Ad Hoc Networks 45 (2016) 47–64

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

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, Cork, Ireland

ARTICLE INFO

Article history:
Received 30 April 2015
Revised 21 January 2016
Accepted 27 February 2016
Available online 10 March 2016

Keywords:
Wireless Sensor Network
Node deployment
Connectivity repair

ABSTRACT

Wireless Sensor Networks in volatile environments may suffer damage which partitions the network, and connectivity must be restored. We investigate the online problem, in which the repairing agent must discover surviving nodes and the damage to the physical and radio environment as it moves around the sensor field to execute the repair. The objectives include minimising the cost of the repair in terms of new radios and distance travelled, and minimising the time to complete the repair. We consider a number of different agent features which we can combine in different configurations. The repairing agent may prioritise either the node cost or the travel distance. The focus of the agent may be local, with a greedy choice of next partition to re-connect, or global, maintaining a plan for all partitions. To handle the developing knowledge of the network conditions, the agent may revert to full replanning when a change is discovered, or try a minimal adjustment of the current plan in order to minimise the computation effort. For each configuration, we develop a number of heuristics for creating the plans. We evaluate the approach in simulation, varying the density of the connectivity graph and the level of damage suffered. We demonstrate that the plan repair 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, algorithms which prioritise mobility cost are preferred for slow agents, while faster moving agents should prioritise the radio node cost.

1. Introduction

Wireless Sensor Networks are becoming increasingly important for monitoring phenomena in remote or hazardous environments, including pollution monitoring, chemical processes, disaster response, and battlefield sensing. 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 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 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. The main objective of the agent is to minimise the cost of the repair, where the main cost factor may be the number of new nodes, the distance travelled, or the time taken to complete the repair, depending on circumstances.

The agent starts with knowledge of the radio and physical environments before the damage, and is aware of the connected subnetwork after the damage, i.e. the set of nodes still successfully transmitting data to the sink. It must plan a deployment of nodes to restore connectivity for designated data streams, and plan 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 may also

* Corresponding author.
E-mail addresses: t.truong@cs.ucc.ie, thuy.th.tr@gmail.com (T.T. Truong), k.brown@cs.ucc.ie (K.N. Brown), cjs@cs.ucc.ie (C.J. Sreenan).

http://dx.doi.org/10.1016/j.adhoc.2016.02.021
1570-8705/

Table 1
Summary of the agent approaches.

Focus	Local (L)	Global (G)
Priority	Node cost (N)	Path cost (P)
Strategy	Replanning (c)	Repairing (f)

discover surviving disconnected components of the 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 planning) are computationally hard, we use heuristic algorithms to generate the plans. We describe possible agents using four different features. The *focus* of the agent describes how much of the remaining problem should be considered when choosing the next data stream to be connected: a *local* focus uses a greedy approach and considers the next data stream in isolation, while a *global* focus chooses the best sequence of streams to connect The planning *priority* determines which cost factor will be prioritised when developing and executing the plans: the number of nodes or the distance travelled. The online planning *strategy* describes how the agent reacts to new knowledge. Full replanning generates a new plan each time the cost of the plan changes significantly, or when the plan becomes infeasible. Plan repair instead searches for a new subplan to connect its current target, and reverts to full replanning only when that is infeasible or has excessive cost. The search *heuristic* describes how the agent makes its selection of data stream to connect or path to be followed. A summary of our approaches is shown in Table 1. Thus we can construct 8 different high-level approaches, with a possibly different set of heuristics for each one.

We evaluate the approaches in simulation on randomly generated problems, assessing the impact of increasing the damage, increasing the number of data streams to be connected, and increasing the number of candidate locations for radio nodes. We show that the costs all increase when we increase the number of disconnected terminals (i.e. the desired locations from which we want to restore the data streams) and the damage levels but do not always increase with the number of candidate locations. In some cases, the global focus and the full replanning strategy, for both node and path priorities, are poorer than locally focused plan-repair approaches. The main cause of this is the initially unknown environment; the global plans are too specific to the initial knowledge, and require long paths or too many additional radio nodes to recover when the initial assumptions prove invalid, while the local plans with small repairs assume less about the environment, and are thus cheaper to adapt. 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 both the time to place nodes and the computation time. Therefore, approaches which produce low mobility cost should be prioritised. For medium speeds, the gap between the path and the node priorities in total restoration time is smaller. However, with a fast moving agent, the higher mobility costs are less significant, and thus the time to place nodes and the computation time become more important. Therefore, the node priority 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 knowledge structures. We then describe the details of our agent features and the different algorithms. We describe the simulation framework, and conclude with the experimental results.

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 predetermined 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 [1–5], 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. [6–14]. 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 precomputes 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

Dai and Chan [16] proactively deploy 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.

Henkel and Brown [17] deploy mobile robotic helper nodes to physically carry the data to the base station. The approach 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

Senel et al. [18–25] 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 centralised manner with the main objective of using the smallest number of additional nodes. Senel et al. [18] uses a spider web approach to reconnect the segments. In [20], the authors propose a Distributed algorithm for Optimised 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 optimise the number of additional nodes that are needed. Senel et al. [22,23] also propose algorithms using minimum Steiner trees where Senel and Younis [22] find the best subsets of three segments and forms a triangular Steiner minimum tree with minimum Steiner points while Lee and Younis [23] use a minimum Steiner tree on the Convex hull and places relay nodes inwards towards the centre of the damage area.

Also [19,21] 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 centralised 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 disc 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: centralised approaches [26–28] 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 localised solutions [29–31] 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 maximise the network coverage.

Other related work in WSNs

Senturk et al. [24] use 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. Senturk et al. [32–34] 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.

Akkaya et al. [35,36] consider on distributed mobile nodes for re-establishing network connectivity. Akkaya et al. [35] relocate some of the sensors to the locations of the failed sensors to re-establish the routes with the sink node based on local information. Senturk et al. [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. Batalin and Sukhatme [37] deploy a robot with unlimited nodes and drops nodes from time to time based on certain ordering rules. Zavlanos and Pappas [38] control the agent's motion to explore the environment while dropping nodes and preserving the connectivity of the network. Poduri et al. [39] assume 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 [40,41,42]. In [40,42], we introduce the network repair problem in the presence of obstacles in a static environment, and propose different heuristics to solve the problem. Truong et al. [41] model 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 provide 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. Chien et al. [44] use iterative repair techniques to support a continuous planning process for autonomous spacecraft control. Lemai et al. [45] and [46], for temporal planning, interleave decision and execution in a dynamic environment to allow plan repair interleaved with execution. Pettersson et al. [47] use a model-free approach which observes and classifies the actual behaviour of the monitored systems into normal or faulty execution. Molineaux et al. [48] dynamically reasons about which goals to pursue in response to unexpected circumstances. Teichteil-Konigsbuch et al. [49] propose a generic and reactive scheme for continuous planning for complex problems. Finally, Kaldeli et al. [50] describe a CSP-based continual planner for web service composition.

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

van der Krogt et al. [51] describe a framework that can use most heuristic planner/search for replanning. Koenig et al. [52] combine 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. Cushing et al. [54] use rewards and penalties to reason about adding or changing goals. Newly ar-

Table 2
Table of notation.

Notation	Description
Subscripts b, B	Used for before damage
Subscripts a, A	Used for after damage
G	Rectilinear grid of locations
V_b	Set of pre-damage candidate locations for radio nodes
C_b	Set of pre-damage potential radio links
V_B	Set of pre-damage locations with live nodes
V_a	Set of post-damage candidate locations for radio nodes
C_a	Set of post-damage potential radio links
V_A	Set of post-damage locations with live nodes
τ	Set of terminals to be connected
I_v	Set of nodes successfully transmitting data to sink
I_c	Set of active links between live nodes
B_b	Set of pre-damage blocked squares
B_a	Set of post-damage blocked squares
V_n	Set of newly added nodes
L	Starting grid location of the agent
P	Path through the grid
d	Agent probe distance
R	Agent transmission range

riving 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.

