Solving coverage problems in wireless sensor networks using cover sets

Dimitrios Zorbas, Dimitris Glynos, Panayiotis Kotzanikolaou and Christos Douligeris

jim@students.cs.unipi.gr

Abstract

To achieve power efficient monitoring of targets by sensor networks, various coverage algorithms have been proposed. These algorithms divide the sensor nodes into *cover sets*, where each cover set is capable of monitoring all targets. Generating the maximum number of cover sets has been proven to be an NP-complete problem and, thus, algorithms producing sub-optimal solutions have been proposed. In this paper we present a novel and efficient coverage algorithm, that can produce both *disjoint* cover sets, i.e. cover sets with no common sensor nodes, as well as *non-disjoint* cover sets. While searching for the best sensor to include in a cover set, our algorithm uses a cost function that takes into account the monitoring capabilities of a sensor, its association with poorly monitored targets, but also the sensor's remaining battery life. Through simulations, we show that the proposed algorithm outperforms similar heuristic algorithms found in the literature, producing collections of cover sets of optimal (or near-optimal) size. The increased availability offered by these cover sets along with the short execution time of the proposed algorithm make it desirable for a wide range of node deployment environments.

1 Introduction

Wireless Sensor Networks (WSNs) are used extensively in many domains, such as military applications [2], environmental monitoring [13], target surveillance [12] and disaster prevention [11]. Each WSN node is equipped with at least one sensing module and one communication module, which enable the collection of monitoring data and their transmission to a base station, which is situated in close proximity to the WSN. The base station may be a desktop computer or a laptop.

Depending on the application and the environment, node deployment and placement can be realised either in a predefined way or randomly. In hostile environments, for example, sensors may be dropped from an aeroplane, resulting in a random placement, where the node density cannot be guaranteed; some areas may contain more sensors than others.

WSN nodes have three modes of operation. In the *active* mode, a sensor can observe the environment and communicate with other sensors (or with the base station). In the *sleep* mode, a sensor cannot monitor or transmit data. The node can change to the active mode, whenever it receives the appropriate signal (either from another sensor or from the base station). Obviously, in the sleep mode a sensor consumes much less energy than in the active mode. Finally, in the *off* mode, the nodes are completely turned off.

Sensors have a limited battery life. If all sensors in a terrain were to be activated at the same time, the lifetime of the network would be equal to the lifetime of a single sensor (supposing that all sensors have almost equal energy resources), say h hours. Clearly, if the uptime of each sensor could be programmed in an efficient and practical way, the lifetime of the WSN could be possibly extended. In order to achieve this lifetime extension, the nodes must be divided into a number of subsets, called *cover sets*, where each cover set is capable of covering all the monitored targets. Sensors belonging to a scheduled cover set are in active mode, while the others are in sleep mode. If the cover sets are *disjoint*, then each sensor is allowed to participate only in one cover set, and, thus, each set has a maximum lifetime of h hours. By periodically switching between cover sets, we can extend the target coverage time to $|C| \cdot h$, where $|C|$ is the number of available cover sets. If the cover sets are *non–disjoint* sets, then any given sensor may participate in more than one cover sets. In this case, the time spent on each scheduled cover set is shortened so as to allow for sensors to participate in multiple sets. In several sensor deployment scenarios, it has been shown that the use of non-disjoint cover sets, may increase the lifetime of the network, if proper scheduling algorithms are used [6].

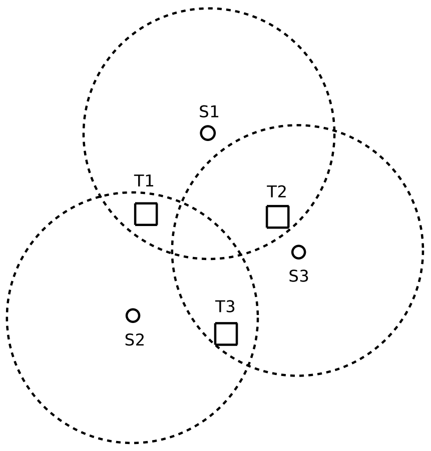


Figure 1: Example topology for sensors and monitored targets

To illustrate the above observation, let us assume that the three targets (T_1, T_2, and T_3) in Figure 1 lie on a field that can be monitored by three sensor nodes (S_1, S_2, and S_3). Node S_1 covers targets T_1 and T_2, node S_2

covers T_1 and T_3 and node S_3 covers T_2 and T_3. If all the sensor nodes were to be activated simultaneously, then the network lifetime would be equal to the standard lifetime h of a single sensor. By dividing the sensors into disjoint sets, the resulting network lifetime would still be h, since for this topology a disjoint algorithm can produce only one cover set (e.g. $C_1 = \{S_1, S_2\}$).

