Ad Hoc Networks 33 (2015) 190–208

Contents lists available at ScienceDirect

Ad Hoc Networks

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

Multi-objective hierarchical algorithms for restoring Wireless Sensor Network connectivity in known environments

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

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

ARTICLE INFO

Article history:
Received 13 December 2014
Received in revised form 5 May 2015
Accepted 7 May 2015
Available online 15 May 2015

Keywords:
Wireless Sensor Network
Node deployment
Connectivity repair

ABSTRACT

A Wireless Sensor Network can become partitioned due to node failure, requiring the deployment of additional relay nodes in order to restore network connectivity. This introduces an optimisation problem involving a tradeoff between the number of additional nodes that are required and the costs of moving through the sensor field for the purpose of node placement. This tradeoff is application-dependent, influenced for example by the relative urgency of network restoration. We propose a family of algorithms based on hierarchical objectives including complete algorithms and heuristics which integrate network design with path planning, recognising the impact of obstacles on mobility and communication. We conduct an empirical evaluation of the algorithms on random connectivity and mobility graphs, showing their relative performance in terms of node and path costs, and assessing their execution speeds. Finally, we examine how the relative importance of the two objectives influences the choice of algorithm. In summary, the algorithms which prioritise the node cost tend to find graphs with fewer nodes, while the algorithm which prioritise the cost of moving find slightly larger solutions but with cheaper mobility costs. The heuristic algorithms are close to the optimal algorithms in node cost, and higher in mobility costs. For fast moving agents, the node algorithms are preferred for total restoration time, and for slow agents, the path algorithms are preferred.

1. Introduction

Wireless Sensor Networks (WSNs) consist of multiple sensing and relay nodes organised into a cooperative network which communicate with each other using radios, exchanging information, making joint decisions on network and sensing configurations, and transmitting their data over multiple hops to one or more sink nodes. The sink nodes have access to the wider world and are more powerful than sensor and relay nodes in terms of memory, power, computation, etc. Many sensor nodes with limited resources are deployed in the vicinity of the phenomenon of interest along with a small number of sink nodes which gather and process data. Sensors play dual roles: generating data and relaying data from other sensors to a sink.

Wireless Sensor Networks have been used widely in industry, science, transportation, civil infrastructure, and security. Many applications expose WSNs to danger such as direct attack in a battlefield or accidental damage from wildlife and weather, and collapsed buildings. In addition, sensor nodes have limited power sources, and thus they can fail due to depleted batteries. Network connectivity can be significantly degraded upon the loss of just a few nodes. The loss may partition the network, in which groups of nodes may only be able to communicate with each other, with no route to the wider network. This leads to a longer delay in the messages or a network that no longer functions due to many nodes being disconnected.

* Corresponding author.
E-mail addresses: t.truong@cs.ucc.ie (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.2015.05.005
1570-8705/

Therefore, to overcome sensor node failure and to restore network connectivity, network repair should be initiated, where we must place new nodes in the environment to restore connectivity to the sink for all sub-partitions. It is this issue that we consider in this paper.

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

In general, WSN applications require the minimum number of deployed nodes not only to reduce the node cost but also to reduce the maintenance cost. Some of these applications might require a quick recovery, e.g. intrusion detection, or air pollution monitoring (which monitors the concentration of dangerous gases), etc. One specific application where the joint optimisation of connectivity restoration and route planning could be pursued is the intrusion detection where a deployed WSN may be destroyed by intruders or by accidents and new nodes need to be positioned quickly to restore the system. Another application could be firefighters search and rescue in a building, where a deployed WSN was wiped out by a fire. However, information for evacuation such as fire spreading and locations of survivors in the building need to be collected quickly. In these scenarios, the joint optimisation can be pursued to minimise the cost of new positioning nodes and also the time to repair the network. Our contribution is the novel problem of simultaneous network connectivity restoration with constrained route planning, in the presence of obstacles, and the development and analysis of a family of hierarchical algorithms including complete algorithms and heuristic algorithms. We assume a known connectivity graph which includes all possible new node locations and existing nodes, and where the edges indicate that two positions could communicate with each other. We also assume a mobility graph over the same set of positions, but where the edges represent a possible motion path between two positions.

We first consider hierarchical objectives, and propose two complete algorithms (Optimal Node Algorithm (N-OPT) and Optimal Path Algorithm (P-OPT)). We then study an algorithm for finding the Pareto set, which contains all non-dominated solutions – that is, solutions that improve on all other solutions on at least one of the objectives. Because the runtimes of the complete algorithms are not very efficient (long runtimes), we propose two heuristic algorithms which prioritise the different objectives: Shortest Cheapest Path algorithm (SCP) prioritises node cost and Integrated Path algorithm (IP) prioritises mobility cost. We evaluate our proposed algorithms on randomly generated graphs, varying the number of terminals and the graph density, comparing the node cost and mobility cost for each algorithm. We also assess the total time for restoration, incorporating CPU time, movement time and installation time, for different speeds of agent. We have found that the SCP algorithm tends to find graphs with fewer nodes, while the IP algorithm finds slightly larger solutions but with cheaper mobility costs. The SCP algorithm is significantly faster, particularly on dense graphs. Both SCP and IP are close to N-OPT in node cost (approximately one extra node for SCP, and two extra nodes for IP compared to N-OPT), and higher in mobility costs than P-OPT (approximately between 16% and 22% for SCP, and between 10% and 13% for IP). In addition, SCP and IP are close to the Pareto frontier in node cost but quite far from the Pareto frontier in mobility cost. We have also found that the capacity of an agent impacts the choice of heuristics. For fast moving agents, the SCP algorithm is faster in total restoration time, and for slow speed moving agent, the IP algorithm will be faster.

In the remainder of the paper, we discuss research background and some related work, then we introduce the network repair problem, followed by details of the proposed algorithms. We then describe the experiments and results, and finish with the conclusions.

2. Background and related work

The wireless communication network and the mobility problem can be represented as graphs. An undirected graph G is a pair (V,E), where V is a set of vertices, and E is a set of edges $E=\{\{x,y\}:x\in V,y\in V\}$. We can augment a graph with a cost function, which is normally either a vertex-weight $w:V\rightarrow N$ assigning a weight to each vertex, or an edge-weight $c:E\rightarrow R$ assigning a weight to each edge. The vertex-weight of G is then $\Sigma_{v\in V}w(v)$, while the edge-weight of G is $\Sigma_{e\in E}c(e)$. A subgraph S of G is a graph $S=(V',E')$, where $V'\subseteq V$ and $E'\subseteq E$. A path P from vertex x_0 to x_n in a graph G is a sequence of oriented edges $\langle(x_0,x_1),(x_1,x_2),\ldots,(x_{n-1},x_n)\rangle$, where $\{x_{i-1},x_i\}\in E$, for each i from 1 to n. The edge weight of a path P is $\Sigma_{e\in P}c(e)$. A circuit is a path in which $x_n=x_0$. A cycle is a circuit in which $x_0(=x_n)$ is the only vertex that appears twice. A connected graph is one in which there is a path between every pair of vertices. A graph is said to be k-connected if it remains connected whenever fewer than k vertices are removed. Given

a graph $G = (V, E)$, a metric closure graph for a set of vertices $V' \subseteq V$ is a complete graph over the nodes in V', where each edge has a weight equal to the shortest path between the two endpoints in the original graph.

A tree is a connected graph with no cycles, and a subtree of a graph is simply a subgraph that is also a tree. For a graph $G = (V, E)$, a spanning tree is a subtree $T = (V, E')$ containing all the vertices in V. A graph may have many spanning trees. The weight of a tree is just the sum of weights of its edges. Different trees can have different weights. A minimum spanning tree for a given graph is a spanning tree with minimum weight for the graph.

Given a graph $G = (V, E)$ and a subset of nodes τ of V called terminals, a Steiner tree for τ in G is a subtree $T = (V', E')$ in which $\tau \subseteq V'$; equivalently, T is a tree in G that spans all vertices of τ. The difference between a spanning tree and a Steiner tree is that the spanning tree spans all the vertices in V while the Steiner tree of τ spans all vertices in τ. A Steiner minimal tree for a given graph $G = (V, E)$ and a subset of terminals $\tau \subseteq V$ is a subgraph of G which is a tree and connects the terminals in τ with minimum weights. The problem of finding a Steiner minimal tree in an edge-weighted graph is NP-hard, and remains NP-hard even if all edge-costs are equal [1–3]. If the edge weights are all the same, then the problem is equivalent to that of finding a minimal vertex-weighted Steiner tree with equal weights.

2.1. Related work

The subject of network restoration for wireless networks has been an active area of research. The different approaches can be classified as (i) deploying redundant nodes so as to be able to cope with a pre-determined number of failures, (ii) use of mobile (actor) nodes that can be moved into position in order to restore connectivity, (iii) dispatching mobile nodes in a pre-emptive manner to avoid failures in connectivity, (iv) the deployment of additional nodes to restore connectivity after failures have occurred, and (v) sensor relocation by mobile robots.

2.1.1. Deploying redundant nodes to achieve a level of connectivity

In [4–8], the goal is to deploy redundant nodes with the intention of achieving k-connectivity. The main idea of these papers are 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 cover the current node if it fails. These networks are suitable for fault-tolerant and multiple failure applications but require many redundant nodes, which may be expensive. Finally, the main focus is not on repairing the damaged network, but on achieving fault tolerance.

2.1.2. Repairing connectivity in Wireless Sensor and Actor Network (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. Like in WSNs, the sensors gather information about the phenomenon/phenomena. The actors usually take decisions and perform suitable actions upon the information collected from those sensors. In this network, ideally, the sensors should be able to communicate with an actor. 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. [9–17]. The papers work on the Wireless Sensor and Actor Network with the assumption that all the sensors are connected and the work is to 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). Ref. [10] is the first work presenting a centralised algorithm to re-establish connectivity for a disconnected actor network. The paper considers the minimum connected dominating set (MCDS) of each sub-network (partition, segment) and picks the appropriate actor(s) to move to sustain the connectivity. Because one node moving to replace a faulty node may also cause other nodes to be disconnected, this paper also selects additional intermediate nodes to perform cascaded movement in order to maintain the network connectivity. This cascaded movement approach is preferred to reduce the energy drain due to movement. The goal is to minimise the number of movements, the maximum distance travelled, and to distribute the movement load among other nodes to prevent the bottleneck problem where a node is depleted of its energy due to long movement.