Joslin and Pollack [55] describe 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. Ayan and Kuter [56] describe 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 required sub-partitions. The agent has to discover the network and physical environments in order to make the right decisions. Our aim is to optimise our use of resources in partially known environments. To represent the problem of exploring an environment to discover mobility paths, we assume an underlying grid representation. The 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. To represent the connectivity problem, we use a connectivity graph, where is node in the graph is assigned a location in the grid model. We give a description of the problem formulation below, with the notation listed in Table 2.

We will model the environment in two phases: the connectivity and mobility environment before the damage, and the connectivity and mobility after the damage. The environment before damage is known completely, from, for example, a survey performed by robot or human agents. The environment after damage is largely unknown, and so part of our problem is to discover as much of the after-damage environment as is needed to complete an effective repair. For the environment before the damage, we have 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. $C_b \subseteq V_b \times V_b$ is a set of potential radio links, and thus (V_b, C_b) is a potential *connectivity* graph. We assume symmetric links are required for the network operation,[1] and so we ignore any asymmetric connections. $V_B \subseteq V_b$ is the set of locations with actual nodes. After damage, the connectivity graph is (V_a, C_a), where $V_a \subseteq V_b$ and $C_a \subseteq C_b$. $V_A \subseteq V_B \cap V_a$ is the set of locations with surviving nodes. The set $\tau \subseteq V_B$, of *terminals*, is the set of locations from which we require sensed data. $I_v \subseteq V_A$ is the set of nodes still successfully transmitting data to our sink, and I_c is the corresponding set of active links. The repairing agent can move from any square into one of its 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 $L \in I_v$.

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 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 is 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, any initial plan is likely to be either infeasible or inefficient, and so while executing it, the agent must sense its environment, update its knowledge and then modify the plan. The agent can probe[2] 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. Fig. 1 shows an illustrative example of the representation.

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 its prior knowledge of the map for reference in planning, but it must 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:

[1] Many protocols across different layers require symmetric links for proper functioning, e.g. the MAC layer may require symmetric links for acknowledgements.

[2] Using radar, video, or sensing or by physically moving.

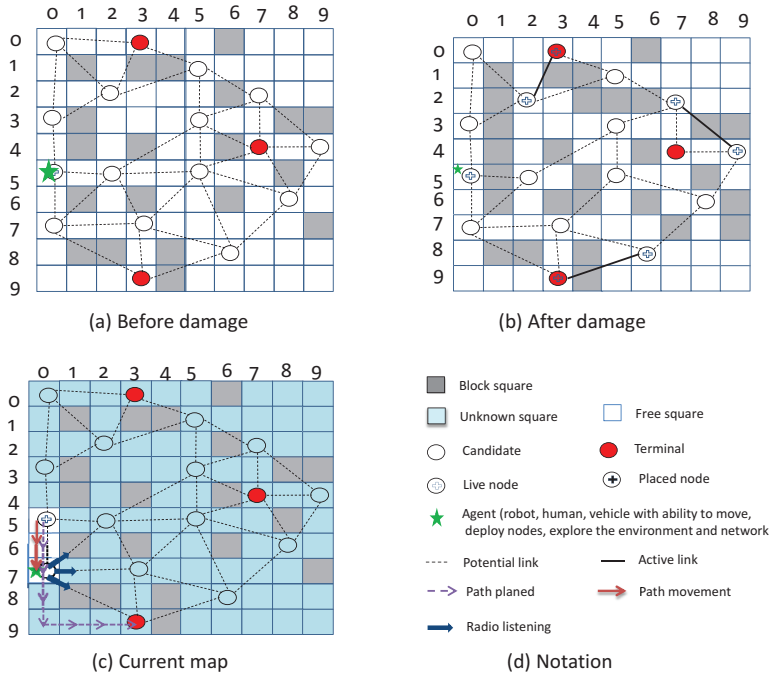


Fig. 1. Example of the network conditions.

- 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:

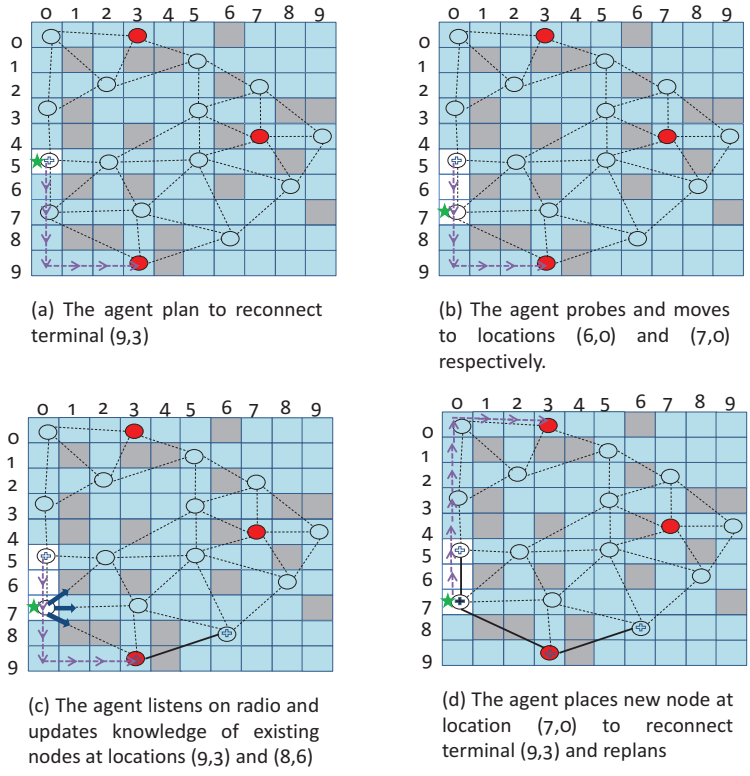
- S_f, locations known to be accessible;
- S_b, locations known to be blocked; and
- S_u, locations whose condition is otherwise unknown.

Initially, $S_f = L$ (the initial location of the agent), $S_b = B_b$, and $S_u = (G - B_b) - \{L\}$.

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

The agent has five possible actions, described below. We base our representation of the actions on the STRIPS-style rules widely used throughout the AI planning literature [57–59], but we extend the rules to allow parametrised actions and arbitrary procedures to represent changes to the agents knowledge structures. The rule pre-conditions, PRE, list the conditions that must be satisfied in the world state for the action to be applied. The delete list, DEL, describes which facts are no longer true after the action completes, and the add list, ADD, describes the new facts which become true after the action complete. The extra procedures, PROC, specify how to update the agent's knowledge after the action.

1. MOVE(u, v): move from square u to square v;
   PRE: at(u) $\wedge$ neighbour(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 neighbour 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;



(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 view as it executes actions.

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, then remove this location from the set of unknown locations N_u and add it into the set of feasible locations N_f.

Fig. 2 shows (a) an initial plan to reconnect the terminal at the location (9,3), (b) execution of the first two movement steps, (c) the updated knowledge after listening for radio communication, and (d) the placement of a new node.

The initial plan (a) is as follows: 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, (b,c), when the agent executes the plan, it will discover changes to the environment:

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 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. The agent revises its plan and (d) drops a new node at location (7,0). This reconnects the terminal at location (9,3), and completes its first task. The agent then repeats the process to reconnect other terminals.