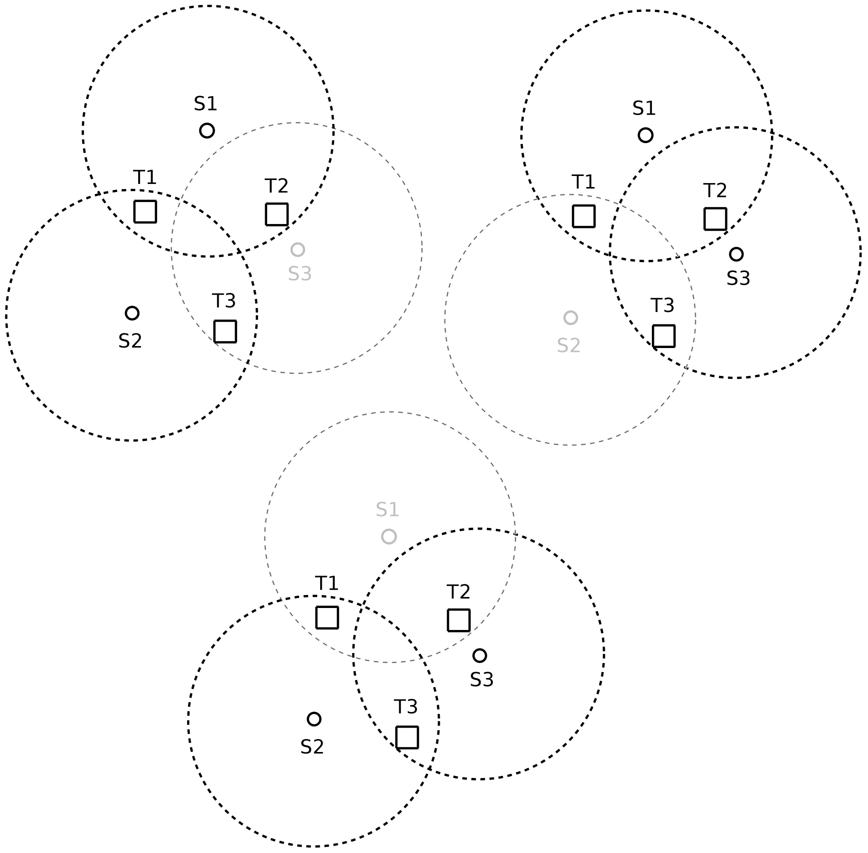


Figure 2: Three generated non-disjoint cover sets

However, if a sensor node in Figure 1 can be part of two cover sets, then the network lifetime can be extended. By creating three non-disjoint cover sets (see Figure 2), each one activated for $0.5 \cdot h$ hours, the total network lifetime can be extended to $1.5 \cdot h$, assuming that the energy consumption during the *sleep* mode is negligible.

Coverage problems are divided into *point coverage* and *area coverage*. The objective in *point coverage* is to cover (monitor) a set of points with a subset of the available sensor nodes [4]. Every point is monitored by at least one sensor. In the *area coverage* problem, the monitored space is divided into smaller areas called *fields* [14]. Each field is uniquely identified by a set of covering sensors that completely cover this field. The goal in area coverage is to produce cover sets of sensors that completely cover all the fields and, consequently, the entire area. As described in [4], the area coverage problem is closely linked to the point coverage problem, since the fields are equivalent to the points mentioned previously[1].

In this paper, we propose a novel centralised coverage algorithm, which can produce both disjoint and non–disjoint cover sets. The algorithm is based on a generic methodology for centralised coverage algorithms that we analyse in Section 3. Its main characteristic is a node selection policy that takes into account the monitoring capabilities of a sensor, its association with poorly monitored targets, but also the sensor's remaining battery life. This policy, described in Section 4, allows for the generation of cover sets in an efficient manner. Furthermore, we simulate the proposed algorithm and compare it to the algorithms of [14] and [6], with respect to the number of generated sets and execution time. Our findings, presented in Section 5, demonstrate results close to the optimal solution in the number of generated sets. Moreover, the proposed algorithm shows improved performance in comparison to similar approaches found in the literature.

Thus, the proposed algorithm extends the availability of sensor networks in two aspects. First, by extending the number of cover sets, it increases the actual coverage time; and, secondly, by producing multiple cover sets it provides high redundancy in terms of target coverage. The latter can be a very desirable characteristic, especially in hostile node deployment environments.

2 Related Work

Cardei and Wu [7] provide a taxonomy of sensor coverage algorithms according to several design criteria, such as:

1. The coverage *objective*, *i.e.* to maximise the lifetime of the network or to minimise the required number of deployed sensors.
2. The *node deployment method*, which may be random or deterministic.
3. The *homogeneous or heterogeneous nature* of the nodes, *i.e.* whether all nodes have a common sensing or communication range.
4. The degree of *centralisation*, *i.e.* centralised vs distributed algorithms, and
5. *additional requirements* for energy efficiency and connectivity.

An additional criterion that can be added to the above list is the *cover set independence*, *i.e.* whether a node appears in exactly one of the generated sets (as in the case of node disjoint coverage algorithms) or not (as in the case of non-disjoint coverage algorithms).

[1]The term *target coverage* is used throughout this text to refer to either area or point coverage scenarios.

Below we describe existing coverage algorithms that can be used in random sensor deployment scenarios with homogeneous device characteristics in terms of battery lifetime.

2.1 Centralised Coverage Algorithms

In a *centralised* coverage algorithm the monitoring schedule is first calculated on the base station and it is then sent to the sensor nodes for execution. The advantage of this scheduling approach is that it requires very low processing power from the sensor nodes, which usually have limited processing capabilities.

2.1.1 Disjoint Centralised Algorithms

Slijepcevic and Potkonjak [14] propose a disjoint centralised algorithm for the area coverage problem. They introduce the idea of the *field* as a set of targets. Two targets belong to the same field if and only if they are covered by the same set of sensors. In particular, the fields are small areas which are produced by the intersection of the coverage limits of sensors and/or the physical limits of the monitored terrain. Every sensor may cover one or more fields and one field is covered by at least one sensor. Their proposed algorithm initially covers the more sparsely populated fields (i.e. the fields that are covered by the smallest number of sensors), which are called *critical* fields. After the selection of a node that covers a critical field, the algorithm excludes all other nodes covering the same field. Thus, it is assured (during the construction of a cover set) that only one node covering a particular critical field shall be selected. The authors state that the complexity of the algorithm is $O(n^2)$, where n is the total number of sensors.

Cardei *et al.* [5] propose an algorithm to solve the same problem using graphs. They construct an undirected graph $G = (V, E)$, where V is the set of sensors and E the set of edges, such that the edge $(u, v) \in E$ if and only if u and v are within each other's sensing range. The goal is to find the maximum number of "dominating sets". To achieve this the authors use a graph colouring technique. As depicted in [15], despite the production of more sets than [14], the dominating sets do not guarantee the coverage of the whole area. The complexity of the heuristic which computes the disjoint sets from the coloured graph is $O(n^3)$.

Cardei and Du [4] propose an algorithm in order to solve the random target coverage problem. This problem is successively formulated as an area coverage problem, which is proved to be an NP-Complete problem. Cardei and Du define the disjoint-set coverage problem, that was first introduced by Slijepcevic and Potkonjak [14], as a generalisation of the 3-SAT problem [9]. They propose a heuristic to compute the disjoint sets. In order to compute the maximum number of covers, they transform the problem into a maximum-flow problem. Then, the result of the maximum-flow problem is solved using *Mixed Integer Programming*, which heuristically produces the final number of cover sets. The results in [4] show a slight improvement in the number of produced sets in comparison to [14] but there is a substantial delay in execution time. The complexity of this algorithm depends on the complexity of the mixed integer programming technique used.

