Planning the Deployment of Multiple Sinks and Relays in Wireless Sensor Networks

Lanny Sitanayah · Kenneth N. Brown · Cormac J. Sreenan

the date of receipt and acceptance should be inserted later

Abstract Wireless sensor networks are subject to failures. Deployment planning should ensure that when a data sink or sensor node fails, the remaining network can still be connected, and so may require placing multiple sinks and relay nodes in addition to sensor nodes. For network performance requirements, there may also be path-length constraints for each sensor node. We propose four algorithms, Greedy-MSP and GRASP-MSP to solve the problem of multiple sink placement, and Greedy-MSRP and GRASP-MSRP for the problem of multiple sink and relay placement. Greedy-MSP and GRASP-MSP minimise the deployment cost, while ensuring that each sensor node in the network is double-covered, i.e. it has two length-constrained paths to two sinks. Greedy-MSRP and GRASP-MSRP deploys sinks and relays to minimise the deployment cost and to guarantee that all sensor nodes in the network are double-covered and noncritical. A sensor node is noncritical if upon its removal, all remaining sensor nodes still have length-constrained paths to sinks. We evaluate the algorithms empirically and show that these algorithms outperform the closely-related algorithms from the literature for the lowest total deployment cost.

Lanny Sitanayah
Mobile & Internet Systems Laboratory, School of Computer Science and IT,
University College Cork, Ireland
Tel.: +353-21-4205396
Fax: +353-21-4205367
E-mail: ls3@cs.ucc.ie

Kenneth N. Brown
Insight Centre for Data Analytics, School of Computer Science and IT,
University College Cork, Ireland
E-mail: k.brown@cs.ucc.ie

Cormac J. Sreenan
Mobile & Internet Systems Laboratory, School of Computer Science and IT,
University College Cork, Ireland
E-mail: cjs@cs.ucc.ie

Keywords Wireless Sensor Networks · Network Deployment Planning · Multiple Sink and Relay Placement

1 Introduction

A Wireless Sensor Network (WSN) is composed of a large number of sensor nodes [1]. Each sensor node is a battery-powered device with limited storage, processing and communication capability. It is able to sense a close-by physical phenomenon, perform a simple computation and send its data wirelessly over a multi-hop network to a special node called a data sink. Unlike sensor nodes which are typically resource-constrained because of a desire to keep them low-cost, small, energy-efficient and easy to deploy, a data sink usually has more energy, storage, processing and communication capabilities allowing it to act as a gateway between sensor nodes and an end-user. This network is subject to failure as the wireless devices and the communication links are unreliable. A sensor node may fail due to limited battery life or hardware malfunction, or may be damaged by weather or human intervention. When some sensor nodes fail, the network may be disconnected and thus it cannot gather information from the isolated area. Even though a sink has more resources than a sensor node, this electronic device may fail too.

To protect against network failure, it is important to plan the topology deployment. In this paper, we define a novel problem for increasing the WSN topology robustness against a single failure by deploying multiple sinks and relays with minimal cost. A network is robust against a single failure if after a failure of a sink, a sensor node or a relay node, each remaining sensor node can deliver its data to a sink through a multi-hop path with an acceptable length. We consider the path length restriction as data latency requirements may be important in WSN applications. To be robust to sink failure, it is necessary to deploy multiple sinks in the network such that each sensor node is *double-covered*, i.e. it has length-bounded paths to two sinks. While we restrict our assumption to k-covered, where $k=2$ in this paper, our solution is also applicable to any integer $k \geq 1$. If $k > 2$, the solution is robust to multiple (up to $k-1$) sink failures. To protect against sensor node failure, it is necessary to place some relay nodes, so that every sensor node is *non-critical*, i.e. when it fails, each remaining sensor node still has at least one length-bounded path to a sink. These relay nodes do not sense, but only forward data from other nodes. Installing sinks and relays comes at a cost that includes not only the hardware purchases, but also the installation and maintenance cost, thus motivating our solution to minimise the total deployment cost.

Our main contribution is a solution that minimises the total cost of sink and relay deployment but ensures the network robustness against a single device failure, either a sink or a sensor node. Our solutions use a greedy algorithm and a local search algorithm based on the Greedy Randomized Adaptive Search Procedure (GRASP) [8] [9] [20]. Firstly, we look at the *Multiple Sink Placement* (MSP) problem to minimise the number of *uncovered* nodes, i.e. nodes

that are not double-covered. We propose Greedy-MSP and GRASP-MSP and show empirically that Greedy-MSP has the shortest runtime but deploys more sinks than GRASP-MSP. On the other hand, our evaluation demonstrates that GRASP-MSP finds comparable solutions in shorter runtimes than a linear programming implementation. The GRASP-MSP performance gives us confidence in using it as the basis for solving the more complex multiple sink and relay placement problem.

After that, we look at the *Multiple Sink and Relay Placement* (MSRP) problem and present Greedy-MSRP and GRASP-MSRP. Both algorithms employ the concept of Length-constrained Connectivity and Rerouting Centrality (l-CRC) introduced in [22] to identify every *critical* node, i.e. a sensor node which if it fails can cause other nodes to lose their length-bounded paths to sinks. Greedy-MSRP deploys sinks and relays separately. It first uses Greedy-MSP to select sinks to minimise the number of uncovered nodes. It then trades sinks for relays to minimise the total deployment cost but ensures that the network is still double-covered and non-critical. To deploy relays, we develop a GRASP algorithm for Multiple Relay Placement (GRASP-MRP). GRASP-MRP extends the GRASP algorithm for Additional Backup Placement (GRASP-ABP) in [22] to make a network topology not only non-critical, but also double-covered.

Unlike Greedy-MSRP that deploys sinks and relays separately, GRASP-MSRP deploys sinks and relays simultaneously in its local search move to minimises the number of uncovered and critical nodes. We demonstrate empirically that GRASP-MSRP runs faster than Greedy-MSRP and the solutions produced by GRASP-MSRP are over 30% less costly than those of Greedy-MSRP. Finally, we evaluate the robustness of GRASP-MSRP's topologies using a network simulator and show that they can tolerate more faults gracefully.

The remainder of this paper is organised as follows. In Section 2 and 3, we describe the background of this work and review the related work on sink and relay deployment algorithms, respectively. We introduce Greedy-MSP and GRASP-MSP in Section 4 and evaluate them in Section 5. Greedy-MSRP and GRASP-MSRP are presented in Section 6 and are evaluated in Section 7. We then evaluate the resulting topologies for robustness, by simulating network operation while nodes are failing in Section 8. We show that the performance of a network is not only influenced by the number of deployed sinks, but more importantly the positions to deploy the sinks. Finally, Section 9 concludes the paper. Parts of this work were presented in [23].

2 Background

A WSN can be modeled as a graph $G=(V,E)$, where V is a set of nodes and E is a set of edges. Each edge connects two nodes that are within transmission range of each other[1], and the two nodes are said to be *adjacent*. A *path* of length

[1] For simplicity we assume bi-directional links, but this could be easily relaxed by specifying a more complex connectivity graph.

t between two nodes v and w is a sequence of nodes $v = v_0, v_1, \ldots, v_t = w$, such that v_i and v_{i+1} are adjacent for each i. A path from a node v to a set of nodes W is simply a path from v to any node $w \in W$. Two nodes are *connected* if there is a path between them. A graph is connected if every pair of nodes is connected. Sensor network topology is an undirected graph and for simplicity, we assume that the graph is connected. $H = (W, E{\downarrow_W})$ is an induced subgraph of $G = (V, E)$ if $W \subset V$ and $E{\downarrow_W} = \{(x, y) \in E;\ x \in W, y \in W\}$ is the subset of edges in E which connect nodes in W.

In a WSN with a data sink, the routing paths from all sensor nodes to the sink form a *rooted tree*, where the sink is the root of the tree. Any node w on a path from a node v to the root is an *ancestor* of v. If w is an ancestor of v, then v is a *descendant* of w. The *subtree rooted at* v is the tree induced by descendants of v rooted at v.

Centrality indices is a core concept in social network analysis [10] [5], used to determine the importance of a node in a network. It was introduced by Bavelas [3] in 1948. A sensor node is identified as a *critical* node if upon its removal, more sensor nodes will have no path of length $\leq l_{\max}$ to a sink, where $l_{\max}$ is the maximum acceptable path length. Otherwise, it is *non-critical*. We define a WSN as non-critical if each sensor $v \in T$ is non-critical. Using *Length-constrained Connectivity and Rerouting Centrality* (l-CRC), a sensor node is critical if its centrality index is above a threshold. l-CRC of a node v is formulated as

$$l\text{-CRC}(v) = < l\text{-CC}(v), l\text{-RC}(v) >, \tag{1}$$

where l-CC is *Length-constrained Connectivity Centrality* and l-RC is *Length-constrained Rerouting Centrality*.

The length-constrained connectivity centrality of a node v is the number of v's descendants that would be either disconnected or pushed over the path length limit $l_{\max}$ when v is removed from the network. It is formulated as

$$l\text{-CC}(v) = |\{w \in D(v);\ d(w, Sink) \leq l_{\max}, d_v(w, Sink) > l_{\max}\}|, \tag{2}$$

where $D(v)$ is the set of node v's descendants in the routing tree, $Sink$ is the set of sinks, $d(w, Sink)$ denotes the shortest path length between node w and $Sink$, while $d_v(w, Sink)$ represents the length of the shortest path from w to $Sink$ which does not visit v. The length-constrained rerouting centrality of a node v is the total percentage of additional length of the shortest paths, which are over the path length limit $l_{\max}$, from v's descendants to the sinks upon removal of v. Note that the calculation only takes v's descendants that are still connected to the routing tree after v is removed, because the sum of distances is only meaningful for a connected graph. The length-constrained rerouting centrality is defined as

$$l\text{-RC}(v) = \sum_{u \in D(v),\ d_v(w, Sink) \neq \infty} \left(\frac{\max\{d_v(w, Sink), l_{\max}\}}{\max\{d(w, Sink), l_{\max}\}} - 1 \right). \tag{3}$$

GRASP [8] [9] [20] is a metaheuristic which captures good features of pure greedy algorithms and random construction procedures. It is an iterative process. In each iteration, it consists of two phases: the construction phase and the local search phase. The construction phase builds a feasible solution as a good starting solution for the local search phase. The probabilistic component of a GRASP is characterised by randomly choosing one of the best possible candidates, instead of the overall best one. Since the solution produced by the construction phase is not necessarily the local optimum, the local search phase is utilised to improve it. A local search algorithm works in an iterative fashion by replacing the current solution by a better one from the neighborhood of the current solution. It terminates when no better solution is found.

3 Related Work

In the literature, the problems of deploying sinks and relays are solved separately. Some sink deployment algorithms have been proposed with objectives to minimise and balance the energy consumption across networks [28] [15], reduce packet delivery latency [29], meet the required lifetime [18] and make the network double-covered for fault-tolerance [7]. Even though the algorithm in [7] is not designed for WSNs, the problem of finding the optimal positions for core nodes in passive optical networks [7] is similar to the problem of finding optimal positions for sinks in WSNs. Similar to our objective, i.e. to minimise the deployment cost, the algorithms proposed in [28] [18] try to minimise the number of sinks deployed, while the number of sinks is given in [15] [29] [7]. In [28], a sink is chosen greedily from a set of candidate locations such that it can cover as many sensor nodes, which are within the hop count bound from the sink, as possible. However, this algorithm does not consider double-covered networks. In [18], the algorithm deploys sinks one by one until the desired network lifetime is reached. It does not make the network double-covered and uses the well-known k-means clustering algorithm to identify the positions of sinks, which can be placed anywhere in the region. The algorithm in [7] also uses the k-means clustering algorithm to place a given number of sinks to make the network double-covered. None of these algorithms consider limits on the lengths of the paths.

The problem of deploying relay nodes for increased reliability has long been acknowledged as a significant problem [6] [19] [12] [16] [13] [14] [22]. Bredin *et al.* [6] develop k-connectivity-repair by finding a minimum-weight vertex k-connected subgraph from a weighted complete graph, adding edges in increasing weight and for each edge, deploying k relays every transmission range distance and $k-1$ relays at endpoints of the edge. Partial k-connectivity-repair by Pu *et al.* [19] is similar to k-connectivity-repair [6], but only places one relay every transmission range distance and none at endpoints. Connectivity-first [12] finds a minimum k-connected spanning graph from a weighted complete graph by adding edges that have the highest contribution to connectivity and the least weight. Misra *et al.* [16] propose connected and survivable relay

node placement. Both connected and survivable assign a weight to each edge equal to the number of candidate relays the edge is incident with. Connected relay node placement computes a low weight connected subgraph, while survivable relay node placement computes a low weight 2-connected subgraph. Relays are then deployed at the candidate locations that appear in the subgraph. Kashyap *et al.* [13] propose k-vertex connectivity, where from a complete graph of cluster heads, it assigns edge weights, finds a minimum cost vertex k-connected spanning subgraph, and deploys relays along the subgraph's edges. Lanza-Gutierrez *et al.* [14] solve the problem of relay node placement using parallel genetic algorithms to reduce computation time.