5. Approach

In this paper, the agent starts by assuming some elements of the unknown sets are available, based on the pre-damage environment, and then replans when errors are discovered.[3] 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 in which we intend to visit to place a node. At the lower level, we have selected one of these locations, and we have a detailed path plan for moving there.

We consider two focuses for the agent. In the greedy or *local* focus (L), the agent considers one terminal at a time, i.e. it picks a single terminal to connect, generates a plan, executes it, and then selects the next terminal until all terminals are connected. In the

[3] An alternative approach, not considered here, would be to use planning under uncertainty, and to select plans which minimise the expected cost.

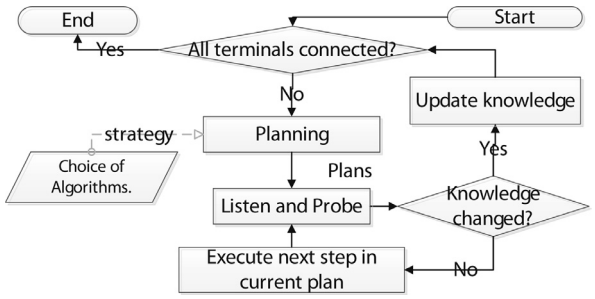


Fig. 3. Flow chart for the agent planning.

global focus (G), the agent 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 (N) and prioritising path cost (P). The node strategy prefers plans that require few nodes and then finds cheap paths for visiting those locations. The path strategy prioritises 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 or infeasible, a new plan is generated, and the process repeats. For the replanning, we also consider two methods. The *repair* method (f) keeps its target terminal, but tries to repair its immediate plan for reconnecting it. For the repair methods in local focus, the agent always keeps the current target but updates new plans when it discovers new knowledge. However, in the globally focused repairing methods, the agent only replans when a cost threshold is exceeded. This strategy is proposed because in general, the time to recompute the plans in local focus is faster than that in global focus. The *replan* method (c) may change its target terminal as soon as it discovers any significant change in the environment, and thus replans each time. The overall approach is shown in Fig. 3.

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)

5.1. Local focus

This focus is based on the belief that the knowledge will change significantly as we explore, and so it is futile to spend time generating a complete plan which will almost certainly 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: first, how to select the terminal which offers the cheapest path in a reasonable time; and second, what should the agent do if it 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, for each of the variants above: node or path priority, and repair or replan. For each of the four configurations (L-N-f, L-N-c, L-P-f, L-P-c), we consider a number of different heuristics for making the choice of node location and path.

The generic local focus and node priority algorithm (L-N-*-*) is shown in Algorithm 1. 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 connected network based on Manhattan distance (L-N-*-MD), and the second is to calculate the optimal number of nodes (or fewest nodes) from a terminal to the connected network (L-N-*-FN). For L-N-*-FN, 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. At each stage, the agent finds a path to the closest one of these nodes using D* Lite [60], which allows us to efficiently integrate new accessibility knowledge into the path planning. As above, if the agent discovers information that changes the node costs (live nodes found, broken links in the current node plan), it recomputes.

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)

The generic local focus and path priority algorithm (L-P-*-*) is shown in Algorithm 2. We have three different heuristics for selecting a target terminal, based on Manhattan distance (L-P-*-MD), shortest distance between the terminal and the agent's location (L-P-*-SD) calculated using D* Lite, or shortest connectivity path to reconnect the terminal to the current network starting from the agent's location (L-P-*-SCP). For L-P-*-SCP, in order to find the cheapest connectivity path for a terminal t, we first find all critical nodes for the current built-up network (N) and the terminal t. The critical nodes for N are candidate locations which directly have potential radio links to at least one node in N (node a, b, c in Fig. 4. There are two cases in finding critical nodes for t. The first case (left figure) is where the terminal is a location without any existing live node. In this case, the critical node for t is t itself and we have to place a new node in the location of t. The second case is when t has an existing live node or it is in a connected component, the critical nodes for t will be all candidate locations which have directly potential radio links to any node in that component (node d and e - right figure).

Now, we find all different combinations to travel from L to t based on D* Lite [60], i.e. 6 and 12 different choices in case 1 and case 2 for the scenario in Fig. 4, respectively. We then select the path which has the lowest mobility cost to travel from L to t. We do the same computation for all the unconnected terminals and then select the terminal which requires the lowest mobility cost as the new target terminal to reconnect. This heuristic requires significant computation time, but should be balanced by improved mobility cost. As before, for each algorithm, the agent will repair or replan as appropriate when it discovers significant changes.

Fig. 5 summarises the classification of the algorithms for the local focus, as described in Table 3.

5.2. Global focus

This focus attempts to plan ahead, selecting the first terminal to reconnect based on the expected cost of completing the connection

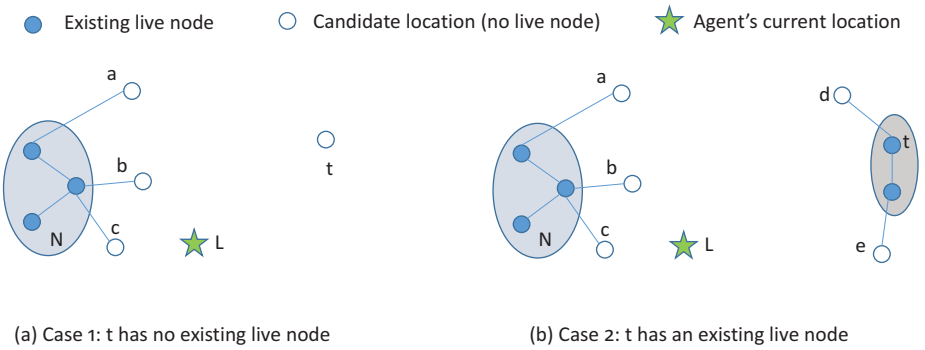


Fig. 4. Two cases in L-P-*-SCP: (a) terminal with an existing live node, (b) terminal with no existing live node.

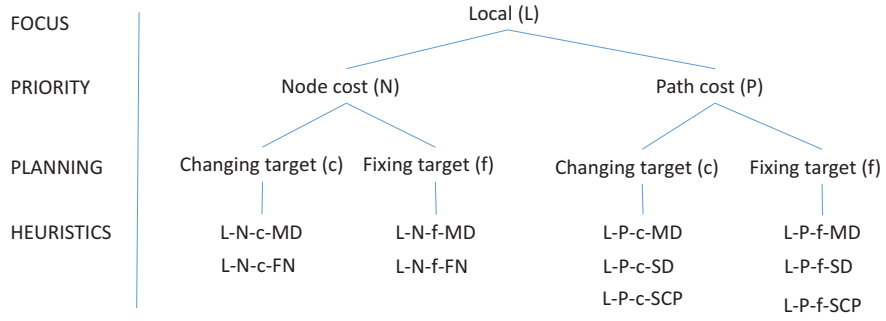


Fig. 5. A classification of the local focus algorithms.

Table 3
Summary of locally focused algorithms. L: local; N: node; P: path; f: fixed terminal; c: change terminal; MD: Manhattan distance; SD: shortest distance; SCP: shortest cheapest path; FN: fewest node.

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

to all the other terminals. The motivation is that the overall plan will be cheaper, and that this might outweigh the cost incurred by recovering from newly discovered damage. As before, the plan 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.

