to be executed now in the plan and the task decompositions occurring in future. Using this graph, it can find and repair only the related part of the plan.

3. Problem Formulation

Network repair is the problem of placing new nodes in the environment to restore connectivity to the sink for all sub-partitions. The agent has to discover the network and physical environments in order to make the right decision. Our aim is to optimise our use of resources in partially known environments. We develop locally and globally focused methods in which many heuristics have been proposed based on the two priorities: prioritising the node cost and prioritising the mobility cost.

To represent the problem of exploring an environment to discover mobility paths, we assume an underlying grid representation. Grid model is a standard representation in robotics problems, where the robot must determine whether a neighbouring square is accessible before it moves into it. We give a description of the problem formulation as follows with the notation listed in Table 2. We will model the environment in two parts: the connectivity and mobility environment before the damage, and the connectivity and mobility after the damage. The environment before damage is known completely, for example, from a survey performed by robot or human agents, and the task is to discover as much of the after-damage environment as is needed to complete an effective repair. 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. This is because many protocols across different layers require symmetric links for proper functioning, e.g. the MAC layer relies on symmetric links for acknowledgements and many routing protocols assume symmetric links. $V_B \subseteq V_b$ 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 four 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 $L \in I_v$.

Notation	Description
G	a rectilinear grid of locations
$V_b \subseteq G$	set of candidate locations for radio nodes
C_b	set of potential radio links between the locations
(V_b, C_b)	a connectivity graph before damage
V_B	set of locations with actual nodes (live nodes) before damage
$V_a \subseteq V_b$	a set of candidate locations for radio nodes after damage
C_a	set of potential radio links between the locations after damage
(V_a, C_a), $V_a \subseteq V_b$, $C_a \subseteq C_b$	a connectivity graph after damage
$\tau \subset V_b$	set of terminals
$V_A \subseteq V_B \cap V_a$	a set of locations with surviving nodes
$I_v \subseteq V_A$	set of nodes still successfully transmitting data to our sink.
$I_c \subseteq I_v \mathrm{x} I_v$	set of active links between nodes in I_v correspondingly
$B_b \subseteq G$	set of blocked squares before damage
$B_a \subseteq B_b$	set of blocked squares after damage
V_n	set of newly added nodes
$\lvert V_n \rvert$	number of nodes to be deployed
$L \in I_v$	starting location of the agent
P	path through the grid
d	a distance the agent can probe
R	transmission range of the agent

Table 2: Table of Notation

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 communication 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 [1] the accessibility of its neighbouring squares up to a distance of d, but cannot probe a square if there is a blocked square in between. 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. We assume 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. Figure 1 shows an illustrative example of the network conditions.

[1] Using radar or sensing or by physically moving

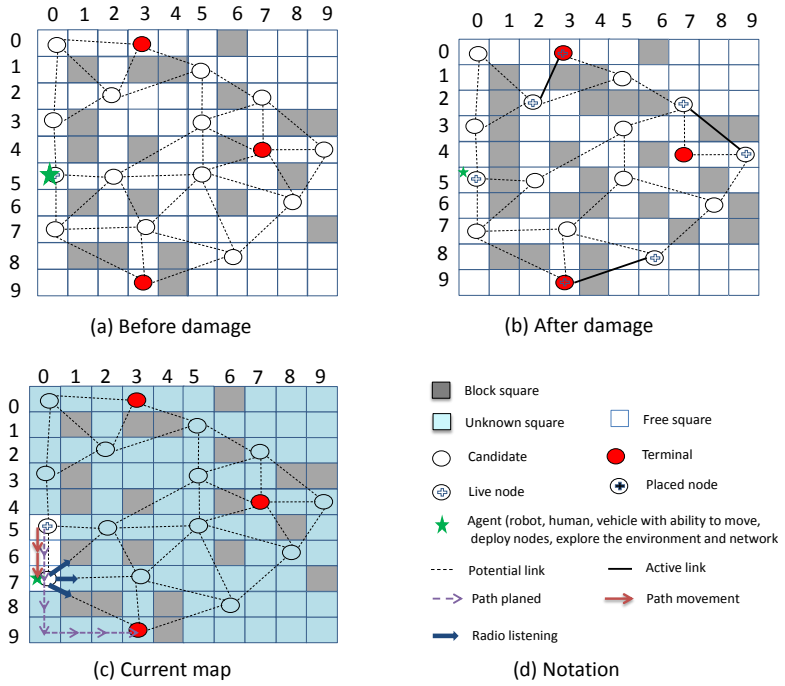


Fig. 1: Example of the network conditions.

4. 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 (locations, radio links, etc.) that are known to be active, those that are known to be damaged, and those that have not been verified. We assume the agent will use the prior knowledge of the map for reference in planning, but it still has to discover if there is any change in the radio and the physical environments.

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 $S_f \subseteq G - B_a$.

We base on the STRIPS-style rules which is widely used throughout the AI planning literature ([57], [58], [59]) to specify the changes to the world state, but we allow arbitrary procedures to represent changes to the agents knowledge structures. 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. The preconditions are the conditions that the agent must meet before the action is performed, the add list is to add new states established after the action occurs, and the delete lists is to remove the states that are no longer valid at the end of the action. 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) $\wedge$ neighbor(u, v) $\wedge$ $v \notin B_a$;
   DEL: at(u);
   ADD: at(v);
   DESCRIPTION: The agent must be at u at the start, which has a neighbor v that is not blocked. After moving, the agent is no longer at u but now at v.
2. LISTEN(u): listen for radio signals at u;
   PRE: at(u);
   PROC: $(\mu, \epsilon) \leftarrow$listen(); //listen() reports live nodes and links
   $N_a \leftarrow N_a \cup \mu$; // any new overheard nodes are active
   $N_f \leftarrow N_f \cup \mu$; // and now known to be in feasible locations
   $N_u \leftarrow N_u - \mu$;
   $E_f \leftarrow E_f \cup \epsilon$; //similarly for links
   $E_u \leftarrow E_u - \epsilon$;
   if $((x \in N_a \| x = u) \&\& y \in N_a \&\& \{x, y\} \notin E_u)$
   then $E_i \leftarrow E_i \cup \{\{x, y\}\}$; //deduce blocked links
   DESCRIPTION: The agent is at location u and tries to listen to any signal from surroundings. The onboard radio will report to the agent the surrounding live nodes in μ and radio links in ϵ. If there is any new discovered live nodes in μ and/or radio links in ϵ, then the agent will update the related lists above. After the update, if there is no reported links for two known active nodes, we can deduce that the communication link between the two nodes is impossible, i.e. blocked link due to obstacles or out of range communication.
3. DROP(u): drop a node at u;
   PRE: at(u);
   PROC: if ($u \in V_a$) then $V_n \leftarrow \{u\}$;
   $N_a \leftarrow N_a \cup \{u\}$;
   DESCRIPTION: If the agent is at location u where u is a candidate location after damage, then the agent will drop a new node at u and update this location into the newly added node list V_n and the active radio node list N_a.
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 $S_u \leftarrow S_u - \{v\}$; $S_f \leftarrow S_f \cup \{v\}$;
   else if (p(u, v)==F) // F if v in range but blocked

then $S_u \leftarrow S_u - \{v\}; S_b \leftarrow S_b \cup \{v\}$;
//p(u,v) reports ? if v not in range
DESCRIPTION: The agent is at location u and tries to probe square v to see if location v is accessible or not. If v is in range and accessible, then remove this location from the set of unknown locations S_u and add it into the set of accessible locations S_f; otherwise, add it into the set of blocked/inaccessible locations S_b. If v is not in range, then no information is updated.

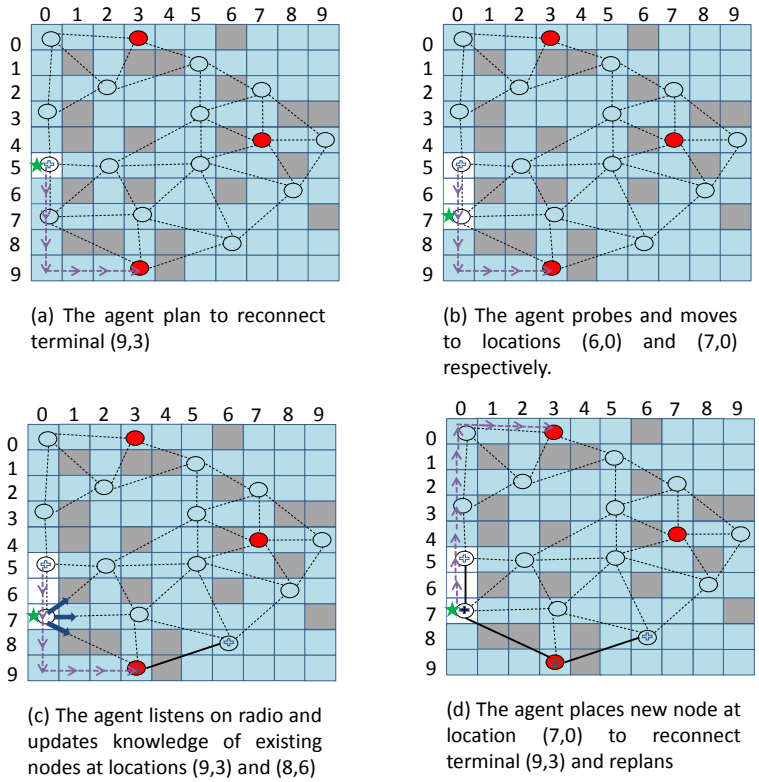
5. INSPECT(u): check if u can take a radio node;
PRE: at(u);
PROC: if (insp(u)==T) // T if $u \in V_a$
then $N_u \leftarrow N_u - \{u\}; N_f \leftarrow N_f \cup \{u\}$;
else $N_u \leftarrow N_u - \{u\}; N_i \leftarrow N_i \cup \{u\}$;
DESCRIPTION: The agent must be at location u. If location u is feasible for placing a node, then remove this location from the set of unknown locations N_u and add it into the set of feasible locations N_f.

Figure 2 shows a sequence of actions to reconnect the terminal at the location (9,3). The first plan is produced based on the initial knowledge:

LISTEN((5,0));
PROBE((6,0));
MOVE((5,0),(6,0));
PROBE((7,0));
MOVE((6,0),(7,0));
LISTEN((7,0));
INSPECT((7,0));
DROP((7,0));
PROBE((8,0));
MOVE((7,0), (8,0));
PROBE((9,0));
MOVE((8,0), (9,0));
PROBE((9,1));
MOVE((9,0), (9,1));
PROBE((9,2));
MOVE((9,1), (9,2));
PROBE((9,3));
MOVE((9,2), (9,3));
LISTEN((9,3));
INSPECT((9,3));
DROP((9,3));

However, as shown in Figure 2, the agent follows the plan and executes the actions:
LISTEN((5,0));
PROBE((6,0)); with $S_f \leftarrow \{(5,0),(6,0)\}$,
MOVE((5,0),(6,0));
PROBE((7,0)); with $S_f \leftarrow \{(5,0),(6,0),(7,0)\}$,
MOVE((6,0),(7,0));
LISTEN((7,0)); with $N_f = N_a \leftarrow \{(5,0),(9,3),(8,6)\}, N_u \leftarrow N_u\ \{(9,3),(8,6)\}$
INSPECT((7,0)); with $N_f \leftarrow \{(5,0),(9,3),(8,6),(7,0)\}, N_u \leftarrow N_u\ \{(7,0)\}$
DROP((7,0)); with $N_a \leftarrow \{(5,0),(9,3),(8,6),(7,0)\}$

At this point, where the agent reaches the location (7,0), it listens on its radio and hears the messages from surviving node at location (9,3) which is also connected with another surviving node at location (8,6). The agent will update the knowledge it has learned. After dropping a new node at location (7,0), the terminal at location (9,3) is reconnected, thus the agent completes the first task (it abandons the current plan as the goal is achieved). The agent then repeats the process to reconnect other terminals.



(a) The agent plan to reconnect terminal (9,3)

(b) The agent probes and moves to locations (6,0) and (7,0) respectively.

(c) The agent listens on radio and updates knowledge of existing nodes at locations (9,3) and (8,6)

(d) The agent places new node at location (7,0) to reconnect terminal (9,3) and replans

Fig. 2: Example of the agent's actions.

5. Approach

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 this paper, the agent assumes that some elements of the unknown sets are available, and then replans when errors are discovered, and is the method 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.