Abrams *et al.* [1] propose a centralised algorithm utilising randomisation to solve the coverage problem. Each generated cover set does not provide complete coverage of targets and the cover sets must be scheduled successively in order to achieve an 80% coverage of the monitored areas. The complexity of this algorithm is $O(nm|P_{max}|)$, where n is the number of sensors available, m the number of generated cover sets and $|P_{max}|$ the maximum number of fields that a sensor covers.

2.1.2 Non-disjoint Centralised Algorithms

In the case of non-disjoint algorithms, nodes may participate in more than one cover sets. In some cases, this may prolong the lifetime of the network in comparison to the disjoint cover set algorithms, but it incurs a higher complexity.

Cardei *et al.* [6] propose a *Linear Programming* (LP) solution to the target coverage problem for non-disjoint cover sets. Although the LP algorithm presents a high complexity $O(m^3n^3)$, where m is the number of covers and n the number of sensors, the authors also propose a greedy algorithm, with a lower complexity $O(dk^2n)$, where d is the number of sensors that cover targets that are associated with a minimum number of sensors and k is the number of targets.

Another LP technique is proposed by Berman *et al.* [3]. The authors first compute a series of cover sets and then they deduce the optimal lifetime for each cover set. Their approach is based on the $(1+\epsilon)$-approximation of the Garg and Könemann algorithm [10], with an approximation factor of $(1+\epsilon)(1+2\ln n)$ for any $\epsilon > 0$.

2.2 Distributed Coverage Algorithms

In *distributed* coverage algorithms a number of sensor nodes perform the required calculations cooperatively and, then, these nodes disseminate the scheduling information to the rest of the sensors. These schemes may require some processing by the sensors involved, but they scale better to accommodate larger networks.

Several distributed algorithms have been proposed in the literature [16, 18, 17, 8]. These approaches use a distributed and localised scheduling scheme, in order to rapidly disseminate the scheduling information throughout the network. The nodes decide cooperatively in an effective way, which of them will remain in sleep mode for a certain period of time. There are extra costs associated with this type of algorithms, due to the increased overhead in message exchanges and the need for synchronisation of participating nodes. Note that in this paper we focus on centralised algorithm problems and, thus, distributed algorithms are outside the scope of our work.

3 Problem Description

In this section, we describe a generic model for centralised coverage algorithms. This model is suitable for *full coverage* algorithms, i.e. algorithms that produce cover sets capable of monitoring all targets, provided that each target is monitored by at least one sensor. In the following section, we will use this model in order to describe the proposed solution.

3.1 Problem Parameters

Let $T_0 = \{t_1, t_2, \ldots, t_k\}$ be the set of targets and $S_0 = \{s_1, s_2, \ldots, s_n\}$ the set of sensor nodes. Each target in T_0 is covered by at least one sensor node in S_0.

Each sensor may have a number of sensing capabilities. The minimum sensing range of these capabilities is R_s. We assume that any target lying within the circle defined by the location of a sensor (circle centre) and range R_s (circle radius) can be monitored by this sensor.

We call N_i the set of "neighbour-sensors" of target t_i. Each neighbour sensor $s_j \in N_i$ is capable of monitoring the target t_i, *i.e.*:

$$\forall\ s_j \in N_i : |t_i - s_j| \leq R_s, N_i \subseteq S_0, t_i \in T_0,$$

where $|t_i - s_j|$ denotes the distance between target t_i and sensor s_j.

The input I of a coverage algorithm is a set of tuples of the form:

$$I = \{(t_1, N_1), \ldots, (t_k, N_k)\}, t_i \in T_0, k = |T_0|,$$

where N_i is the set of "neighbour-sensors" of target t_i. The set I contains one tuple per target in T_0. It must be noted that a specific sensor may appear in more than one "neighbour-sensor" sets.

The coverage algorithm produces a collection $C = \{C_1, \ldots, C_m\}$ of m "cover sets". Each cover set C_p is a subset of the available sensors ($C_p \subseteq S_0$) and it must cover all targets found in T_0. The number of occurrences w ($w \in \mathbb{N}^*$) of a sensor in the output cover sets, depends on the type of the algorithm. In a *node-disjoint* algorithm, each sensor is allowed to participate only in one cover set ($w = 1$), *i.e.*:

$$\forall\ i, j : C_i \cap C_j = \emptyset,\ i, j \in [1, m], i \neq j.$$

For *non-disjoint* algorithms, a node can be part of multiple cover sets, i.e. $w > 1$.

The objective of a coverage algorithm is to extend the lifetime of the network by maximising $|C|$, where $|C|$ is the cardinality of the generated collection C of cover sets.

3.2 Generating Cover Sets

As described in Section 2, there exist a number of approaches that deal with the cover set generation problem in sensor networks. One of these, the greedy heuristic approach, examines only a subset of all possible solutions of a given instance of the problem, but it is capable of producing near-optimal results in a relatively short amount of time. In this subsection we present the steps followed by a greedy heuristic algorithm when generating cover sets.

Figure 3 shows the general structure of a centralised greedy coverage algorithm. The algorithm consists of two nested loops. The outer loop checks for the availability of sensors (step 3) while the inner loop checks for the existence of targets that have yet to be covered by the sensors of a given cover set (step 7).

The algorithm selects one sensor at a time for inclusion to a cover set (step 8). A cover set is considered complete once it contains the necessary sensors to cover all targets (step 7). Each complete cover set is added to the cover set collection (step 15). The algorithm terminates (step 16) returning the cover set collection, when there are no more available sensors to use (steps 3, 9).