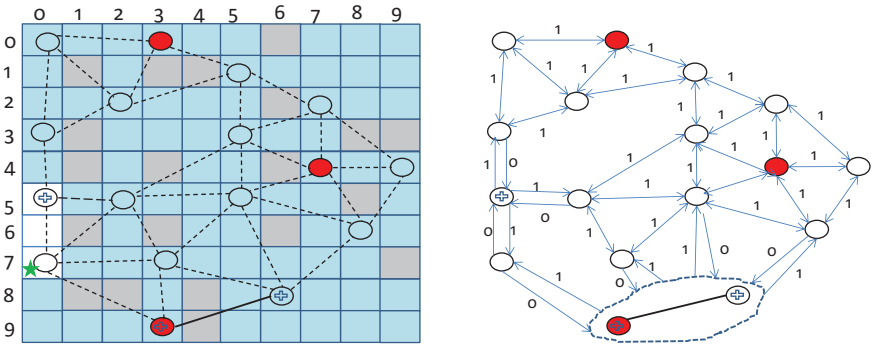
Note that the underlying problems, even when there is no damage, are computationally hard. The task of finding in a graph a minimal set of nodes which connect a terminal set is the minimal Steiner tree in graphs problem, and is NP-hard [61]. Given a set of nodes in a mobility graph, the task of finding 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 again consider heuristic approaches for generating the full plans.

The generic global focus and node priority algorithm (G-N-*)

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)

is shown in Algorithm 3. The aim is to find the smallest set of nodes which reconnects all terminals, and then to find the shortest 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. Fig. 6 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 algorithm 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



(a) Current map (b) Directed weighted connectivity graph

Fig. 6. Example of a directed weighted connectivity graph extracted from a current map.

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

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 (G-N-c), 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 (G-N-f), 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. Changes of type (i) do 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.

The generic global focus and path priority algorithm (G-L-*) is shown in Algorithm 4. The aim is 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. Fig. 7 is an illustrative example of building a weighted connectivity graph and finding a Steiner tree which spans all the 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)

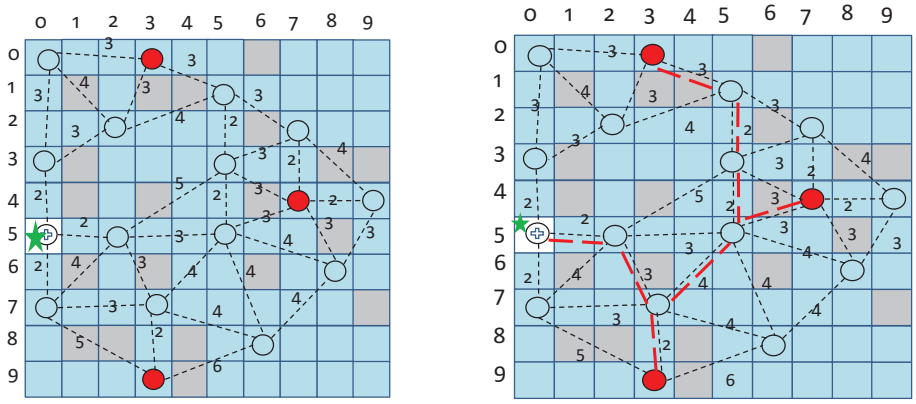
Again, when we discover knowledge that changes the cost of the plan, we revise the plan. We consider the same triggers for full replanning (G-P-c) and repairing (G-P-f) as above, except that change of type (i) would change the path cost. Therefore, in this case of (i) the full replanning will restart 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.

The classification of the global focus algorithms is shown in Fig. 8.

5.3. The adapted DORMS approach

We were unable to find an approach in the literature which could be applied to our problem without significant changes. As summarised in Section 2, previous approaches have tended to assume uniform radio propagation, and free space for movement, and the approaches concentrate on the number of nodes required to make the repair. Some papers do consider physical obstacles for the path planning, but assume full knowledge of the physical and radio environment, and so it is difficult to find a baseline algorithm to compare against without significantly altering the algorithm. We have selected the DORMS approach [20] for our comparison, since it has a clear design principle which can be adapted to the new environment. 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. Therefore, we have adapted the original DORMS, maintaining the original idea of finding a central point from which to reconnect the various partitions, based on Steiner trees in the graph overlaid on the physical environment. We try to use the same algorithms and subroutines as in our own heuristic algorithms, in order to give a fair comparison.

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. Fig. 9 shows an example of the adapted DORMS reconnecting 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 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.



(a) A weighted connectivity graph in a grid

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

Fig. 7. Example of a weighted connectivity graph and a Steiner tree spanning all terminals.

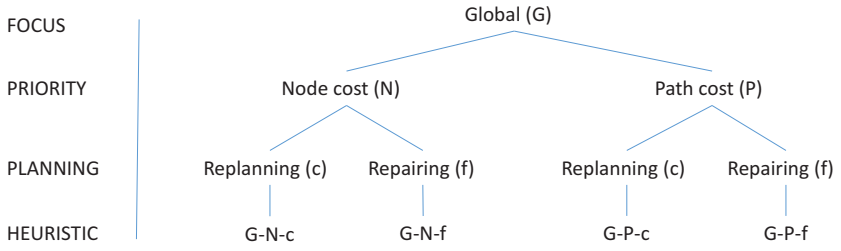


Fig. 8. The classification of the global focus algorithms.

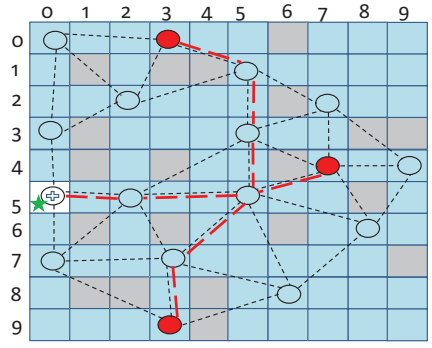


Fig. 9. Example of adapted DORMS. A node plan (red lines) connecting all terminals to a central point (5,5). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

6. Experiments

We now evaluate our algorithms empirically on randomly generated sensor fields and environments, to compare the quality of solutions and the run times of the different algorithm variants. Our aim is to determine the trade-offs between the different algorithm variants, and to assess which particular configurations of focus, priority, planning and heuristic are most suited to different problem characteristics. Note that we are evaluating the repair algorithms rather than any network protocols, and so we use a custom Java simulation of the sensor field, rather than a network simulator.

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 m,[4] 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 the damage, and we remove r% of the radio links. We ensure the problems are feasible—i.e. that there is a set of reachable locations for which nodes would reconnect all terminals—by screening for infeasibility and re-generating if required. In each case, the algorithms only probe a square that the agent intends to move into. For setting up the values for the cost threshold β and the live node threshold γ in the globally focused repairing method (G-P/N-f) (Section 5.2), we note that G-P-f triggers replanning when the agent finds at least β surviving nodes and the expected total path length exceeds the original path length by γ steps, and G-N-f only triggers replanning when it finds at least β surviving nodes. If the values are too high, then the agent rarely changes 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 $\beta = 4$ and $\gamma = 3$ are competitive compared to the local heuristics. In all experiments, we create $a = 15$ 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

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

Table 4
Experiment setup.

Vary	Candidates	Terminals	Damage levels	Grid granularity
	50,100,150	5,10,15	⟨10%, 10%⟩ ⟨20%, 20%⟩ ⟨30%, 30%⟩	45 × 45, 150 × 150, 200 × 200, 300 × 300
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	45 × 45	45 × 45	45 × 45	–
Number of live nodes—α	15	15	15	15
Cost threshold—β	4	4	4	4
Live node threshold—γ	3	3	3	3

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 to the adapted DORMS algorithm in Algorithm 5. We present the experimental results in three sections: local focus, global focus, and agent speed.

Algorithm 5: Adapted DORMS Algorithm.