The relay placement algorithm proposed in [22], GRASP-ABP, is also a GRASP-based local search algorithm. It uses *Length-constrained Connectivity and Rerouting Centrality* (l-CRC) to identify critical nodes. After identifying the critical nodes, relays are deployed around those nodes to preserve backup paths if they die. By restricting attention to nodes with high centrality, we can focus on the most important nodes. Thus, l-CRC allows us to trade off the deployment cost against the robustness of the network. GRASP-ABP with l-CRC has been shown to deploy fewer relays compared to the algorithm in [6] [19].

4 Multiple Sink Placement (MSP)

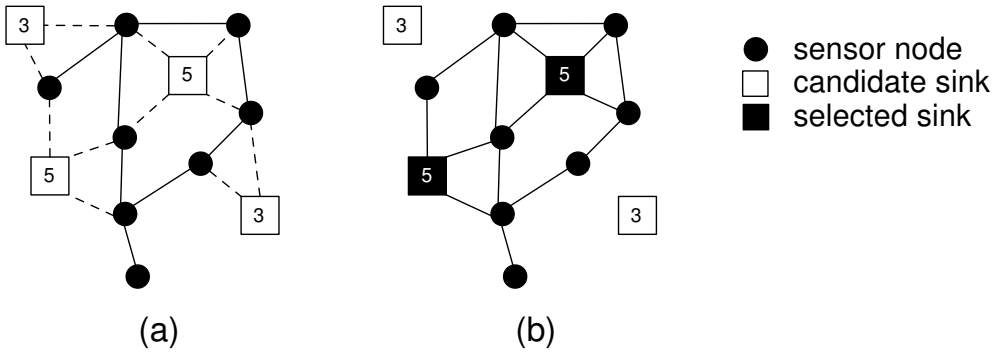

Fig. 1 Illustration of the MSP problem. (a) A WSN with four candidate sinks and (b) the double-covered WSN where $l_{\max}=3$.

We partition nodes into a set of sensors T and sinks S. In the graph representation of a WSN, $V=T\cup S$. Note that at this stage we do not use relays. A sensor is *double-covered* if and only if it has at least two paths of length $\leq l_{\max}$ to two sinks in S. If a sensor is not double-covered, it is *uncovered*. We define a WSN as double-covered if each sensor $v \in T$ is double-covered. In the *multiple sink placement problem*, given a graph $G=(T\cup A_{\mathrm{S}}, E)$, where A_{S} is a set of candidate locations for sinks with a non-negative cost function

$c : A_S \rightarrow \mathbb{R}$, we find a minimum cost subset $S \subseteq A_S$ such that $H = (T \cup S, E\downarrow_{T \cup S})$ is double-covered.

We illustrate this problem in Figure 1. Figure 1(a) illustrates a WSN with four candidate locations to deploy sinks, where the numbers represent the costs. In order to make the WSN double-covered with $l_{max} = 3$, we need to deploy the two sinks as shown in Figure 1(b). The total cost of sink deployment in this example is 10 units.

To solve the multiple sink placement problem, we present Greedy-MSP, a greedy-based algorithm, and GRASP-MSP, a local search algorithm that uses the GRASP technique [8] [9] [20]. To speed-up our algorithms' processing time, we compute the shortest path from all sensors to all candidate sinks once in the beginning and store the length of the shortest path in *Distance* table, while the parent of each sensor is stored in *Parent* table. A parent of a sensor node is the next node on the shortest path to a candidate sink. For example, for $G = (T \cup A_S, E)$, $Distance_G(v, w)$ shows the length of the shortest path from a sensor $v \in T$ to a candidate sink $w \in A_S$, while $Parent_G(v, w)$ shows the parent of a sensor v on the shortest path to a candidate sink $w \in A_S$.

4.1 Greedy Algorithm for Multiple Sink Placement (Greedy-MSP)

Greedy-MSP is similar to the *Heuristic Opt Multisink Place* (HOMP) algorithm by Xu and Liang [28], but instead of deploying a minimum number of sinks to make a network single-covered, it considers double-covered networks. Greedy-MSP takes as input the original graph $G = (T \cup A_S, E)$, the set T of sensors, the set A_S of candidate sinks, the cost function c, the pre-computed $Distance_G$ table, and the maximum acceptable path length l_{max}. Greedy-MSP selects the best sinks one by one until all sensors are double-covered. In each iteration, the greedy move picks the best sink from the set of candidate sinks that can minimise the number of uncovered sensors. The best sink is the one that together with the previously chosen sinks can minimise the number of uncovered sensors. In Greedy-MSP, if two or more sinks offer the same number of uncovered sensors, the lowest cost sink is selected. If there are ties, we select one randomly. The iteration stops when all sensors are double-covered or all candidate sinks have been selected.

4.2 Greedy Randomised Adaptive Search Procedure for Multiple Sink Placement (GRASP-MSP)

To tackle the issue of local minima, we present GRASP-MSP to solve the multiple sink placement problem. As with other GRASP-based algorithms, GRASP-MSP consists of two steps: *construction phase* to construct an initial feasible solution and *local search phase* to explore the neighbourhood of the initial solution, looking for lower cost solutions.

4.2.1 Construction Phase

In the construction phase, we find S, an initial set of sinks. Instead of selecting the best candidate sink from A_S to be put in S, which can minimise the number of uncovered sensors, we add randomisation to the initial solution by choosing a sink from A_S randomly. This random selection is repeated until the network is double-covered or all candidate sinks have been chosen.

4.2.2 Node-based Local Search

Let S be the set of sinks. We explore the neighbourhood of the current solution by adding a new sink $s \in A_S \setminus S$ into S that can eliminate some existing sinks from S to reduce the total sink cost. This move must always ensure that the network is double-covered.

4.2.3 Algorithm Description

The GRASP-MSP pseudocode is given in Algorithm 1. It takes as input the original graph $G = (T \cup A_S, E)$, the set T of sensors, the set A_S of candidate sinks, the cost function c, the pre-computed $Distance_G$ table, the maximum acceptable path length $l_{\max}$, and the number of iterations ($max\_iterations$). In each iteration, the construction phase to find the initial set of sinks S is executed in line 3. The local search starts with the initialisation of the best set and the best cost in line 6. The loop from line 7 to 22 searches for the best move, i.e. finding a new sink $r \in A_S \setminus S$ that can eliminate as many existing sinks from S as possible. The loop from line 9 to 15 tries to find the set $Z \subseteq S$ of existing sinks that are safe to be removed after the insertion of r. The sinks in Z are safe to be removed if all sensors in $H = (T \cup S \cup \{r\} \setminus Z, E{\downarrow}_{T \cup S \cup \{r\} \setminus Z})$ are double-covered. In line 16, we check if the new solution reduces the total cost of the current best solution. If the total cost can be reduced, we reset the set of the best set in line 17. If the total cost is the same, we keep this new solution in the set of the best set as shown from line 19 to 21. When all moves have been evaluated, we check in line 23 if an improving solution has been found. If the moves produce a better solution, the set of sinks S is updated in line 24 by selecting one best set randomly from the set of the best set. Then, the local search continues. If, at the end of the local search, we find a better solution compared to the best solution found so far, we update in line 29 the set of sinks and the lowest total cost found. The best sink set S^* is returned in line 32.

5 Evaluation of Greedy-MSP and GRASP-MSP

In this section, we evaluate Greedy-MSP and GRASP-MSP, and we show that while Greedy-MSP has the shortest runtime, GRASP-MSP finds the lowest cost sink deployment compared to other closely-related algorithms.

Algorithm 1: GRASP-MSP

Input: $G, T, A_S, c, Distance_G, l_{max}, max\_iterations$
Output: S^*

$best\_value \leftarrow \infty$;
for $i \leftarrow 1$ to $max\_iterations$ **do** /* Construction phase */
  Find S by choosing sinks from A_S randomly;
  repeat /* Local search phase */
    $solution\_updated \leftarrow$ **false**;
    $best\_set_0 \leftarrow S$; $best\_cost \leftarrow \sum_{v \in S} c_v$; $best\_num\_set \leftarrow 1$;
    foreach $r \in A_S \setminus S$ **do**
      $Z \leftarrow \emptyset$;
      foreach $t \in S$ **do**
        $Z \leftarrow Z \cup \{t\}$; $H \leftarrow (T \cup S \cup \{r\} \setminus Z, E{\downarrow}_{T \cup S \cup \{r\} \setminus Z})$;
        Calculate $num\_uncovered$ in H using $Distance_G$ and l_{max}
        if $num\_uncovered > 0$ **then**
          $Z \leftarrow Z \setminus \{t\}$;
        end
      end
      if $\sum_{v \in S \cup \{r\} \setminus Z} c_v < best\_cost$ **then**
        $best\_num\_set \leftarrow 0$;
      end
      if $\sum_{v \in S \cup \{r\} \setminus Z} c_v \leq best\_cost$ **and** $S \cup \{r\} \setminus Z \notin best\_set$ **then**
        $best\_set_{best\_num\_set} \leftarrow S \cup \{r\} \setminus Z$; $best\_cost \leftarrow \sum_{v \in S \cup \{r\} \setminus Z} c_v$;
        $best\_num\_set \leftarrow best\_num\_set + 1$;
      end
    end
    if $best\_cost < \sum_{v \in S} c_v$ **then**
      $S \leftarrow$ select a set randomly from $best\_set$;
      $solution\_updated \leftarrow$ **true**;
    end
  until $solution\_updated =$ **false**;
  if $\sum_{v \in S} c_v < best\_value$ **then** /* Best solution update */
    $S^* \leftarrow S$; $best\_value \leftarrow \sum_{v \in S} c_v$;
  end
end
return S^*;

GRASP-MSP finds comparable solutions to a linear programming model, with marginally faster runtime. We evaluate the performance of the algorithms using the following metrics:

1. ***Average number of sinks needed and average total sink cost***. We compare the effectiveness of the algorithms in finding the minimum cost solution to the same problem. We expect that GRASP-MSP deploys the fewest sinks and has the lowest cost solution compared to other algorithms. We also expect Greedy-MSP's solution to have slightly more sinks than GRASP-MSP's, but it should be comparable to other multiple sink placement algorithms.
2. ***Average runtime***. We expect both Greedy-MSP and GRASP-MSP to be more efficient than the other algorithms.

The results presented here are based on the average of 20 randomly generated network deployments that are simulated 31 times each. We presented the average based on the median values because our data is not symmetrically distributed. The network consists of 100 sensor nodes deployed within randomly perturbed grids, where a sensor node is placed in a unit grid square of 8m $\times$ 8m and the coordinates are perturbed. To get sparse networks (average degree 2-3), we generate more grid points than the number of nodes. We use 11 $\times$ 11 grid squares to randomly deploy 100 sensor nodes. 25 candidate sinks are also distributed in a grid area, where a candidate occupies a unit grid square of 18m $\times$ 18m. The coordinates for the sensors and candidate sinks are given in [21]. Both sensor nodes and sinks use 10-metre transmission range, which is realistic for 0 dBm transmission power in indoor environments [27].

We compare Greedy-MSP and GRASP-MSP to *Minimise the Number of Sinks for Fault-Tolerance* (MSFT), *Cluster-Based Sampling for Multiple Sink Placement* (CBS-MSP) and the linear programming solution. Since there are no existing algorithms in the literature that share the same objectives as ours, we take the closest approaches and modify them to be comparable. MSFT and CBS-MSP are algorithms based on the well-known k-means clustering algorithm. MSFT is the modification of *Minimise the Number of Sinks for a Predefined Minimum Operation Period* (MSPOP) [18]. MSPOP is similar to Greedy-MSP in the sense that both algorithms greedily place sinks until an objective is met. However, in MSPOP, sinks can be deployed anywhere and the objective is a required network lifetime. As the modification of MSPOP, MSFT has candidate locations and keeps adding sinks until the network is double-covered. CBS-MSP is the modification of *Cluster-Based Sampling* (CBS) proposed in [7]. In CBS, the number of sinks is given and the objective is to minimise the total road distance from all nodes to the sinks where each node is required to be double-covered. Unlike CBS, CBS-MSP tries to reduce the number of sinks and thus the deployment cost. We implement CBS-MSP using path length to represent distance between two nodes and also we have path length restrictions. In each iteration, both CBS and CBS-MSP try to find the best sink locations to ensure the network is double-covered. The k-means clustering algorithm is used in these algorithms to divide the network into clusters and to find the position of each sink, which is in the centre of a cluster.