We have two focuses: a greedy/locally focused method which repairs each terminal at a time, i.e. it picks a terminal to connect, generates a plan, executes it, and then selects the next terminal until all terminals are connected; and a globally focused method, which generates a plan for reconnecting all terminals before it begins to execute the plan. For each focus, we consider two strategies for solving the problem: prioritising node cost and prioritising path cost. The node heuristics prefer plans that require few nodes and then finds cheap paths for visiting those locations. The path heuristics prioritise mobility cost, and prefers cheap paths that would allow the agent to restore connectivity. In all cases, the process is iterative. The first

plan is generated based on the pre-damage map. As the agent starts to move through the environment, it discovers information about mobility and connectivity, and updates its knowledge. When the current plan is considered too expensive, a new plan is generated, and the process repeats.

The overall approach, applicable to both heuristic approaches and to both full replanning and repairing, is shown in Figure 3.

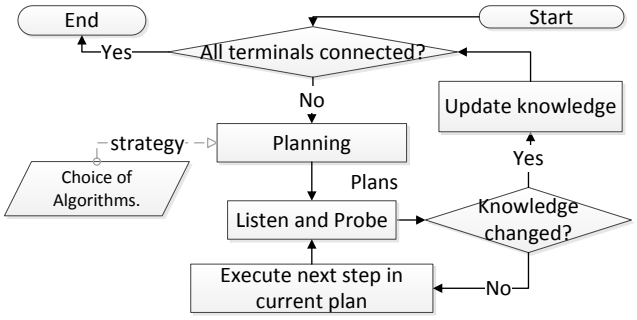


Fig. 3: Flow chart for the agent planning.

5.1. Focus 1: local focus

This focus is based on the belief that the knowledge will change significantly as we explore, and so it is fuetile to spend time generating a complete plan which will almost definitely change. Instead, it takes a greedy approach, picking the best terminal to reconnect first. Once it has connected that terminal (or discovered significant changes), it will move on and consider a second terminal. There are two important questions: firstly, how to select the terminal which offers the cheapest path in a reasonable time; and secondly what happens if the agent discovers new knowledge while connecting a target terminal, whether it continues to reconnect that terminal or generate a new plan which might change the target terminal. To evaluate the approach, we propose a set of greedy algorithms: algorithms which generate new plans whenever the agent discovers new knowledge which may change the target terminal, and algorithms which fix the target terminal until that terminal gets connected. We also propose a number of different choices for selecting the terminal to reconnect.

Algorithm 1: Local Node (L-N) Algorithm

1 find $t \in \tau$, the terminal requires least node cost to connect to the network
2 find N, the set of required nodes to reconnect t
3 find P, the cheapest path to the nearest node in N
4 return (N, P)

Strategy 1: prioritising node cost: In this strategy, all local approaches that prioritise node cost will be labelled with L-N- (for Local-Node-). We have two different choices: changing the target terminal every time knowledge changes (L-N-c-) and fixing the terminal regardless of changes to knowledge (L-N-f-), until that terminal is reconnected. For selecting which terminal to connect next, based on the node cost, we have two different heuristics : the first heuristic is to estimate the number of nodes from a terminal to the network based on Manhattan distance (L-N-x-MD, where x is "c" or "f" and MD stands for Manhattan Distance), and the second is to calculate the optimal number of nodes from a terminal to the network (L-N-x-FN, FN stands for Fewest Node). In the second heuristic, the agent first builds a directed weighted connectivity graph. Each candidate location will be a vertex, with connected components merged into supernodes. Each potential link will be 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 runs Dijkstra's algorithm to find the cheapest path from the current network to each terminal, where the cheapest path will

be the one with fewest additional nodes. The agent then selects the terminal which requires the fewest nodes. Then, at each stage, the agent finds a path to the closest one of these nodes using D* Lite ([60]). As above, if the agent discovers information that changes the node costs (live nodes found, broken links in the current node plan), it recomputes.

In summary, we have the L-N- algorithm presented in Algorithm 1 with different heuristics: L-N-c-MD, L-N-f-MD for Local Node for changing/fixing target terminal based on Manhattan distance, and L-N-c-FN, L-N-f-FN for Local Node for changing/fixing target terminal based on calculating the terminal with least number of required nodes.

Strategy 2: prioritising mobility/path cost: In this strategy, all local approaches that prioritise path cost will be labelled with L-P-. Similarly to the above strategy, we have two different choices: possibly changing (L-P-c) and fixing (L-P-f) the target terminal. We encounter the same question as above: how to select the first unconnected terminal to be reconnected. As this strategy is based on the path cost, the agent always looks for the closest terminal. We have three different heuristics for selecting a target terminal: based on Manhattan distance (L-P-c/f-MD), or based on shortest distance calculated using D* ([60]) between the terminal to the agent's location (L-P-c/f-SD), or shortest connectivity path to reconnect the terminal to the current network starting from the agent (L-P-c/f-SCP). The third heuristic requires more computation that is complicated, and therefore we expect a longer runtime. However, we also expect better results in mobility cost as this heuristic takes into account the agent's location to restore connectivity for a terminal to the network. After selecting a target terminal, the agent then searches for the cheapest connectivity path which connects the terminal to the network. It then begins to execute the plan, until it discovers blocked squares, blocked radio links, or live nodes. At that point, it updates its knowledge of the environment, and depending on the choices of algorithms, it might recompute, possibly finding a new terminal and a new path (L-P-c) or it finds new path to the current target terminal (L-P-f).

Algorithm 2: Local Path (L-P) Algorithm

1 find $t \in \tau$, the closest terminal to the agent
2 find N, the set of required nodes to reconnect t
3 find P, the cheapest path to the nearest node in N
4 return (N, P)

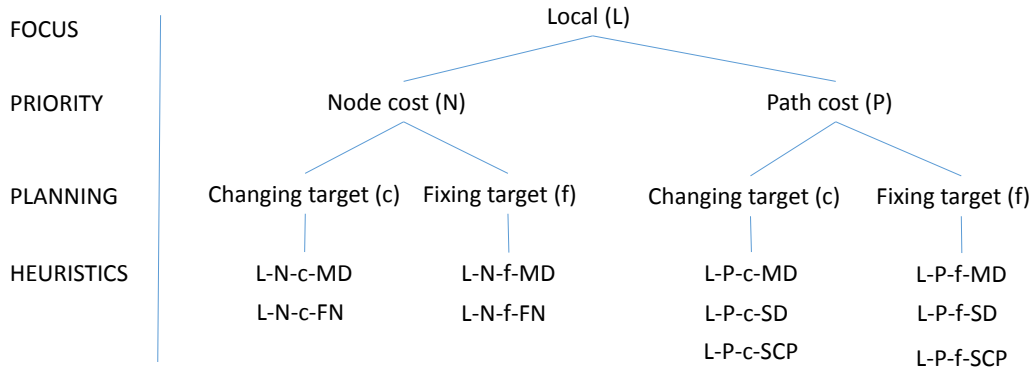


Fig. 4: A tree reprensentation for the local focus.

In summary, we have the L-P algorithm presented in Algorithm 2, and L-P-c-MD, L-P-f-MD for Local Path for changing/fixing target terminal based on Manhattan distance, L-P-c-MD, L-P-f-MD for Local Path for changing/fixing target terminal based on shortest distance, and L-P-c-MD, L-P-f-MD for Local Path for changing/fixing target terminal based on calculating the shortest connectivity path.

Algorithm	Description
L-N-c-MD	Manhattan distance for finding the closest terminal to the network to estimate the terminal which requires fewest number of nodes to the network, with D*lite for the search-while-moving part, change the target terminal if cost might change.
L-N-c-FN	Fewest nodes for finding the closest terminal to the network, with D*lite for search-while-moving, change the target terminal if cost might change.
L-N-f-MD	Same as L-N-c-MD but fix the target terminal until it is connected
L-N-f-FN	Same as L-N-c-FN but fix the target terminal until it is connected
L-P-c-MD	Manhattan distance for finding the closest terminal to the agent, with D*lite for the search-while-moving part, change the target terminal if cost might change.
L-P-c-SD	shortest-path-in-grid for finding the closest terminal to the agent, with D*lite for search-while-moving, change the target terminal if cost might change.
L-P-c-SCP	shortest-connectivity-path for finding the closest terminal to the agent, with D*lite for search-while-moving, change the target terminal if cost might change.
L-P-f-MD	Same as L-P-c-MD but fix the target terminal until it is connected
L-P-f-SD	Same as L-P-c-SD but fix the target terminal until it is connected
L-P-f-SCP	Same as L-P-c-SCP but fix the target terminal until it is connected

Table 3: Summary of locally focused approaches. L-P(Local-Path), L-N (Local-Node), c(change), f(fix), MD(Manhattan Distance), SD(Shortest Distance), SCP(Shortest Cheapest Path), FN (Fewest Node)

Figure 4 shows a tree representation that summarises the heuristics for the local focus, and the descriptions of these heuristics are in Table 3.

5.2. Focus 2: global focus

This focus will produce a plan to reconnect all the terminals. 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. We also have two techniques for adapting changes while planning: repairing will consist of generating a new path plan to the same selected location, and 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 ([61]). 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 ([62]) 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.

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. Figure 5 shows an illustrative example of extracting a directed connectivity graph from a current map. The agent then finds a Steiner node set N connecting all terminals using Steiner-MST ([63]) 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 [60] and then select the Steiner node with the shortest path (the nearest Steiner node) to our current location.

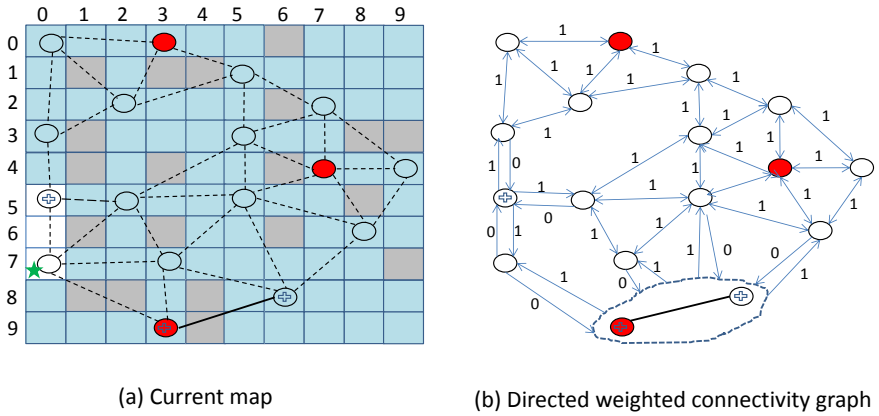


Fig. 5: Example of a directed weighted connectivity graph extracted from a current map.

Algorithm 3: Global Node (G-N) Algorithm

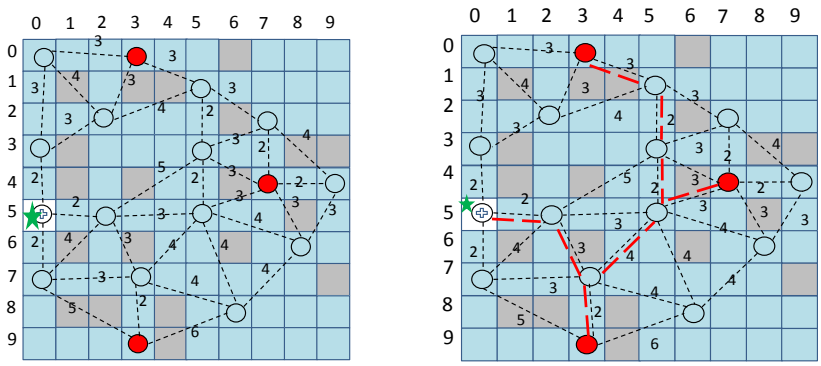
1 find D, the directed weighted connectivity graph
2 find N, the Steiner nodes in D
3 find P, the cheapest path to the nearest node in N
4 return (N, P)

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 repairing, in which for case (iv) we use D* Lite to continue searching for the current target, until we discover β 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 repairing 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. In summary, we have the G-N algorithm (in Algorithm 3) with two different planning techniques: full replanning (G-N-c) and repairing (G-N-f).

Strategy 2: prioritising mobility/path 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 compute these paths many times as we discover blocked locations. In the weighted connectivity graph, we then search for a low-cost Steiner tree. We use the Steiner-MST heuristic ([63]) 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. Figure 6 is an illustrative example of building a weighted connectivity graph and finding a Steiner tree which spans all the terminals.