Abbasi et al. [9] uses cascaded movement for a number of suitable actors toward the failure node. It proposes a Distributed Actor Recovery Algorithm (DARA) which is a localised, distributed algorithm to efficiently restore the connectivity of the actor network. In this paper, the actor nodes are selected to move based on three simple rules: the actors with the lowest degree first, i.e. connectivity degree or a number of connected neighbour actors, and then for those actors with the same degree, the selection will be the actors which have the shortest distance to the failed node, and finally for those actors with equal distances, they select the actors with smallest ID. DARA prevents further partitioning that result from moving nodes by pursuing cascaded node relocation and uses the three simple rules above to select the moving nodes.

Akkaya and Janapala [11] focuses on achieving the maximal actor coverage while meeting the actor connectivity requirements (all the actors in the network must stay connected). It applies repelling forces to spread actors in the area of interest for maximised connected coverage of all the actors. In this work, all actors will be involved in movement. A neighbour j of node i will form a force F_{ji} on node i which has its direction from node j to node i, and also node i will form a force F_{ij} on node j where $F_{ij} = -F_{ji}$ (opposite direction). The composite force F_i of a node i will be defined by the addition of two forces from two neighbours, and then the actor can move in the direction of that force a distance that is proportional to its magnitude. The travel distance will be restricted to sustain the connectivity with the neighbours.

Unlike the above approaches that change the network topology by cascaded or block movement, Tamboli and Younis [14] proposes a Coverage Conscious Connectivity Restoration (C^3R) algorithm which strives to keep most of the network topology intact. This approach will involve the neighbours of the faulty node in movement. For example, if a node i has failed, then the neighbours N of node i will be involved in movement to recover node i unless they are performing another recovery for another node. In particular, they will make a schedule and follow the schedule to substitute the faulty node. This approach gives a better solution in coverage but requires more movement compared to the above approaches.

Zamanifar et al. [13] also uses a different movement scheme to move all disconnected partitions towards one another based on its proactive restoration policies, i.e. it uses a number of policies to find suitable candidate actors in the neighbours of the failed node and then moves those candidate actors towards each other. After that, it re-applies the policies for space replacement (selecting the next candidate actors following the policies and moving them toward the previous moved actors). However, the work only connects two sub-networks at a time.

In contrast, [15,17] apply block movement for only the smallest partition for minimal topology changes. In [15], the authors propose a Least-Disruptive topology Repair (LeDiR) algorithm which relies on the local view of a node to relocate the least number of nodes to cover the disconnection. Upon the detection of the network partitioning, LeDiR identifies the smallest block and limits the movement to that block only.

The repair methods listed in this category are for restoring the connectivity in Wireless Sensor and Actor Networks for a single node failure at a time only. The work is extended in [18] to deal with multiple failures. It proactively pre-computes cut-nodes and Connected Dominating Set (CDS), and designates the appropriate neighbours to cover them if they fail and then applies cascaded movement for replacement. Therefore, this work involves all dominatees to the cut-node. Although there are many papers on restoring damage in Wireless Sensor and Actor Networks, the solutions are quite limited. Most of the work focuses on dealing with a single failure and re-connecting just two networks at a time, and with an assumption that mobility is unimpeded by obstacles in free space.

2.1.3. Dispatching mobile nodes to avoid disconnection

Dai and Chan [19] proactively deploys additional helper mobile nodes, controlling their trajectories in response to predicted network disconnection events. The work assumes that the mobile nodes are always fast enough to reach the desired destination in case of a predicted disconnection event, and that a full map of the physical terrain and radio environment is available. Details of how to determine the number of mobile nodes that are needed and the related path planning are not provided.

Henkel and Brown [20] deploys mobile robotic helper nodes to physically carry the data to the base station or the helper nodes will carry traffic from one disconnected site to the base station. The work 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.

2.1.4. Deploying additional nodes to repair the connectivity

Senel et al. [21–25] 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. [21] uses a spider web approach to reconnect the segments. In [24], the authors propose a Distributed algorithm for Optimized Relay node placement using Minimum Steiner tree (DORMS). This approach forms a connectivity chain from each segment toward a centre point and then seeks to optimise the number of additional nodes that are needed.

Also [22,23] model the area into 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 works assume a free space where nodes can move freely to achieve minimum travel distance or other goals (minimum number of nodes, etc.). The works also assume uniform transmission range where each node has equal transmission range as a disk (centre at the node's location and radius as the transmission range). The assumption is not realistic due to radio propagation obstacles and physical movement obstacles. The last one is closest to our research but different in two respects, firstly in that we optimise both the number of additional nodes as well as the path length needed for their deployment, and secondly in that we explicitly take into account the impact of obstacles that can alter both the available paths and the ability of nodes to communicate directly.

Our work in this paper extends from the paper [26] where we introduce the network repair problem with node and movement cost constraints in the presence of obstacles. We focus on multiple failures and placing new nodes in the environment to restore connectivity for all sub-partitions to a sink. In [26], we simply define two heuristics, one

prioritises the node cost and the other prioritises the movement cost. In this paper, we develop two complete algorithms, and an algorithm to find the Pareto set, and compare all the algorithms in more extensive experiments.

2.1.5. 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 [27–29] 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 [30–32] where each robot may carry at most one sensor and makes decision that depends only on locally detected information.

2.1.6. Other related work

Senturk et al. [33] uses game theory to reconnect the network. Again, the work assumes a free space where the mobile nodes can move freely and new nodes can be placed in any positions. Some papers [34,35] 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.

There is also research on topology control using mobile agents. Batalin and Sukhatme [36] deploys a robot with unlimited nodes and drops nodes from time to time based on certain ordering rules. Zavlanos and Pappas [37] controls the agent's motion to explore the environment while dropping nodes and preserving the connectivity of the network. Poduri et al. [38] assumes a mobile sensor network where nodes can use repulsion and attraction forces to arrange the topology. These papers focus on topology control and deployment but do not consider repair/restoration after damage has occurred.

3. The network repair problem

The network repair will focus on placing new nodes in the environment to restore connectivity to the sink for all sub-partitions. Our aim is to optimise our use of resources in the static fully observed problem. We develop two complete algorithms, two heuristic algorithms, and an algorithm to find the Pareto set, and compare all the algorithms in experiments.

In this paper, we assume symmetric links are required for the network operation, and so we ignore any asymmetric connections. This is because many protocols across different layers require symmetric links for proper functioning, e.g. the MAC layer relies on symmetric links for acknowledgements and many routing protocols assume symmetric links. We assume a survey has been completed, and so we know all possible accessible locations for the radio (relay and/or sensor) nodes and the potential viable radio links between them. We assume a set of terminal nodes which are the desired locations from which we want to get data reports but we cannot due to the damage, and our aim is to select enough new locations for radio nodes to reconnect those terminal nodes. We are considering the extreme version of the problem, where everything has been destroyed except the connected component around the sink node. We represent this problem as a set τ, where each $v \in \tau$ is a terminal, and a connectivity graph $G_C = (V, E_C)$, where $\tau \subseteq V$ and each vertex $v \in V$ is a possible location, and each edge $(v_i, v_j) \in E_C$ represents a potential radio link between the two locations. We assume all radio nodes have the same cost, and so we associate a unit vertex weight function w with G_C, such that $w(v) = 1$ for each $v \in V$, representing the cost of positioning a radio at that position. Our first aim is then to find a Steiner tree S in G_C for τ; that is, a connected set of vertices that includes all terminals in τ. If we find a Steiner minimal tree (minimising $w(S)$ where $w(S)$ is the total cost of positioning radio nodes at all vertices in S) then we ensure as few radio nodes are used as possible.

We assume accessibility paths are known between the different candidate radio positions, and that there is a known cost of moving between any pair of positions, where the cost may represent time, energy, distance or hazard. We represent this as a graph $G_M = (V, E_M)$ with an associated edge cost function c. For any set $V' \subseteq V$, a circuit P in G_M that visits all vertices in V' represents a tour for the agent, and we can compute the associated path cost. For any given Steiner tree $S = (V', E'_C)$, a circuit P in G_M that visits every vertex $v \in V'$ then represents a possible tour in which the agent can place all necessary radio nodes to reconnect the network. Minimising $c(P)$ ensures that the cheapest circuit is selected. Finding a cheapest circuit in a graph which visits all specified vertices of a graph is the set TSP (generalisation of the Traveling Salesman (TSP)) problem [39]. Note that the two objectives may conflict: a larger Steiner tree may allow a cheaper path, and a more expensive path may be required for a smaller Steiner tree. The problem of finding a minimal Steiner tree and the problem of finding a minimal cost circuit visiting a set of vertices in a graph are both NP-hard [2,39,40]. The summary of notation is in Table 1.

We can now state the formal problem:

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

As an example, Fig. 1 shows a connectivity graph and a mobility graph for a set of terminals $\tau = \{t_1, t_2, t_3\}$ and a set of candidate locations $\{a, b, d, e, f, g, h, j\}$. The minimal Steiner tree in the connectivity graph has the vertex set $\{t_1, t_2, t_3, a, d, g, f\}$. However, this is not an easy tree for the agent to create, since there is no short path between

d and g. The Steiner tree $\{t_1, t_2, t_3, h, b, e, f, g, j\}$ requires two extra radio nodes, but allows a shorter circuit. Which of these solutions should be selected will depend on the relative cost of the radio nodes compared to the path cost. Applications which require the use of expensive radio nodes will prefer the first tree, while applications which involved high mobility costs (high energy movement, etc.) will prefer the second tree.

The novelty of this problem formulation is in the integration of the connectivity restoration problem and the route planning problem over graphs with the same vertex set. We first consider hierarchical objectives, and propose two complete algorithms which prioritise the different objectives, Optimal Node Algorithm (N-OPT) first finds the optimal node cost, and then finds the optimal path for that node cost while Optimal Path Algorithm (P-OPT) finds the optimal path that reconnects the network and then finds the optimal node cost for that path. We then study an algorithm for finding the Pareto set, which contains all non-dominated solutions. We then propose two heuristic algorithms, prioritising the two different objectives. Shortest Cheapest Path (SCP) prioritises node cost, and first tries to optimise the number of nodes required to heal the network; it then tries to optimise the path cost which visits all the new nodes. Integrated Path (IP) integrates the two objectives by adding weights into the connectivity graph to approximate the mobility cost of establishing each link, and then searches for the cheapest tree that connects all required nodes. We evaluate our proposed algorithms on a randomly generated graphs, varying the number of terminals and the graph density, comparing the node cost and mobility cost for each algorithm. We also assess the total time for restoration, incorporating CPU time, movement time and installation time, for different speeds of agent.