MSFT, CBS-MSP and GRASP-MSP are all randomised local search algorithms. As a stopping criterion, we allow a maximum number of iterations, *MaxIter*, and compare versions with different values. We consider values of 1, 10 and 100 for MSFT and CBS-MSP, and restrict to values of 1 and 10 for GRASP-MSP, since experience on similar sized problems has shown that higher iterations rarely produce better solutions. In our Greedy-MSP and GRASP-MSP, if two or more moves offer the same solution, we select one randomly. We also consider Greedy-MSP-All and GRASP-MSP-All, where if there exists more than one best move, we evaluate them all.

For comparison, we implement a binary linear programming model, with the objective to minimise the total sink cost, i.e.

$$\min \sum_{j \in S} c_j x_j \tag{4}$$

subject to the following constraints

$$\sum_{j \in S} l_{ij} x_j \geq 2;\ \forall i \in T \tag{5}$$

$$x_j \in \{0, 1\};\ j \in S \tag{6}$$

c_j is the cost of a candidate sink x_j. Constraint (5) guarantees each sensor node has at least two length-bounded paths to two sinks. l_{ij} has value 1 if the shortest path length from sensor node i to sink j is less than $l_{\max}$, and otherwise 0. In the second constraint, a candidate sink is either selected to be deployed or not. The binary linear programming is implemented in Matlab, while the other algorithms are written in C++.

We first assume all candidate sinks have the same cost and consider two cases where the maximum acceptable path length from each sensor node to a sink is either 6 or 10. The average number of sinks deployed by each algorithm is shown in Figure 2 and the average runtime is in Table 1. Firstly, the results show that all variants of simulated GRASP-MSP deploy the same average number of sinks. However, more iterations and computation of all best moves incur longer runtime. Secondly, the results show that GRASP-MSP even with only *MaxIter* = 1 already has the same average number of sinks as the linear programming solution (the Wilcoxon signed-rank test, with p-value < 0.0001), but with shorter runtime (p-value < 0.0001). It also outperforms MSFT, CBS-MSP and Greedy-MSP. For example, it requires fewer sinks than MSFT with *MaxIter* = 100 and has faster runtime than MSFT with *MaxIter* = 1, both with p-values less than 0.0001. Greedy-MSP has the shortest runtime, but it places more sinks compared to GRASP-MSP. We also observe that the average number of deployed sinks decreases when the maximum path length increases because each sink in the network can cover more sensor nodes.

We also consider a more realistic scenario where the deployment costs for all candidate sinks are different, for example due to cabling and installation costs. We compare the performance of the algorithms using the same sink cost (c_{S}), i.e. 3 units and different sink costs, which are selected randomly in the interval [3, 6]. The results for the average total sink cost and the average runtime with $l_{\max} = 6$ are presented in Figure 3 and Table 2, respectively. These results show similar trends to the previous ones when we vary the maximum path lengths. That is, GRASP-MSP achieves the same cost as the linear programming solution, while Greedy-MSP has the shortest runtime in all scenarios.

We evaluate the performance of GRASP-MSP against the linear programming solution when the density of the network increases. We double the number of sensor nodes from 100 to 200 while keeping the area fixed. As a result,

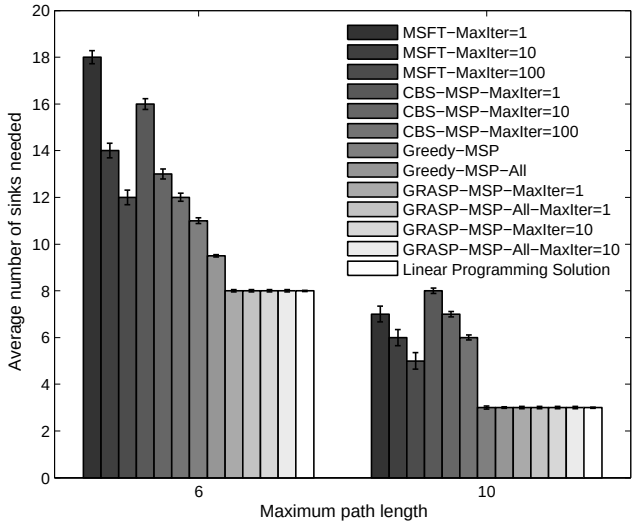

Fig. 2 Average number of sinks needed for multiple sink placement algorithms versus maximum path length

Table 1 Multiple sink placement algorithms' average runtime with different maximum path length

Algorithms	**Average runtime (sec)**	
	$l_{\max}=6$	$l_{\max}=10$
MSFT-MaxIter=1	0.4179	0.0337
MSFT-MaxIter=10	4.1518	0.3067
MSFT-MaxIter=100	41.2484	3.1048
CBS-MSP-MaxIter=1	1.0188	0.0220
CBS-MSP-MaxIter=10	9.7818	0.2146
CBS-MSP-MaxIter=100	97.4409	2.0861
Greedy-MSP	0.0022	0.0021
Greedy-MSP-All	7.9480	0.0059
GRASP-MSP-MaxIter=1	0.0088	0.0064
GRASP-MSP-All-MaxIter=1	0.0151	0.0120
GRASP-MSP-MaxIter=10	0.0766	0.0455
GRASP-MSP-All-MaxIter=10	0.1218	0.0871
Linear Programming Solution	0.0727	0.0867

the average degree of a sensor node increases from 3 to 7. We present the average number of deployed sinks out of 25 candidate sinks in Figure 4 and the average runtime in Table 3. Compared to the linear programming, GRASP-MSP with $MaxIter = 1$ has the shortest runtime and the same average number of sinks with p-values less than 0.0001.

At this stage, finding the linear programming solution is sufficient for the multiple sink placement problem. Nevertheless, the GRASP-MSP performance gives us confidence to use the same local search technique for the more com-

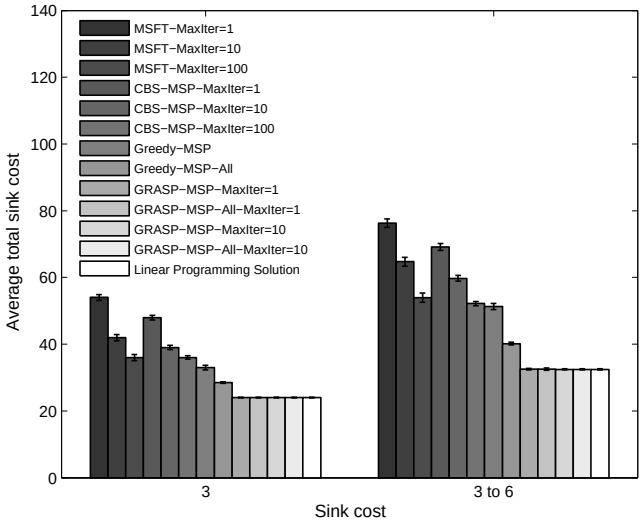

Fig. 3 Average total sink cost for multiple sink placement algorithms versus sink cost

Table 2 Multiple sink placement algorithms' average runtime with different sink cost

Algorithms	**Average runtime (sec)**	
	$c_S = 3$	$c_S = 3$ **to** 6
MSFT-MaxIter=1	0.4179	0.4135
MSFT-MaxIter=10	4.1518	4.1578
MSFT-MaxIter=100	41.2484	41.5128
CBS-MSP-MaxIter=1	1.0188	0.9970
CBS-MSP-MaxIter=10	9.7818	9.9196
CBS-MSP-MaxIter=100	97.4409	99.0951
Greedy-MSP	0.0022	0.0017
Greedy-MSP-All	7.9480	8.2004
GRASP-MSP-MaxIter=1	0.0088	0.0152
GRASP-MSP-All-MaxIter=1	0.0151	0.0131
GRASP-MSP-MaxIter=10	0.0766	0.1365
GRASP-MSP-All-MaxIter=10	0.1218	0.1198
Linear Programming Solution	0.0727	0.1086

Table 3 Multiple sink placement algorithms' average runtime with different average degree

Algorithms	**Average runtime (sec)**	
	average degree $= 3$	**average degree** $= 7$
GRASP-MSP-MaxIter=1	0.0088	0.0167
GRASP-MSP-All-MaxIter=1	0.0151	0.0306
GRASP-MSP-MaxIter=10	0.0766	0.1290
GRASP-MSP-All-MaxIter=10	0.1218	0.2535
Linear Programming Solution	0.0727	0.0735

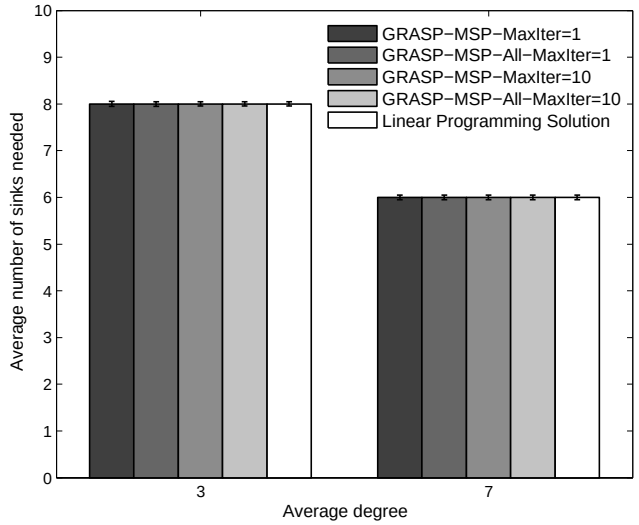

Fig. 4 Average number of sinks needed for GRASP-MSP and the linear programming solution versus average degree

plex multiple sink and relay placement problem, where a linear programming solution is not available.

6 Multiple Sink and Relay Placement (MSRP)

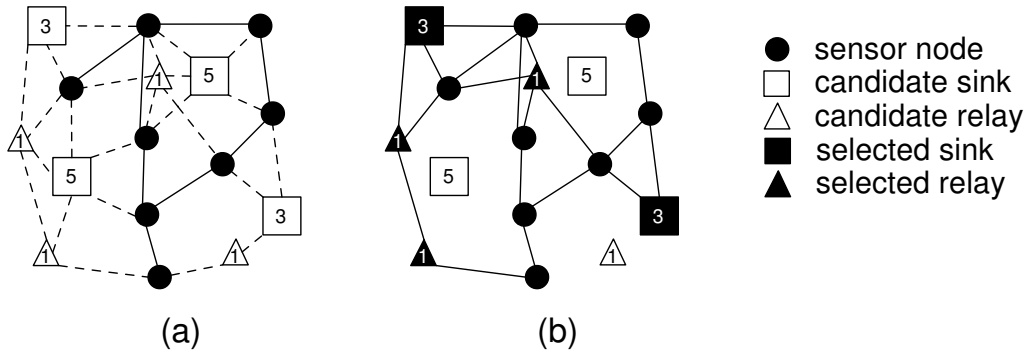

Fig. 5 Illustration of the MSRP problem. (a) A WSN with four candidate sinks and four candidate relays, and (b) the double-covered and noncritical WSN where $l_{\max} = 3$.

For the sink and relay placement, nodes are partitioned into a set of sensors T, relays R and sinks S. In the graph representation, $V = T \cup R \cup S$. In the *multi-*

ple sink and relay placement problem, given a graph $G=(T\cup A_{\mathrm{R}}\cup A_{\mathrm{S}}, E)$, where A_{R} and A_{S} are sets of candidate locations for relays and sinks, respectively, we find minimum cost subsets $R\subseteq A_{\mathrm{R}}$ and $S\subseteq A_{\mathrm{S}}$ such that $H=(T\cup R\cup S, E\downarrow_{T\cup R\cup S})$ is double-covered and non-critical. The relay and sink candidate locations are associated with a non-negative cost function $c : A_{\mathrm{R}}\cup A_{\mathrm{S}}\rightarrow\mathbb{R}$. We assume that a relay is cheaper than a sink because sinks usually are assumed to be powered, have more memory, processing and WiFi/ethernet backhaul.

The multiple sink and relay placement problem is illustrated in Figure 5. Figure 5(a) shows a WSN with four candidate sinks and four candidate relays. The numbers in the figure represent the costs. In this example, if we choose the two 5-unit sinks, we only need to deploy the relay at the bottom-left corner to make the network double-covered and non-critical. The total cost is 11 units. However, if we choose the two 3-unit sinks as shown in Figure 5(b), we need three additional relays. The total cost of this deployment is the lowest, i.e. 9 units.

In this section, we present Greedy-MSRP and GRASP-MSRP to solve the multiple sink and relay placement problem. Both algorithms use the concept of Length-constrained Connectivity and Rerouting Centrality (l-CRC) [22] to identify critical sensors. A sensor is critical if its centrality index is above a given threshold. We can raise the threshold to trade-off the deployment cost against the robustness of the network. However, in this chapter, we only assume zero threshold for full reliability. After the identification of critical sensors, some candidate relays are selected to be deployed. We identify relays that need to be deployed by finding the shortest path from each descendant of each critical sensor to a sink bypassing each critical sensor.