(a) A weighted connectivity graph in a grid

(b) A Steiner tree (red dotted lines) in the weighted connectivity graph

Fig. 6: Example of a weighted connectivity graph and a Steiner tree spanning all terminals.

Algorithm 4: Global Path (G-P) Algorithm

```
1 find W, the weighted connectivity graph, using D* Lite
2 find N, the Steiner nodes in W, using Steiner-MST
3 find P, the cheapest path to the nearest node in N
4 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 repairing 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 repairing uses D* Lite to continue searching for the current target, until the expected total path length

exceeds the original path length by γ steps, which triggers full replanning. In summary, we also have the Global Path algorithm presented in 4 with two different planning techniques: full replanning (G-P-c) and repairing (G-P-f).

Similarly, a tree representation for the global focus is in Figure 7.

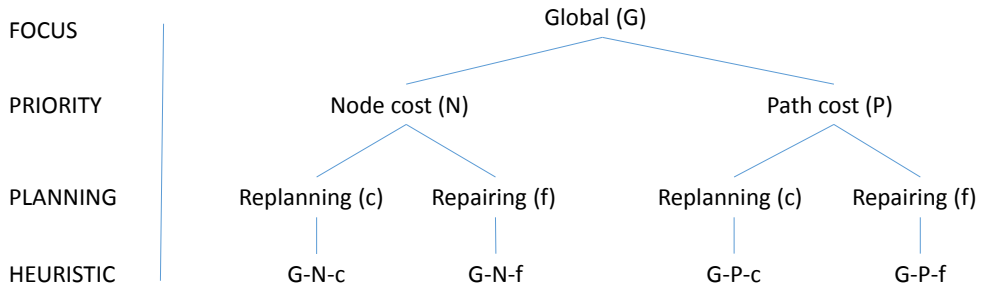


Fig. 7: A tree reprensentation for the global focus.

5.3. The adapted DORMS approach

As a comparison, we adapt the DORMS approach [20] 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. We select DORMS because the algorithm is one of the best classic approaches to solve the problem, and it is the closest one to our approach with some similarities as stated in Section 2. We knew in advance that it would not be fair to compare with DORMS, this centrialized algorithm with free land for movement and full knowledge of the radio and physical environments would behave very badly in our setting (partial known environments with obstacle awareness). We believe that the phenomenon will be similar with other existing work because of the different objectives and assumptions. Therefore, we have adapted the original DORMS and used this version as a baseline.

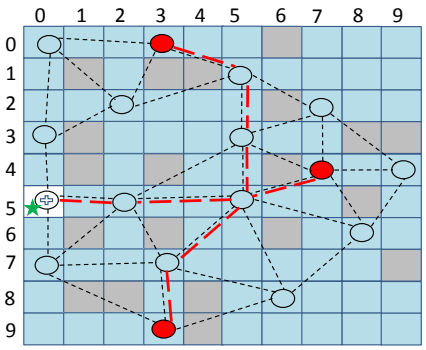


Fig. 8: Example of adapted DORMS. A node plan (red lines) connecting all terminals to a central point (5,5).

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. Figure 8 shows the adapted DORMS reconnects all terminals to a central point (5,5) of the grid. 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

Algorithm 5: Adapted DORMS Algorithm