4. Optimal Node Algorithm (N-OPT)

This approach first finds the optimal node cost and for that it finds the optimal path cost. The pseudo-code of N-OPT is as in Algorithm 1. We use the algorithm SMT-generation in [41] which generates all the Steiner Minimal Trees for a fixed number of terminals on the connectivity graph (Algorithm 2), and then for each Steiner Minimal Tree, we apply the Branch and Bound algorithm in [40] to find an optimal path visiting all the nodes in the mobility graph (Algorithm 3).

Table 1
Table of notation.

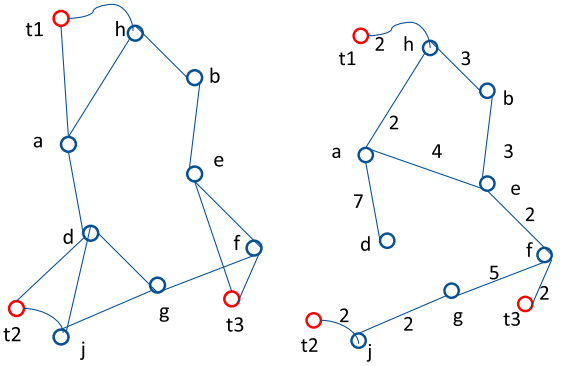
Notation	Description
V	Set of possible locations/positions for radio nodes
E_C	Set of potential radio links between locations
$G_C = (V, E_C)$	A connectivity graph
$w(v), v \in V$	Cost of positioning a radio node at location v
E_M	Set of accessibility paths between locations
$G_M = (V, E_M)$	A mobility graph
$c(e = (v_i, v_j)) \in E_M$	Cost of moving from v_i to v_j or vice versa in G_M
P	A path in G_M
$\tau, \tau \subset V$	Set of terminals
A	Set of live/existing nodes

Algorithm 1. N-OPT Algorithm

Data: A graph $G_C = (V, E_C)$, a graph $G_M = (V, E_M)$, a set of terminals $\tau \subset V$.
Result: a set of required nodes N for τ in G_C and an optimal path P visiting all nodes in N in G_M

```
1  begin
2      L = SMT-generate(G_C, τ) (Algorithm 2)
3      mincost = INFINITY
4      for each MST tree T in L do
5          G_T(V_T, E_T) = metric_closure(V_T, G_C)
6          Path = BranchAndBound-search(G_T, V_T) (Algorithm 3)
7          if cost(Path) < mincost then
8              mincost = cost(Path)
9              N = V_T, all nodes in T
10             P = Path
11     return < N, P >
```

In order to generate all the Steiner Minimal Trees (SMTs), the algorithm firstly constructs the set $R \subseteq V \setminus \tau$ such that $|R| \leqslant |\tau| - 2$. To understand this set R, let T be a Steiner tree on G for τ and $T(V)$ be the set of vertices in tree T. Then the set of Steiner nodes will be $S = T(V) \setminus \tau$, which can be divided into two partitions R and L. The nodes in R, called routers, are the Steiner nodes with a degree greater than 2, and the nodes in L, called linkers, are the Steiner nodes with the degree equal to 2. For any Steiner Minimal Tree T connecting the terminal set τ, it is proved that $|R| \leqslant |\tau| - 2$ [41]. Now, for each set of routers R, the algorithm constructs a complete graph $G_{R\tau} = (V_{R\tau}, E_{R\tau})$ with $V_{R\tau} = R \bigcup \tau$ and the weight of an edge $uv \in E_{R\tau}$ is equal to the minimum path length between two nodes u and v. In other words, it constructs a metric closure graph (complete graph) for the set $V_{R\tau} = R \bigcup \tau$ in the graph G. We then find all minimum spanning trees in the graph $G_{R\tau}$, using the algorithm in [42]. We then consider each edge in that tree in turn and, if the end points are not yet connected in the new tree, select the corresponding shortest path from the original graph. If this shortest path contains no more than 1 vertex already added, we add all edges into



(a) Connectivity Graph (b) Mobility Graph

Fig. 1. Example connectivity graph and mobility graph where t_1, t_2 and t_3 are terminals.

the tree, with their required vertices; if it contains more than 1 vertex already added, then we add those edges that are not already in the graph, and their associated vertices if needed. Because we only consider the set of Steiner nodes to re-connect the terminals, we will remove those Steiner trees which output the same Steiner nodes but different edges or output a tree whose nodes are a superset of any set of nodes of the trees found so far.

The time complexity of Algorithm 2 is dominated by two main components: the computation for the shortest path for all pairs $u, v \in V$ in line 2, and the constructing $G_{R\tau}$ in line 4 and finding all minimum spanning trees at line 5 for each graph $G_{R\tau}$ generated in line 3. The first component is in line 1 which can be done in $O(|V|^3)$. For the second component, firstly, we have the total number of graphs $G_{R\tau}$ satisfying the condition in line 3 is $O(|V(G_{R\tau})|^{|\tau|-2})$ where $V(G_{R\tau})$ is the set of vertices in the graph $G_{R\tau}$. Each single execution of line 4 takes $O(|V(G_{R\tau})|^2)$ time by using the distances calculated in line 2. Line 5 takes $O(|V(G_{R\tau})|.|E(G_{R\tau})|\log(|E(G_{R\tau}|) + |V(G_{R\tau})|^2)$ [42]. Therefore, by neglecting the constants, the total time complexity of Algorithm 2 is $O(|V|^3 + (|V(G_{R\tau})|^{|\tau|-2}).((|V(G_{R\tau})|^2) + (|V(G_{R\tau})|.|E(G_{R\tau})|\log(|E(G_{R\tau}|) + |V(G_{R\tau})|^2))))$.

Algorithm 2. SMT-Generate Algorithm

```
Data: A graph G_C = (V, E_C), a set of terminals τ ⊂ V.
Result: A list L of Steiner Tree T = (V', E'_C) for τ in G_C.
1  begin
2      Compute shortest path for all pairs u, v ∈ V
3      for each R ⊆ V\τ such that |R| ≤ |τ| − 2 do
4          Construct − G_Rτ
5          LT' = Generate_Minimum_Spanning_Trees(G_Rτ) ([42])
6          for each tree T' ∈ LT' do
7              T ← 0
8              for each edge e_uv ∈ E(T_τ) do
9                  if node u and node v are not connected in T then
10                     P = shortest_path(e_uv) \\ compute in line 1
11                     if P contains less than two vertices in T then
12                         Add P to T
13                     else
14                         Add to T all edges in P which are not already in the tree T, and do not create a
                           cycle in T
15                 if T is connected and includes all terminals then
16                     break
17             bool replaced = true
18             for each element T_i in L do
19                 if V_T isSubsetOf(V_Ti) then
20                     Remove T_i from L
21                 else if replaced==true then
22                     if V_Ti isSubsetOf(V_T) then
23                         replaced=false
24             if replaced==true then
25                 Add T_i into L
26     return L
```

After listing all the Steiner Minimal Trees (all the trees which have the same minimum number of required nodes), N-OPT algorithm will find for each SMT an optimal path in the mobility graph which visits all the nodes in the tree, and then selects the SMT tree which has the least path cost. For each SMT tree T, it will find a metric closure graph for all vertices V_T in T in the mobility graph G_M, which is a complete graph over the nodes, where each edge has a weight equal to the shortest path between the two endpoints in the original graph. We obtain the metric closure graph by repeated application of Dijkstra's algorithm. Then the algorithm applies Branch and Bound search (Algorithm 3) for finding a tour visiting all nodes V_T in the graph.

Algorithm 3. Branch and Bound search

```
Data: A graph G = (V, E), and cost function w
Result: A tour P visiting all node in V with minimum total edge cost.
1  begin
2      Find LB (lower bound) and initialise r
3      //r is the root of the search tree Add r into a set S, S = {r}
4      initialise bestSol variable and bestCost=∞
5      while S is not empty do
6          Select a node n in S such that LB(n)≤ LB(p) ∀p ∈ S
7          Extract n from S
8          if n is a tour then
9              if cost(n) < bestCost then
10                 bestCost = cost(n)
11                 bestSol = n
12         else
13             if LB(n) < bestCost then
14                 create leftChild n_1 and calculate LB(n_1)
15                 create rightChild n_2 and calculate LB(n_2)
16                 if LB(n_1) < bestCost then
17                     Add n_1 into S
18                 if LB(n_2) < bestCost then
19                     Add n_2 into S
20     Return P=bestSol
```

The Branch and Bound search is based on a principle that if the best solution found so far is less than the lower bound (LB) for a subset, we do not need to explore this subset. However, if this subset has a lower bound which is less than the best cost so far, we will need to explore the subset as it might contain a better solution. To find a lower bound for a tour for visiting all nodes V in a graph $G = (V, E)$, we note that the cost of any tour is $\geqslant 1/2\sum_{v\in V}$ (Sum of the costs of the two least cost edges adjacent to v). Thus a lower bound on the cost of any tour is $\geqslant 1/2\sum_{v\in V}$ (Sum of the costs of the two least cost edges adjacent to v). Now we want a lower bound on the cost of a subset of tours. Note that a tour will be defined by a set of edges that must be in the tour and a set of edges that may not be in the tour. These constraints allow us to branch from a node (a search point) two possible branches: first branch (leftChild) contains the next unvisited edge and the second branch (rightChild) must not contain the edge. From these, we have different choices of the two lowest cost edges at each node (search point). Each time it branches, by creating two children of a node (leftChild and rightChild), it also checks which edges must be included or excluded by two rules: (i) an edge must be included if excluding this edge would make it impossible for a vertex to have two edges adjacent to it in the tour, and (ii) an edge must be excluded if it would cause any vertex to have more than two edges adjacent to it in the tour or create a cycle in the partial solution (a solution is not a tour but has a cycle). When branching the two children, however, the algorithm only adds into the search tree the children which have a lower bound is less than the best cost so far. The algorithm will terminate when the search tree is empty. This algorithm is based on [40], and the time complexity of Branch and Bound in the worst case is as high as that of exhaustive search. However, in real-life test cases it proved to speed up the search considerably because the Branch and Bound algorithm can be used to help us prune the search tree.

5. Optimal Path Algorithm (P-OPT)

This approach finds the optimal path that reconnects the network and then finds the optimal node cost for that path. The algorithm will list all the Steiner Trees (STs) (instead of all Steiner Minimal Trees as above) which connects terminals in the connectivity graph and then performs Branch and Bound search on the mobility graph for each Steiner tree to find the optimal path. P-OPT will produce the optimal path which also connects the terminals. The pseudo code of P-OPT is in Algorithm 4.