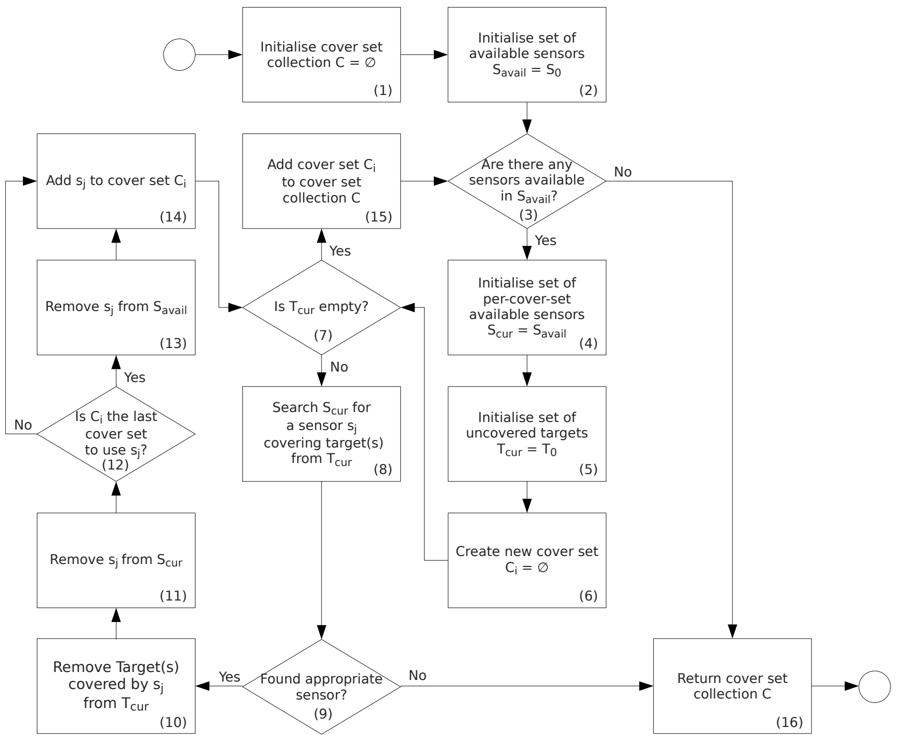

Initialise cover set collection C = ∅ (1)
Initialise set of available sensors $S_{avail} = S_0$ (2)
Are there any sensors available in S_{avail}? (3)
No
Yes
Initialise set of per-cover-set available sensors $S_{cur} = S_{avail}$ (4)
Initialise set of uncovered targets $T_{cur} = T_0$ (5)
Create new cover set $C_i = \emptyset$ (6)
Is T_{cur} empty? (7)
Yes
No
Search S_{cur} for a sensor s_j covering target(s) from T_{cur} (8)
Found appropriate sensor? (9)
Yes
No
Remove Target(s) covered by s_j from T_{cur} (10)
Remove s_j from S_{cur} (11)
Is C_i the last cover set to use s_j? (12)
Yes
No
Remove s_j from S_{avail} (13)
Add s_j to cover set C_i (14)
Add cover set C_i to cover set collection C (15)
Return cover set collection C (16)

Figure 3: Generating cover sets with the greedy heuristic approach

Two sets are used to keep track of the available sensors. The first one, S_{avail}, holds the sensors that will be available for use in future cover sets. The second one, S_{cur}, holds the sensors that are available for use in the current cover set. To avoid multiple inclusions of a particular sensor in a cover set, a sensor is immediately removed from S_{cur} once it has been selected. In algorithms producing non-disjoint sets, step 12 checks if the selected sensor will be available (i.e., it will have the necessary power) for use in other sets apart from the one currently being built. If this is not the case, then that sensor must be removed from the set of available sensors S_{avail} (step 13). In node-disjoint set generation, each sensor is only used once in the generated cover sets and, thus, sensors are immediately removed from S_{avail}, once they have been used.

The number of sets generated depends on the sensor selection strategy employed in step 8. In the following subsection we discuss the various parameters that node selection strategies must take into account in order to assist in the production of more cover sets.

3.3 Maximising the output

3.3.1 Finding the upper bound

The target with the smallest "neighbour-sensor" set places an upper bound on the number of generated cover sets. If the smallest "neighbour-sensor" set includes x sensors, then a coverage algorithm can produce at most $x \times w$ cover sets. We call this upper-bound, the *theoretical maximum*, which can be trivially computed from the input I of the algorithm. The *theoretical maximum* can serve as a hint for both heuristic and "branch & bound" algorithms, signifying the point where the algorithm has developed an optimal (or near optimal) solution and no further searching is required.

Unfortunately, the *theoretical maximum* does not always provide the actual maximum number of sets that can be generated for a given input I. As described above, once all the sensors from small "neighbour-sensor" sets have been used (w times each), the algorithm will not be able to produce any other cover sets. We will call targets that are associated with such small "neighbour-sensor" sets, *Critical Targets*. When selecting the next sensor to be added to the currently generated cover set, there is a higher probability that this sensor will cover a Critical Target if the network is densely populated (*i.e.*, a sensor covers many targets) rather than sparsely populated (*i.e.*, each sensor covers few targets). It is, thus, possible for a cover set of a dense network to include more than one neighbours of a given Critical Target. Each additional neighbour used, that covers the same Critical Target, shrinks by one the total number of generated cover sets. The algorithm will quickly run out of sensors covering this Critical Target and it will eventually produce less cover sets than the *theoretical maximum*. There is no efficient way, to the best of our knowledge, of pre-calculating the maximum possible number of cover sets, without actually trying all sensor combinations.

3.3.2 Managing the Critical Targets

One can limit the number of sensors covering Critical Targets within a cover set by first selecting a sensor covering the most Critical Target and then making sure not to select any other sensor from this target's "neighbour-sensor" set. This method, introduced by [14], requires a search for unused sensors covering Critical Targets, each time a new cover set needs to be generated. A different approach is to sort candidate nodes according to a cost function that incorporates a weight describing the number of Critical Targets a sensor covers. In this way, a cover set generation routine can use the sorted candidate list to select a node that covers the least number of Critical Targets possible. This is the approach used in our work. It adds flexibility to the algorithm in the cases where choosing more than one sensors covering Critical Targets is the only way of generating more cover sets.