```
Result: N: node plan; P: path plan.
while there are unconnected terminals do
    centre = Find_Centre_Location()
    for each t ∈ τ do
        p_t = Shortest_Connectivity_Path(t, centre, G_C)
    for each t ∈ τ do
        ta = Find_adjacent_terminal(t)
        G' = bounded_graph(p_t, p_t a, centre, G_C)
        T_t = Steiner_Minimal_Tree(t, ta, centre)
    while any t ∈ τ 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
```

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.

6. Experiments

Our focus is on evaluating repair algorithms rather than network protocols, so we use a custom Java simulation. We evaluate our algorithms empirically on randomly generated maps, to compare the quality of solutions for the two different measures, and to compare their runtime. 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[2], we allow a potential radio link with probability 0.85 (to simulate the RF propagation effects). For the map after 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. In 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 repairing, the cost threshold is $\beta = 4$, and the live node threshold is $\gamma = 3$ for all experiments. Note that the repairing only triggers replanning when the agent finds at least β surviving nodes in the local node heuristics and the expected total path length exceeds the original path length by γ steps in the local path heuristics. If the values are too high, then the agent will rarely change the current plans. This may cause extremely high costs in repairing. However, if the values are too small, it is sensitive to the changes in the network and environments, and thus updates the plans frequently. There is a tradeoff between the time to repair and the node/path costs. We have tried different values for these thresholds and the selected values of β and γ are competitive compared to the global heuristics. In all experiments, we set $a = 15$ live nodes and the results are the average of 50 runs at each data point. We have tried different values for the parameters above and different grid sizes, etc. in the experiment, and we received the same relative performance for our algorithms and heuristics.

A summary of the experiments is in Table 4. We study the effects of different numbers of candidate locations, different numbers of terminals and different levels of damage. We compare our heuristic algorithms

[2]Chosen to lie well within the maximum range of the popular TmoteSky sensor node [64]

to the adapted DORMS algorithm in Algorithm 5. We present the experimental results in two sections: local focus and global focus.

Vary	Candidates 50,100,150	Terminals 5,10,15	Damage levels $< 10\%, 10\% >$ $< 20\%, 20\% >$ $< 30\%, 30\% >$	Grid Granularity 45x45, 150x150, 200x200, 300x300
Existing live nodes	15	15	15	15
Terminals	5	-	5	5
Candidates	-	100	100	100
Block	90	90	90	90
Links	9 squares	9 squares	9 squares	9 squares
More blocks	10%	10%	-	20%
Removed links	10%	10%	-	20%
Grid size	45x45	45x45	45x45	-
Number of live nodes - α	15	15	15	15
Cost threshold- β	4	4	4	4
Live node threshold - γ	3	3	3	3

Table 4: Experiment Setup

6.1. Local focus

We have proposed many different heuristics and performed many experiments in different settings, varying number of candidates, terminals, damage levels, grid size and different moving speeds of the agent. Plotting all of the heuristics in all experiments will be hard to read. In addition, in some experiments, the performance of the heuristics follows the same patterns. Therefore, to make the results more compact, in this section, we first show the relative performance of the proposed heuristics in the first set of the experiments with $c = 100$, $t = 5$, the damage level set to ($b = 10\%$, $r = 10\%$) and the grid is at 45×45. This set of experiments presents the relative performance of the proposed heuristics based on their priorities, planning techniques and strategies. We then focus on different sets of experiments (varying c, t, the damage level and the grid size). However, in these sets, we only show the top heuristics among the proposed heuristics and plot any of the others if they have different patterns or their relative performance is significantly changed compared to the first set.

We have run the experiments with all the proposed heuristics and we have found that (i) for the local path-based heuristics (L-P-c/f-MD/SD/SCP), L-P-c/f-MD/SD produce similar results in all cases, and they are all worse than the L-P-c/f-SCP in both node cost and path cost. These algorithms are very fast to run. However, the short computation times for L-P-c/f-MD/SD are not good enough to compensate for the high node cost and path cost that they produce. These results lead to longer total restoring times (the total time to restore the network) compare to L-P-c/f-SCP. The main reason for L-P-c/f-SCP outperform these algorithms is that L-P-c/f-SCP require longer runtimes to perform complicated computation for the costs based on current knowledge while the others (L-P-c/f-MD/SD) use heuristics to estimate the costs. Due to these results, in the following figures, we only pick L-P-c/f-SCP to represent for the local path-based heuristics to compare with the others; (ii) for the local node-based heuristics (L-N-c/f-MD/FN), L-N-c/f-FN outperform L-N-c/f-MD in node costs but they are worse than those in path costs in all cases. With similar required runtimes, changing the target terminal produces better results in node cost but then behaves badly in the mobility cost compared to fixing the target terminal, and thus causes longer total restoring times in all cases. Due to these variant results, we keep all the heuristics in these local node-based approaches in the following figures when comparing with other heuristics in different categories.

Relative performance of the locally focused heuristics

The relative performance of the locally focused heuristics is shown in Figure 9. Firstly, we compare between changing and fixing the target terminal. Changing the target terminal produces better results than fixing the target terminal in general, e.g L-N-c-FN is better than L-N-f-FN, and L-N-c-MD is slightly better

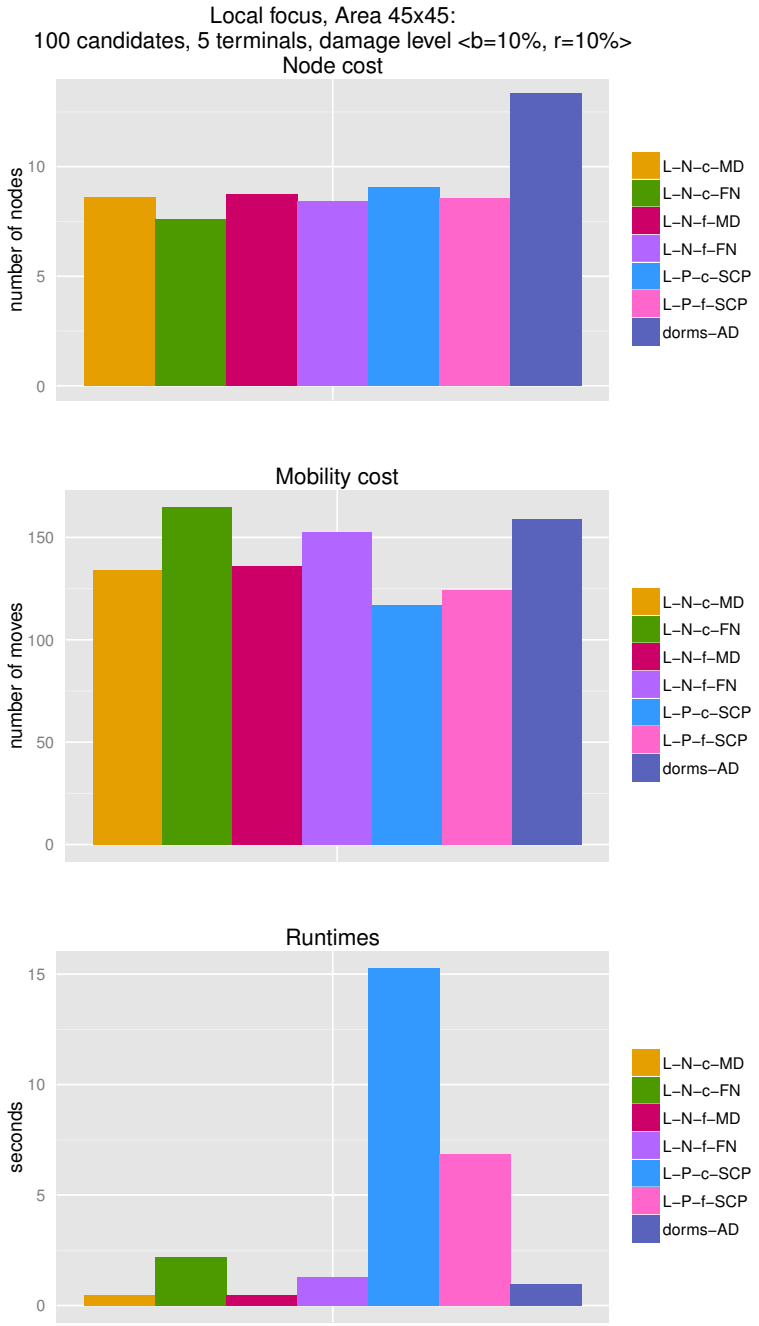


Fig. 9: Local focus, area 45x45: 100 candidates, 5 terminals, damage level <b=10%, r=10%>

than L-N-f-MD in node cost, and L-P-c-SCP requires less mobility cost than L-P-f-SCP. However, changing the target terminal requires longer runtimes than fixing the target terminal, e.g. the runtimes of L-P-c-SCP and L-N-c-FN are approximately double the runtimes of L-P-f-SCP and L-N-f-FN respectively. This is due to the fact that the agent always uses new updated knowledge to calculate the next target terminal, and the unknown environment might require the plan to be changed frequently.

Secondly, we examine the results from different priorities (node or path cost). For node cost, L-N-c/f-FN are better than L-N-c/f-MD as these heuristic prefer cheapest node plans and always perform complicated computation at each step using the current updated knowledge to calculate the node cost while L-N-c/f-MD use heuristic estimate the node cost. L-P-c/f-SCP althrough prioritise the mobility cost but still produce good results in the node cost because they also consider a good connectivity path (radio path) from the terminal to the network. L-N-c-FN is the top heuristic. The adapted DORMS is the worst. For path cost, as the node based heuristics do not prioritise the mobility cost, the results are quite high. L-P-c-SCP is the top heuristic because it prioritises the path cost and always uses new knowledge to calculate the cost with heavy computation which considers the whole connectivity and mobility graphs at the time. L-P-f-SCP is the second heuristic, slightly worse than L-P-c-SCP because it also performs the same computation like L-P-c-SCP but fixing the target terminal which sometimes ignores the new discoveries in the environment. For runtime, L-P-c-SCP is the longest due to the same reason above. However, even L-P-c-SCP requires twice the runtimes of L-P-f-SCP (or 7.5 more seconds) but it produces less number of moves in mobility cost (roughly 4 moves which is approximately 24.4m) and this reduced mobility cost can compensate for the extra runtime above. The heuristics (L-N-c/f-MD) are the fastest as they use heuristics to estimate the cost.

Overall, L-N-c-FN is the top algorithm in node cost and L-P-c-SCP is the top algorithm in mobility cost, and L-P-f-SCP is the second top algorithm in both node cost and mobility cost. They all require longer runtime than the other algorithms because they perform complicated computation (Shortest Cheapest Path, Fewest Node) for the costs based on current knowledge while the others use simple heuristics (Manhattan Distance, Shortest Distance) to estimate the costs. The DORMS-AD approach is consistently poorer on both measures, with up to 50% higher costs compared to the best heuristic in each case. It is competitive on runtime.

As this set of experiments does not show the effects of environmental settings. In the next step, we will show the behaviors of the heuristics in different scenario setup, varying nubmer of candidates, terminals, different dammage level, grid size and different agent's moving speed. We note in the above experiment that L-N-f-FN does not show any benefit in improving the costs, therefore, we will skip this heuristic in the following figures.