Algorithm 4. P-OPT Algorithm

```
Data: A graph G_C = (V, E_C), a graph G_M = (V, E_M),
      a set of terminals τ ⊂ V.
Result: a set of required nodes N for τ in G_C and
        an optimal path P visiting all nodes in N in G_M
1  begin
2    L = ST-generate(G_C, τ) (Algorithm 5)
3    mincost = ∞
4    for each ST tree T in L do
5      G_T(V_T, E_T) = metric_closure(V_T, G_C)
6      Path = BranchAndBound-search(G_T, V_T) (Algorithm 3)
7      if cost(Path) < mincost then
8        mincost = cost(Path)
9        N = V_T, all nodes in T
10       P = Path
11     if cost(Path) = mincost then
12       N' = V_T, all nodes in T
13       if cost(N') < cost(N) then
14         N = N'
15         P = Path
16   return < N, P >
```

Because the original algorithm in [41] only lists all Steiner Minimal Trees for a graph, we modified this algorithm to generate all Steiner trees for that graph (ST-Generate Algorithm 5). The algorithm firstly constructs the set $R \subseteq V \setminus \tau$ such that $|R| \leqslant |\tau| - 2$. It then finds all spanning trees in the graph $G_{R\tau}$, using the Algorithm in [42]. We then consider each edge in that tree in turn and, if the end points are not yet connected in the new tree, select the corresponding shortest path from the original graph. If this shortest path contains no more than 1 vertex already added, we add all edges into the tree, with their required vertices; if it contains more than 1 vertex already added, then we add those edges that are not already in the graph, and their associated vertices if needed. Because we only consider the set of Steiner nodes to re-connect the terminals, we will remove those Steiner trees which output the same Steiner nodes or output a tree which is a super tree of any current tree found so far.

As discussed above, the only difference between Algorithms 2 and 5 is that the former lists all minimum spanning trees for graph $G_{R\tau}$ while the later generates all spanning trees for that graph. In the worst case, where all the spanning trees have the same weight, then listing all minimum spanning trees is the same as listing all spanning trees. Therefore, the time complexity of Algorithm 5 is equal to that of Algorithm 2.

Algorithm 5. ST-Generate Algorithm

```
Data: A graph G_C = (V, E_C), a set of terminals τ ⊂ V.
Result: A list L of Steiner Tree T = (V', E'_C) for τ in G_C.
1  begin
2    Compute shortest path for all pairs u, v ∈ V
3    for each R ⊆ V\τ such that |R| ≤ |τ| − 2 do
4      Construct − G_Rτ
5      LT' = Generate_Spanning_Trees(G_Rτ) ([42])
6      for each tree T' ∈ LT' do
7        T ← 0
8        for each edge e_uv ∈ E(T_τ) do
9          if node u and node v are not connected in T then
10           P = shortest_path(e_uv) \\ compute in line 1
11           if P contains less than two vertices in T then
12             Add P to T
13           else
14             Add to T all edges in P which are not already in the tree T, and do not create
               a cycle in T
15         if T is connected and includes all terminals then
16           break
17       bool replaced = true
18       for each element T_i in L do
19         if V_T isSubsetOf(V_Ti) then
20           Remove T_i from L
21         else if replaced==true then
22           if V_Ti isSubsetOf(V_T) then
23             replaced=false
24       if replaced==true then
25         Add T_i into L
26   return L
```

Each Steiner tree produces a set of required nodes to reconnect the terminals. The algorithm then applies a Branch and Bound algorithm (Algorithm 3) to find the optimal path visiting those nodes in order to populate new radio nodes.

6. The Pareto set

The Network Repair Problem has two objective functions to be optimised simultaneously. For a multi-objective problem, the Pareto optimal solutions are those solutions for which there is no other feasible solution that is better in all objectives. A solution s_1 dominates a solution s_2 if and only if s_1 is no worse than s_2 in all objectives, and s_1 is strictly better than s_2 in at least one objective. In order to assess the quality of the solutions developed by the previous algorithms, we now develop a simple algorithm for finding the Pareto set for the multi-objective network repair problem.

To find the Pareto set, we find all alternative solutions for the problem, and then remove any dominated solutions. First, we find all the Steiner trees for connecting all terminals (Algorithm 5). Then, we find the optimal path for each Steiner tree (Algorithm 3). We store all the alternative solutions with node cost and optimal path cost for those nodes in each solution $T_i = \langle N_i, P_i \rangle$, i.e. a solution which needs N_i extra nodes and costs P_i unit in mobility.

Obviously, both the cheapest path and cheapest node solutions always belong to the Pareto set, and in fact, they are its endpoints. We then find the other solutions (if any) on the Pareto set using Algorithm 6. We first sort the alternative solutions according to one of the objectives (in this case, the node cost). The algorithm starts with the cheapest node cost solution and skips successive solutions in order of increasing node cost until it finds one with a cheaper path cost. This solution is then added to the frontier and the search is restarted from it.

Algorithm 6. Finding Pareto Set Algorithm

```
Data: List of T_1 =< N_1, P_1 >,
      T_2 =< N_2, P_2 >, ..., T_m =< N_m, P_m > in increasing order of node costs
      where ties are broken by order of path costs.
Result: A Pareto Frontier F.
begin
    i = 1
    while true do
        Add T_i to F
        Find smallest j > i such that P_j < P_i
        if there is no such j then
            break;
        i = j
```

Although N-OPT, P-OPT and the Pareto algorithm all generate optimal solutions, they will have high runtimes, and so are not expected to scale to larger problems. Therefore, we propose two heuristic algorithms, Shortest Cheapest Path and Integrated Path, which prioritise different objectives.

7. The Shortest Cheapest Path (SCP) algorithm

Our heuristic assigns an ordering to the objectives. The SCP (Algorithm 7) first tries to minimise the number of nodes required to connect all terminals. Given a minimal set of nodes, we then try to find the cheapest circuit for the agent that visits all of those nodes.

Algorithm 7. Shortest Cheapest Path algorithm

```
Data: A graph G_C = (V, E_C), graph G_M = (V, E_M),
      an edge cost function c for G_M, a set of terminals τ ⊂ V.
Result: Number of nodes placed and a tour in G_M visiting those nodes.
begin
    T = (V', E') = Steiner_MST(G_C, τ);
    P = Greedy_TSP(V', G_M);
    return [V',P] ;
```

Algorithm 8. Steiner-MST Algorithm

```
Data: A graph G_C = (V, E_C), a set of terminals τ ⊂ V.
Result: A Steiner Tree T = (V', E'_C) for τ in G_C.
begin
    G_τ(τ, E_τ, w_τ) = metric_closure(τ, G_C)
    T_τ = Minimum_Spanning_Tree(G_τ)
    T ← 0
    for each edge e_uv ∈ E(T_τ) do
        if node u and node v are not connected in T then
            P = shortest_path(e_uv)
            if P contains less than two vertices in T then
                Add P to T
            else
                Add to T all edges in P which are not already in the tree T, and do not create a cycle
                in T
            if T is connected and includes all terminals then
                break;
    return T
```

Since the Minimum Steiner Tree in Graphs problem is NP-Hard [3], we use a heuristic algorithm, adapted from [43] (Algorithm 8). First, we create a metric closure graph for the terminals, which is a complete graph over the terminal nodes, where each edge has a weight equal to the shortest path between the two endpoints in the original graph. We obtain the metric closure graph by repeated application of Dijkstra's algorithm [44]. We then find a minimum spanning tree in that graph, using Kruskal's algorithm [44]. We then consider each edge in that tree in turn and, if the end points are not yet connected in the new tree, select the corresponding shortest path from the original graph. If this shortest path contains no more than 1 vertex already added, we add all edges into the tree, with their required vertices; if it contains more than 1 vertex already added, then we add those edges that are not already in the graph, and their associated vertices if needed.

The complexity of Algorithm 8 is based on three components. The first component is the complexity of calculating the metric closure graph for τ which we repeatedly applied Dijkstra's algorithm (which has complexity of $O(|V|^2)$ [44]) for each terminal in τ. The second is the complexity of finding a minimum spanning tree in the graph $G_\tau = (\tau, E_\tau)$ (which we use Kruskal algorithm which yields $O(E_\tau \log(\tau))$ in complexity [44]). Finally, the third one is the complexity of replacing each edge in $E(T_\tau)$. Therefore, it is $O(|\tau||V|^2) + O(E_\tau \log(\tau)) + O(E(T_\tau)) = O(|\tau||V|^2)$.

Algorithm 9. Greedy-TSP Algorithm

```
Data: A graph G = (V, E).
Result: A tour visiting all nodes in V.
begin
    sort the edges in the increasing order of costs
    r = ∅ //empty set
    while r is not a tour do
        n = first(E) //get the first element in E
        if n does not cause a vertex to have degree three or more AND does not form a cycle unless it
        completes the tour then
            add n into r
    return r
```

For the problem of finding the shortest circuit (Algorithm 10), we take all the nodes in the Steiner tree created above, and then for those vertices create a metric closure graph from the mobility graph G_M. We then apply [40]'s Greedy-TSP heuristic (Algorithm 9) which is based on Kruskal's algorithm – we sort the edges in increasing order of cost, and we then iteratively add the lowest cost edge which does not increase any vertex's degree to 3, and which does not create a cycle unless it completes the tour.

The complexity of Algorithm 10 is that of calculating the metric closure graph for V' in G_M which we repeatedly applied Dijkstra's algorithm (which has complexity of $O(|V|^2)$) for each node in V', plus the complexity of Greedy-TSP (Algorithm 9). Therefore, it is $O(|V'||V|^2) + O(|V'|^2 \log|V'|) = O(|V'||V|^2)$.