A coverage algorithm terminates either when it has reached the *theoretical maximum* number of generated cover sets, or when it has run out of sensors capable of covering the given set of targets. Running out of sensors is not attributed only to Critical Targets though. A poor selection of sensors may cause targets to be double (or even triple) covered with auxiliary sensors that could have been used elsewhere more appropriately (and would have, thus, helped in the generation of more cover sets). Therefore, each cover set must contain a minimal amount of sensors, leaving as many sensors as possible available for the next cover sets to utilise.

3.3.3 Selecting the appropriate candidate

While selecting the appropriate sensor for a cover set, the sensor selection routine has to deal with different types (classes) of sensors. Each class is characterised by its coverage status over the already covered and uncovered targets (see Figure 4). Following the naming convention introduced in [19], we use the following 4 classes of sensors:

- **class Best**: The sensor covers all uncovered targets and none of the already covered ones.
- **class Good**: The sensor covers a subset of the uncovered targets and none of the already covered ones.
- **class OK**: The sensor covers all of the uncovered targets and a subset (or all) of the already covered ones.
- **class Poor**: The sensor covers a subset of the uncovered targets and a subset (or all) of the already covered ones.

Members of the *Best* class are always the preferred candidates for inclusion in cover sets. If no member of the *Best* class exists, the selection routine must try one of the other classes. We cannot enforce a strict ordering of preference (e.g. $Good \to OK \to Poor$) in the other classes, because the results produced turn out to be far from the optimum. To clarify this point, one may consider a scenario where it is obviously preferable to select a *Poor* sensor, covering one already covered target and five uncovered, rather than a *Good* sensor covering just one uncovered target. Thus, instead of enforcing a strict order of preference according to class, it is preferable to integrate the class information

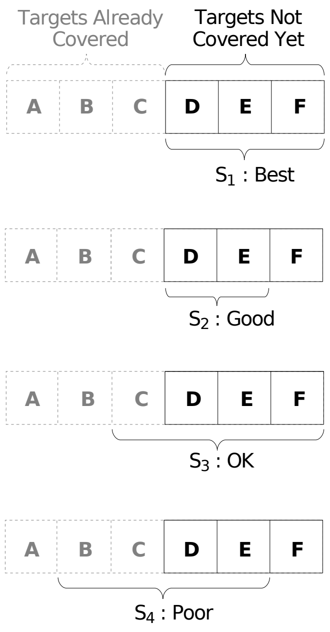


Figure 4: The four classes of sensor candidates: Best, Good, OK and Poor

(uncovered/covered targets) within a cost function. Thus, the node selection strategy becomes more agile, avoiding double-covers and promoting nodes that cover as many uncovered targets as possible.

4 The proposed algorithm

The key characteristic of the proposed algorithm is a uniform node selection strategy that tries to select the smallest possible number of nodes that cover Critical Targets. To achieve this, the algorithm uses a cost function, called *Critical Control Factor - CCF*, that takes into account the coverage status of sensor candidates, their relation to Critical Targets, and their remaining battery life.

The proposed algorithm is based on our previous work, the B{GOP} algorithm of [19], but it improves it in two significant ways; firstly, the CCF-based algorithm is capable of producing both non-disjoint and disjoint cover sets. Secondly, its enhanced cost function (CCF) considers additional node characteristics such as the remaining battery life of sensors.

Two variations of the algorithm are described. The *Static–CCF* algorithm, uses a weight to describe the association of a sensor to Critical Targets. This weight is computed only once per sensor, in the beginning of the algorithm, and it remains constant, until the termination of the algorithm. On the other hand, in the *Dynamic–CCF* variation, the weight changes dynamically, mirroring the changes happening to the set of Critical Targets during the execution of the algorithm. This requires the recalculation of Critical Targets each time a new cover set is constructed. The dynamic approach is more commonly found in the literature and hence we developed the second variation, for the purpose of testing this in combination with the cost function of Static–CCF.

4.1 Static–CCF

4.1.1 Setup

During the setup phase, Static–CCF uses input I (see Section 3.1) to construct the following auxiliary constants:

- The initial target set T_0.
- The initial sensor set S_0.
- The "neighbour-sensor" set N_i of sensors covering the target $t_i \in T_0$.
- The "target-coverage" set P_j for each sensor $s_j \in S_0$, containing the targets sensor s_j is capable of monitoring, *i.e.* $t_i \in P_j$ iff $s_j \in N_i$.
- The maximum number of sensors per "neighbour-sensor" set, $\mu = \max(|N_1|, \ldots, |N_k|), k = |T_0|$.
- The *badness* attribute B_j, which measures the *accumulative criticality* of the targets covered by sensor s_j. The badness attribute is computed according to the following formula:

  $$B_j = \sum_{i=1}^{|P_j|} (\mu - |N_i| + 1)^3 ,$$

  where N_i is the "neighbour-sensor" set of the i-th element of the "target-coverage" set P_j. This attribute is further analysed in section 4.1.3.
- The maximum badness: $B_{max} = \max(B_1, \ldots, B_n)$, $n = |S_0|$.
- The theoretical maximum number of possible generated sets, $max\_sets = w \cdot \min(|N_1|, \ldots, |N_k|)$, $k = |T_0|$.

During the execution of the algorithm, set L keeps track of the available battery life of each sensor. Initially, all the sensors have a battery life of w sets, assuming that all generated sets will provide coverage for the same amount of time.

4.1.2 Description

The Static–CCF algorithm is a greedy heuristic, mapped to three nested loops: the "Sensor availability check" loop, the "Uncovered target check" loop and the "Sensor applicability check" loop (see Algorithm 1).

The "Sensor availability check" loop is responsible for adding cover sets to the collection C, provided that there are still sensors available to utilise. It creates an empty cover set C_{cur} and it initialises the set of currently uncovered targets T_{cur} (with the targets found in T_0) and the set of currently available sensors S_{cur} (with the sensors found in S_{avail}). It then passes control to the "Uncovered target check" loop which is responsible for populating the cover set C_{cur} with sensors. Once C_{cur} has been populated, it is added to the collection C. If the algorithm detects that it has generated the maximum number of possible cover sets (max_sets) or it has run out of available sensors, it stops searching for further sets and it returns the current collection of cover sets C.