Varying the number of candidates

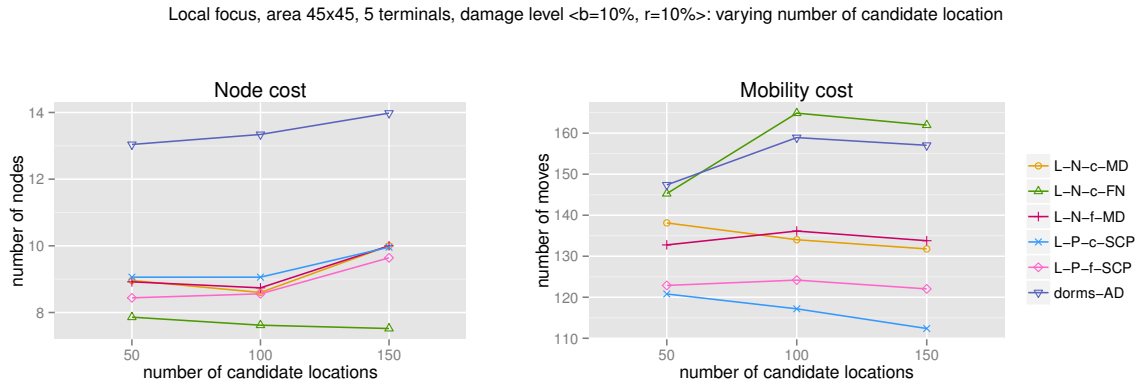


Fig. 10: Local focus, area 45x45, 5 terminals, damage level <b=10%, r=10%>: varying number of candidate locations

We now 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 ($b = 10\%, r = 10\%$). As the connectivity is defined by distance, so packing more nodes into a space increases the density. Therefore, increasing the number of candidates from 50 to 150 will create problems with increasing connectivity graph density. The results are shown in Figure 10 and Table 5.

Comparing between changing and fixing the target terminal, for node cost, L-P-f is slightly better than

L-P-c, and L-N-f is slightly worse than L-N-c. As this is the node cost, and L-N prioritises the node cost, therefore, L-N is better than L-P in node cost in average. In addition, and L-N-c/f-FN are better than L-N-c/f-MD. This is due to the fact that L-N-c/f-FN spend more time to find better solutions by exploiting the current updated knowledge while L-N-c/f-MD use heuristic to estimate the node cost. We expect the cost will reduce when the density is increased. However, this only happens for the heuristic L-N-c-FN. It is because that we do not increase the number of existing live nodes (15 live nodes in all experiments). When we increase the density, we also reduce the percentage of live nodes compared with the total candidate locations. Those live nodes might not aid in reducing the additional nodes as increasing the number of candidates might put the existing nodes further away from the terminals' locations.

	50	100	150
L-N-c-MD	0.07	0.49	5.85
L-N-c-FN	0.24	2.18	34.75
L-N-f-MD	0.07	0.48	5.80
L-P-c-SCP	1.99	15.28	92.90
L-P-f-SCP	1.57	6.86	31.14
dorms-AD	0.20	0.99	10.92

Table 5: Local focus, area 45x45, 5 terminals, damage level <b=10%, r=10%>: Runtime (sec) vs number of candidate locations.

For the mobility cost, firstly we examine the results of path priority heuristics. Changing the target terminals produces better results compared to fixing it. This is due to the fact that the agent always uses new updated knowledge to calculate the next target terminal. L-P-c-SCP is always the top heuristic. Secondly, we examine the results of node priority heuristics. As they do not prioritise the mobility cost, the results are quite high. L-N-c-FN is far worse than L-N-f-FN and they are both worse than L-N-c/f-MD. L-N-c-MD is worse than L-N-f-MD at the start (50 candidate locations), but then becomes slightly better than L-N-f-MD when the density is increased. Thirdly, the path priority heuristics perform better than the node priority heuristics in all cases. Finally, as the graphs become denser, and there will be more choice in where to place the nodes, and so the path costs can be reduced mostly for path priority heuristics. For the node priority heuristics, as they try to reduce the node cost, they might want to combine with the existing nodes to lower the node cost but they could lose the chance to put the required nodes close together (to reduce the path cost). Therefore, the path cost might increase for them.

The runtime for all algorithms increases with the increasing number of candidates. Changing target requires longer computation time than fixing the target in most cases. As the graphs become denser, there is little impact on node costs, but the path costs drop. The DORMS-AD approach is competitive on runtime.

Overall, L-N-c-FN is the top algorithm in node cost, L-P-c-SCP is the top algorithm in mobility cost, and L-P-f-SCP is the second top algorithm in both node cost and mobility cost, L-N-f-FN performs well in node cost but worse in mobility cost. They all require longer runtime than the other algorithms with the same reason like before.

Varying the number of terminals

Next, 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 ($b = 10\%, r = 10\%$). In this setting, as expected, all the costs increase, more or less scale linearly, with the increasing number of terminals to be connected. L-N-c-FN is the top heuristic in node cost and L-P-c-SCP is the top heuristic in path cost. The results is shown in Figure .21 in the Appendix.

Varying the damage level

We now vary the damage level (b, r) 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 11 where the top heuristics in node cost (L-N-c-FN) and in path cost (L-P-c-SCP) are presented. We also compare these top heuristics with their counterparts for different planning techniques (L-N-f-FN and L-P-f-SCP respectively). We expect the node cost rises as it requires more nodes in most cases to compensate for heavier damage, and the fixing plan requires more costs when the damage level is increased. However, from damage level <20%, 20%> to <30%, 30%>, the node costs in both heuristics are similar. The reason is because we do not count the infeasible solutions, and

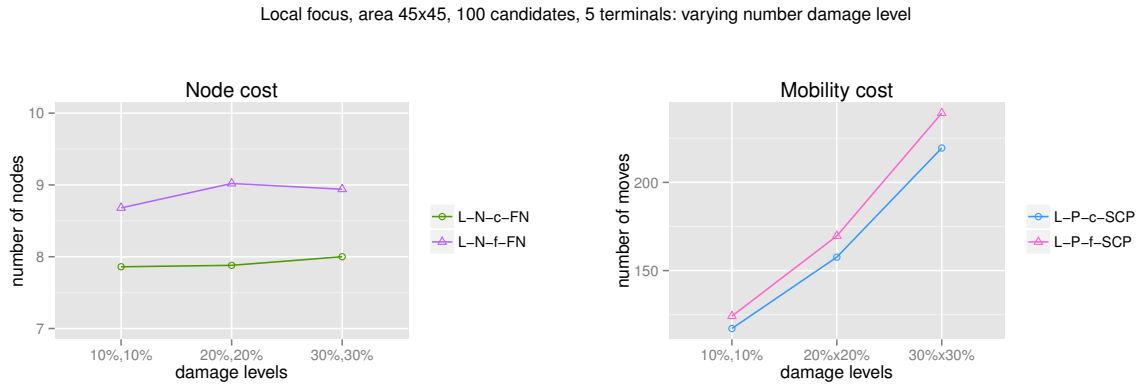


Fig. 11: Local focus, area 45x45, 100 candidates, 5 terminals: varying damage levels.

when the damage level is high, there are less chance to create feasible solutions and these solutions most likely to have the similar numbers of required nodes between terminals. 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 causes the cost to increase highly with heavy damage. The figure shows the effect of heavier damage is more severe in mobility cost compared to the node cost, e.g. approximately, the mobility cost increases 70% and the node cost increases 20% from <20%, 20%> to <30%, 30%> compared to those with damage level from <10%, 10%> to <20%, 20%>. The reason for this could be the number of candidates is still good enough for connectivity among all terminals. Note that we only consider the solvable solutions, therefore, when the damage level increases, the agent is still able to find alternative nearby candidates to connect the terminals. However, the agent has to change the routes frequently due to the obstacles blocking the route. As expected, when the damage level increases, fixing the plan requires more mobility cost compared to updating the plan.

Varying the grid granularity

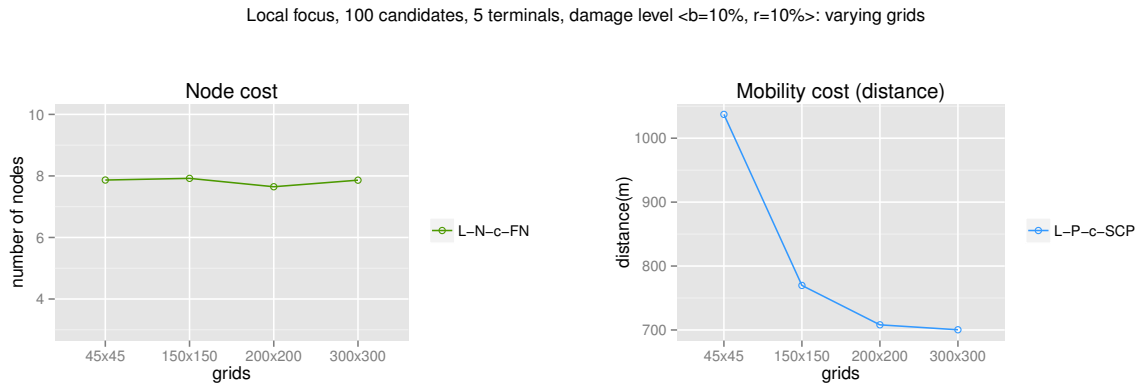


Fig. 12: Local focus, 100 candidates, 5 terminals, damage level <b=10%, r=10%>: varying the granularity of the mobility grid.

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 150x150, 200x200 and 300x300 with the square size of 2m, 1.5m and 1m respectively. The results are shown in Figure 12 and Table 6, where we now plot mobility cost as the path distance and the top heuristics are presented like before.

	45x45	150x150	200x200	300x300
L-N-c-FN	2.56	45.13	122.39	598.19
L-P-c-SCP	29.11	37.98	72.96	314.25

Table 6: Local focus, 100 candidates, 5 terminals, damage level <b=10%, r=10%>: Runtime (sec) vs grid granularity levels.

As expected, the number of required nodes shows only minor variation because the number of candidate locations and terminals are all the same. However, the mobility costs by number of moves (from square to square) obviously increases with the finer grids. We are interested in 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. The travel distance seems to reduce approximately 35% at the end. This is because we are able to find paths between obstacles which would have been blocked with the larger squares. The runtimes, howevever, increase significantly as the finer grids mean more options to explore.

Assessing the different movement speeds of the agent

	V=0.1ms^{-1}	V=1.4ms^{-1}	V=4ms^{-1}
L-P-c-SCP	**8099.08**	**845.33**	482.63
L-N-c-MD	9476.87	907.47	492.65
L-N-c-FN	11222.78	1023.06	512.72
L-P-f-SCP	8542.33	**851.15**	**466.79**
L-N-f-MD	9338.68	916.37	494.98
L-N-f-FN	10433.28	988.84	516.19
dorms-AD	10995.86	1148.85	656.96

Table 7: Local focus, area 45×45: Total Restoring Time (sec) with different agent's speed, 100 candidates, 5 terminals, and <10%, 10%> for damage level.

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 consist of 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 three scenarios, one representing a small robot which moves at 0.1ms^{-1}, the second representing a human walking at average speed 1.4ms^{-1}, and the third 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 $6.66m \times 6.66m$.

First, we look at the relative performance of all the locally focused heuristics for the set of experiments with 100 candidates, 5 terminals, <10%, 10%> damage level, and the area is set to 45×45. The result is shown in Table 7. For the slow moving agent, the path priority approaches are faster, since the extra runtimes are recovered by shorter paths for the agent. L-P-c-SCP algorithm is the fastest, despite requiring the longest runtime. DORMS is consistently slower. At the human average walking speed, L-P-c-SCP is still the top heuristic. L-P-f-SCP can be competitive with few seconds slower than L-P-c-SCP. This is due to the fact that L-P-f-SCP, although poorer in quality than L-P-c-SCP, it still offers good solutions in both node cost and path cost, and the runtime is significantly faster than L-P-c-SCP. As the movement speed increases to 1.4ms^{-1}, the higher path cost drops in importance compared to the runtime. This also reduces the gap of total time to restore between the node priority approaches and the path priority approaches. For the fast agent, the L-P-c-SCP is unable to compensate for the longer runtime and extra nodes in some cases. Node priority heuristics become the better approaches for scalability. As the movement speed of the agent increases, the runtime and the time taken to place the nodes becomes more significant, and the node priority approaches benefit from their lower node costs and faster runtime. However, as we mentioned before, L-