Algorithm 10. GreedyTour

```
Data: A set of vertices V', a graph G_M = (V, E_M, c)
Result: a tour in G_M visiting all nodes in V'
begin
    G'' = (V'', E'', w'') = metric_closure(V', G_M)
    P = Greedy_TSP(G'')
    return [P]
```

Fig. 2 shows the SCP algorithm being applied to the example of Fig. 1. First we create the metric closure on the terminal nodes, then we create a spanning tree from that (Fig. 2(a)). We then extract the corresponding Steiner tree from the connectivity graph (b). We extract

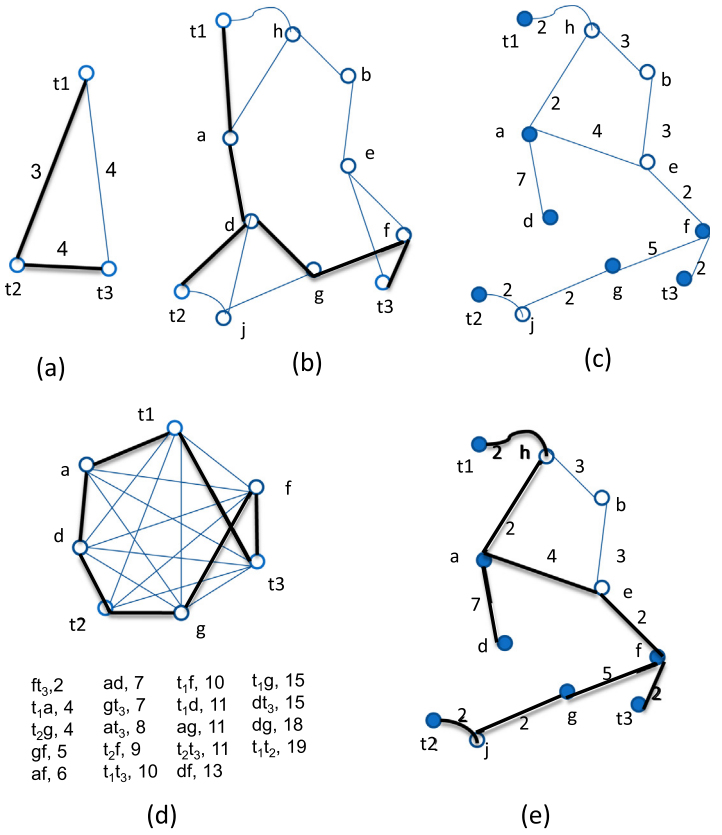


Fig. 2. A sample execution of Shortest Cheapest Path.

the new locations (c), construct the metric closure for those vertices (d), and then create the tour (e). This requires 7 new nodes in total, with a mobility cost of 56 units.

8. The Integrated Path IP algorithm

The SCP algorithm prioritises the number of nodes over the mobility cost. The IP approach attempts to combine the two objectives, adding approximate mobility costs into the connectivity graph, and then searching for a minimal Steiner tree on this modified graph (Algorithm 11). First, for each edge in the connectivity graph we compute the shortest path between the vertices in the mobility graph using Dijkstra's algorithm, and add that as an edge cost to the connectivity graph. We then apply the SCP algorithm on this new problem, with the difference being in the step where we create the Steiner tree from the minimum spanning tree of the metric closure, since the edges in the connectivity graph now have non-uniform costs.

The complexity of Algorithm 11 is based on three components. The first component is the complexity of calculating the weighted connectivity graph for V which we repeatedly applied Dijkstra's algorithm (which has complexity of $O(|V|^2)$) for each node in V. The second is the complexity of Steiner-MST (Algorithm 8). The third is the complexity of Greedy-TSP (Algorithm 10). Therefore, it is $O(|V|^3) + O(|\tau||V|^2) + O(|V'|^2 \log V') = O(|V|^3)$.

Algorithm 11. Integrated Path Algorithm

Data: A graph $G_C = (V, E_C)$, graph $G_M = (V, E_M)$,
an edge cost function c for G_M, a set of terminals $\tau \subset V$.
Result: Number of nodes placed and a tour in G_M visiting those nodes.
1 **begin**
2 $G_C* = (V, E_C, w*) = weighted\_connectivity\_graph(G_C, G_M)$
3 $T = (V', E') = Steiner\_MST(G_C*, \tau)(Algorithm 8)$
4 $P = Greedy\_TSP(V', G_M)$ (Algorithm 9)
5 return $[V', P]$

Fig. 3 shows the IP algorithm being applied to the example of Fig. 1. First we create the weighted connectivity graph (Fig. 3(a)), then metric closure on the terminal nodes, followed by its spanning tree from (b). We then extract the corresponding Steiner tree from the weighted connectivity graph (c). We extract the new locations (d), and then create the tour (e). This requires 9 new nodes in total, with a mobility cost of 42 units.

9. Adapted DORMS

For the purpose of comparison, we select the algorithm in [24] (discussed in Section 2.1) which appears closest to

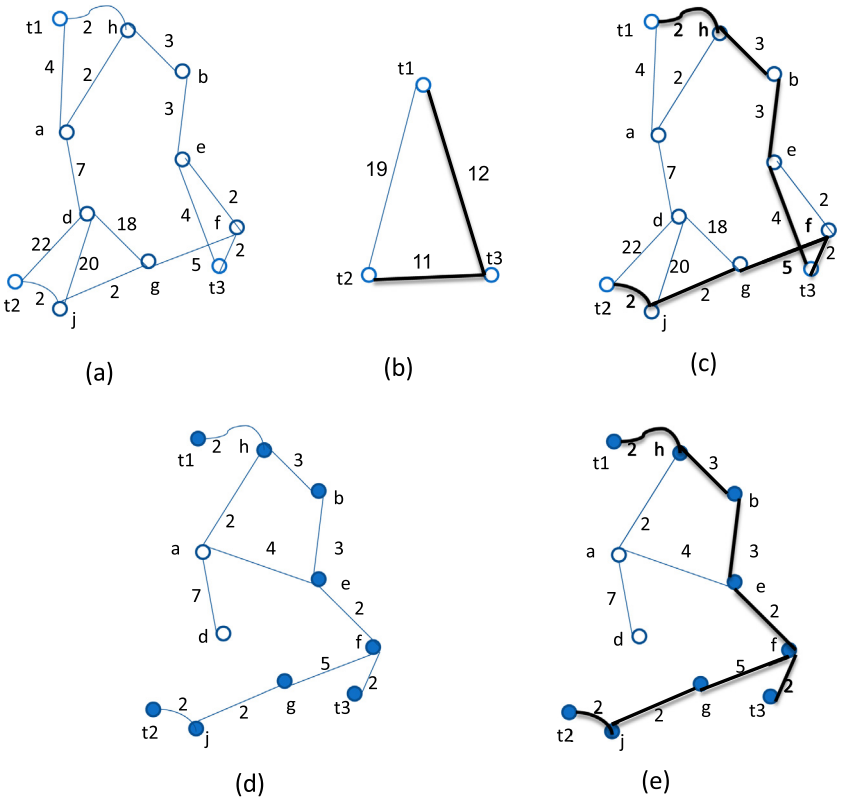


Fig. 3. A sample execution of Integrated Path.

our approach, and which has a smaller number of deployed nodes compared to other similar algorithms. This algorithm, called DORMS, will re-connect each partition toward a centre point and then seeks to optimise the total additional nodes that are needed. However, DORMS assumes free space for mobility, and so the mobility paths are simply straight lines. In our adaptation, we select a location which is closest to the centre of the area. We use A* search [45] to find the shortest path from each terminal toward the centre location in the connectivity graph, i.e. A* uses a best-first search and finds a least-cost path from a start node to a goal node. Then for each pair of adjacent terminals, we find a graph which contains all nodes and edges in the connectivity graph which are in the smallest area which is bounded by the two connectivity paths from the terminals toward the centre location. Then we find a Steiner minimal tree (Algorithm 8) in that graph which spans the two terminals and the centre location. After finding all Steiner minimal trees for all pair of adjacent terminals with the centre location, each terminal is now considered in two separate Steiner minimal trees formed by its neighbouring terminals, the algorithm chooses the best Steiner minimal trees among those trees to reconnect all the terminals. After finding the set of required nodes, the algorithm applies the Greedy-TSP in Algorithm 10 to find the path to visit those locations. The adapted DORMS (DORMS-AD) algorithm is given in Algorithm 12.

Algorithm 12. The Adapted Dorms Algorithm

Data: A graph $G_C = (V, E_C)$, a set of terminals $\tau \subset V$.
Result: A list L of required nodes.
begin
  centre = Find_Centre_Location()
  for *each* $t \in \tau$ **do**
    $p_t = Find\_Shortest\_Connectivity\_Path(t, centre, G_C)$
  for *each* $t \in \tau$ **do**
    $ta = Find\_adjacent\_terminal(t)$
    $G' = Find\_bounded\_graph(p_t, p_{ta}, centre, G_C)$
    $T_t = Steiner_Minimal_Tree(G', t, ta, centre)$
  while *any* $t \in \tau$ *is not in* L **do**
    $T = Find\_Smallest\_tree\_for\_t()$
    Add T into L

10. Evaluation

Our focus is on evaluating repair algorithms rather than network protocols, so we use a custom Java simulation. We evaluate the above algorithms on randomly generated problems, to compare the quality of their solutions on both objectives, and on their runtimes. To generate the problems, we create graphs from an underlying spatial grid, to represent a physical area for the deployment. The wireless connectivity graph is based on the distance between points in the grid, but modified by randomly placed obstacles. Similarly, the mobility graph is computed from grid

locations and the presence of obstacles. Note that in these experiments we are not evaluating the operation of the network, and so we do not use a network simulator.

We generate graphs within a rectangular area consisting of $n \times m$ squares, each of size 10 units squared. Within this space, we place o mobility obstacles, where each obstacle is a random polygon contained within a randomly selected pair of neighbouring cells. For each square, we generate a random position within it; if that position is inside an obstacle, we discard it, otherwise we designate it as a candidate location. Each obstacle is given a random weight w between 0 and 1, representing the difficulty it creates for the agent to traverse it, and such that any obstacle with a weight greater than 0.2 is assumed to be not able to be traversed.

We then create the connectivity graph by adding edges indicating that two candidate locations are within wireless transmission range. For each pair of locations, if they are within 10 units apart, we add an edge with probability 0.85; if they are between 10 and 20 units apart, we add an edge with probability 0.2. This is to simulate the RF propagation caused by the multi-path effects. For the mobility graph, we add an edge between any pair of locations which are less than 45 units apart and which can be connected by a straight line that does not cross an obstacle. The weight of the edge is simply the length of the connecting line. For any pair of locations separated by a distance of less than 45 and which has a straight line that traverses all obstacle with a weight less than or equal to 0.2, we add those edges into the mobility graph, the cost of the edge is the distance plus 10∗weight for each obstacle it crosses (to add the difficulty when traversing the obstacles). The grid is only used to create an underlying structure for the abstract graphs, allowing us to generate connectivity links and mobility costs based on geometric properties. In general, we can evaluate our algorithms in any graphs. We have tried different values and different grid sizes, etc. in our evaluation, and we received the same relative results for our algorithms in node cost, mobility cost and runtimes.

10.1. A glance at Pareto optimality

First, we want to compare the relative quality of the solutions produced by the different algorithms. We do this by running all algorithms and comparing to the Pareto set. As the runtime of Pareto search (Section 6) explodes, even for a small problem size, here we only plot three random instances of the experiments with small problem settings. Figs. 4–6 show all solutions of minimal Steiner sets associated with their minimal paths, the Pareto set and the results of N-OPT, P-OPT, SCP, IP algorithms for selected instances with 4, 5, 6 terminals (T) to be connected.