Result: N: node plan; P: path plan.
while *there are unconnected terminals* **do**
  centre = Find_Centre_Location()
  for *each* $t \in \tau$ **do**
    $p_t = Shortest\_Connectivity\_Path(t, centre, G_C)$
  for *each* $t \in \tau$ **do**
    $ta = Find\_adjacent\_terminal(t)$
    $G' = bounded\_graph(p_t, p_t a, centre, G_C)$
    $T_t = Steiner_M inimal_T ree(t, ta, centre)$
  while *any* $t \in \tau$ *is not in* L **do**
    $T = Find\_Smallest\_tree\_for\_t()$
    Add T into L
  find P, the cheapest path to the nearest node in N

6.1. Local focus

Relative performance of the locally focused heuristics

We first evaluate the relative performance of all the local focus algorithms on a standard problem type, with $c = 100$ candidates, $t = 5$ terminals, the damage level set to $(b = 10\%, r = 10\%)$ and the grid dimensions at 45 × 45.

For the path priority algorithms (L-P-*-*) (Fig. 10), the MD and SD heuristics are almost identical. In addition, there is negligible difference between the c (replan) and f (repair) versions of MD/SD, and so for the remainder of the paper, unless there is a reason to show a specific algorithm's result, we show the results of L-P-c-MD algorithm to represent the four L-P-*-MD/SD.[5] We also note the difference in runtime for L-P-c-SCP and L-P-f-SCP, and so we retain both.

For the node priority algorithms (Fig. 11), L-N-c-MD has similar results to L-N-f-MD, and so we will omit the c variant. The FN heuristic improves on the MD heuristic for node costs, but is poorer in mobility cost. Again, for the FN heuristic, there is some variation between the c and f versions, and so we will retain L-N-f-MD, L-N-c-FN and L-N-f-FN for further analysis.

The relative performance of the selected locally focused heuristics, and Dorms-AD is shown in Fig. 12. Dorms-AD is significantly poorer than the other algorithms in node cost, while the path priority algorithm are surprisingly competitive in node costs. For mobility cost, the MD heuristic with node priority is also surprisingly competitive. There is a small variation between the c and f versions for L-N-*-FN and L-P-*-SCP, and the c versions produces marginally better results in the best performing algorithms for each cost factor. However, in terms of runtime, the replanning methods are clearly slower, and this may outweigh the gains from full replanning. We note that Dorms-AD is significantly faster than the SCP heuristic, but is still dominated by the L-N-f-MD algorithm. In summary, L-N-c-FN is the top algorithm in node cost and L-P-c-SCP is the top algorithm in mobility cost, although L-P-f-SCP may be a better choice because of the lower runtime.

Table 5
Local focus, area 45 × 45, five terminals, damage level ⟨b=10%, r=10%⟩: Runtime (s) vs. number of candidate locations.

	50	100	150
L-N-c-FN	0.24	2.18	34.75
L-N-f-MD	0.07	0.48	5.80
L-N-f-FN	0.16	1.28	19.08
L-P-c-MD	0.13	0.65	7.09
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

Varying the number of candidates

We now consider the impact of varying numbers 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 connectivity is dependent on the distance, packing more nodes into a space increases the density of the connectivity graph. We expect this to improve the solution costs, but at the expense of runtime. The results are shown in Fig. 13 and Table 5.

The first thing to note is that for most algorithms, the solution cost increases as the number of candidate locations increase. We believe this is an artefact of the experimental setup—although the number of candidate locations increases, the number of live nodes after damage does not increase, and the random placement of the live nodes is thus less constrained. This appears to allow, in some instance, clustering of the live nodes away from the terminal lo-

[5] Although we omit the results, we have run the same set of experiments for all algorithms, and can confirm that the omitted results are consistent with this analysis.

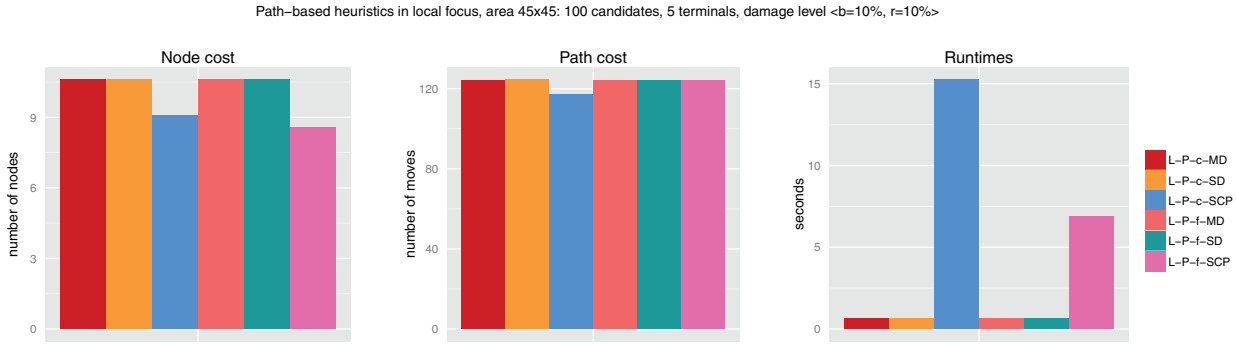


Fig. 10. Path-based heuristics in local focus, area 45 × 45: 100 candidates, five terminals, damage level $\langle b = 10\%, r = 10\% \rangle$.

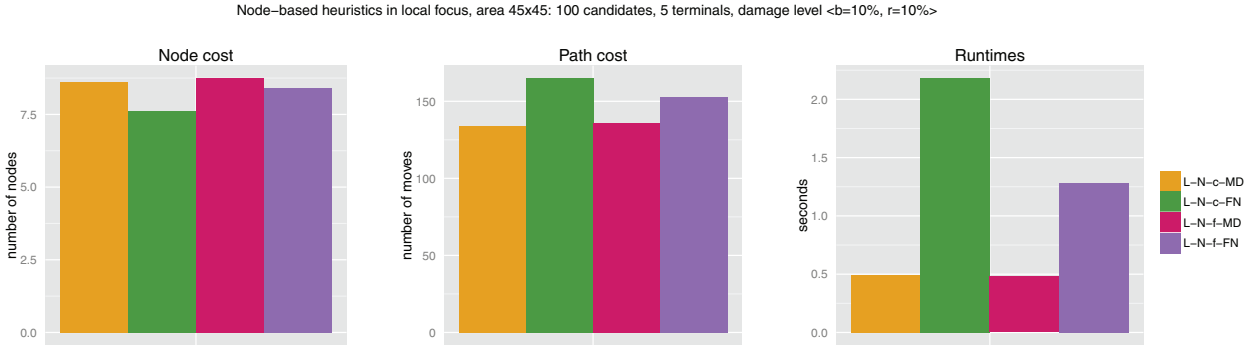


Fig. 11. Node-based heuristics in local focus, area 45 × 45: 100 candidates, five terminals, damage level $\langle b = 10\%, r = 10\% \rangle$.

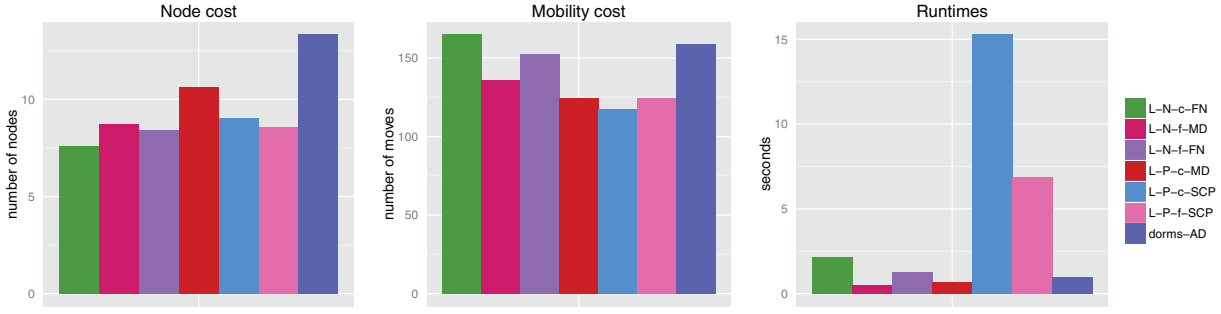


Fig. 12. Local focus, area 45 × 45: 100 candidates, five terminals, damage level $\langle b = 10\%, r = 10\% \rangle$.