The "Uncovered Target Check" loop is responsible for selecting a sensor node from a set of candidates. Once the best candidate has been selected ("Sensor Applicability Check" loop), it is removed from the currently available sensors set S_{cur} and it is added to the current cover set C_{cur}. Additionally, all the targets monitored by this sensor are removed from T_{cur}. If the selected sensor's battery life does not permit its use in other cover sets, then the sensor is removed from S_{avail} (the set of available sensors for use in future cover sets). The "Uncovered Target Check" loop exits when there are no more targets to cover, thus signifying that the cover set C_{cur} is ready for inclusion in the collection C.

The "Sensor Applicability Check" loop is responsible for sorting node candidates and for selecting the top scoring nodes according to the CCF cost function. The objectives of the CCF cost function are threefold:
a) to promote candidates that cover as few already covered targets as possible (thus minimising the probability of double-covering a target),
b) to minimise the probability of selecting critical nodes (i.e., nodes that cover Critical Targets), and
c) to promote candidates that have more battery time available.

$$\begin{aligned} CCF(T_{cur}, P_j, B_j, L_j) = \alpha \cdot & \frac{coverage(P_j, T_{cur})}{|T_{cur}|} \\ & + \beta \cdot \left(1 - \frac{B_j}{B_{max}}\right) \\ & + \gamma \cdot \frac{L_j}{w}. \end{aligned}$$

The *coverage* function used in CCF describes the coverage status of a sensor, *i.e.* it measures the number of uncovered targets in relation to the number of already covered targets the sensor monitors. It is computed by the following formula:

$$coverage(P_j, T_{cur}) = \frac{uncovered}{(covered + 1)^r} = \frac{freq(P_j, T_{cur})}{(freq(P_j, T_0) - freq(P_j, T_{cur}) + 1)^r},$$

where *uncovered* is the number of targets covered by the sensor that have not been covered previously, *covered* is the number of already covered targets the sensor is capable of covering and $r = 1 - \frac{|T_{cur}|}{T_0}$. To test the coverage status of a sensor and compute the *uncovered* and *covered* values, the utility function $freq(P_j, T)$ is used, which counts the targets in T that a sensor s_j covers, i.e. $freq(P_j, T) = |P_j \cap T|$. The *coverage* function is analysed in Section 4.1.3.

The weights α, β and γ remain constant throughout the execution of the algorithm, with $\alpha, \beta, \gamma \in (0, 1)$ and $\alpha + \beta + \gamma = 1$. Their values can be tuned according to the nature of the examined problem and are further examined in Section 4.1.3.

The CCF function is always called for sensors that cover at least one of the remaining targets, and thus $\frac{coverage(P_j, T_{cur})}{|T_{cur}|} \in (0, 1]$. Furthermore, CCF is not called for sensors that have exhausted their battery life on previously created cover sets, hence $\frac{L_j}{w} \in (0, 1]$. Since $\left(1 - \frac{B_j}{B_{max}}\right) \in [0, 1)$, it follows that $CCF(T_{cur}, P_j, B_j, L_j) \to (0, 1)$.

4.1.3 Parameter analysis

The badness attribute. In the CCF algorithm we do not classify targets as Critical and non-Critical. Instead, we evaluate the criticality level of each target by measuring the number of sensors in its associated "neighbour-sensor" set. In this way, a target covered by x sensors is considered less critical than a target covered by $x - 1$ sensors.

The badness attribute of a sensor describes the accumulative criticality level of all targets covered by this sensor. We can use this attribute to prioritise the selection of sensor nodes that exhibit a low badness value (i.e., they are not as heavily associated with highly critical targets).

As mentioned in Section 4.1.1, the badness attribute of sensor s is computed according to the following formula:

$$B_s = \sum_{i=1}^{|P_s|} (\mu - |N_i| + 1)^3$$

where N_i is the "neighbour-sensor" set of the i-th element of the "target-coverage" set P_s, and μ is the maximum number of sensors per "neighbour-sensor" set, i.e. $\mu = \max(|N_1|, \ldots, |N_k|), k = |T_0|$.

The major component of the badness attribute, the criticality of a target t_i, is measured through $\mu - |N_i|$, since a target with a small "neighbour-sensor" set is more critical than a target with a larger "neighbour-sensor" set. We add one to this value, so that even targets with the maximum number of sensors in their "neighbour-sensor" set, will receive a positive criticality value.

The badness attribute of a sensor is the sum of criticality values of all targets covered by this sensor. By raising $\mu - |N_i| + 1$ to the power of 3, we decrease the probability of two or more sensors having the same badness value, since we expand the range of possible badness values. Such a collision would not be acceptable, since it could lead to random node selections among sensors that have the same badness value but cover targets with different levels of criticality.

Obviously, the appropriate power value depends on the problem parameters (number of sensors, number of targets, coverage status of targets). Through simulations we have found that the power of 3 is acceptable for most realistic scenarios. In table 1 we show the relation between the power value and the number of unique badness values generated for two sensor deployment scenarios. The first scenario consists of 350 sensors covering 40 targets in a 2-dimensional terrain of $600m^2$, while the second scenario consists of 500 sensors and 50 targets in a 3-dimensional terrain of $800m^3$.

Table 1: Measurements of the badness attribute in a 2-dimensional and a 3-dimensional scenario

	2d scenario			**3d scenario**		
Power	**Min. badness**	**Max. badness**	**Number of unique values**	**Min. badness**	**Max. badness**	**Number of unique values**
1	4	180	91	177	984	244
2	16	4278	147	4906	75512	304
3	64	113256	150	127236	6853916	304
4	256	3570050	150	3563194	664514348	304
5	1024	124952816	150	overflow		

The *coverage* function. As described in the previous section, the *coverage* function describes the coverage status of a sensor and it is computed as follows:

$$coverage(P_s, T_{cur}) = \frac{uncovered}{(covered+1)^r}.$$

The objectives of the coverage function are (i) to promote nodes that cover as many uncovered targets as possible, and (ii) to penalise nodes that cover already covered targets (thus avoiding the double-covering of targets). Obviously, the two objectives are satisfied from the fraction $\frac{uncovered}{covered+1}$ (we add one to the denominator to avoid division by zero). Moreover, the use of the exponent $r = 1 - \frac{|T_{cur}|}{|T_0|}$ (with $r \in (1, 0]$) gradually increases the penalty of nodes that cover already covered targets as the algorithm starts dealing with targets of higher criticality. Since our algorithm prioritises the selection of nodes that cover targets of lower criticality, it processes targets of higher criticality at a latter stage (i.e. when r converges to 1).