The figures show that there is a wide range of solution costs, and there is clearly a trade-off to be managed between node cost and mobility cost. We only plot the results of the minimum paths for the minimal Steiner sets, and yet in each case node cost varies by up to 50%, and mobility cost varies by 40–70%. We see that in all three cases both SCP and IP are within 1 node of the optical node cost (shown by N-OPT). However, their mobility costs range from 20% to 50% higher than the optimal cost (P-OPT), with IP consistently better than SCP, although not by a large margin.

10.2. More experiments

For the rest of the experiments, we consider two different problem sizes: (i) a 5×10 grid, and thus a maximum of 50 candidate locations, and 10 obstacles, and (ii) a 10×10

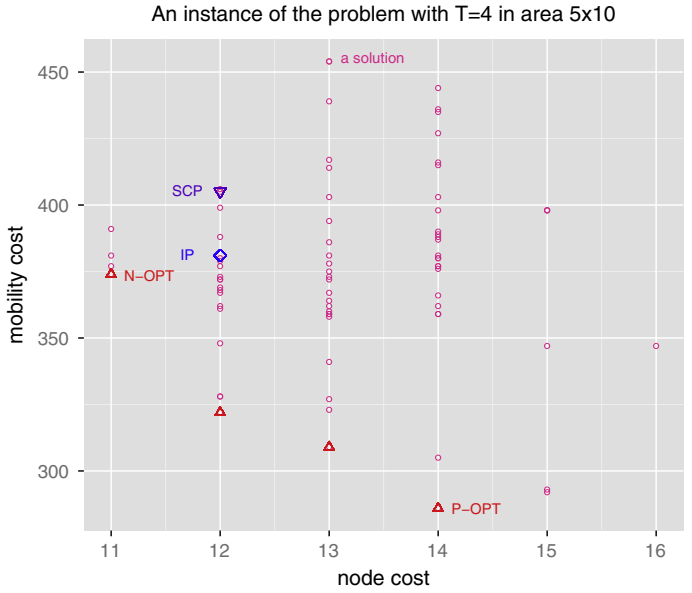


Fig. 4. An instance of the problem with $T = 4$ (number of terminals to be connected) in area 5×10. All solutions are displayed in the figure. The points which belong to the Pareto set are denoted as red triangles. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

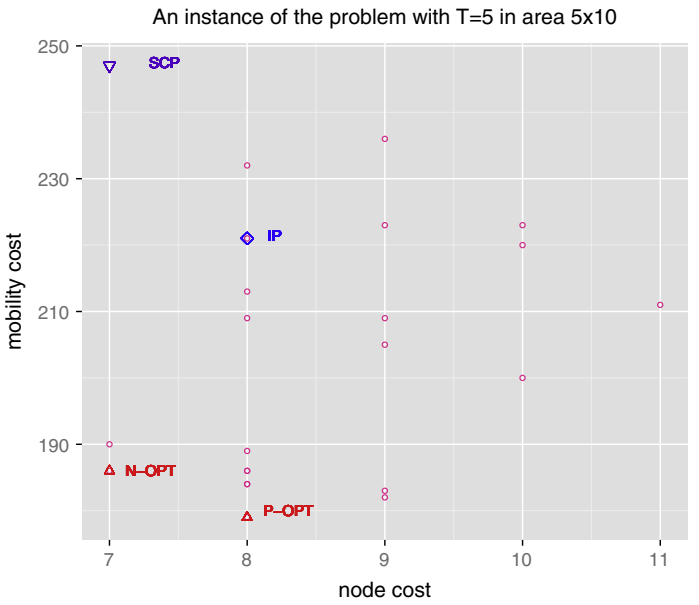


Fig. 5. An instance of the problem with $T = 5$ (number of terminals to be connected) in area 5×10. All solutions are displayed in the figure. The points which belong to the Pareto set are denoted as red triangles. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

grid, and thus a maximum of 100 candidate locations, and 20 obstacles. For each of the two problem sizes, for each data point, we generate 50 instances, and present the average solution cost (mobility cost, number of nodes needed) and runtime. For each instance, we randomly select candidate locations as terminals.

We compare the proposed algorithms including the adapted DORMS. As the network repair includes both the node cost (defined by the number of nodes placed) and the mobility cost (or the cost of moving for the agent to place new nodes at required locations), we are interested in the relative performance of the proposed algorithms in

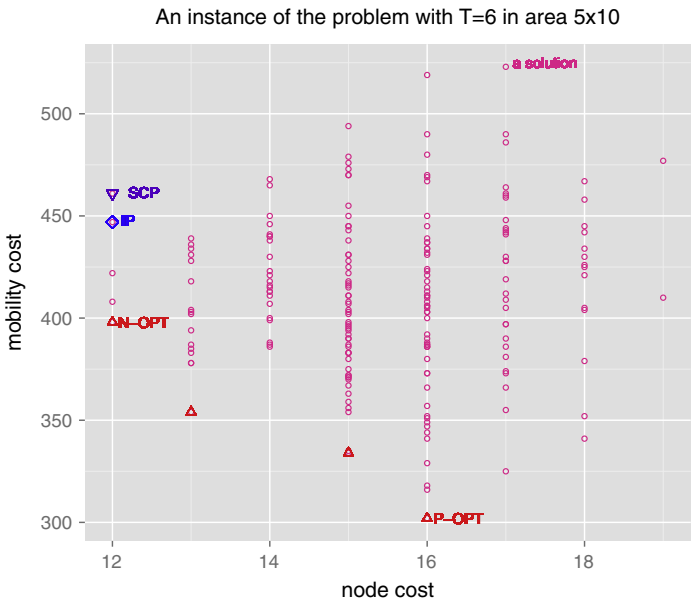


Fig. 6. An instance of the problem with $T = 6$ (number of terminals to be connected) in area 5×10. All solutions are displayed in the figure. The points which belong to the Pareto set are denoted as red triangles. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

terms of node cost and mobility cost. We also evaluate their runtime (computation time), and the total time to restore connectivity for the network.

10.2.1. Node cost

Fig. 7 shows the number of nodes placed by each algorithm, as we vary the number of terminals from 4 to 7. As the number of terminals to be connected increases, the number of nodes required to connect the terminals by all algorithms also rises. The number of nodes required by DORMS-AD is noticeably higher than the others because DORMS-AD connects all terminals to a centre point which could require more nodes compared to the SCP and IP which calculate a minimal Steiner tree. Note that the IP algorithm finds a MST on the weighted connectivity graph while the SCP finds a MST on the original connectivity graph. Therefore, the IP algorithm requires more nodes than the SCP algorithm, although on average no more than 1.5 extra nodes. The SCP requires no more than one extra node compared to the N-OPT algorithm. Obviously, the node costs of all algorithms increase with the problem size (from 5×10 to 10×10), while the relative costs between the different algorithm remains the same. We note that the complete P-OPT algorithm, which prioritises mobility cost, is competitive on node cost with the heuristic SCP algorithm, which prioritises node cost.

10.2.2. Mobility cost

Fig. 8 shows the mobility costs as we vary the number of terminals from 4 to 7. Again, the costs rise as we increase the number of terminals. DORMS-AD is again noticeably poorer than the other algorithms with the same reason above. SCP is between 6% and 9% poorer than IP, which in turn is between 10% and 15% poorer than P-OPT. This is because the SCP has the node cost in higher priority than the path cost while the IP intends to find a better path to connect the terminals. We note that the complete N-OPT, which prioritises node cost, consistently produces mobility paths that are shorter than the heuristic algorithm IP which attempts to balance node and mobility costs.

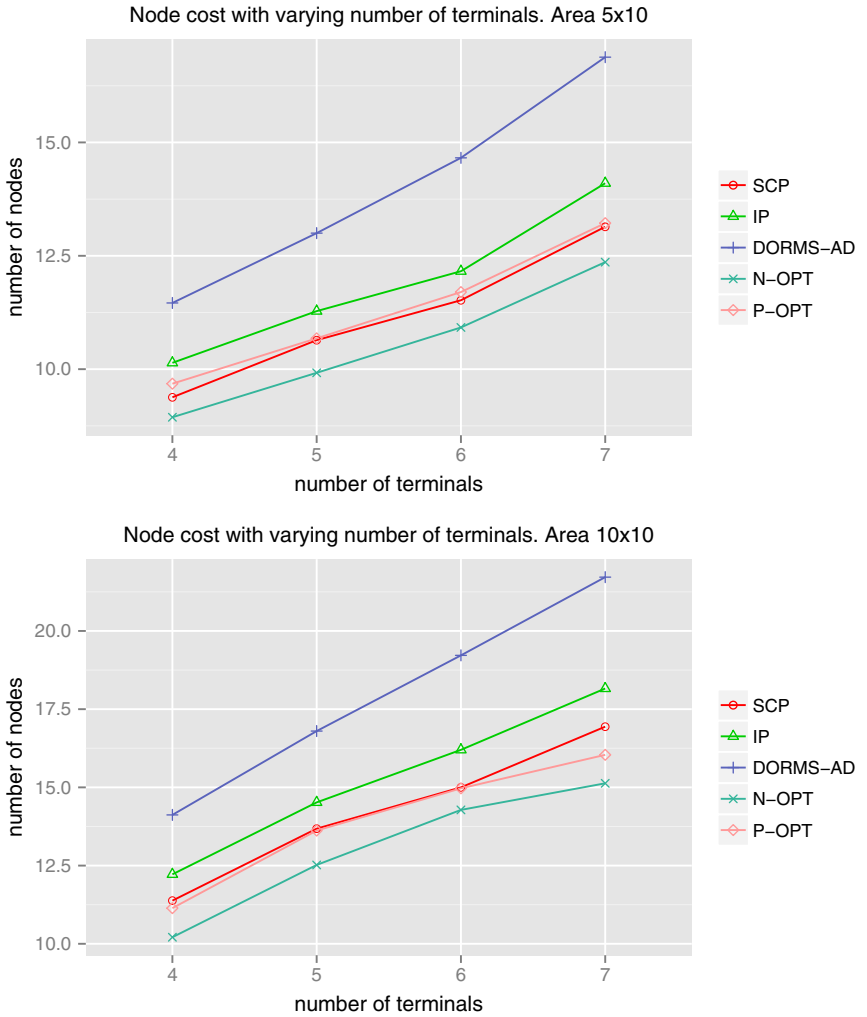


Fig. 7. Node cost with varying number of terminals.