We first present Greedy-MSRP. Greedy-MSRP uses Greedy-MSP to deploy a minimum number of sinks and GRASP-MRP to deploy a minimum number of relays. Before presenting Greedy-MSRP, we will look at GRASP-MRP, which is a modification of GRASP-ABP from [22] to tackle both double-covered and non-critical requirements.

6.1 Greedy Randomised Adaptive Search Procedure for Multiple Relay Placement (GRASP-MRP)

6.1.1 Construction Phase

In the construction phase, we find $R\subseteq A_{\mathrm{R}}$, an initial set of relays from the candidate relays, to minimise the number of uncovered and critical sensors. Initially R is an empty set. Given sets of sensors T and sinks S, we firstly find and store the length of the shortest path of each sensor to the sinks in $\mathit{Distance}_H$ table, where $H=(T\cup R\cup S, E\downarrow_{T\cup R\cup S})$. If a sensor $v\in T$ is uncovered, we try to place some relays to construct a path to a sink $w\in S$ if $\mathit{Distance}_H(v,w)>l_{\max}$ but the pre-computed $\mathit{Distance}_G(v,w)\leq l_{\max}$. We choose the relays that appear on the shortest path from v to w by tracing the path in $\mathit{Parent}_G(v,w)$. If a sensor needs two paths to make it double-covered,

this step is repeated twice. After some relays are selected from the candidate relays, we have the set of relays R that minimises the number of uncovered sensors. We then rebuild $H = (T \cup R \cup S, E\downarrow_{T \cup R \cup S})$ and check if critical sensors exist using centrality calculation. If a sensor $v \in T$ is critical, we deploy relays that appear on the shortest path from each descendant of v to a sink $w \in S$ bypassing v, as long as the shortest path length is $\leq l_{\max}$. The randomisation of the initial solution is obtained by randomly selecting parents in the shortest paths if there are hop count ties.

6.1.2 Node-based Local Search

Let R be the set of relays. The local search's move tries to add a new relay $r \in A_{\mathrm{R}} \backslash R$ into R that can eliminate as many existing relays from R as possible. This move must always ensure that the network is double-covered and non-critical.

Algorithm 2: GRASP-MRP

Input: $G, T, S, A_{\mathrm{R}}, l_{\max}, max\_iterations$
Output: R^*
$best\_value \leftarrow \infty$;
for $i \leftarrow 1$ to $max\_iterations$ **do** `/* Construction phase */`
  Find initial R;
  repeat `/* Local search phase */`
    $solution\_updated \leftarrow$ **false**;
    $best\_set \leftarrow R$; $best\_number \leftarrow |R|$;
    foreach $r \in A_{\mathrm{R}} \setminus R$ **do**
      $Z \leftarrow \emptyset$;
      foreach $t \in R$ **do**
        $Z \leftarrow Z \cup \{t\}$; $H \leftarrow (T \cup R \cup \{r\} \setminus Z \cup S, E\downarrow_{T \cup R \cup \{r\} \setminus Z \cup S})$;
        Calculate $Distance_H$;
        Calculate $num\_uncovered$ and $num\_critical$ in H using $Distance_H$ and $l_{\max}$;
        if $num\_uncovered > 0$ **or** $num\_critical > 0$ **then**
          $Z \leftarrow Z \setminus \{t\}$;
        end
      end
      if $|R| - |Z| + 1 < best\_number$ **then**
        $best\_set \leftarrow R \cup \{r\} \setminus Z$; $best\_number \leftarrow |R| - |Z| + 1$;
      end
    end
    if $best\_number < |R|$ **then**
      $R \leftarrow best\_set$;
      $solution\_updated \leftarrow$ **true**;
    end
  until $solution\_updated =$ **false**;
  if $|R| < best\_value$ **then** `/* Best solution update */`
    $R^* \leftarrow R$; $best\_value \leftarrow |R|$;
  end
end
return R^*;

6.1.3 Algorithm Description

Algorithm 2 shows the pseudocode for GRASP-MRP. It takes as input the original graph $G=(T\cup A_{\mathrm{R}}\cup A_{\mathrm{S}}, E)$, the set T of sensors, the set S of sinks, the set A_{R} of candidate relays, the maximum acceptable path length $l_{\max}$, and the number of iterations (*max_iterations*). In each iteration, the construction phase to find the initial set of relays R to minimise the number of uncovered and critical sensors is executed in line 3. The local search starts with the initialisation of the best set and the best number of relays in line 6. The loop from line 7 to 20 searches for the best move to find a new relay $r \in A_{\mathrm{R}} \setminus R$ that can eliminate as many existing relays from R as possible. The loop from line 9 to 16 finds the set $Z \subseteq R$ of existing relays that are safe to be removed after the insertion of r. The relays in Z are safe to be removed if all sensors in $H=(T\cup R\cup\{r\}\setminus Z\cup S, E{\downarrow}_{T\cup R\cup\{r\}\setminus Z\cup S})$ are double-covered and non-critical. We check in line 17 if the new solution has fewer relays than the current best solution. If the number of relays can be reduced, the best set and the best number of relays are updated in line 18. When all local search moves have been evaluated, we check if an improving solution has been found in line 21. If the moves produce a better solution, the set of relays R is updated in line 22. Then, the local search continues. If, at the end of the local search, we find a better solution compared to the best solution found so far, we update in line 27 the set of relays and the least number of relays used. Finally, the best relay set R^* is returned in line 30.

6.2 Greedy Algorithm for Multiple Sink and Relay Placement (Greedy-MSRP)

After describing GRASP-MRP, we are now ready to present Greedy-MSRP for the multiple sink and relay placement problem. Greedy-MSRP uses Greedy-MSP to select a minimum number of sinks to make a network double-covered and GRASP-MRP to deploy a minimum number of relays to make the network non-critical. Greedy-MSRP takes as input the original graph $G=(T\cup A_{\mathrm{R}}\cup A_{\mathrm{S}}, E)$, the set T of sensors, the set A_{R} of candidate relays, the set A_{S} of candidate sinks, the cost function c, the pre-computed $Distance_G$ table, the maximum acceptable path length $l_{\max}$, and the number of iterations (*max_iterations*) for GRASP-MRP. Initially, Greedy-MSRP calls Greedy-MSP to find the set of sinks. Since the cost of a sink is assumed to be more expensive than the cost of a relay, Greedy-MSRP tries to reduce the total deployment cost by trading some sinks with relays. The deployed sinks are removed one by one, starting from the last one inserted into the set, and more relays are added in the network. However, this swap must ensure that the network is always double-covered and non-critical. If all sensors are double-covered and non-critical, GRASP-MRP is called to find the set of relays.

6.3 Greedy Randomised Adaptive Search Procedure for Multiple Sink and Relay Placement (GRASP-MSRP)

We now present GRASP-MSRP to solve the multiple sink and relay placement problem. Unlike Greedy-MSRP that deploys relays after finding the minimal set of sinks, GRASP-MSRP finds the least deployment cost by placing sinks and relays at the same time. We give the detailed algorithm below.

6.3.1 Construction Phase

In the construction phase, we find $R \subseteq A_{\mathrm{R}}$ and $S \subseteq A_{\mathrm{S}}$ as our initial sets of relays and sinks, respectively. Initially R and S are empty sets. We then alternate the sink and relay addition during this process. We deploy some sinks to minimise the number of uncovered sensors and then some relays to minimise the number of both uncovered and critical sensors. Note that we do not add more sinks if at some point the network is already double-covered. After the addition of either sinks or relays, $H = (T \cup R \cup S, E{\downarrow}_{T \cup R \cup S})$ is rebuilt. The process of sink and relay addition is repeated until the network is double-covered and non-critical.

We need at least two sinks for a double-covered WSN, so we firstly choose two sinks randomly from A_{S}. We then deploy a subset of relays from A_{R} to minimise the number of uncovered and critical sensors. If a sensor $v \in T$ is uncovered, we place some relays to construct a path to a sink $w \in S$ if $\mathit{Distance}_H(v, w) > l_{\max}$ but $\mathit{Distance}_G(v, w) \leq l_{\max}$. We choose the relays that appear on the shortest path from v to w by tracing the path in $\mathit{Parent}_G(v, w)$. If the sensor needs two paths to make it double-covered, we repeat this step twice. If a sensor $v \in T$ is critical, we deploy relays that appear on the shortest path from each descendant of v to a sink $w \in S$ bypassing v, as long as the shortest path length is $\leq l_{\max}$. After the relay deployment, we place sinks again. In order to add the randomisation to the initial solution, we randomly select parents in the shortest paths if there are hop count ties.

6.3.2 Node-based Local Search

Let R be the set of relays and S be the set of sinks. We look for a lower cost solution by adding either a new relay $r \in A_{\mathrm{R}} \setminus R$ into R or a new sink $s \in A_{\mathrm{S}} \setminus S$ into S that can eliminate some existing relays from R and sinks from S to minimise the total cost as possible. Given that the cost of a sink is higher than the cost of a relay, we also try to minimise the total cost by adding some relays into R when we eliminate an existing sink from S. The local search moves are performed to reduce the total cost, but must ensure that the network is always double-covered and non-critical in each iteration.

Algorithm 3: GRASP-MSRP

Input: $G, T, A_{\mathrm{R}}, A_{\mathrm{S}}, c, l_{\max}, max\_iterations$
Output: R^*, S^*

$best\_value \leftarrow \infty$;
for $i \leftarrow 1$ to $max\_iterations$ **do** /* Construction phase */
  Find initial R and S; $W \leftarrow R \cup S$;
  repeat /* Local search phase */
    $solution\_updated \leftarrow$ **false**;
    $best\_set_0 \leftarrow W$; $best\_cost \leftarrow \sum_{v \in W} c_v$; $best\_num\_set \leftarrow 1$;
    foreach $r \in A_{\mathrm{R}} \cup A_{\mathrm{S}} \setminus W$ **do**
      $Y \leftarrow \{r\}$; $Z \leftarrow \emptyset$; $X \leftarrow \emptyset$;
      foreach $t \in W \cup X$ **do**
        $Z \leftarrow Z \cup \{t\}$; $X \leftarrow \emptyset$;
        $H \leftarrow (T \cup W \cup Y \setminus Z, E{\downarrow}_{T \cup W \cup Y \setminus Z})$;
        Calculate $Distance_H$;
        Find $uncovered\_set$ in H using $Distance_H$ and $l_{\max}$;
        if $|uncovered\_set| > 0$ **then**
          $X \leftarrow X \cup \{$Find relays to minimise $|uncovered\_set|\}$;
        end
        $H \leftarrow (T \cup W \cup X \cup Y \setminus Z, E{\downarrow}_{T \cup W \cup X \cup Y \setminus Z})$;
        Calculate $Distance_H$;
        Find $critical\_set$ in H using $Distance_H$ and $l_{\max}$;
        if $|critical\_set| > 0$ **then**
          $X \leftarrow X \cup \{$Find relays to minimise $|critical\_set|\}$;
        end
        $H \leftarrow (T \cup W \cup X \cup Y \setminus Z, E{\downarrow}_{T \cup W \cup X \cup Y \setminus Z})$;
        Calculate $Distance_H$;
        Calculate $num\_uncovered$ and $num\_critical$ in H using $Distance_H$ and $l_{\max}$;
        if $num\_uncovered = 0$ **and** $num\_critical = 0$ **then**
          $Y \leftarrow Y \cup X$; $Z \leftarrow Z \setminus X$;
        end
        if $num\_uncovered > 0$ **or** $num\_critical > 0$ **then**
          $Z \leftarrow Z \setminus \{t\}$;
        end
      end
      if $\sum_{v \in W \cup Y \setminus Z} c_v < best\_cost$ **then**
        $best\_num\_set \leftarrow 0$;
      end
      if $\sum_{v \in W \cup Y \setminus Z} c_v \leq best\_cost$ **and** $W \cup Y \setminus Z \notin best\_set$ **then**
        $best\_set_{best\_num\_set} \leftarrow W \cup Y \setminus Z$; $best\_cost \leftarrow \sum_{v \in W \cup Y \setminus Z} c_v$; $best\_num\_set \leftarrow best\_num\_set + 1$;
      end
    end
    if $best\_cost < \sum_{v \in W} c_v$ **then**
      $W \leftarrow$ select a set randomly from $best\_set$;
      $solution\_updated \leftarrow$ **true**;
    end
  until $solution\_updated =$ **false**;
  if $\sum_{v \in W} c_v < best\_value$ **then** /* Best solution update */
    $R^* \leftarrow \emptyset$; $S^* \leftarrow \emptyset$;
    foreach $v \in W$ **do**
      if $v \in A_{\mathrm{R}}$ **then**
        $R^* \leftarrow R^* \cup \{v\}$;
      else
        $S^* \leftarrow S^* \cup \{v\}$;
      end
    end
    $best\_value \leftarrow \sum_{v \in W} c_v$;
  end
end
return R^*, S^*;

6.3.3 Algorithm Description

The GRASP-MSRP pseudocode is given in Algorithm 3. Generally, its concept is similar to GRASP-MSP in Algorithm 1 with some key differences. The key differences are the inclusion of candidate relays as one of its inputs, the identification of critical sensors, the deployment of relays to minimise the number of uncovered and critical sensors, and the repetitive computation of the shortest path from all sensors to all sinks due to the addition and elimination of relays. The detailed description of the pseudocode is given below.