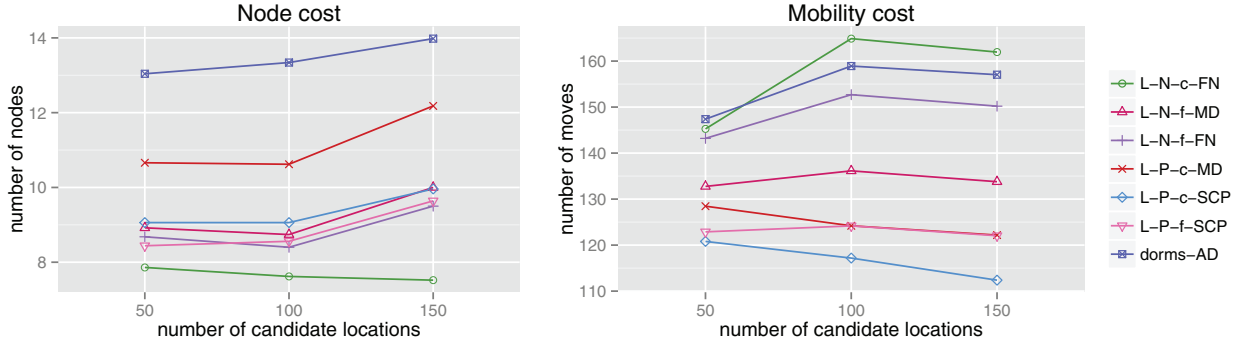


Fig. 13. Local focus, area 45 × 45, five terminals, damage level $\langle b$=10%, r=10%$\rangle$: varying number of candidate locations.

cations, and so requiring longer connectivity paths. The algorithm that specifically prioritise the fewest number of nodes (L-N-f-FN) is the only one to show reducing node costs. For mobility cost, L-P-c-SCP shows the strongest improvement, with its more complex heuristic and full replanning able to exploit the greater opportunities to place nodes. Thus for both cost measures, the algorithms that gave the best results show higher relative performance when we increase the candidate locations. The runtime for all algorithms increases significantly with the increasing number of candidates.

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 in-

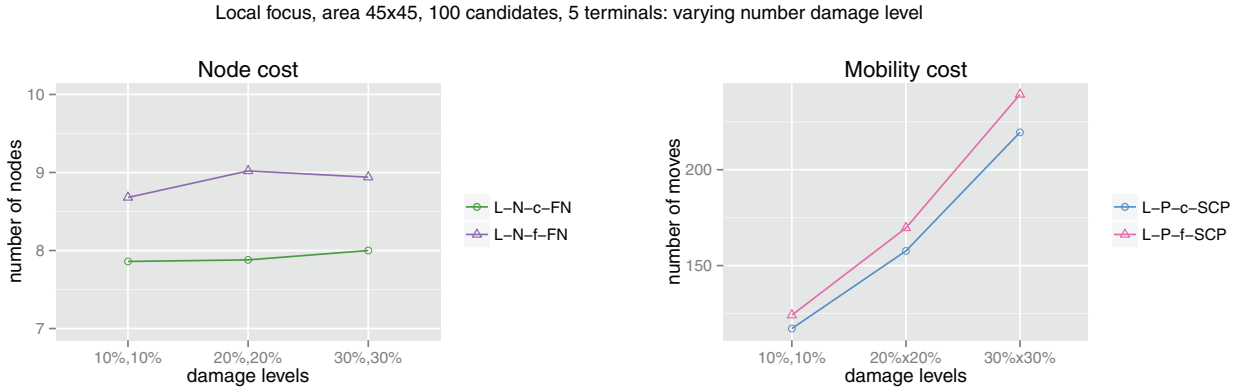


Fig. 14. Local focus, area 45 × 45, 100 candidates, five terminals: varying damage levels.

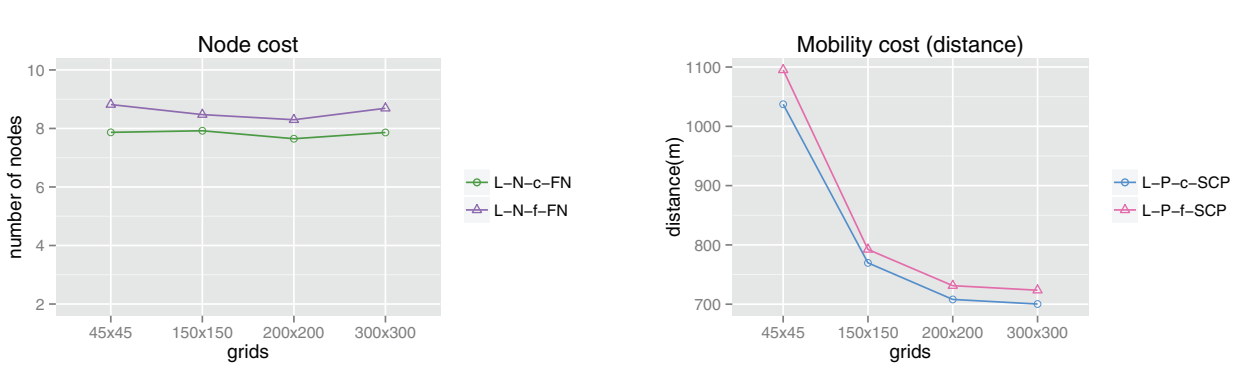


Fig. 15. Local focus, 100 candidates, five terminals, damage level ⟨b=10%, r=10%⟩: varying the granularity of the mobility grid.

crease, more or less scaling linearly, with the increasing number of terminals to be connected, and so the results are omitted.

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 Fig. 14 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). As the damage level increases, we expect the node and mobility costs to rise, as more nodes and longer paths are need to reconnect. However, from damage level ⟨20%, 20%⟩ to ⟨30%, 30%⟩, the node costs for both heuristics are similar. We believe this is due to the fixed size of the area, and so feasible problems for high damage tend to require similar numbers of nodes. 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. For both node and mobility costs, the replanning method (c) continues to produce better quality solutions.

Varying the grid granularity

We note that the representation of the physical area may have an impact on the results—smaller grid squares allow a more finely detailed model of physical obstacles, and may enable paths to be found that are effectively hidden by less finely grained representation, but increasing the number of grid squares is likely to increase computation time. To assess the impact of this, we vary the granularity of the grid. First, we generate a problem for a 300 m × 300 m area with a 45 × 45 grid (square size 6.66 m), 100 candidate locations, five terminals, and damage level (20%, 20%). We then fix the physical locations for candidates, terminals, and blocked squares. We also fix the radio links before and after damage for those candidate locations. We model the area using finer grids of 150 × 150, 200 × 200 and 300 × 300 with the square size of 2 m, 1.5 m and 1 m respectively. The results are shown in Fig. 15 and Table 6. Note that we now plot path distance for the mobility cost instead of simply the number of grid squares. Since all the algorithms behave consistently, we plot just the two algorithms show the best performance for the two measures.

Table 6
Local focus, 100 candidates, five terminals, damage level ⟨b=10%, r=10%⟩: runtime (s) vs. grid granularity levels.