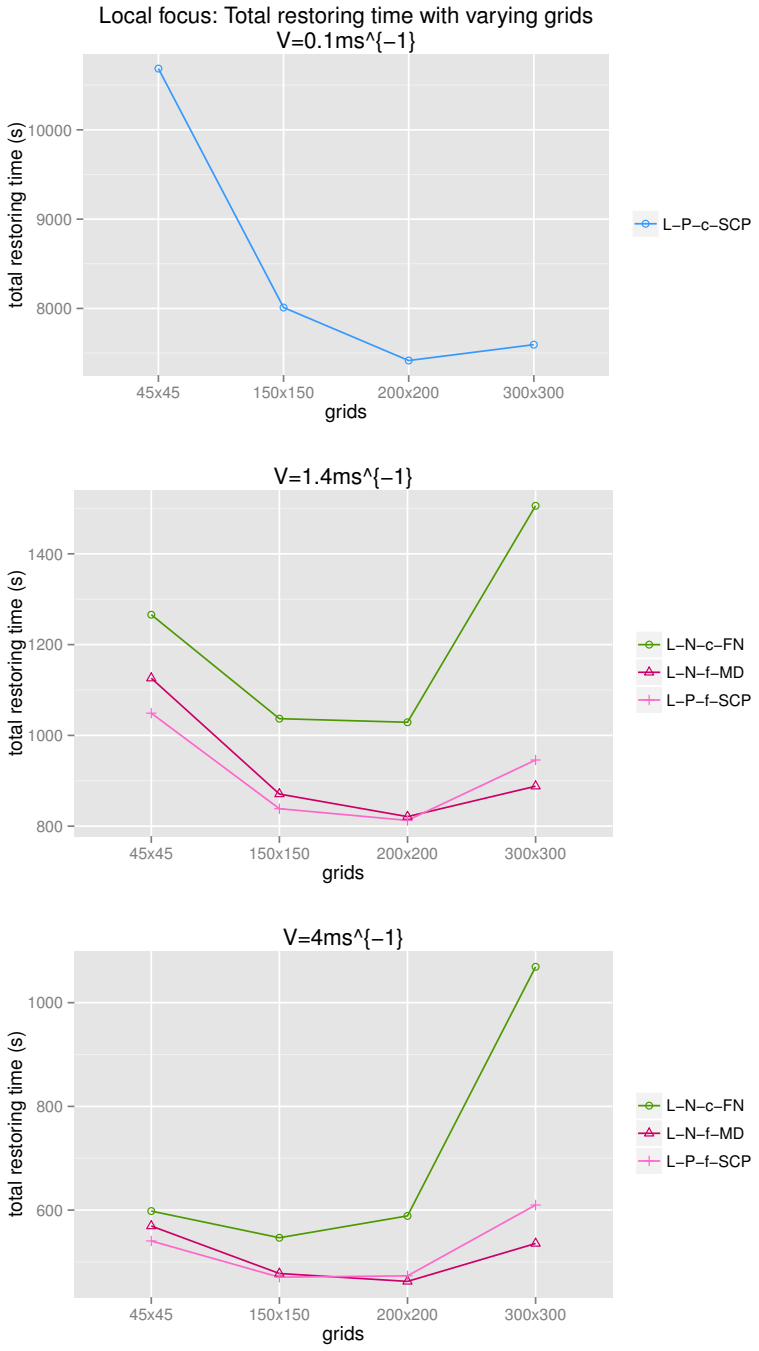


Fig. 13: Local focus: Total restoring time with different mobility grids.

P-f-SCP is the second top in node and mobility costs in most cases, therefore requires as less total time to repair as the node priority approach.

We now show the total restoring time of the top heuristics for the granularity of the mobility grids (Figure 13). With the slow moving agent, the total restoring time consistently reduces with the finer grids as the mobility cost is reduced, and the L-P-c-SCP always remains the fastest. At the end, the total restoring time

increases because the finer grid requires longer computation time and the reduced mobility cost can not compensate for the longer runtime. The rest of the heuristics follow this pattern and is skipped in the figure for better visualisation. For the human walking speed agent, the restoration time starts to drop at first, but then rises significantly at 300x300 with the same reason above. With 300×300 grid, L-P-f-SCP starts to lose to L-N-f-MD because of its heavy computation time. For the fast moving the agent, the restoration time starts to drop at first, but then starts to rise. 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 is reduced. L-N-f-MD becomes the top with 300x300 grid due to its fast runtime. We also spot the misbehave of L-N-c-FN in Figure 13 for V=1.4ms^{-1} and V=4ms^{-1}. This is because the heuristic requires long runtime for heavy computation but does not produce good results in path cost which can not compensate for the best results in node cost in total restoring time.

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 L-N-c-FN heuristic always offers the lowest node costs and this algorithm would be a good choice for applications where node cost dominates. For applications that prefer the path cost, L-P-c-SCP would be the suitable choice. In other situations, the time to restore the network may be a greater consideration, in which case L-P-f-SCP is most suitable. There is an exception of L-N-f-MD which could be considered for a scenario where we have a fast moving agent with a finer grid.

6.2. Global focus

Similarly to the locally focused search, we evaluate heuristics in globally focused search in simulation. In the following parts, we will show the relative performance of all globally focused heuristics for a set of experiments with 100 candidate locations, 5 terminals, damage level set to <b=10%, r=10%> in 45×45 grid. We then show the results of the top heuristics (G-N-c for node cost and G-P-c for path cost) in varying the number of candidate locations, the number of terminals, the damage levels, and varying different grid granularity in more details.

Relative performance of the globally focused heuristics

Compared to the locally focused heuristics, for node cost, L-N-c-FN and G-N-c are quite competitive. However, the other global approaches seem to lose to the local approaches due to they are optimising too much and counterpart the benefit of global knowledge. Each time the agent discovers new knowledge or significantly changes in the knowledge, it will abandon all the plans and recalculate new plans. The affects will be larger in global plans when the agent is already in the middle of the plans repairing the network. For path cost, G-P-c is competitive with L-P-c/f-SCP and the other global approaches are better than the other local approaches. The global heuristics require longer runtimes than the local heuristics in general.

The relative performance of the global heuristics is shown in Figure 14. The repair methods incur approximately 13% extra node cost, for both path priority and node priority heuristics. The path priority approach is surprisingly 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 repairing. For mobility costs, the repairing method is worse than the full replanning method for path priority. Both methods for path priority are better than the methods for node priority. The DORMS-AD approach is consistently poorer on both measures, with up to 50% higher costs compared to the best heuristic in each case.

For runtimes, the repairing methods in both approaches are faster than the full replanning methods. The path approaches are slower than the node approaches respectively due to the computation of weighted connectivity graphs. In the node priority approach, the full replanning takes closer to 100% more time to complete, compared to the repairing, while in the path priority approach, the full replanning takes approximately 400% more time to complete, compared to the repairing. Full replanning methods require longer runtimes than the repairing methods due to the fact that they recompute the plans every time the agent discovers new knowledge that might change the costs while the repairing methods only do the replanning when the agent finds significant changes in the costs. DORMS is competitive on runtime, becoming the second fastest algorithm for the most dense problems.

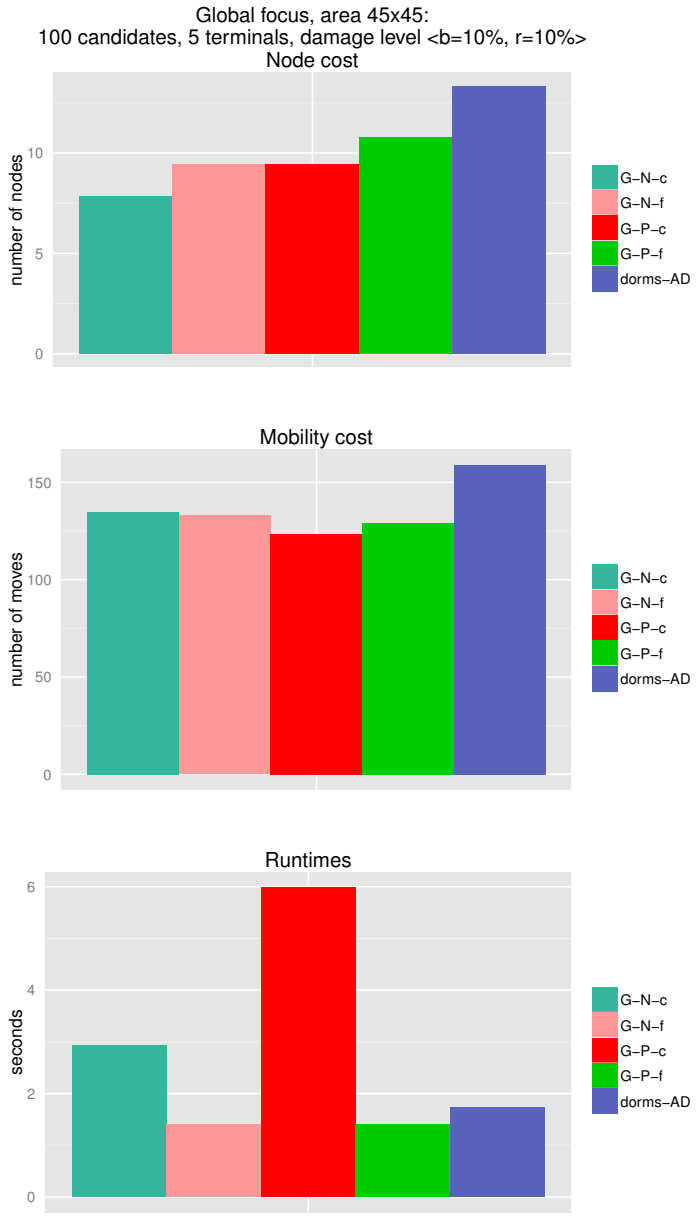


Fig. 14: Global focus, Area 45x45: 100 candidates, 5 terminals, damage level <b=10%, r=10%>

Varying the number of candidates

We now 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 15.

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

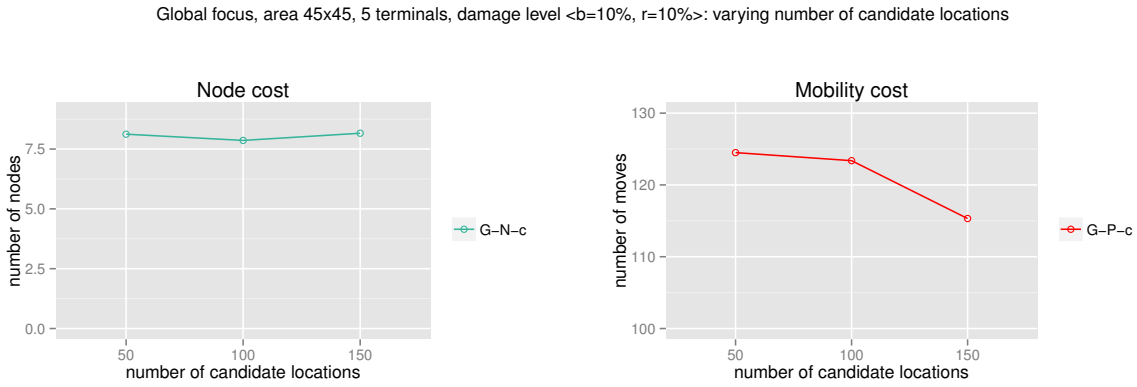


Fig. 15: Global focus, area 45 × 45, 5 terminals, damage level <b=10%, r=10%>: varying number of candidate locations.

in where to place them, and so the path costs can be reduced. Further, 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.

Varying the number of terminals

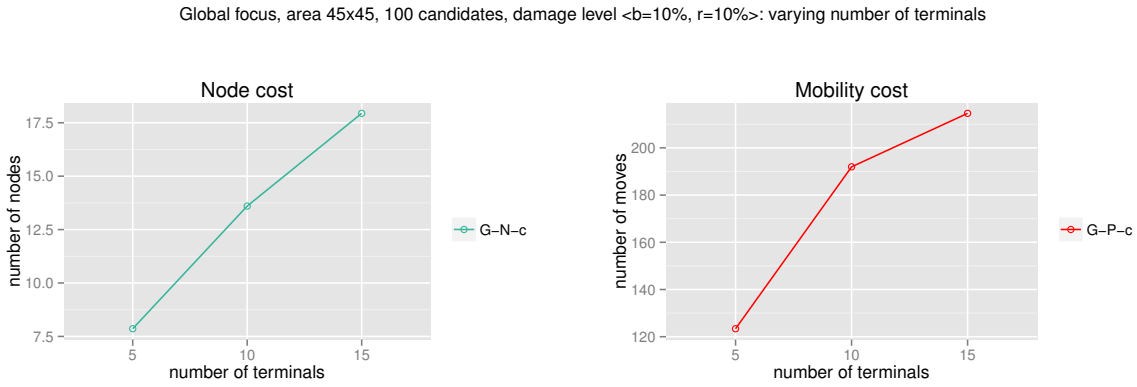


Fig. 16: Global focus, area 45 × 45, 100 candidates, damage level <b=10%, r=10%>: varying number of terminals.

Next, 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 16. We only show the top heuristics in each case and the rest of the heuristics follow the patterns as in the relative performance results.

As expected, all the costs increase with the increasing number of terminals to be connected. The G-N-c is the top heuristic in node cost, and the G-P-c is the top in path cost. The number of required nodes and the mobility cost are both reduced from 10 terminals to 15 terminals compared to those costs from 5 terminals to 10 terminals. This is because the agent starts to reuse the newly added nodes when we increase the number of terminals with the same area.

Varying the damage level

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 17.

All costs rise as it requires more nodes to compensate for heavier damage. The G-N-c and G-P-c remain the tops in node and path costs respectively. The mobility cost is rising significantly as the damage increases.

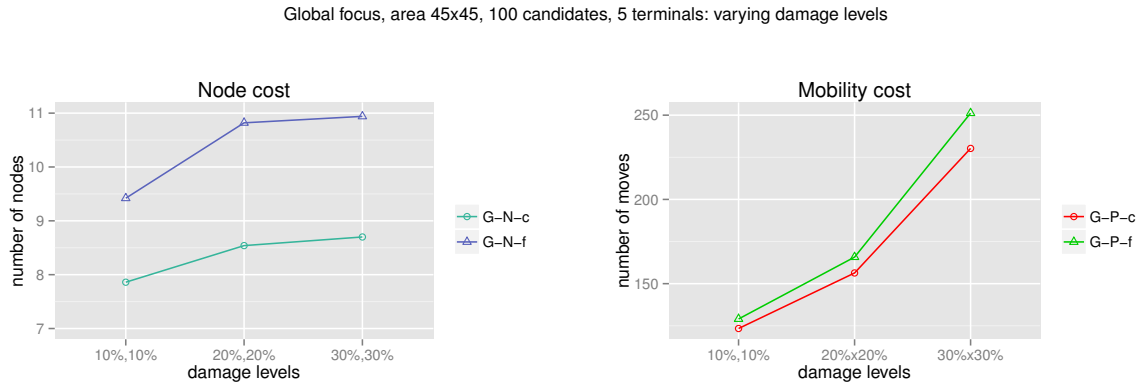


Fig. 17: Global focus, area 45×45, 100 candidates, 5 terminals: varying damage levels.

This is due to the fact that the number of candidate locations decreases, and there are more obstacles blocking the route which causes the lack of knowledge of the true mobility problem. It appears that the global focus is optimising too much which counterparts the benefit of global knowledge in heavier damage, and therefore, the node cost of G-N-c is a bit higher than L-N-c-FN (approximately 0.6) compared to the local heuristics.

Varying the grid granularity

As in local focus, we vary the granularity of the grid. The patterns of the results for global focus are similar to the local focus where the number of required nodes show only minor variation because we have the same area, number of candidates and terminals and damage level. 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 runtime increase for the algorithms as the finer grids mean more options to explore. More detail of the graphs is in Figure .22 in the Appendix.

Assessing the different movement speeds of the agent

Finally, we consider the time to restore the network with three classes of the agent (slow speed agent, medium speed agent and fast moving agent) as above. For the 45×45 grid in the $300m \times 300m$ area, the individual squares are of size $6.66m \times 6.66m$.

	V=0.1ms^{-1}	V=1.4ms^{-1}	V=4ms^{-1}
G-N-c	9222.874	**879.351**	**462.174**
G-N-f	9146.163	916.545	505.063
G-P-c	**8511.878**	**874.069**	492.178
G-P-f	8926.930	938.740	539.330
dorms-AD	10995.86	1157.953	666.057

Table 8: Global focus, area 45×45: Total restoring times (sec) with different agent's speed, 100 candidates, 5 terminals, <b=10%, r=10%> for damage level.

We first look at the relative performance of all the globally focused heuristics for the set of experiments with 100 candidates, 5 terminals, damage level set to <b=10%, r=10%>, and the area is set to 45×45. The results are shown in Table 8. For the slow moving agent, 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 medium speed agent, improving the mobility cost becomes less important, the gap of the total restoration time between the G-N-c and G-P-c becomes small, and the G-N-f is faster than the G-P-f on average. For the fast agent, 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 heuristic. 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.