GRASP-MSRP takes as input the original graph $G=(T\cup A_{\mathrm{R}}\cup A_{\mathrm{S}}, E)$, the set T of sensors, the set A_{R} of candidate relays, the set A_{S} of candidate sinks, the cost function c, the maximum acceptable path length $l_{\max}$, and the number of iterations (*max_iterations*). In each iteration, the construction phase to find initial sets of relays R and sinks S is executed in line 3. The local search starts with the initialisation of the best set and the best cost in line 6. The loop from line 7 to 39 searches for the best move, i.e. finding either a new relay or a new sink $r\in A_{\mathrm{R}}\cup A_{\mathrm{S}}\setminus W$ that can eliminate as many existing relays and sinks from W as possible, where $W=R\cup S$. The loop from line 9 to 32 tries to find the set $Z\subseteq W\cup X$ of existing relays and sinks that are safe to be removed after the insertion of Y. Y is the set of new relays and sinks that are added during the iteration. X is the set of new relays that are added to the network to reduce the total cost when a sink is removed from the network.

The algorithm checks for uncovered sensors in line 13. If some exist, it tries to deploy some relays in line 15. The identification of critical sensors is performed in line 19. If some exist, relays are added in line 21. Note that we try to minimise the total cost by adding some relays when we eliminate a sink. These relays are saved in X as shown in line 15 and 21, which later will be included in Y, the set of new relays and sinks to be inserted, if X helps the network become double-covered and non-critical. The network is checked if it is double-covered and non-critical in line 25. Note that there is repetitive computation of the shortest path from all sensors to all sinks in line 12, 18 and 24 due to the addition and elimination of relays. The relays and sinks in Z are safe to be removed if without Z all sensors are double-covered and non-critical. In line 33, we check if the new solution improves the total cost of the current best solution. If the total cost is reduced, we reset the best set in line 34. If the total cost is the same, we keep this new solution in the set of the best set as shown from line 36 to 38. When all moves have been evaluated, we check in line 40 if an improving solution has been found. If the moves produce a better solution, the set of relays and sinks W is updated in line 41 by selecting one best set randomly from the set of the best set. After that, the local search continues.

If, at the end of the local search, we find a better solution compared to the best solution found so far, we update from line 46 to 54 the set of relays, the set of sinks, and the lowest total cost found. The best sets R^* of relays and S^* of sinks are returned in line 57.

7 Evaluation of Greedy-MSRP and GRASP-MSRP

We evaluate the performance of Greedy-MSRP and GRASP-MSRP using the following metrics:

1. ***Average total sink and relay cost***. We want to compare the total deployment cost that resulted from each algorithm, which includes the cost of sinks and the cost of relays.
2. ***Average number of devices***, which is divided into number of sinks and number of relays. We present this metric as we cannot infer how many sinks and relays are deployed from the total cost metric. We expect to see that when the sink cost increases, the number of sinks decreases. This happens because some sinks are traded for relays to reduce the deployment cost.
3. ***Average runtime***. We also evaluate the efficiency of the algorithms by comparing the algorithms' runtime.

We follow the same experiment setting as for the evaluation of the multiple sink placement problem in Section 5, where each network consists of 100 sensor nodes in grid squares of 8m $\times$ 8m and 25 candidate sinks in grid squares of 18m $\times$ 18m. In addition, we also have 81 candidate relays distributed evenly in grid squares of 10m $\times$ 10m. The coordinates for the sensors, candidate sinks and candidate relays are given in [21].

We compare Greedy-MSRP and GRASP-MSRP against *Minimise the Number of Sinks and Relays for Fault-Tolerance* (MSRFT) and *Cluster-Based Sampling for Multiple Sink and Relay Placement* (CBS-MSRP). MSRFT and CBS-MSRP extend MSFT and CBS-MSP, respectively, to find the best locations to deploy sinks and use GRASP-MRP to deploy relays. These two algorithms start by finding the best locations for two sinks before utilising GRASP-MRP to deploy relays. The number of sinks is gradually increased and GRASP-MRP is used to deploy relays until the network becomes double-covered and non-critical. In the experiments, we only use 100 and 200 as the maximum iteration (*MaxIter*) for MSRFT and CBS-MSRP. The maximum iteration for GRASP-MRP is 10.

We evaluate the deployment cost of the algorithms by varying the sink costs (c_S), i.e. 3, 6, randomly in the interval [3, 6], and 10 units, while the relay cost is fixed at 1 unit. The average total sink and relay cost suggested by each algorithm with $l_{\max}=6$ is presented in Figure 6 and the average runtime is in Table 4. We also show the average numbers of sinks and relays for each algorithm from Figure 7 to Figure 14.

The results in Figure 6 show that GRASP-MSRP has the lowest average total deployment cost compared to other algorithms. For example, comparing GRASP-MSRP with $MaxIter = 1$ to CBS-MSRP with $MaxIter = 200$ for sink cost = 3 gives a p-value less than 0.0001. This is because GRASP-MSRP has the fewest sinks if we compare the average number of deployed sinks from Figure 7 to Figure 14. The evaluation also shows that we are able to trade-off GRASP-MSRP's runtime for a reduced cost when we increase the number of iterations from 1 to 10. When we further increase the number of iterations

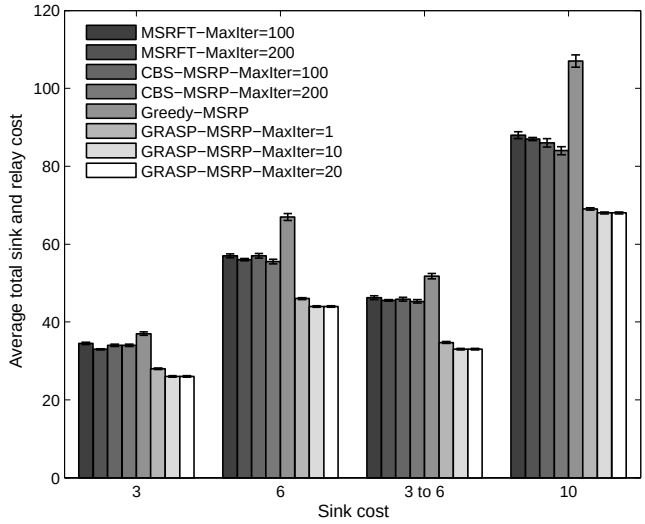

Fig. 6 Average total sink and relay cost for multiple sink and relay placement algorithms versus sink cost

Table 4 Multiple sink and relay placement algorithms' average runtime with different sink cost

Algorithms	**Average runtime (sec)**			
	$c_S = 3$	$c_S = 6$	$c_S = 3$ **to** 6	$c_S = 10$
MSRFT-MaxIter=100	61.7472	60.4956	61.7661	60.0905
MSRFT-MaxIter=200	74.6617	68.9032	69.6122	68.7917
CBS-MSRP-MaxIter=100	60.3329	62.7394	62.2207	61.6556
CBS-MSRP-MaxIter=200	68.2127	72.5465	73.7754	68.6779
Greedy-MSRP	137.3733	130.5268	139.4527	146.7378
GRASP-MSRP-MaxIter=1	20.2190	21.3075	26.6085	24.3780
GRASP-MSRP-MaxIter=10	194.8030	218.3831	254.7753	230.9070
GRASP-MSRP-MaxIter=20	405.8351	449.6994	534.1611	464.4079

to 20, the results are similar to the results of 10 iterations, but with higher runtime.

Greedy-MSRP is outperformed by the two k-means clustering-based algorithms, MSRFT and CBS-MSRP. Firstly in terms of the average total deployment cost, Greedy-MSRP deploys more sinks than MSRFT and CBS-MSRP as shown from Figure 10 to Figure 14. Secondly for the runtime, Greedy-MSRP is slower because it tries to find the lowest cost solution by solving the multiple sink and relay placement problem from the number of sinks = n to 2, where n is the number of sinks found by Greedy-MSP. On the other hand, MSRFT and CBS-MSRP start from the number of sinks = 2 and stop when the network is double-covered and non-critical.

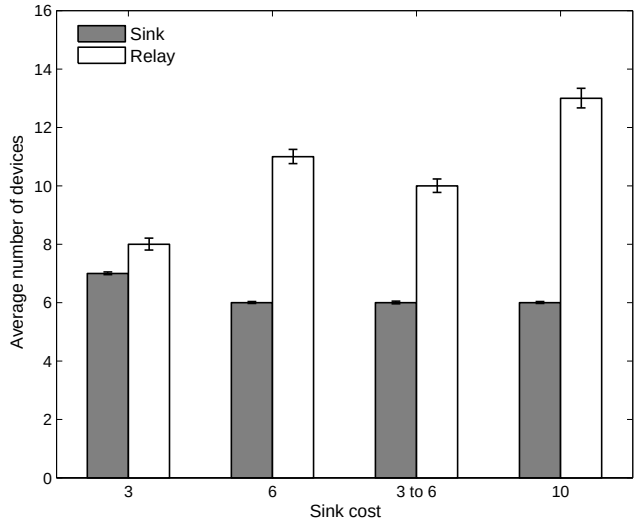

Fig. 7 Average numbers of sinks and relays for GRASP-MSRP with $MaxIter = 1$ versus sink cost

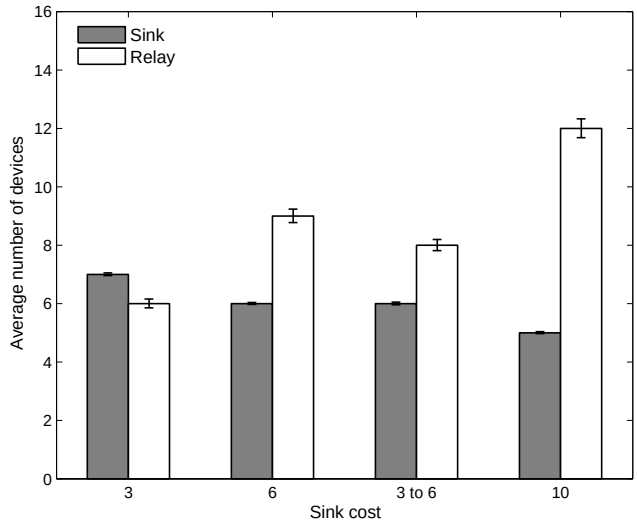

Fig. 8 Average numbers of sinks and relays for GRASP-MSRP with $MaxIter = 10$ versus sink cost

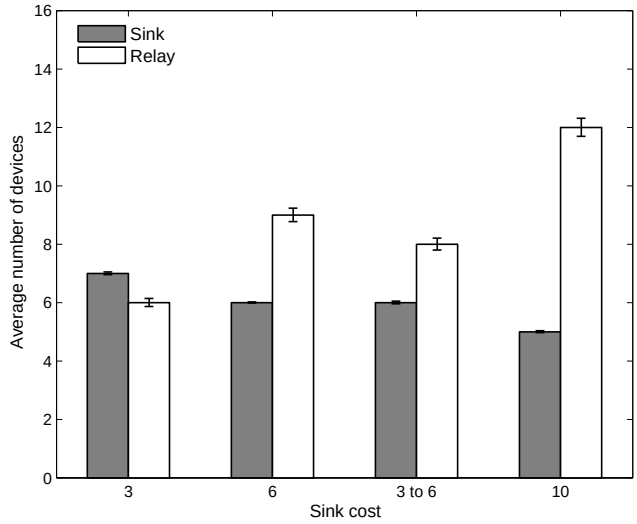

Fig. 9 Average numbers of sinks and relays for GRASP-MSRP with $MaxIter = 20$ versus sink cost