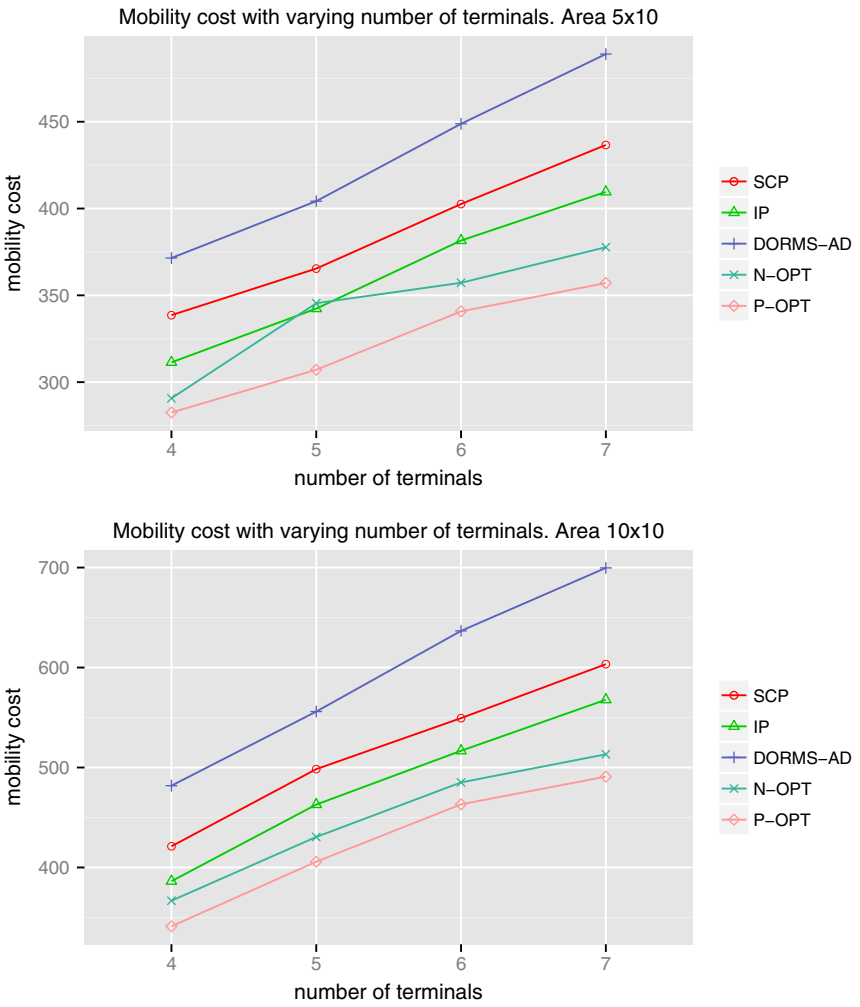


Fig. 8. Mobility cost with varying number of terminals.

10.2.3. Runtime

Table 2 shows the runtime for the three heuristics. N-OPT and P-OPT were designed to examine the extremes of the multi-objective problem, and were not expected to be competitive in runtime, and so are not included in the table. For example, on average for 4 terminals, N-OPT requires 915.35 s and P-OPT takes 7252.37 s to run an instance. P-OPT requires lots of effort to find the optimal path that can reconnect the terminals as it searches over the mobility graph for the cheapest path for all Steiner trees found in the connectivity graph to reconnect the terminals while the N-OPT only needs to do the search over the mobility graph for the smallest Steiner trees connecting the terminals in the connectivity graph. The SCP and DORMS-AD algorithms are significantly faster compared to IP algorithm, with a speed-up factor between 10 and 23. As shown in Section 8, the runtime of IP is dominated by the computation time of the weighted connectivity graph which depends on the network size (size of connectivity graph and mobility graph).

Table 2
Runtime (s) with varied number of terminals.

	4	5	6	7
(a) Area 5 × 10				
SCP	0.08	0.10	0.11	0.13
DORMS-AD	0.10	0.12	0.14	0.18
IP	1.19	1.22	1.21	1.36
(b) Area 10 × 10				
SCP	0.80	0.99	1.10	1.27
DORMS-AD	1.00	1.23	1.44	1.66
IP	18.45	19.05	18.42	19.32

10.2.4. Density of the possible candidate locations in the network

We note that the number of terminals for a fixed size problem does not have a significant impact on the runtime, but that there is a significant difference as we increase the network size. Therefore, we perform another experiment in which we vary the density of the candidate locations. More specifically, we vary the maximum number of candidate locations able to be placed in a single grid square from

Table 3
Runtime (s) with varied density and the number of terminals.

Runtime (s)	SCP	IP	DORMS-AD
Location density = 1, $T = 4$	0.03	0.30	0.03
Location density = 1, $T = 5$	0.03	0.31	0.03
Location density = 1, $T = 6$	0.04	0.32	0.04
Location density = 1, $T = 7$	0.05	0.34	0.05
Location density = 2, $T = 4$	0.09	3.65	0.11
Location density = 2, $T = 5$	0.11	3.58	0.13
Location density = 2, $T = 6$	0.12	3.71	0.14
Location density = 2, $T = 7$	0.15	3.72	0.18
Location density = 3, $T = 4$	1.32	132.36	1.73
Location density = 3, $T = 5$	1.70	137.04	2.16
Location density = 3, $T = 6$	1.92	136.73	2.54
Location density = 3, $T = 7$	2.21	131.11	2.89
Location density = 4, $T = 4$	6.49	1180.93	8.91
Location density = 4, $T = 5$	7.29	1124.07	10.39
Location density = 4, $T = 6$	9.41	1130.51	12.79
Location density = 4, $T = 7$	10.81	1109.95	14.34

1 to 4. During generation, for each square, we randomly select the number of candidate locations to be placed, and then select their positions. As we increase this parameter, the number of edges per location increases significantly, and particularly so for the mobility graph. The results are shown in Table 3. For SCP, the runtime increases highly with the density parameter. For IP, the runtime appears to grow dramatically. Note that the density parameter relates to the density of the locations packed into the same geographic area, and so a higher value involves more nodes as well has a higher average degree for each node in the graph. SCP complexity is the product of the square of the number of candidate locations and the size of the Steiner tree. As more locations are added, the density increases, and we expect the Steiner tree to grow sub-linearly. However, the complexity of the IP algorithm is the cube of the number of nodes, and so increasing the density parameter should have a more significant effect, as indicated by the increasing runtime.

In Fig. 9, we examine the results of density parameter = 4 in more detail. Although the IP algorithm saves about 20% of the mobility costs, it is two orders of magnitude slower in runtime (Table 3), and requires slightly more nodes on average compared to the SCP algorithm.

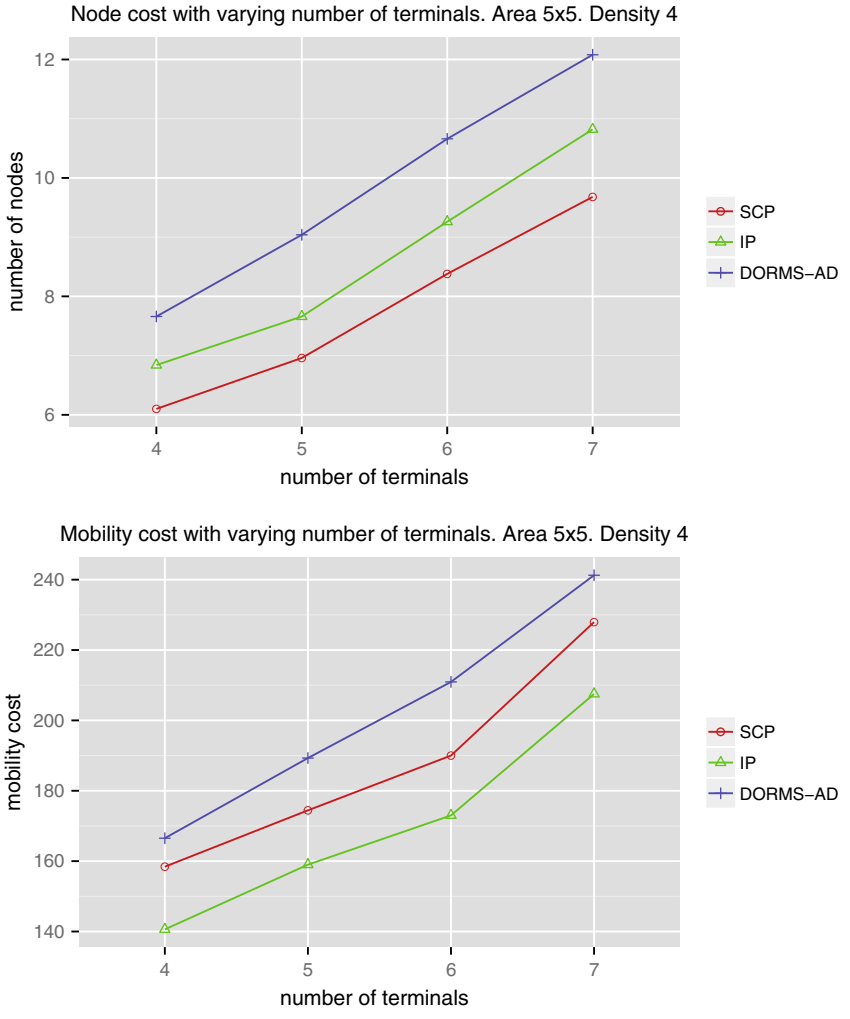


Fig. 9. Node and mobility costs with density 4, area 5×5.

10.2.5. Total restoration time

To decide which algorithm should be selected in practical applications will require greater knowledge of the relative cost of the new radio nodes versus the mobility costs. We should also take into account the runtime of the algorithm, since as the graphs grow in size, the runtime will become more significant: waiting for the algorithm to complete may remove any benefit gained from a shorter path. If we regard the mobility cost as the time to traverse the circuit, we can then consider the total time for restoring the network as being the runtime of the algorithm plus travelling time of the agent, plus any time required to place the nodes in position. We assume it takes the robot 30 s to position a new node and the distance for the grid squares is 10 m. We consider three scenarios, the first represents a small robot moving at the speed of 0.1 ms^{-1}, the second represents a human walking speed agent at 1.4 ms^{-1}, and the third represents a fast moving vehicle over a rough terrain at the speed of 4 ms^{-1}. The results are shown in Tables 4–6.