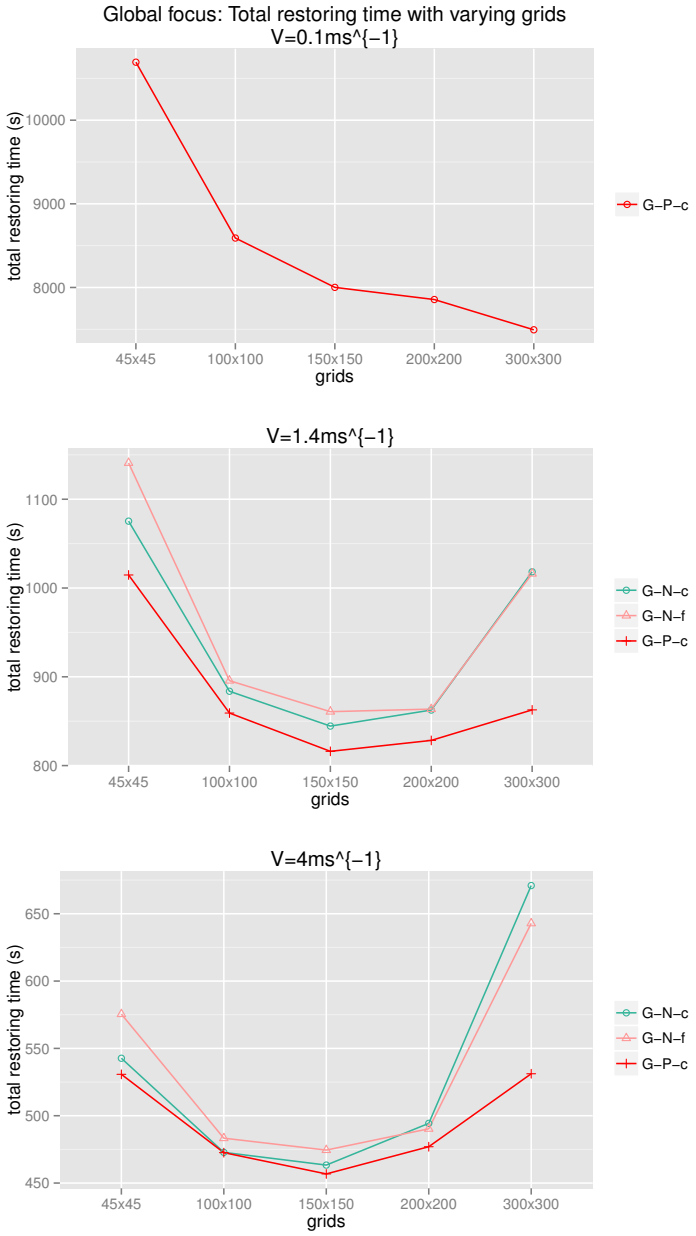


Fig. 18: Global focus, 100 candidates, 5 terminals, damage level <b=10%, r=10%>: Total restoring time with varying mobility grids.

Figure 18 shows the results for the granularity of the mobility grids for the top heuristics - G-P-c. The others follow the same patterns, except the G-N-c and G-N-f behave badly with 300 × 300 grid due to their heavier computation for the directed weighted connectivity graph, and thus the runtimes increase significantly with this finer grid. With the slow moving agent, the total restoring time for all methods almost consistently reduces with the finer grids (they follow the same pattern as the G-P-c in the figure), and the G-P-c always remains the fastest. For the medium and 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.

We now look at the different performance between the global and local heuristics. The global heuristics lose to L-P-c-SCP and L-P-f-SCP when the speed of the agent is at $0.1ms^{-1}$ and $1.4ms^{-1}$, respectively. This is due to the fact that the global heuristics require longer runtimes than the local ones in general, and also L-P-c/f-SCP is more flexible in adapting to changes in the environments compared to the global approaches where they have to abandon the global plans for new discovered knowledge. When the speed of the agent is at $4ms^{-1}$, G-N-c is the most competitive because it provides good results in node cost and path cost and seems reasonable in runtime.

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 G-N-c 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 G-P-c is most suitable. This heuristic should also be prefered in applications that prefer the path cost.

7. Summary

We have defined the new problem of simultaneous network repair and autonomous exploration and route planning in the presence of unknown obstacles, and the repairing agent must discover the damage as it makes the repair. We consider local and global focuses, for each we develop many heuristics based on the two priorities: prioritising the node cost and prioritising the mobility cost. We evaluate the approaches in different focuses in simulation, assessing the impact of increasing damage, increasing nodes to be connected, and increasing locations for radio nodes. Overall, the costs all increase when we increase the terminals and the damage levels. On average, there are two top heuristics in node cost: L-N-c-FN and G-N-c, and three top heuristics in path cost: L-P-c-SCP, L-P-f-SCP and G-P-c. We note that although the local heuristics L-N-c-FN and L-P-c/f-SCP require longer runtimes compared to those global heuristics as their computations are heavier, they still give one of the best solutions in node cost and path cost respectively. In some cases, the full replanning methods in both node and path approaches lose to those local heuristics. This is due to the unknown environments which make the initial global decision poorer.

In addition, we show that different agent speeds have a significant impact on performance, and must be taken into account when selecting the algorithm. With slow speed agent, the time to move through the sensor field outweighs the time to place nodes and the computation time. There are three top heuristics to use in this case: the L-P-c-SCP, L-P-f-SCP, and the G-P-c due to their slow mobility costs. With medium speed, the gap between the path and the node approaches in total restoration time is smaller. However, with a fast moving agent, the speed means that the higher mobility costs are less significant, and thus the time to place node and the runtime become more important. Therefore, the node based approaches may be preferred in this situation.

There are many areas for potential future work. The paper uses the graph-based model where the candidate locations are decided and fixed into a number of positions. Future work will address the real world of potential areas for node placement where continuous positions can be investigated. We should also develop a hierarchical clustering approach, which will allow us to handle dense mobility graphs, by initially merging adjacent location nodes into a super-node to reduce the complexity. There is a need to develop distributed algorithms, allowing multiple agents and sensor nodes to collaborate to determine the damage to the network in large scale problems. The work in this paper should be extended to consider a continually changing network and environment, and to consider other constraints in the network such as delivery latency, network throughput, etc.. Future work can also implement the network repair scenario with a real robot.

Acknowledgment

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

Appendix

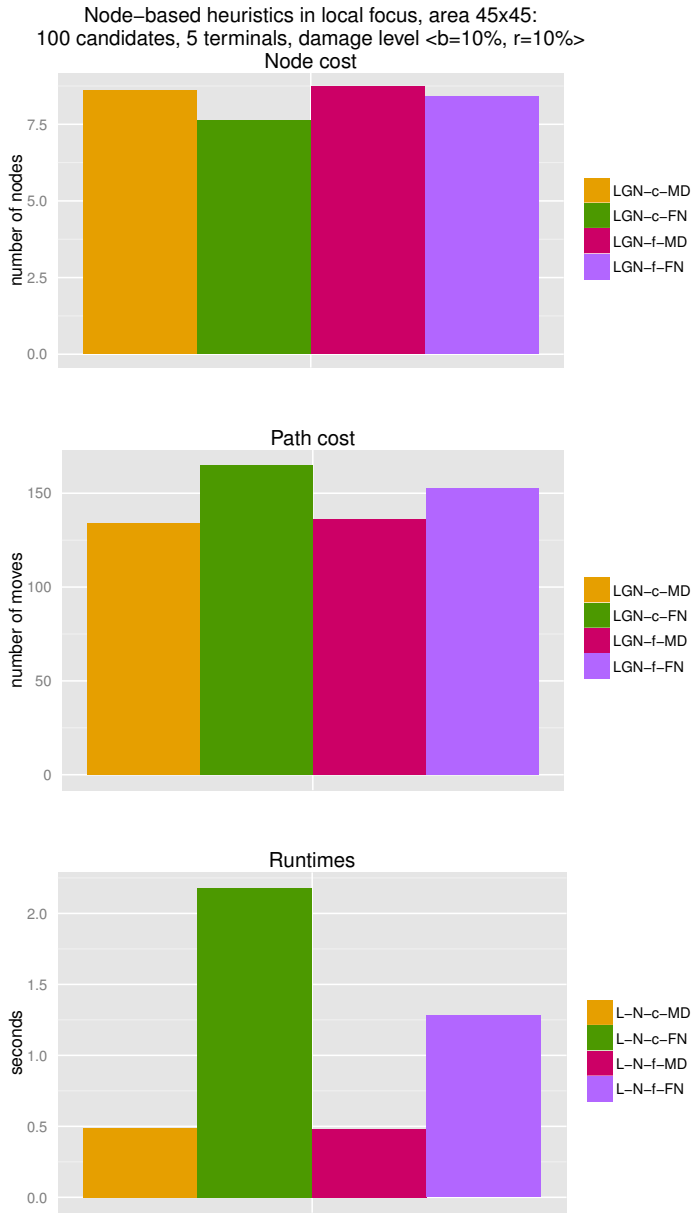


Fig. .19: Node-based heuristics in local focus, area 45x45: 100 candidates, 5 terminals, damage level <b=10%, r=10%>.

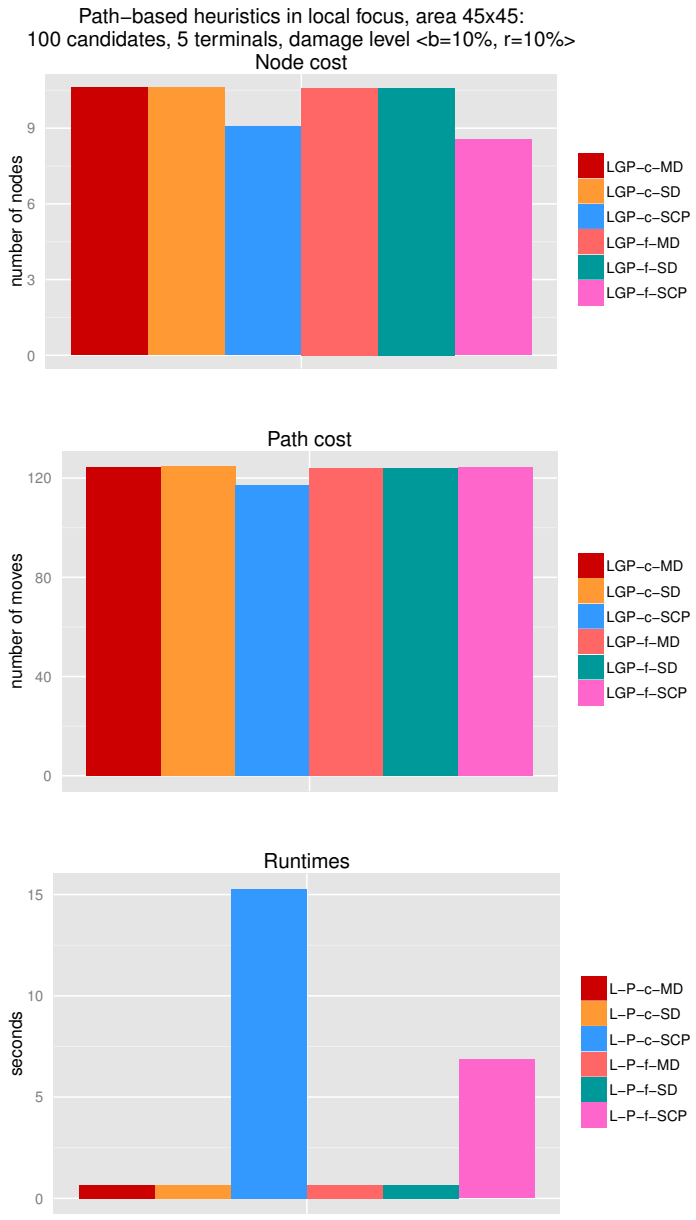


Fig. .20: Path-based heuristics in local focus, area 45x45: 100 candidates, 5 terminals, damage level <b=10%, r=10%>.

	45x45	150x150	200x200	300x300
G-N-c	1.872	15.505	46.725	247.647
G-P-c	5.157	8.327	21.775	97.639

Table .9: Global focus, 100 candidates, 5 terminals, damage level <b=10%, r=10%>: Runtime (sec) vs grid granularity levels.

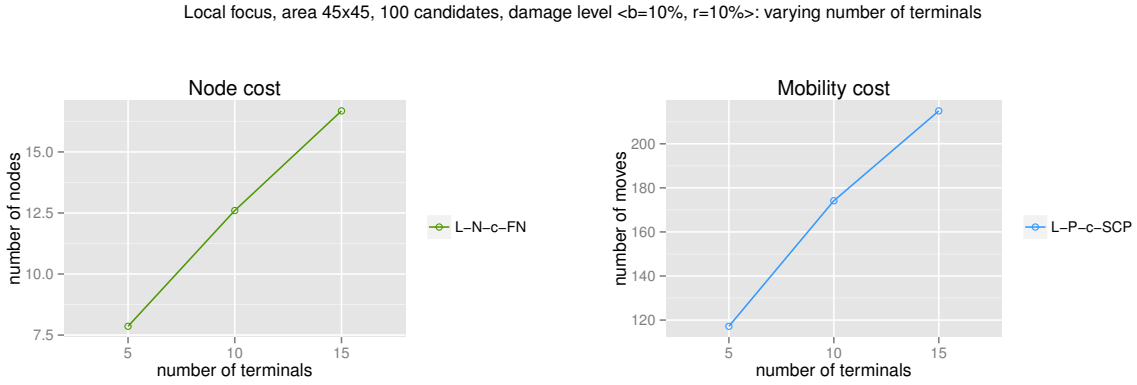


Fig. .21: Local focus, area 45x45, 100 candidates, damage level <b=10%, r=10%>: varying number of terminals.

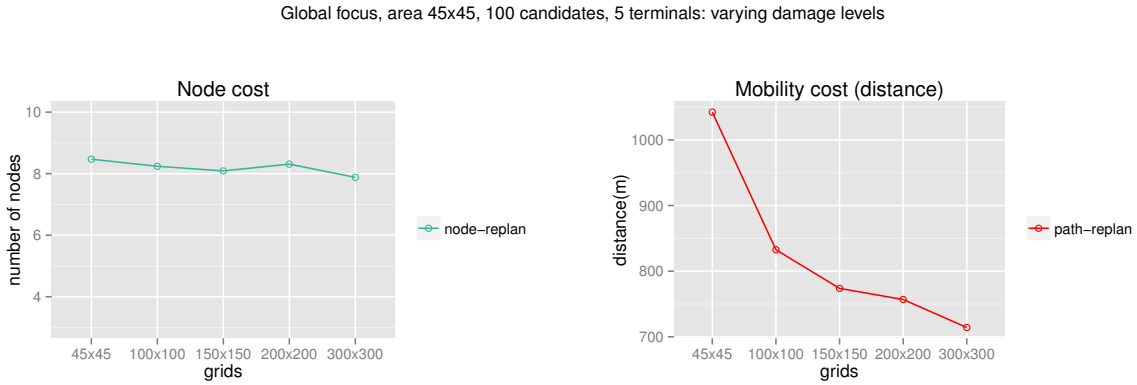
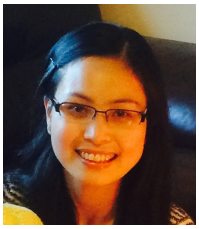


Fig. .22: Global focus, 100 candidates, 5 terminals, damage level <b=10%, r=10%>: varying the granularity of the mobility grid.

[1] B. Khelifa, H. Haffaf, M. Merabti, D. Llewellyn-Jones, Monitoring connectivity in wireless sensor networks., in: IEEE Symposium on Computers and Communications (ISCC), 2009, pp. 507–512.
[2] H. M. Almasaeid, A. E. Kamal, On the Minimum k-Connectivity Repair in Wireless Sensor Networks., in: IEEE International conference on Communications (ICC), 2009, pp. 1–5.
[3] N. Atay, B. Burchan, Mobile Wireless Sensor Network Connectivity Repair with K-Redundancy., in: 8th International Workshop on the Algorithmic Foundations of Robotics, Vol. 57, 2008, pp. 35–49.
[4] L. Sitanayah, K. N. Brown, C. J. Sreenan, A fault-tolerant relay placement algorithm for ensuring k vertex-disjoint paths in wireless sensor networks", Journal of Ad Hoc Networks 23 (2014) 145–162.
[5] L. Sitanayah, K. N. Brown, C. J. Sreenan, Planning the deployment of multiple sinks and relays in wireless sensor networks, Journal of Heuristics (2014) 1–36.
[6] A. A. Abbasi, K. Akkaya, M. Younis, A distributed connectivity restoration algorithm in wireless sensor and actor networks, in: 32nd IEEE conference on Local Computer Networks (LCN), 2007, pp. 496–503.
[7] F. Senel, K. Akkaya, M. F. Younis, An Efficient Mechanism for Establishing Connectivity in Wireless Sensor and Actor Networks., in: IEEE conference on Global Telecommunications (GLOBECOM), 2007, pp. 1129–1133.
[8] K. Akkaya, S. Janapala, Maximizing connected coverage via controlled actor relocation in wireless sensor and actor networks, Journal of Computer and Telecommunications Networking 52 (14) (2008) 2779–2796.
[9] K. Akkaya, F. Senel, Detecting and connecting disjoint sub-networks in wireless sensor and actor networks, Journal of Ad Hoc Networks 7 (2009) 1330–1346.
[10] A. Zamanifar, M. Sharifi, O. Kashefi, Self Actor-Actor Connectivity Restoration in Wireless Sensor and Actor Networks, in: 1st Asian conference on Intelligent Information and Database Systems (ACIIDS), 2009, pp. 442–447.
[11] N. Tamboli, M. Younis, Coverage-aware connectivity restoration in mobile sensor networks, IEEE International Conference on Communications (2009) 1–5.
[12] A. Abbasi, M. Younis, U. Baroudi, Restoring connectivity in wireless sensor-actor networks with minimal topology changes, in: IEEE International conference on Communications (ICC), 2010, pp. 1–5.
[13] M. Sir, I. Senturk, E. Sisikoglu, K. Akkaya, An optimization-based approach for connecting partitioned mobile sensor/actuator networks, in: 3rd International Workshop on Wireless Sensor, Actuator and Robot Networks (INFOCOM), 2011, pp. 525–530.
[14] A. Abbasi, M. Younis, U. Baroudi, Recovering from a node failure in wireless sensor-actor networks with minimal topology changes, IEEE Transactions on Vehicular Technology 62 (2013) 256–271.
[15] K. Akkaya, F. Senel, A. Thimmapuram, S. Uludag, Distributed recovery from network partitioning in movable sensor/actor networks via controlled mobility, IEEE Transaction on Computers 59 (2010) 258–271.
[16] L. Dai, V. Chan, Helper node trajectory control for connection assurance in proactive mobile wireless networks, in: 16th International conference on Computer Communications and Networks (ICCCN), 2007, pp. 882–887.
[17] D. Henkel, T. Brown, Delay-tolerant communication using mobile robotic helper nodes, in: 6th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks and Workshops (WiOPT), 2008, pp. 657–666.
[18] F. Senel, M. Younis, K. Akkaya, A robust relay node placement heuristic for structurally damaged wireless sensor networks, in: IEEE 34th conference on Local Computer Networks (LCN), 2009, pp. 633–640.
[19] S. Lee, M. F. Younis, Qos-aware relay node placement in a segmented wireless sensor network, in: IEEE International conference on Communications (ICC), 2009, pp. 1–5.