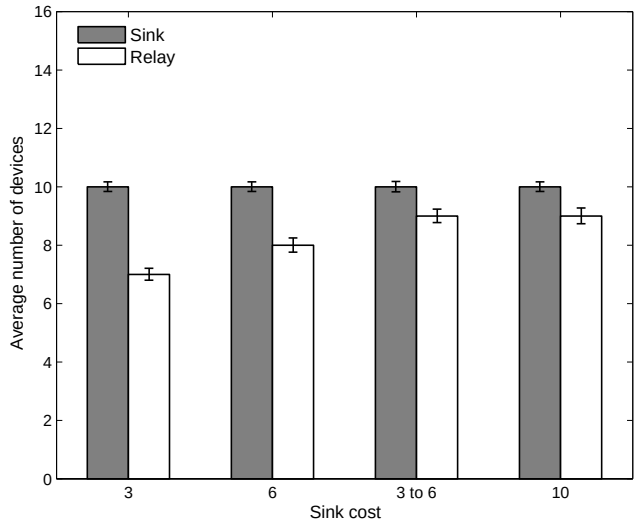

Fig. 10 Average numbers of sinks and relays for Greedy-MSRP versus sink cost

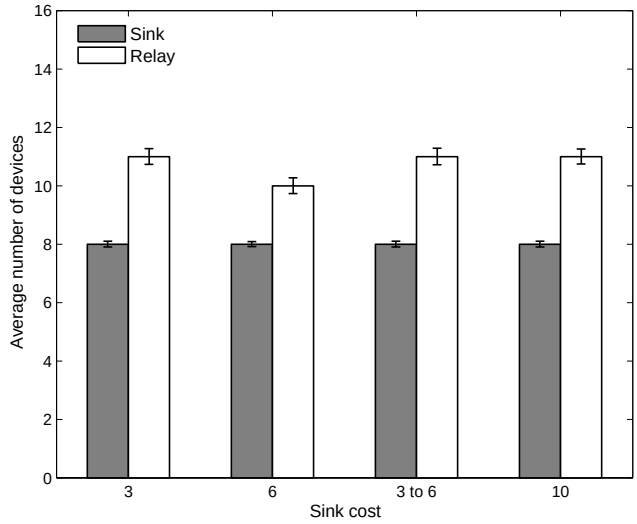

Fig. 11 Average numbers of sinks and relays for MSRFT with $MaxIter = 100$ versus sink cost

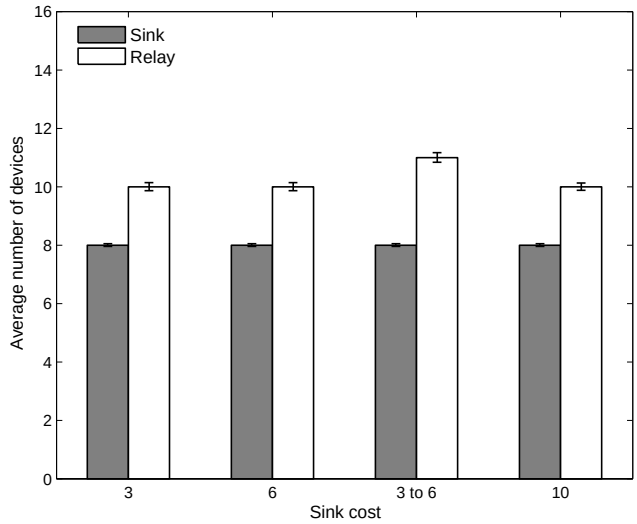

Fig. 12 Average numbers of sinks and relays for MSRFT with $MaxIter = 200$ versus sink cost

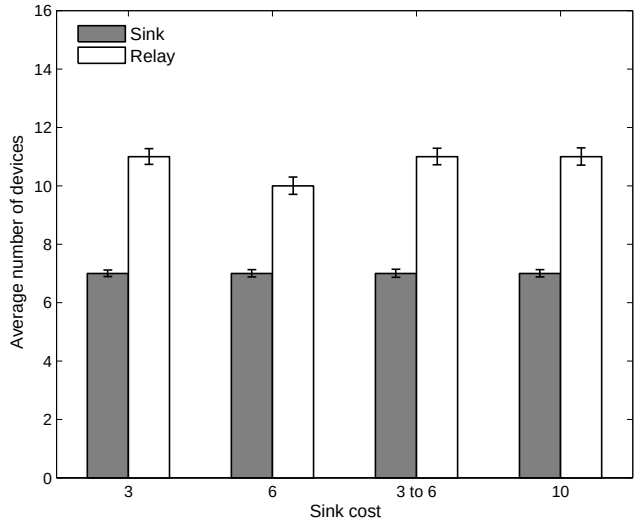

Fig. 13 Average numbers of sinks and relays for CBS-MSRP with $MaxIter = 100$ versus sink cost

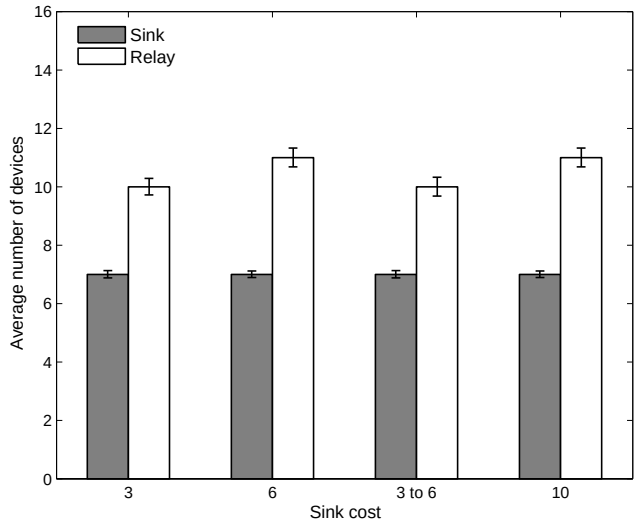

Fig. 14 Average numbers of sinks and relays for CBS-MSRP with $MaxIter = 200$ versus sink cost

GRASP-MSRP can swap sinks with relays to reduce the average total deployment cost. Therefore, when the sink cost increases, the number of sinks decreases because more sinks are exchanged with relays as shown in Figure 7 to Figure 9.

Since GRASP-MSRP gives the best solution for the multiple sink and relay placement problem, we further investigate its performance under various experiment settings by using $MaxIter = 10$. Firstly, we increase $l_{\max}$ from 6 to 10, while keeping the sink cost fixed at 3 units. The results are depicted in Figure 15. When $l_{\max}$ is increased from 6 to 10, the average number of required sinks drops significantly from 7 to 3, while the average number of relays does not increase much. The average runtime of GRASP-MSRP also increases from 194.8030 seconds to 349.7281 seconds.

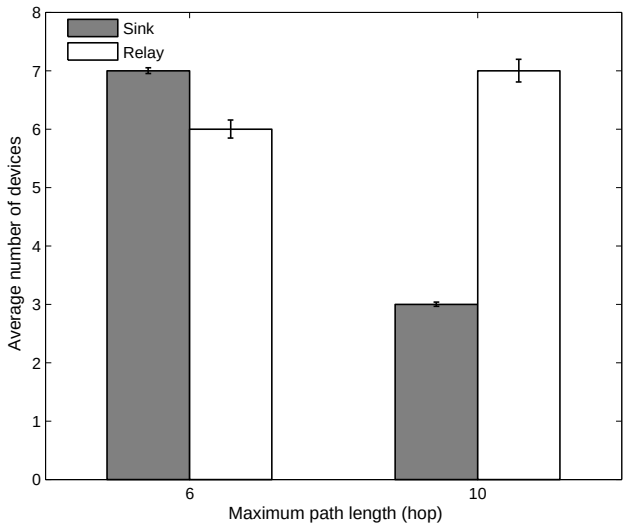

Fig. 15 Average numbers of sinks and relays for GRASP-MSRP versus maximum path length

We then try to increase the number of candidate relays from 81, which are evenly distributed in grid squares of 10m × 10m, to 196 candidate relays in grid squares of 6m × 6m. We use a fixed sink cost of 3 units and $l_{\max} = 6$. As shown in Figure 16, when we have more candidate relays, the average numbers of deployed sinks slightly decrease because the local search is more likely to find common relays that appear on the shortest paths. The average runtime of GRASP-MSRP increases from 194.8030 seconds for 81 candidate relays to 1030.0060 seconds for 196 candidates.

We also increase the number of candidate sinks from 25 in grid squares of 18m × 18m to 81 candidates in grid squares of 10m × 10m. The results are presented in Figure 17. When the number of candidate sinks increases, the

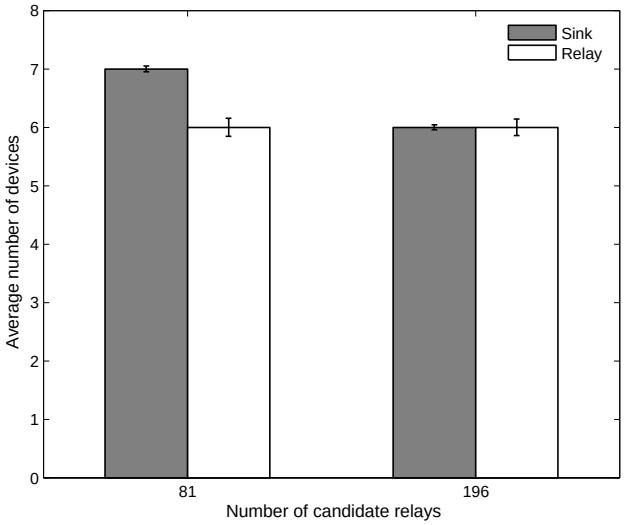

Fig. 16 Average numbers of sinks and relays for GRASP-MSRP versus number of candidate relays

local search can find better sink positions to cover a network. Therefore, the network requires fewer sinks. Since the sinks are better positioned, it also needs fewer relays. The average runtime of GRASP-MSRP increases from 194.8030 seconds for 25 candidate sinks to 843.7186 seconds for 81 candidates.

Finally, we evaluate the performance of GRASP-MSRP when the number of sensor nodes increases. In the first case, we increase the number of sensor nodes from 100 to 200 while keeping the area fixed. This affects the density of the network, where the average degree of a sensor node increases from 3 to 7. For this evaluation, we use 25 candidate sinks and 81 candidate relays. When the network becomes denser, the average numbers of required sinks and relays decrease as depicted in Figure 18. This happens because when the network density increases, path lengths from a sensor node to sinks become shorter and a sensor node has more neighbours that help finding alternate routes to sinks when it fails. However, GRASP-MSRP takes longer time to compute the solution, i.e. from 194.8030 seconds to 321.8777 seconds when the average degree increases.

In the second case, we increase the number of sensor nodes from 100 to 300 while keeping the average degree fixed, so we enlarge the area. As a result, we need more candidate sinks and relays. For this experiment, we use 81 candidate sinks and 225 candidate relays for 300 sensor nodes. When the network area becomes bigger, more sinks and relays are required to be deployed as shown in Figure 19. The average runtime of GRASP-MSRP increases from 194.8030 seconds to 20,534.8851 seconds.

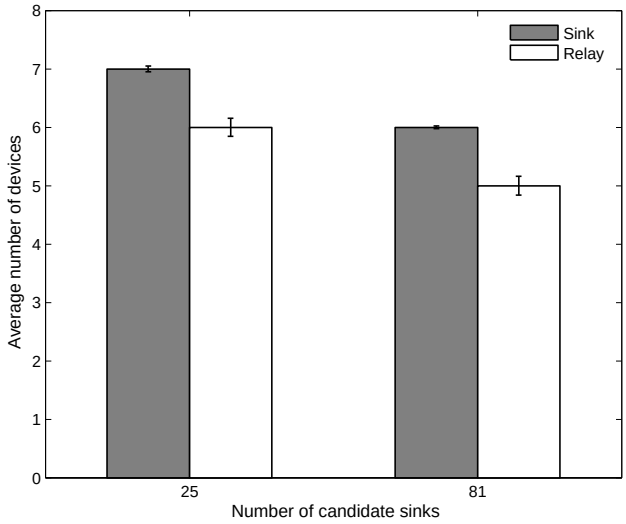

Fig. 17 Average numbers of sinks and relays for GRASP-MSRP versus number of candidate sinks