	45 × 45	150 × 150	200 × 200	300 × 300
L-N-c-FN	2.56	45.13	122.39	598.19
L-P-c-SCP	29.11	37.98	72.96	314.25

As expected, the number of required nodes shows only minor variation, since the grid size does not affect the connectivity graph. The mobility cost reduces noticeably as the grid granularity increases, as the agent is able to find paths between the obstacles. The runtimes, however, do increase significantly, and have a higher impact on the node priority algorithms.

6.2. Global focus

We now consider the approaches which have a global focus, and which select the next terminal to re-connect based on the expected cost of completing the plan after that. In Fig. 16, we plot the node cost, mobility cost and runtime for the four global focus approaches, with Dorms-AD for reference, on the standard setup of 100 candidate locations, five terminals, damage level set to ⟨b=10%, r=10%⟩ in a 45 × 45 grid. Each global focus algorithm is plotted immediately before the best corresponding local focus approach,

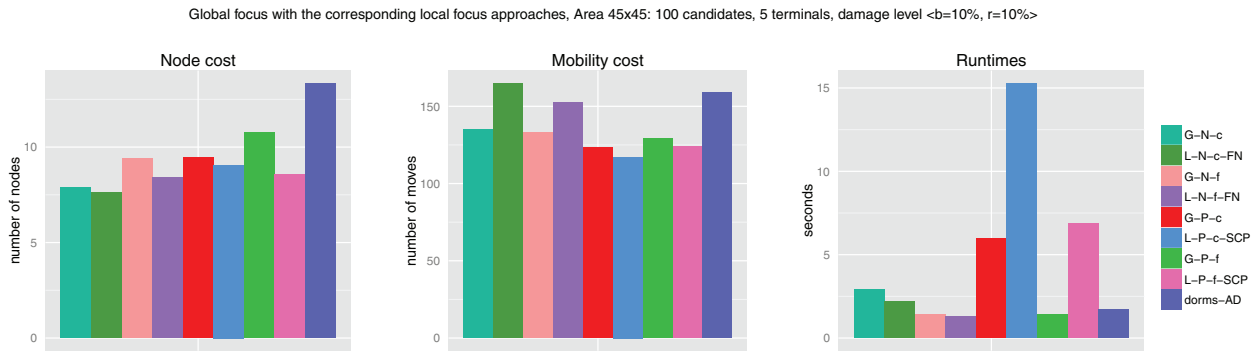


Fig. 16. Global focus with the corresponding local focus approaches, Area 45 × 45: 100 candidates, five terminals, damage level ⟨*b*=10%, *r*=10%⟩.

for comparison. We see that the node costs for the global approaches are all higher than the corresponding local approaches. This may appear to be a counter-intuitive result, since the global focus approach attempts to take a wider view of the problem, and optimises the plan to reconnect all terminals, rather than greedily focusing on the first terminal which may incur extra costs when extended. However, the issue is the need to discover the state of the environment as we execute the plan. The global approach is planning with incorrect or incomplete knowledge, and the optimisations it makes for later stages in the plan may turn out to be misguided, and must be revised. Thus both locally focused and globally focus approaches frequently have to replan after reconnecting the first terminal, and the local approaches benefit from having first connected at lower cost. The advantage of more intense optimisation in the global focus is lost. For mobility cost, the node priority approaches show better performance for the global approaches, but the path priority approaches follow the same pattern as for the node cost graph. For runtime, there is a clear difference between the node and path priority approaches. The local approaches with node priority faster than the global version. The selected local approaches (L-P-*-SCP) with path priority are slower than the global versions. This is due to the expensive procedure for computing the shortest path to each terminal, which is recomputed each time the agent changes target terminal.

We omit the graphs and results varying the features in the experimental set up (number of candidate locations, number of terminals, damage level and grid granularity), since the global focus patterns are the same as for the locally focused approaches.

6.3. The impact of agent movement speeds

The mobility costs in the above analysis are associated only with the distance travelled. However, for repairing a disconnected network, one of the most important objectives is the time required to complete the repair. Total time to repair is influenced by distance, but also by the movement speed of the agent, the time to probe for radio messages and obstacles, the time to place each new node, and the time to plan and replan during execution. We now consider the impact of movement speed on the time to repair. We consider three scenarios, one representing a small robot which moves at 0.1 m s^{-1}, the second representing a human walking at average speed 1.4 m s^{-1}, and the third representing a larger vehicle moving over rough terrain at 4 m s^{-1}. For the 45 × 45 grid in the 300 m × 300 m area, the individual squares are of size 6.66 m × 6.66 m. In each scenario, we assume that it takes the agent 30 s to position a new node.

First, we look at the relative performance of all the algorithms for the standard setup of 100 candidates, five terminals, ⟨10%, 10%⟩ damage level, with the grid size set to 45 × 45. The result is shown in Table 7. For the slow moving agent, the path priority approaches are faster than the node priority approaches, since the path length will be dominant, and outweigh the higher runtimes. The local path priority algorithms are faster than their global counterparts. The global node variants are faster than their local counterparts which are affected by their higher path costs. At the human average walking speed, L-P-c-SCP is still the fastest algorithm, although the relative improvement over the repair method L-P-f-SCP is less, and the lower runtimes start to benefit. The global node approaches are now competitive with the global path approaches, as the path length advantage is less important. For the fast agent, the global node replanning method (G-N-c) has become the fastest, as the high runtimes for the replanning approach become more significant than the improvement in path distance, and as the time taken to place a new radio node also becomes more significant. For all three scenarios, Dorms-AD is either the slowest or the second slowest algorithm in the table, with the performance getting relatively poorer as the agent speed increases. This shows the importance of choosing a quality heuristic that incorporates knowledge of the restrictive physical environment - the fast computation of Dorms-AD is not enough to balance the higher node and path costs it generates.

Table 7
Impact of agent speed on total time to repair (seconds): 100 candidates, five terminals, ⟨10%, 10%⟩ damage, 45 × 45 grid.

	V=0.1 m s^{-1}	V=1.4 m s^{-1}	V=4 m s^{-1}
L-N-c-FN	11229	1023	512
L-N-f-MD	9344	916	494
L-N-f-FN	10441	988	516
L-P-c-MD	8601	910.68	526.25
L-P-c-SCP	**8098**	**845**	482
L-P-f-SCP	8538	851	466
G-N-c	9222	879	**462**
G-N-f	9146	916	505
G-P-c	8511	874	492
G-P-f	8926	938	539
dorms-AD	10995	1148	656

Since varying the grid granularity had the biggest impact on runtimes, we now show the relationship between total restoring time and grid granularity (Fig. 17). The impact of the grid granularity is expected to be high with heavier damage, therefore, we run this experiment set with damage level of ⟨*b*=20%, *r*=20%⟩. Again, we omit the results of global focus because they produce similar patterns. With the slow moving agent, the time to repair generally reduces with the finer grids, although starts to increase at the end because of the high runtime. For the human walking speed, the restoration time is lowest for the middle granularities, for the same reason as above. The FN heuristic suffers most from this increase,

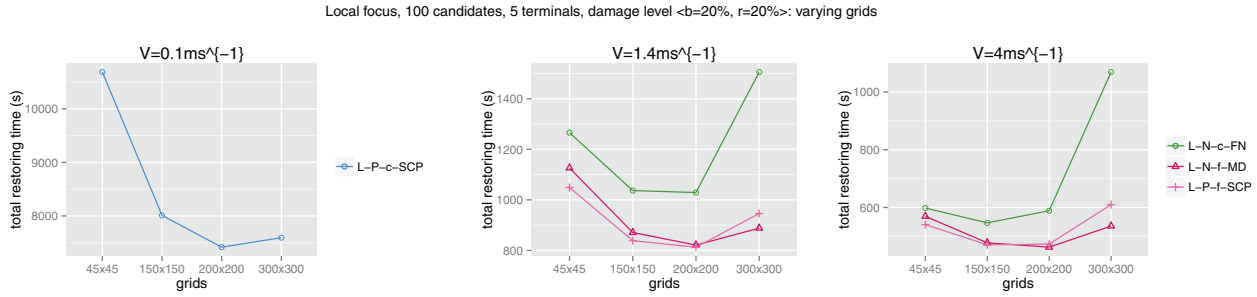


Fig. 17. Local focus, 100 candidates, five terminals, ⟨20%, 20%⟩ damage level: Total restoring time with different mobility grids.

as its path costs are very high, and it becomes uncompetitive. For the fast moving the agent, the pattern is similar. We note that the MD heuristic is now starting to outperform the others, its reasonable mobility and path costs are combined with the fast runtime.