The CCF weights. The weights α, β and γ are constants, with $\alpha, \beta, \gamma \in (0, 1)$ and $\alpha + \beta + \gamma = 1$. Their values can be tuned according to the nature of the examined problem. For example, by increasing α, the algorithm pays more attention to the coverage status of sensors and produces cover sets with a smaller number of nodes. A larger β value would make the algorithm less tolerant to the selection of nodes that cover highly critical targets, while a larger γ value would prioritise the selection of nodes with a higher remaining lifetime.

Figure 5 shows the relation between α, β and the number of generated sets $|C|$ for two deployment scenarios. The γ value is equal to $1 - \alpha - \beta$. The first scenario, shown in Figure 5a, is an example of dense sensor deployment, with 350 sensors covering 40 targets in a 2-dimensional terrain of $361m^2$, while the second scenario (Figure 5b) is an example of sparse sensor deployment, using the same number of sensors and targets in a larger terrain of $961m^2$.

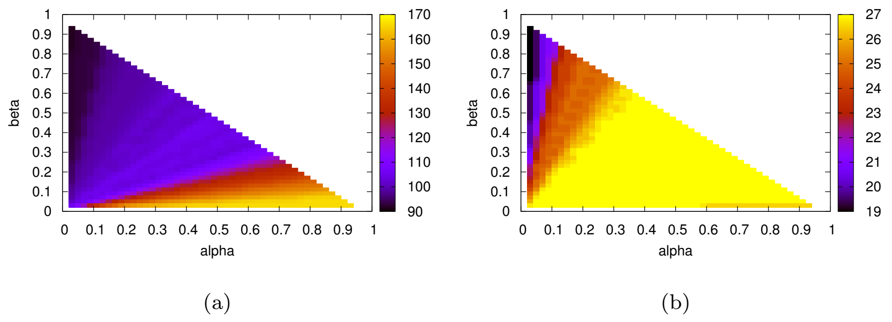


Figure 5: Relation between α, β and the number of generated sets (z-colour) in dense (a) and sparse (b) node deployment scenarios

To limit the number of tests needed in order to find the optimal values for α and β for a given scenario, one may divide the triangular solution space into sub-triangles and examine their centroids, as representative points of the solution space occupied by each sub-triangle. Once a centroid has been selected for evaluation, its corresponding α and β values will be fed to the coverage algorithm and the number of generated cover sets will be recorded (centroid score). The next step would be to further subdivide each of the sub-triangles. A *best-first* technique may be applied here, dividing first the triangle with the highest scoring centroid. The subdivision may continue for a number of user-specified rounds, or until a satisfactory solution has been reached (e.g. a combination of α and β that yields the theoretical maximum number of generated sets). Ultimately, the output of this process will be the coordinates of the centroid (i.e. the values of α and β) that was responsible for the highest number of generated sets.

In both scenarios of Figure 5, we can see that the optimal values for α and β are clustered together in a single subregion of the solution space. Also, solutions that lie near to this subregion tend to produce a greater number of generated sets than the ones that lie further away from it. From these two observations we can deduce that a local optimum search strategy may be sufficient for finding the optimal values for α and β, in scenarios similar to those presented in Figure 5.

Table 2 shows the number of iterations needed by a local optimum search heuristic in order to produce the optimal values for α and β (i.e. the values that will result in the production of the maximum number of generated sets) in scenarios with 350 sensors, 40 targets and variable terrain size. In each iteration, the heuristic divides a triangle into 6 sub-triangles (2 for each median) and evaluates their centroids. The triangle with the highest scoring centroid becomes the triangle that will be examined in the next iteration. Table 2 shows that this approach is capable of finding optimal values for α and β, using only a small number of iterations and thus, in a relatively small amount of time.

Table 2: Number of local optimum search iterations needed in order to achieve the theoretical maximum number of cover sets in a 2-dimensional scenario with 350 sensors, 40 targets and variable terrain size

Terrain Size m^2	**Number of iterations**	**Local optimum search exec. time (sec)**
361	2	5.76
484	1	2.20
625	2	3.65
778.4	1	1.52
961	1	1.34
1156	1	1.18
1369	1	1.26
1600	4	4.88
1764	2	1.53
1989	1	1.05

4.1.4 Algorithm Analysis

In order to select a single sensor for inclusion in a generated set, the algorithm must test all available sensors for their monitoring capacity over the currently uncovered targets. Once a sensor with a suitable monitoring capacity has been chosen, the remaining sensors will be tested with regard to the targets the previously chosen sensor had left uncovered. Once a generated set is complete (*i.e.*, it covers all the targets), the algorithm will start building a new set and it will re-initialise the set of uncovered targets T_{cur} to the initial set of targets T_0.

The algorithm will terminate either when it has run out of sensors ($S_{cur} = \emptyset$) or when it has reached the maximum number of possible generated sets ($|C| = max\_sets$). As explained in Section 3, the theoretical maximum number of generated sets is sometimes impossible to achieve, since the algorithm exits prematurely, as it has no more sensors to utilise. Hence, the total number of available sensors introduces an upper bound to the execution time of the algorithm. The longest run of Static–CCF would have included all available sensors in the generated sets and would have used each sensor w times. In that case, the time taken to produce the generated sets from n sensors and k targets would have been proportional to the following product:

$$w \sum_{i=0}^{n-1} (n-i)(k - i \bmod k), \text{ where } n = |S_0|, \ k = |T_0|.$$

We can, thus, deduce that the total running time of the Static–CCF algorithm is $O(wn^2k)$.

Proposition:

Static–CCF is capable of generating at least one cover set, if one exists.

Proof:

If G is a cover set, then:

$$\forall \ t_i \in T_0, \ \exists \ s_j \in G \ : t_i \in P_j, \tag{1}$$

where $G \subseteq S_0$, $G \neq \emptyset$, $P_j \subseteq T_0$ and $P_j \neq \emptyset$.