For the slow small robot, prioritising the mobility cost results in a faster restoration time for all parameter settings, as the mobility costs outweigh the time to place nodes and the increased runtime. For a human, the time to restore for the two heuristics is similar as the node cost, path cost and runtime are balanced. For the vehicle, prioritising the node cost becomes more important, since the reduction in mobility cost by the path-based algorithms has difficulty compensating for the increased runtime and the increased node-placement cost. Thus, the WSN restoration problem is subtle, with the choice of approach clearly dependent on the details of the specific problem. Solution methods must take into account the main objectives (minimising infrastructure and minimising time), but also consider the capabilities of the agent that will implement the eventual solution.

Table 4
Total restoring time (s) with a slow speed agent $v = 0.1$ ms^{-1}.

	4	5	6	7
(a) Area 5 × 10				
SCP	3667.28	3973.50	4371.71	4760.53
IP	3419.39	3763.02	4181.81	4519.96
DORMS-AD	4058.90	4432.52	4927.94	5396.18
(b) Area 10 × 10				
SCP	4554.00	5395.19	5945.10	6543.27
IP	4247.85	5084.05	5671.62	6242.13
DORMS-AD	5242.80	6064.83	6944.84	7650.07

Table 5
Total restoring time (s) with a medium speed agent $v = 1.4$ ms^{-1}.

	4	5	6	7
(a) Area 5 × 10				
SCP	523.33	580.31	633.28	706.20
IP	527.82	584.15	638.56	716.90
DORMS-AD	609.26	678.86	760.51	855.84
(b) Area 10 × 10				
SCP	643.04	767.37	843.53	940.46
IP	660.96	785.32	873.51	969.70
DORMS-AD	768.76	902.35	1032.81	1153.04

Table 6
Total restoring time (s) with a high speed agent $v = 4$ ms^{-1}.

	4	5	6	7
(a) Area 5 × 10				
SCP	366.13	410.65	446.36	503.49
IP	383.24	425.20	461.40	526.75
DORMS-AD	436.78	491.18	552.14	628.82
(b) Area 10 × 10				
SCP	447.50	535.98	588.45	660.32
IP	481.62	570.39	633.60	706.08
DORMS-AD	545.06	644.22	737.21	828.19

Table 7
IP or SCP.

Criteria	SCP	IP
Energy (movement)	No	Yes
Expensive nodes	Yes	No
Small network, low speed agent	No	Yes
Small network, medium speed agent	Yes	Yes
Small network, high speed agent	Yes	No
Large network, low speed agent	No	Yes
Large network, medium speed agent	Yes	Yes
Large network, high speed agent	Yes	No
Very dense deployment	Yes	No

Table 7 gives a summary of the conditions under which we would prefer one algorithm to the other. For high node costs, or fast moving agents, we expect to prefer the SCP algorithm, while for cases where energy costs are significant, or where agents are relatively slow, then the IP algorithm will be preferred. The optimal solutions can be used for small network or can be run offline for the expensive node or energy costs.

11. Summary

We have defined the new problem of simultaneous network connectivity restoration with constrained route planning, in the presence of obstacles, in a static observed problem. We formalise the problem as a multi-objective problem of minimising a Steiner tree in a connectivity graph and minimising a tour of the nodes in that tree in a mobility graph. We present two complete algorithms: Optimal Node Algorithm (N-OPT) first finds the optimal node cost, and then finds the optimal path for that node cost while Optimal Path Algorithm (P-OPT) finds the optimal path that reconnects the network and then finds the optimal node cost for that path. We then study an algorithm for finding the Pareto set, and present two heuristic algorithms, Shortest Cheapest Path (SCP) and Integrated Path (IP). Shortest Cheapest Path prioritises node cost, first optimises the number of nodes required to heal the network, then optimises the path which visits all the new node positions. Integrated Path combines the two objectives by adding weights into the connectivity graph to approximate the mobility cost of establishing each link, and then searches for the cheapest tree that connects all existing nodes. We conducted an empirical evaluation of the two algorithms on random connectivity and mobility

graphs. We compared our proposed heuristic algorithms with the optimal solutions and an adaptive existing algorithm and analyse the choice of the algorithms in each specific application. The SCP algorithm tends to find graphs with fewer nodes, while the IP algorithm finds slightly larger solutions but with cheaper mobility costs. The SCP algorithm is significantly faster, particularly on dense graphs. Both SCP and IP are close to N-OPT in node cost (approximately one extra node for SCP, and two extra nodes for IP compared to N-OPT), and higher in mobility costs than P-OPT (approximately between 16% and 22% for SCP, and between 10% and 13% for IP). In addition, SCP and IP are close to the Pareto frontier in node cost but quite far from the Pareto frontier in mobility cost. We also evaluate the total restoration time as a function of an agent's speed for the choice of heuristics. For fast moving agents, the SCP algorithm is faster in total restoration time, and for slow speed moving agent, the IP algorithm will be faster.

There are many areas for potential future work. The paper uses the graph-based model where the candidate locations are decided and fixed into a number of positions. Future work will address the real world of potential areas for node placement where continuous positions can be investigated. It would have been certainly desirable to further analyse the Pareto sets and validate the efficiency of each multi-objective approach under consideration with some well-known Pareto metrics. We should also develop a hierarchical routing approach, which will allow us to handle dense mobility graphs, by initially merging adjacent location nodes into a super-node to reduce the complexity. There is a need to develop distributed algorithms, allowing multiple agents and sensor nodes to collaborate to determine the damage to the network in large scale problems. The work in this paper should be extended to consider a continually changing network and environment. Future work can also implement the network repair scenario with a real robot.

Acknowledgments

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

References

[1] F.K. Hwang, D.S. Richards, P. Winter, The Steiner tree problem, in: Annals of Discrete Mathmatics, 1992.
[2] H.J. Promel, A. Steger, The Steiner tree problem. A tour through graphs, algorithms, and complexity, Adv. Lect. Math. (2002).
[3] G. Robins, A. Zelikovsky, Tighter bounds for graph Steiner tree approximation, J. Discr. Math. 19 (2005) 122–134.
[4] 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.
[5] 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.
[6] N. Atay, B. Burchan, Mobile wireless sensor network connectivity repair with k-redundancy, in: 8th International Workshop on the Algorithmic Foundations of Robotics, vol. 57, 2008, pp. 35–49.
[7] 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.
[8] L. Sitanayah, K.N. Brown, C.J. Sreenan, Planning the deployment of multiple sinks and relays in wireless sensor networks, J. Heurist. (2014) 1–36.
[9] 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.
[10] 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.
[11] K. Akkaya, S. Janapala, Maximizing connected coverage via controlled actor relocation in wireless sensor and actor networks, J. Comput. Telecommun. Network. 52 (14) (2008) 2779–2796.
[12] K. Akkaya, F. Senel, Detecting and connecting disjoint sub-networks in wireless sensor and actor networks, J. Ad Hoc Networks 7 (2009) 1330–1346.
[13] A. Zamanifar, M. Sharifi, O. Kashefi, Self actor–actor connectivity restoration in wireless sensor and actor networks, in: 1st Asian Conference on Intelligent Information and Database Systems (ACIIDS), 2009, pp. 442–447.
[14] N. Tamboli, M. Younis, Coverage-aware connectivity restoration in mobile sensor networks, in: IEEE International Conference on Communications, 2009, pp. 1–5.
[15] 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.
[16] M. Sir, I. Senturk, E. Sisikoglu, K. Akkaya, An optimization-based approach for connecting partitioned mobile sensor/actuator networks, in: 3rd International Workshop on Wireless Sensor, Actuator and Robot Networks (INFOCOM), 2011, pp. 525–530.
[17] A. Abbasi, M. Younis, U. Baroudi, Recovering from a node failure in wireless sensor–actor networks with minimal topology changes, IEEE Trans. Veh. Technol. 62 (2013) 256–271.
[18] 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.
[19] 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.
[20] D. Henkel, T. Brown, Delay-tolerant communication using mobile robotic helper nodes, in: 6th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks and Workshops (WiOPT), 2008, pp. 657–666.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[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] 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.
[27] R. Falcøn, 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.
[28] 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.
[29] R. Falcøn, 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.

[30] K. Magklara, D. Zorbas, T. Razafindralambo, Node discovery and replacement using mobile robot, in: Ad Hoc Networks – 4th International ICST Conference, 2012, pp. 59–71.
[31] 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.
[32] H. Li, A. Barnawi, I. Stojmenovic, C. Wang, Market-based sensor relocation by robot team in wireless sensor networks, Ad Hoc Sensor Wireless Networks 22 (3–4) (2014) 259–280.
[33] 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.
[34] 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.
[35] 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.
[36] 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. Robot. 23 (4) (2007) 661–675.
[37] M.M. Zavlanos, G.J. Pappas, Distributed connectivity control of mobile networks, IEEE Trans. Robot. 24 (6) (2008) 1416–1428.
[38] 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.
[39] M. Fischetti, J.-J. Salazar-Gonzalez, P. Toth, The generalized traveling salesman and orienteering problems, in: G. Gutin, A.P. Punnen (Eds.), The Traveling Salesman Problem and Its Variations, Combinatorial Optimization, vol. 12, Springer, 2004, pp. 609–662.
[40] E.L. Lawler, J. Lenstra, A. Rinnooy Kan, D. Shmoys, The Traveling Salesman Problem, John Wiley & Sons, 1985.
[41] M.C. Dourado, R.A. de Oliveira, F. Protti, Generating all the Steiner trees and computing Steiner intervals for a fixed number of terminals, Electron. Notes Discr. Math. 35 (2009) 323–328.
[42] K. Sorensen, G.K. Janssens, An algorithm to generate all spanning trees of a graph in order of increasing cost, J. Pesquisa Oper. 25 (2005) 219–229.
[43] B.Y. Wu, K.-M. Chao, Spanning Trees and Optimization Problems, Chapman & Hall/ CRC Press, USA, 2004.
[44] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., The MIT Press, 2001.
[45] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybernet. 4 (2) (1968) 100–107.

Thuy T. Truong received a Ph.D. degree in Computer Science from University College Cork in 2014, and MSc degree in Advanced Distributed System from the University of Leicester, UK in 2009, and a BSc degree in Computer Science and Engineering from Ho Chi Minh University of Technology, Vietnam in 2007. Currently, she is a post-doctoral researcher in Insight/MISL Lab, University College Cork under guidance of Dr. Kenneth N. Brown and Prof. Cormac Sreenan. Her research interests are in network embedded systems and wireless networks.

Kenneth N. Brown joined UCC Computer Science Department as a senior lecturer in 2003, where he is the UCC Deputy Director of Insight, the centre for data analytics, and the UCC Principal Investigator on CTVR, the telecommunications research centre. Prior to that 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.