7. Conclusions

Wireless Sensor Networks are being deployed in volatile environments, and thus are subject to damage and must be repaired. The damage may involve failed nodes or broken links, and may cause partitions in the network. New sensors or relay nodes must be placed into the sensor field to reconnect data streams from important sensors. However, movement through the field to make the repair is likely to be constrained, and thus the choice of where to place new nodes will be influenced by the cost of moving through the field to the chosen position. Further, in volatile environments, it is likely that the physical environment will also be subject to damage. The knowledge of the repairing agent will thus be incomplete: it will be aware of those nodes still connected to the sink, and of the location of other nodes before damage, but will not know which nodes and links have survived beyond the connected component; similarly, it may have a model of the environment before damage, but will not be aware of changes to the physical environment that may hinder its access.

We have defined the new problem of simultaneous network repair and exploration, in the presence of unknown physical obstacles, in which the repairing agent must discover the damage as it makes the repair. We have developed an algorithmic framework which allows agents to take a greedy or local focus in selecting the next terminals to reconnect, or a global focus which considers all target terminals simultaneously when choosing its next target. We develop algorithms which prioritise the node cost or the path cost. For adapting to newly discovered knowledge of the network and the environment, we allow two variants, one which tries to repair the sub-plan for reconnecting the current target terminal, and one which replans everything as soon as it discovers any significant change. For each possible approach within this framework, we develop a number of heuristics for generating the plan. We evaluate our algorithms empirically in a simulated sensor field, and we study the effect of varying the parameters of the simulation, including the number of possible locations for placing node, the number of target terminals, the level of damage to the field, and the internal representation of the physical environment. We measure the quality of our algorithms in terms of total node cost, total path length, and runtime. We also consider the total time to repair, which combines the movement time with the time to place new nodes and to compute the plans. For reference we compare our algorithms against an adapted version of the Dorms algorithm - the Dorms algorithm was originally designed assuming free movement through the sensor field, and we adapted it take account of the physical objects in its execution. In almost all of our experimental settings, almost all of our approaches outperform the adapted Dorms in node cost, path cost and total time to repair. The physical environment is closely interlinked with the radio connectivity problem, and these experiments demonstrate the need to take account of the physical environment when planning and executing a repair. Our experiments also demonstrate the pitfalls of premature optimisation when knowledge of the problem is incomplete. The algorithms with a global focus do not perform as well as the algorithms with a greedy local focus. The plans generated with the global focus are better for the initial assumptions, but as the agent discovers changes to those assumptions, it has already executed costly actions, and it costs more to recover the plans to adapt to the new knowledge. The local approaches, although starting with a short-sighted view of the problem, make less commitment to the initial assumptions, and have incurred less cost when they must also replan, and so their final solutions are better. Further, the experiments quantify the tradeoff between node cost and path cost, and measure the impact of the movement speed of the repairing agent, and thus offer guidance to network repair operators in selecting a specific strategy. In all cases, the L-N-c-FN algorithm, which used a greedy re-planning approach to prioritise the node cost, offers the lowest node cost, and thus algorithms of this type should be preferred where node cost dominates. Where the distance travelled is the important metric, the L-P-c-SCP algorithm, which attempts to optimise path length at the same time as planning the node location, performs best, and algorithms of this type should be preferred. For slow moving agents, the path cost dominates, and algorithms which generate the shortest path should be preferred. For fast moving agents, the path cost is less important, and the time taken for computation and for placing the node into the environment start to dominate, and in those cases algorithms which prioritise node cost should be preferred.

There are many areas for future work. We have assumed a graph-based model where the candidate locations for nodes are limited. It is likely that in real-world scenarios, the choice of location would be less restrictive, and the repairing agent must experiment with small adjustments to the positions. Our approach does not consider the quality of the network beyond being connected, and future approaches should consider parameters such as latency and throughput, and so avoiding long multi-hop connectivity paths or bottleneck nodes. 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, including, for example, UAVs to scan the sensor field and provide information to the repairing agent.

Acknowledgment

This work was funded by CTVR, the telecommunications research centre, funded by Science Foundation Ireland (10/CE/I1853) and the HEA PRTLI4 project NEMBES.

References

[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: Eighth International Workshop on the Algorithmic Foundations of Robotics, 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, J. 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, J. Heuristics 21 (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, J. Comput. Telecommun. Networking 52 (14) (2008) 2779–2796.

[9] K. Akkaya, F. Senel, Detecting and connecting disjoint sub-networks in wireless sensor and actor networks, J. 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: First 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, in: IEEE International Conference on Communications, 2009, pp. 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: Third 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 Trans. Vehic. Technol. 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 Trans. Comput. 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: Sixth 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, J. Parallel Distrib. Comput. 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 Trans. Comput. 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, Comput. Networks 56 (12) (2012) 2788–2804.

[24] I.F. Senturk, S. Yilmaz, K. Akkaya, Connectivity restoration in delay-tolerant sensor networks using game theory, J. Ad Hoc Ubiquitous Comput. 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. Falcó3n, 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—Fourth 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 Soft Comput. 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 Sens. 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, J. Network Comput. Appl. 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 Trans. Robotics 23 (4) (2007) 661–675.

[38] M.M. Zavlanos, G.J. Pappas, Distributed connectivity control of mobile networks, IEEE Trans. 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 Trans. Mobile Comput. 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, J. Ad hoc Network 33 (2015) 190–208.

[43] K.L. Myers, Cpef: a continuous planning and execution framework., AI Mag. 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: Fifth 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, J. Autonom. Agents Multi-Agent Syst. 19 (3) (2009) 332–377.

[47] O. Pettersson, L. Karlsson, A. Saffiotti, Model-free execution monitoring in behavior-based robotics, IEEE Trans. Syst. Man Cybern. 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, 1, 2003, pp. 247–253.

[52] S. Koenig, M. Likhachev, D. Furcy, Lifelong planning A*, J. Artif. Intell. 155 (1–2) (2004) 93–146.

[53] C. Fritz, S.A. Mcllraith, Monitoring plan optimality during execution, in: Seventh 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, 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, Artif. Intell. 2 (3–4) (1971) 189–208.

[58] R.E. Fikes, P.E. Hart, N.J. Nilsson, Learning and executing generalized robot plans, Artif. Intell. 3 (1972) 251–288.

[59] R.E. Fikes, P.E. Hart, N.J. Nilsson, Some New Directions in Robot Problem Solving, Technical Report AI-TN-068, Stanford 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, J. Appl. Math. 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, Tmote Sky Datasheet, http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf, 2006.

Thuy T. Truong received a Ph.D. degree in Computer Science from University College Cork in 2014, and M.Sc. degree in Advanced Distributed System from the University of Leicester, UK in 2009, and a B.Sc. 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 robotic network repair in wireless networks.

Kenneth N. Brown is a professor in the Department of Computer Science at University College Cork, where he is the Deputy Director of Insight at UCC, 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 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.