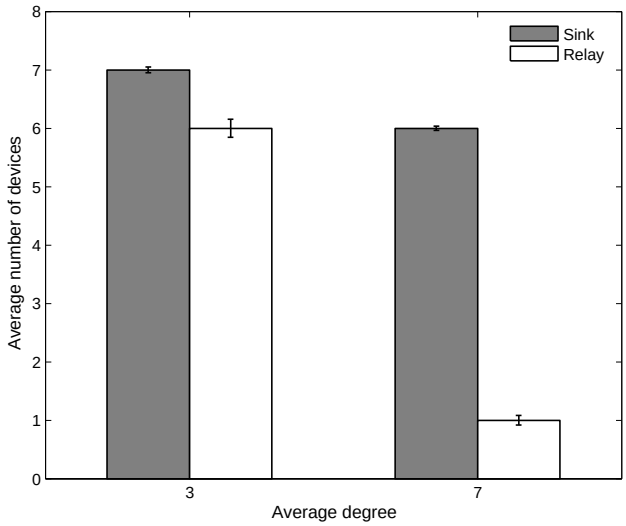

Fig. 18 Average numbers of sinks and relays for GRASP-MSRP versus average degree

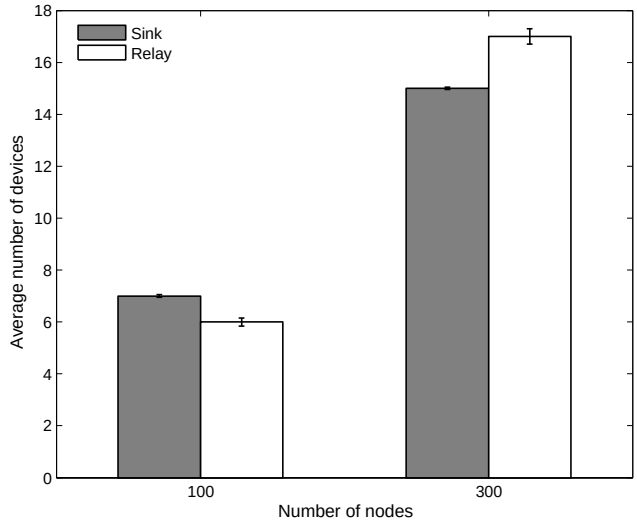

Fig. 19 Average numbers of sinks and relays for GRASP-MSRP versus number of nodes

Table 5 Multiple sink and relay placement algorithms' average runtime for 300 sensors, 81 candidate sinks and 225 candidate relays

Algorithms	**Average runtime (sec)**
MSRFT-MaxIter=100	4,601.2030
CBS-MSRP-MaxIter=100	4,482.8590
Greedy-MSRP	14,055.4217
GRASP-MSRP-MaxIter=1	2,020.4032
GRASP-MSRP-MaxIter=10	22,416.5930

For the case of 300 sensors, 81 candidate sinks and 225 candidate relays, we compare the deployment cost of MSRFT, CBS-MSRP, Greedy-MSRP and GRASP-MSRP in Figure 20 using one topology that is simulated 31 times. In this experiment, we only use 100 as the maximum iteration (*MaxIter*) for MSRFT and CBS-MSRP, 10 for GRASP-MRP, 1 and 10 for GRASP-MSRP. The sink and relay costs are 3 units and 1 unit, respectively, and $l_{\max} = 6$. Comparing the average total costs of GRASP-MSRP with $MaxIter = 1$ to MSRFT with $MaxIter = 100$ and to CBS-MSRP with $MaxIter = 100$ give p-values less than 0.0001. The average runtime is presented in Table 5.

We compare the runtime measurements for GRASP-MSRP with variety of network sizes. The topologies used in this experiment is summarised in Table 6 and the datasets are provided in [21]. In this experiment, we use $MaxIter = 1$, sink cost = 3 units, relay cost = 1 unit, and $l_{\max} = 6$. Each topology is simulated 31 times and the average runtime is depicted in Figure 21 with log scale in the y-axis.

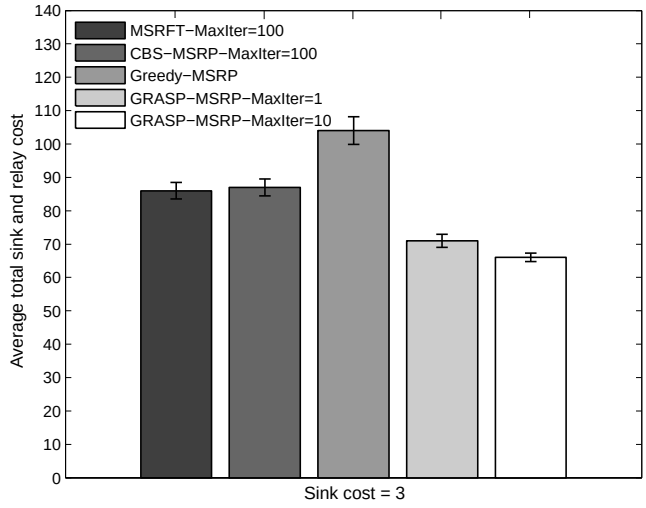

Fig. 20 Average total sink and relay cost for multiple sink and relay placement algorithms for 300 sensors, 81 candidate sinks and 225 candidate relays

Table 6 Topologies used to measure GRASP-MSRP's runtime with variety of network sizes

Num. sensors	Num. sinks	Num. relays	Area size
100	25	81	88m × 88m
200	36	144	128m × 128m
300	81	225	152m × 152m
400	100	289	168m × 168m
500	121	361	192m × 192m

8 Evaluation of Network Performance

The previous sections present and evaluate algorithms for solving the sink and relay placement problem, and we demonstrate how they can reduce the cost of providing solutions that meet the criteria which are based on protecting against single failures. The original high-level objective, though, was to design network topologies that were robust to failures. It is important to revisit that objective, and consider how well our problem definition and heuristic solutions satisfy it. To do this, we have implemented network protocols in a network simulator, and we test topologies generated by our algorithms while nodes fail.

We compare topologies generated by GRASP-MSRP and GRASP-ABP [22]. In GRASP-MSRP topologies, sinks and relays are deployed so all sensor nodes are double-covered and noncritical. With GRASP-ABP, four sinks are placed at the four corners of the networks and relays are deployed so all sensor nodes are noncritical, but not double-covered. In the simulation, we use topologies of

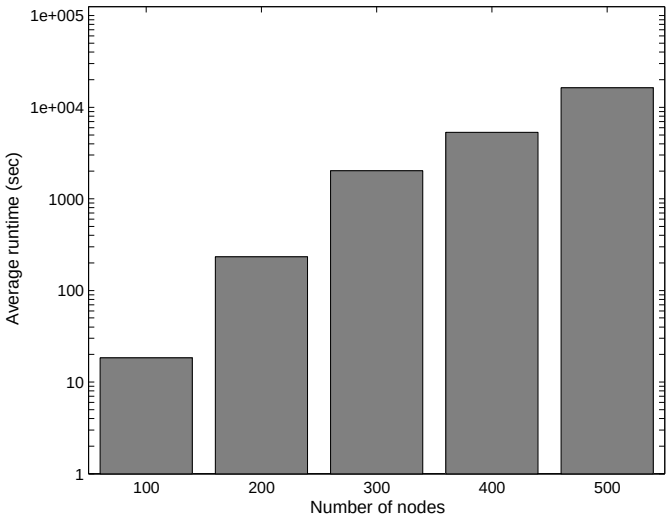

Fig. 21 GRASP-MSRP's average runtime with variety of network sizes

GRASP-MSRP with three and six sinks. To obtain the topologies with three sinks, we set ($l_{\max} = 10$), fix the sink cost and get the resulting topologies generated by GRASP-MSRP. Similarly for the case of six sinks, we set ($l_{\max} = 6$).

We take the resulting topologies and deploy, in simulation, sensor nodes, relay nodes and sinks according to the deployment plans. Then, we evaluate the network performance for each topology. We want to show that having more sinks and placing them at the best locations improves robustness of the networks. In the simulation, the GRASP-MSRP topologies with three sinks achieve around 5% improvement in connectivity and delivery ratio compared to the GRASP-ABP topologies with four sinks after several failures. This confirms the importance of not only having multiple sinks in the networks, but also placing them at the best positions.

In this section, we use the following metrics for the evaluation of network performance:

1. ***Packet delivery ratio*** measures the number of packets successfully received by the sink over the number of packets generated by source nodes.
2. ***Average per packet latency*** measures the average per packet transmission time through a multi-hop network.
3. ***Connectivity*** measures the percentage of live sensor nodes that are still connected to the sink through multi-hop communication.

Table 7 Simulation parameters in Ns-2

Simulation parameters	Default value
Hardware	Tmote sky
Transmit power	52.2 mW
Receive/idle listening power	59.1 mW
Sleep power	3 μW
Radio propagation model	Two-ray ground
CSThresh_	3.65262e-10
RXThresh_	3.65262e-10
Pt_	5.35395e-05
Traffic load	0.1 packets/node/sec
MAC protocol	ER-MAC
TDMA slot size	50 ms
TDMA sub-slot size	5 ms

8.1 Details of Simulation

The topologies are evaluated in the open-source network simulator ns-2 version 2.33 [26]. Ns-2 is a discrete-event, packet-level network simulator that is widely used for WSN and other network simulations. We simulate multi-hop data gathering using ER-MAC [24] [25] with its forward-to-parent routing mechanism. We choose ER-MAC because of its flexibility to adapt to topology changes, where nodes can modify their schedules and routing decisions locally.

ER-MAC is a hybrid MAC protocol designed for emergency response WSNs. It adopts a time-division multiple access (TDMA) approach to schedule collision-free transmission toward the sink. With its *normal mode*, sensor nodes wake up to transmit and receive messages according to their schedules, but otherwise switch into power-saving sleep mode. When an emergency event occurs, nodes involved in the emergency monitoring change their MAC protocol autonomously to *emergency mode* to allow contention in TDMA slots to cope with large volumes of traffic. Because our purpose in this simulation is to evaluate the designed topologies, not the communication protocol, we assume that all nodes operate in the normal mode of ER-MAC, where nodes can only send packets in their own transmit slots.

Our parameters used in the ns-2 simulation are based on Tmote sky hardware [17] as shown in Table 7. Tmote Sky's transmission ranges and power were reported in [11]. We used the Lagrange Interpolating Polynomial [2] based on the known six points to find transmission power for the transmission range used in our simulation, i.e. 10 metres.

The simulation results presented are based on the average of five topologies that are simulated 31 times each, enough to achieve a 95% confidence in the standard error interval. Note that we do not show error bars in line graphs to improve their readability. In each simulation, we simulate a data gathering, where all sensor nodes are the source nodes that generate packets with a fixed interval. They also forward other nodes' packets toward the sink. Source nodes

generate one packet every 10 seconds. Therefore, the traffic load is 0.1 packets/node/sec. Relay nodes do not generate packets, but only forward them, and are used from the start of the simulation. Our works assume point-based failures, where the failed devices are scattered in the network. We do not assume that relay nodes are more robust than sensor nodes, so they too may fail during the simulation period. During the simulation, we increase the number of dead nodes gradually by killing one node, either a sensor node or a relay node, in each time step.

Since a common problem is node death due to energy depletion, which is either caused by normal battery discharge, short circuits or leakage due to broken packaging [4], we consider the probability of node death in our simulation to be proportional to the work done. That is, instead of selecting dead nodes randomly, we bias the node death towards the nodes that have done most work. Firstly, we list all live nodes (sensors and relays) in increasing order of energy consumption, count the total energy used, and calculate a weight for each node by using the ratio of own-energy to the total-energy. We then use these weights to generate an array of numbers between 0 and 1 representing bins of different size. The weight of a bin that associates with node a is the total weight of all nodes whose weight $\leq a$'s weight, including a. To select a node to be killed, we generate a random number between 0 and 1, and select the first bin whose weight is larger than the generated number. Doing it this way means, for example, if node a has 3 times the energy consumption of node b, the bin size for node a is always 3 times the bin size for node b, and so is always 3 times as likely to be killed. With this model, it is likely that nodes closer to the sink are the first to die because they have much higher relay loads and drain their batteries quickly.

8.2 Evaluation of Network Topologies with Multiple Sources and Multiple Sinks

In each simulation, we simulate a data gathering for 3,000 seconds and kill one node every 250 seconds, either a sensor node or a relay node, starting from the 250^{th} second. We gather statistical data every 250 seconds and plot the results. Therefore, the statistics after one node dies are plotted at the 500^{th} second.

The delivery ratio while nodes are failing is depicted in Figure 22, where the GRASP-MSRP topologies with six sinks achieve the highest ratio. The second highest delivery ratio is not achieved by the topologies of GRASP-ABP with four sinks, but by GRASP-MSRP with three sinks. The delivery ratio gap between these two simulation results is around 5% after several failures. From this simulation, we not only show that having more sinks gives us better performance, but also that placing them at the best locations is more important.

The latency of high priority packets is shown in Figure 23. As expected, the topologies of GRASP-MSRP with six sinks have the lowest latency, i.e.