[20] S. Lee, M. Younis, Recovery from multiple simultaneous failures in wireless sensor networks using minimum steiner tree, Journal of Parallel Distributed Computing 70 (2010) 525–536.
[21] S. Lee, M. F. Younis, EQAR: Effective QoS-Aware Relay Node Placement Algorithm for Connecting Disjoint Wireless Sensor Subnetworks, IEEE Transactions on Computers 60 (12) (2011) 1772–1787.
[22] F. Senel, M. Younis, Optimized connectivity restoration in a partitioned wireless sensor network., in: GLOBECOM, IEEE, 2011, pp. 1–5.
[23] S. Lee, M. Younis, Optimized relay node placement for connecting disjoint wireless sensor networks, Computer Networks 56 (12) (2012) 2788–2804.
[24] I. F. Senturk, S. Yilmaz, K. Akkaya, Connectivity restoration in delay-tolerant sensor networks using game theory, Journal of Ad Hoc and Ubiquitous Computing 11 (2/3) (2012) 109–124.
[25] M. Won, R. Stoleru, H. Chenji, W. Zhang, On optimal connectivity restoration in segmented sensor networks., in: Proceeding of 10th European conference on Wireless Sensor Networks, 2013, pp. 131–148.
[26] R. Falcón, X. Li, A. Nayak, I. Stojmenovic, The one-commodity traveling salesman problem with selective pickup and delivery: An ant colony approach., in: IEEE Congress on Evolutionary Computation, 2010, pp. 1–8.
[27] L.-M. Mou, X.-L. Dai, A novel ant colony system for solving the one-commodity traveling salesman problem with selective pickup and delivery., in: ICNC, 2012, pp. 1096–1101.
[28] R. Falcón, X. Li, A. Nayak, I. Stojmenovic, A harmony-seeking firefly swarm to the periodic replacement of damaged sensors by a team of mobile robots., in: ICC, 2012, pp. 4914–4918.
[29] K. Magklara, D. Zorbas, T. Razafindralambo, Node discovery and replacement using mobile robot, in: Ad Hoc Networks - 4th International ICST Conference, 2012, pp. 59–71.
[30] Y. Wang, A. Barnawi, R. F. de Mello, I. Stojmenovic, Localized ant colony of robots for redeployment in wireless sensor networks, Multiple-Valued Logic and Soft Computing 23 (1-2) (2014) 35–51.
[31] H. Li, A. Barnawi, I. Stojmenovic, C. Wang, Market-based sensor relocation by robot team in wireless sensor networks, Ad Hoc & Sensor Wireless Networks 22 (3-4) (2014) 259–280.
[32] I. F. Senturk, K. Akkaya, On the performance of sensor node repositioning under realistic terrain constraints, in: IEEE 37th conference on Local Computer Networks (LCN), 2012, pp. 336–339.
[33] I. F. Senturk, K. Akkaya, Energy and terrain aware connectivity restoration in disjoint mobile sensor networks, in: 12th IEEE

International Workshop on Wireless Local Networks (LCN), 2012, pp. 767–774.
[34] I. F. Senturka, K. Akkayaa, S. Yilmazb, Relay placement for restoring connectivity in partitioned wireless sensor networks under limited information, Ad Hoc Networks 13 (2014) 487–503.
[35] K. Akkaya, I. F. Senturk, S. Vemulapalli, Handling large-scale node failures in mobile sensor/robot networks, Journal of Network and Computer Applications 36 (2013) 195–210.
[36] I. F. Senturk, K. Akkaya, S. Yilmaz, Relay placement for restoring connectivity in partitioned wireless sensor networks under limited information, Ad Hoc Networks 13 (2014) 487–503.
[37] M. Batalin, 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 23 (4) (2007) 661–675.
[38] M. M. Zavlanos, G. J. Pappas, Distributed connectivity control of mobile networks, IEEE Transactions on Robotics 24 (6) (2008) 1416–1428.
[39] S. Poduri, S. Pattem, B. Krishnamachari, G. S. Sukhatme, Using local geometry for tunable topology control in sensor networks, IEEE Transactions on Mobile Computing 8 (2009) 218–230.
[40] T. T. Truong, K. N. Brown, 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.
[41] T. T. Truong, K. N. Brown, C. J. Sreenan, Autonomous discovery and repair of damage in wireless sensor networks, in: 38th Annual IEEE Conference on Local Computer Networks, 2013, pp. 450–458.
[42] T. T. Truong, K. N. Brown, C. J. Sreenan, Multi-objective hierarchical algorithms in restoring wireless sensor network connectivity in known environments, Journal of Ad hoc Network (2015) 190–208.
[43] K. L. Myers, Cpef: A continuous planning and execution framework., AI Magazine 20 (4) (1999) 63–69.
[44] S. Chien, R. Knight, A. Stechert, R. Sherwood, G. Rabideau, Using iterative repair to improve responsiveness of planning and scheduling, in: 5th International conference on Artificial Intelligence Planning and Scheduling (AIPS), 2000, pp. 300–307.
[45] S. Lemai, F. Ingrand, Interleaving temporal planning and execution in robotics domains., in: D. L. McGuinness, G. Ferguson (Eds.), Association for the Advancement of Artificial Intelligence (AAAI), 2004, pp. 617–622.
[46] P. Doherty, J. Kvarnström, F. Heintz, A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems, Journal of Autonomous Agents and Multi-Agent Systems 19 (3) (2009) 332–377.
[47] O. Pettersson, L. Karlsson, A. Saffiotti, Model-free execution monitoring in behavior-based robotics, IEEE Transaction on Systems, Man and Cybernetics, Part B 37 (4) (2007) 890–901.
[48] M. Molineaux, M. Klenk, D. W. Aha, Goal-driven autonomy in a navy strategy simulation., in: Twenty-Fourth AAAI Conference on Artificial Intelligence, 2010, pp. 1548–1554.
[49] F. Teichteil-Konigsbuch, C. Lesire, G. Infantes, A generic framework for anytime execution-driven planning in robotics, in: IEEE International conference of Robotics and Automation (ICRA), 2011, pp. 299–304.
[50] E. Kaldeli, A. Lazovik, M. Aiello, Continual planning with sensing for web service composition., in: Association for the Advancement of Artificial Intelligence (AAAI), 2011, pp. 1198–1203.
[51] R. P. van der Krogt, M. M. de Weerdt, C. Witteveen, A resource based framework for planning and replanning, in: Proceedings of the 2003 IEEE/WIC International Conference on Intelligent Agent Technology, Vol. 1, 2003, pp. 247–253.
[52] S. Koenig, M. Likhachev, D. Furcy, Lifelong planning A*, Journal in Artificial Intelligence 155 (1-2) (2004) 93–146.
[53] C. Fritz, S. A. McIlraith, Monitoring plan optimality during execution, in: 7th International conference on Automated Planning and Scheduling (ICAPS), 2007, pp. 144–151.
[54] W. Cushing, J. Benton, S. Kambhampati, Replanning as a deliberative re-selection of objectives, in: Technical Report, Arizona State University, 2008.
[55] D. Joslin, M. E. Pollack, Least-cost flaw repair: A plan refinement strategy for partial-order planning, in: Proceedings of the Twelfth National Conference on Artificial Intelligence, 1994, pp. 1004–1009.
[56] N. F. Ayan, U. Kuter, Hotride: Hierarchical ordered task replanning in dynamic environments, in: ICAPS 2007 Workshop on Planning and Execution for Real-World Systems, 2007.
[57] R. E. Fikes, N. J. Nilsson, Strips: A new approach to the application of theorem proving to problem solving, Artificial Intelligence 2 (3-4) (1971) 189–208.
[58] R. E. Fikes, P. E. Hart, N. J. Nilsson, Learning and Executing Generalized Robot Plans, Artificial Intelligence 3 (1972) 251–288.
[59] R. E. Fikes, P. E. Hart, N. J. Nilsson, Some new directions in robot problem solving, Tech. Rep. AI-TN-068, Standford Research Institute (Menlo Park, CA US) (1972).
[60] S. Koenig, M. Likhachev, D* lite, in: Association for the Advancement of Artificial Intelligence (AAAI), 2002, pp. 476–483.
[61] M. Garey, R. Graham, D. Johnson, The Complexity of Computing Steiner Minimal Trees, Journal of Applied Mathematics 32 (4) (1977) 835–859.
[62] E. L. Lawler, J. Lenstra, A. Rinnooy Kan, D. Shmoys, The Traveling Salesman Problem., John Wiley & Sons, 1985.
[63] B. Y. Wu, K.-M. Chao, Spanning Trees and Optimization Problems, Chapman & Hall / CRC Press, USA, 2004.
[64] Moteiv Corporation, http://www.eecs.harvard.edu/ konrad/ projects/shimmer/references/tmote-sky-datasheet.pdf, Tmote Sky Datasheet (2006).



THUY T. TRUONG (t.truong@cs.ucc.ie) received a Ph.D. degree in Computer Science from University College Cork in 2014, and MSc degree in Advanced Distributed System from the University of Leicester, UK in 2009, and a BSc degree in Computer Science and Engineering from Ho Chi Minh University of Technology, Vietnam in 2007. Currently, she is a post-doctoral researcher in Insight/MISL Lab, University College Cork. Her research interests are in network embedded systems and wireless networks.



KENNETH N. BROWN (k.brown@cs.ucc.ie) joined UCC Computer Science Department as a senior lecturer in 2003, where he is the UCC Deputy Director of Insight, the centre for data analytics, and the UCC Principal Investigator on CTVR, the telecommunications research centre. Prior to joining UCC in 2003, he was a lecturer at the University of Aberdeen, a Research Fellow at Carnegie Mellon University, and a Research Associate at the University of Bristol. His research interests are in the application of AI, optimisation and distributed reasoning, with a particular focus on wireless networks.



CORMAC J. SREENAN (cjs@cs.ucc.ie) received the Ph.D. degree in computer science from Cambridge University. He is a full professor of computer science at University College Cork (UCC) in Ireland. Prior to joining UCC in 1999 he was on the Research Staff at AT&T Labs-Research, Florham Park, NJ, and at Bell Labs, Murray Hill, NJ. He is currently on the editorial boards of IEEE Transactions on Mobile Computing, ACM Transactions on Sensor Networks, and ACM/Springer Multimedia Systems Journal. He is a member of the IEEE and the ACM and was elected a Fellow of the BCS in 2005. His research interests include wireless sensor networks, mobile networks and video streaming.