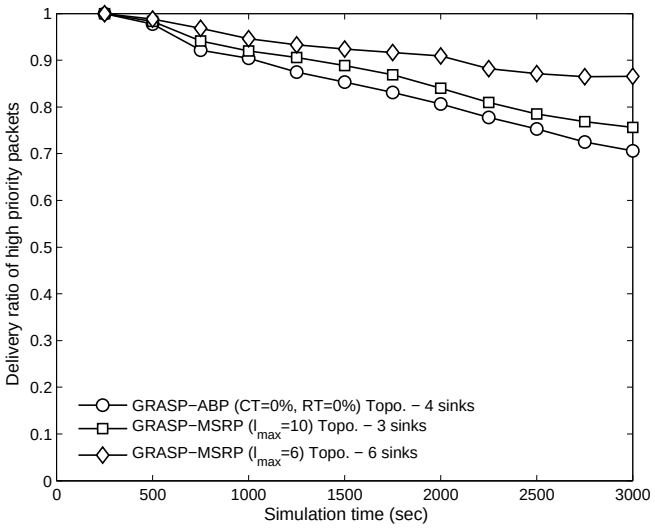

Fig. 22 Delivery ratio where a node dies every 250 seconds

around 0.6 second, followed by the topologies with three sinks, i.e. 2 seconds in average. The latency of the GRASP-ABP topologies with four sinks is slightly higher than 3 seconds because most of the nodes have longer paths when the sinks are deployed at the corners of the networks.

The network connectivity for this simulation set is presented in Figure 24. This results correspond with the delivery ratio as shown in Figure 22, where the topologies of GRASP-MSRP with six sinks offer the best performance, followed by the topologies with three sinks. The GRASP-ABP topologies with four sinks have 5% lower connectivity than the topologies of GRASP-MSRP with three sinks after several failures due to the sinks' positions. In this simulation, we show that if we have higher budget to deploy more sinks, we can get better network performance, especially if we place them at the best locations.

9 Conclusion

We define the problem of increasing network robustness by protecting it against one single failure, of either a sink or a sensor node. We design a network to be double-covered and non-critical. Double-covered means each sensor node must have at least two length-bounded paths to two sinks. Non-critical means all sensor nodes must have at least one length-bounded path to a sink after any single sensor node failure. Our novel contributions are solutions to minimise the deployment cost of sinks and relays.

We firstly look at the multiple sink placement problem and propose Greedy-MSP and GRASP-MSP to minimise the total sink cost. Both algorithms solve

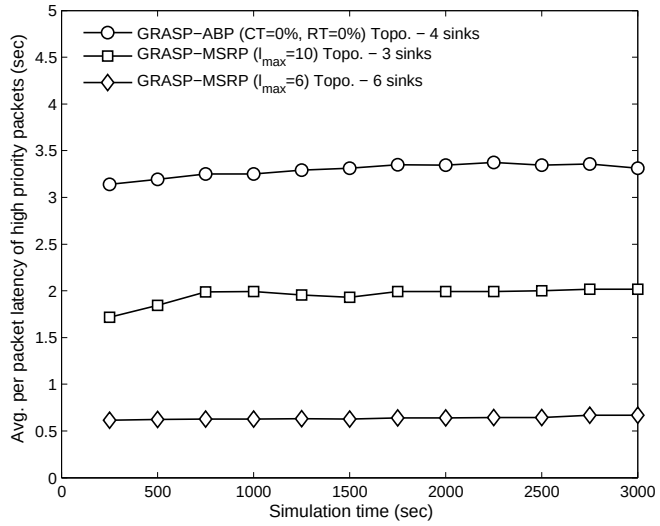

Fig. 23 Latency where a node dies every 250 seconds

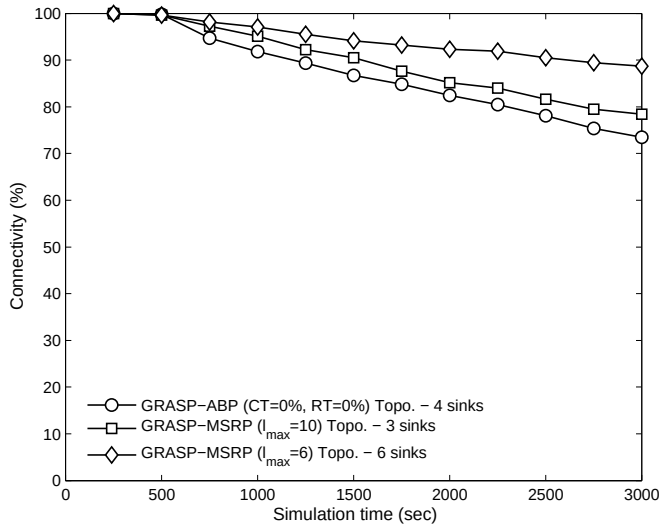

Fig. 24 Connectivity where a node dies every 250 seconds

the multiple sink placement problem by ensuring that each sensor node in the network is double-covered. Empirically, Greedy-MSP has the shortest runtime, but GRASP-MSP achieves comparable cost to the linear programming solution with shorter runtime. GRASP-MSP's results justify the use of local search to solve the multiple sink and relay placement problem, where a linear programming solution is not available.

We then propose Greedy-MSRP and GRASP-MSRP to solve the multiple sink and relay placement problem, where we want the designed topologies to be double-covered and non-critical. Our results show that the k-means clustering-based algorithms outperform Greedy-MSRP in terms of lower cost solution and shorter runtime because they have better sink positions. On the other hand, GRASP-MSRP outperforms the other algorithms with the lowest cost solutions and the shortest runtime. The GRASP-MSRP results also show that more sinks are exchanged with relays when the sink cost increases to reduce the total deployment cost.

In this paper, if both the number of uncovered nodes and the number of critical nodes in a network are zero, we guarantee robustness against single failure. However, we do also benefit when there are multiple failures. In a multiple sink network, the performance of the network is not only influenced by the number of deployed sinks, but more importantly the positions to deploy the sinks. We show in simulations that the best performance is achieved by topologies with six sinks where the placements were optimised (GRASP-MSRP). Note that topologies with only three sinks but where the placements were optimised (GRASP-MSRP) are better than topologies with four sinks in fixed positions (GRASP-ABP).

Our future work will include evaluation for larger networks with more sensor nodes, and more candidate locations for sinks and relays. We will also compare the network performance of topologies resulted from other algorithms.

Acknowledgements This research is funded by the Irish Higher Education Authority PRTLI-IV research program through the NEMBES project and by the Science Foundation Ireland as part of the CTVR project (SFI CSET 10/CE/I1853).

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless Sensor Networks: A Survey. Computer Networks **38**(4), 393–422 (2002)
2. Archer, B., Weisstein, E.W.: Lagrange Interpolating Polynomial. MathWorld–A Wolfram Web Resource. (2011). Available at `http://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html` [1 September 2011]
3. Bavelas, A.: A Mathematical Model for Group Structure. Human Organizations **7**, 16–30 (1948)
4. Beutel, J., Römer, K., Ringwald, M., Woehrle, M.: Deployment Techniques for Wireless Sensor Networks. In: G. Ferrari (ed.) Sensor Networks: Where Theory Meets Practice, pp. 219–248. Springer (2009)
5. Brandes, U.: On Variants of Shortest-Path Betweenness Centrality and Their Generic Computation. Social Networks **30**(2), 136–145 (2008)

6. Bredin, J.L., Demaine, E.D., Hajiaghayi, M., Rus, D.: Deploying Sensor Networks with Guaranteed Capacity and Fault Tolerance. In: Proc. 6th ACM Int'l Symp. Mobile Ad Hoc Networking and Computing (MobiHoc'05), pp. 309–319 (2005)
7. Cambazard, H., Mehta, D., O'Sullivan, B., Quesada, L., Ruffini, M., Payne, D., Doyle, L.: A Combinatorial Optimisation Approach to the Design of Dual-Parented Long-Reach Passive Optical Networks. In: Proc. 23rd IEEE Int'l Conf. Tools with Artificial Intelligence (ICTAI'11), pp. 785–792 (2011)
8. Feo, T.A., Resende, M.G.C.: A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. Operations Research Letters **8**, 67–71 (1989)
9. Feo, T.A., Resende, M.G.C.: Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization **6**, 109–133 (1995)
10. Freeman, L.: Centrality in Social Networks Conceptual Clarification. Social Networks **1**(3), 215–239 (1979)
11. Guo, Y., Kong, F., Zhu, D., Tosun, A.S., Deng, Q.: Sensor Placement for Lifetime Maximization in Monitoring Oil Pipelines. In: Proc. 1st ACM/IEEE Int'l Conf. Cyber-Physical Systems (ICCPS), pp. 61–68 (2010)
12. Han, X., Cao, X., Lloyd, E.L., Shen, C.C.: Fault-tolerant Relay Node Placement in Heterogeneous Wireless Sensor Networks. IEEE Trans. on Mobile Computing **9**(5), 643–656 (2010)
13. Kashyap, A., Khuller, S., Shayman, M.: Relay Placement for Fault Tolerance in Wireless Networks in Higher Dimensions. Computational Geometry: Theory and Applications **44**(4), 206–215 (2011)
14. Lanza-Gutierrez, J.M., Gomez-Pulido, J.A., Vega-Rodriguez, M.A., Sanchez-Perez, J.M.: A Parallel Evolutionary Approach to Solve the Relay Node Placement Problem in Wireless Sensor Networks. In: Proc. 15th Ann. Conf. Genetic and Evolutionary Computation (GECCO'13), pp. 1157–1164 (2013)
15. Mahmud, S., Wu, H., Xue, J.: Efficient Energy Balancing Aware Multiple Base Station Deployment for WSNs. In: Proc. 8th European Conf. Wireless Sensor Networks (EWSN'11), pp. 179–194 (2011)
16. Misra, S., Hong, S.D., Xue, G., Tang, J.: Constrained Relay Node Placement in Wireless Sensor Networks to Meet Connectivity and Survivability Requirements. In: Proc. 27th Ann. IEEE Conf. Computer Communications (INFOCOM'08), pp. 281–285 (2008)
17. Moteiv: Tmote Sky Datasheet (2010). Available at `http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf` [30 April 2010]
18. Oyman, E.I., Ersoy, C.: Multiple Sink Network Design Problem in Large Scale Wireless Sensor Networks. In: Proc. IEEE Int'l Conf. Communications (ICC'04), pp. 3663–3667 (2004)
19. Pu, J., Xiong, Z., Lu, X.: Fault-Tolerant Deployment with k-connectivity and Partial k-connectivity in Sensor Networks. Wireless Communications and Mobile Computing **9**(7), 909–919 (2008)
20. Resende, M.G.C., Ribeiro, C.C.: Greedy Randomized Adaptive Search Procedures. In: F. Glover, G. Kochenberger (eds.) State of the Art Handbook in Metaheuristics, pp. 219–249. Kluwer Academic Publishers (2002)
21. Sitanayah, L.: GRASP-MSRP Dataset (2014). Available at `http://www.cs.ucc.ie/~ls3/grasp_msrp/dataset/` [25 June 2014]
22. Sitanayah, L., Brown, K.N., Sreenan, C.J.: Fault-Tolerant Relay Deployment Based on Length-Constrained Connectivity and Rerouting Centrality in Wireless Sensor Networks. In: Proc. 9th European Conference on Wireless Sensor Networks (EWSN'12), pp. 115–130 (2012)
23. Sitanayah, L., Brown, K.N., Sreenan, C.J.: Multiple Sink and Relay Placement in Wireless Sensor Networks. In: Proc. 1st Workshop Artificial Intelligence for Telecommunications and Sensor Networks (WAITS'12), 20th European Conf. Artificial Intelligence (ECAI'12), pp. 18–23 (2012)
24. Sitanayah, L., Sreenan, C.J., Brown, K.N.: ER-MAC: A Hybrid MAC Protocol for Emergency Response Wireless Sensor Networks. In: Proc. 4th Int'l Conf. Sensor Technologies and Applications (SENSORCOMM'10), pp. 244–249 (2010)
25. Sitanayah, L., Sreenan, C.J., Brown, K.N.: A Hybrid MAC Protocol for Emergency Response Wireless Sensor Networks. Ad Hoc Networks **20**, 77–95 (2014)

26. VINT: The Network Simulator - ns-2 (2010). Available at `http://www.isi.edu/nsnam/ns/` [30 April 2010]
27. Wenming, S., Chuanhe, H., Mingkai, S., Yong, C., Zhe, C.: Indoor Localization Scheme in Wireless Sensor Networks Using Spatial Information. In: Proc. Int'l Conf. Wireless Communications, Networking and Mobile Computing (WiCOM'06), pp. 1–5 (2006)
28. Xu, X., Liang, W.: Placing Optimal Number of Sinks in Sensor Networks for Network Lifetime Maximization. In: Proc. IEEE Int'l Conf. Communications (ICC'11) (2011)
29. Youssef, W., Younis, M.: Intelligent Gateway Placement for Reduced Data Latency in Wireless Sensor Networks. In: Proc. IEEE Int'l Conf. Communications (ICC'07), pp. 3805–3810 (2007)