Let us suppose that a set G does exist (covering all the targets in T_0 with sensors from S_0) but our algorithm has failed to produce a cover set. This means that during the construction of the first cover set, the algorithm failed to find a sensor s_j capable of covering a subset of T_{cur}, *i.e.*,

$$\forall t_i \in T_{cur}, \nexists s_j \in S_{cur} : t_i \in P_j. \tag{2}$$

The set C_{cur} contains the sensors already selected for inclusion in the cover set. Since this is the algorithm's first attempt to construct a cover set, all sensors not found in the current sensor set S_{cur} are members of C_{cur}, *i.e.*

$$S_0 = C_{cur} \cup S_{cur},\ C_{cur} \cap S_{cur} = \emptyset. \tag{3}$$

All targets covered by sensors in C_{cur} have been removed from the current point set T_{cur}. Hence from (3) we have:

$$\forall t_i \in T_{cur}, \nexists s_j \in C_{cur} : t_i \in P_j. \tag{4}$$

¿From (2), (3), (4) we have:

$$\forall t_i \in T_{cur}, \nexists s_i \in S_0 : t_i \in P_j. \tag{5}$$

Statement (5) is false, since we can rewrite (1) for $T_{cur} \subseteq T_0$ and $G \subseteq S_0$, as follows:

$$\forall t_i \in T_{cur}, \exists s_j \in S_0 : t_i \in P_j. \tag{6}$$

Thus, our initial hypothesis proves to be false; if a solution to the coverage problem exists, Static–CCF will have a sufficient number of sensors to generate at least one cover set. □

4.1.5 Optimisations

During the "Sensor Applicability Check" (see Algorithm 1) the $freq(P,T)$ function is used to measure the coverage status of a sensor with respect to the current and initial sets of targets (*i.e.*, $freq(P_s, T_{cur})$ and $freq(P_s, T_0)$ respectively). Function $freq(P,T)$ calculates the number of members found in the intersection of sets P and T which requires at least $min(|P|, |T|)$ checks[2]. By minimising the number of unnecessary calls to $freq(P,T)$, the total execution time of the algorithm can be reduced.

The initial number of targets a sensor covers ($freq(P_s, T_0)$) can be computed during the *Setup* phase, since it remains constant throughout the algorithm runtime.

Instead of calculating the current coverage $freq(P_s, T_{cur})$ upon request, we can have this value pre-calculated, whenever T_{cur} changes:

- $freq(P_s, T_{cur})$ will be equal to the initial $freq(P_s, T_0)$, whenever the current target set is initialised to the original target set (*i.e.*, $T_{cur} = T_0$).

[2]Should a hash-table be employed for set member lookups.

- As soon as a sensor has been selected for the generated set, the targets this sensor covered are removed from the current set of targets T_{cur}. The coverage value of other sensors covering any of these targets is decremented as required.

The optimisations mentioned above make the coverage checks much more efficient and have been used in the simulation software described in Section 5.1.

4.2 Dynamic–CCF

While constructing a cover set, a greedy coverage algorithm removes sensors from the set of available nodes S_{avail}. If N'_i represents the sensors that could cover target t_i during the execution of the algorithm, then N'_i should be updated each time S_{avail} changes.

Instead of defining a Critical Target as a target t_i that is covered by a small amount of sensors in the terrain, one can define the Critical Target as a target within the execution context of the algorithm that has a corresponding "neighbour-sensor" set N'_i with the *lowest* cardinality. This provides us with a more accurate view of the targets that can be effectively covered by a small amount of sensors (i.e., they might have not started as Critical Targets but they became such during the cover set generation process). The Dynamic–CCF algorithm uses N'_i to recalculate the Critical Targets at the beginning of each cover set. This approach provides the CCF function with a more accurate measurement of critical nodes, but comes at the cost of a continuous recalculation of "neighbour-sensor" sets.

During the setup phase, Dynamic–CCF uses input I to calculate T_0, S_0, N and *max_sets*. The algorithm structure is very similar to that of Static–CCF (see Algorithm 2). Function *recompute_neighbour_sets* is responsible for recalculating the "neighbour-sensor" sets, while function *recompute_min_cardinality* is responsible for calculating the minimum cardinality of the new "neighbour-sensor" sets. These functions aid Dynamic–CCF in the discovery of new Critical Targets.

The new CCF cost function used is:

$$CCF(T_{cur}, P_s, L_s, H_s) = \alpha \cdot \frac{coverage(P_s, T_{cur})}{|T_{cur}|} + \beta \cdot H_s + \gamma \cdot \frac{L_s}{w},$$

$$CCF(T_{cur}, P_s, L_s, H_s) \to (0, 1],$$

where H_s is an attribute of sensor s that signifies whether it is safe to select this sensor or not. H_s enables the algorithm to skip over sensors that double-cover Critical Targets. Each time a new cover set is to be constructed, all sensors are considered *harmless* and, thus, $\forall s \in S_{cur}, H_s = 1$. If a selected node covers a set of Critical Targets K, then Dynamic–CCF marks all nodes covering the same Critical Targets as *harmful* ($\forall\ t_i \in K, \forall\ s \in N'_i : H_s = 0$). In this way, *harmless* nodes get a bonus of $\beta \cdot H_s$ in the CCF cost function and are, thus, preferred for inclusion in the cover set, while $harmful$ nodes (covering Critical Targets) are penalised and considered with a lower priority.

Dynamic–CCF incurs a higher order of complexity, due to the "neighbour-sensor" set recalculations and critical node management tasks. Its total running time is $O(wn^2k + wn^2 + wnk)$ for n sensors, k targets and w maximum allowed participations of any sensor in the output cover sets.

5 Algorithm Evaluation

In order to evaluate the performance of the CCF algorithm, we have conducted a set of simulations. In this section we discuss our simulation findings and we show how the proposed algorithm compares to other similar approaches found in the literature.

5.1 Simulation Environment

Our simulation environment consists of two families of Perl scripts. The first family of scripts is responsible for generating terrains of targets and sensors, while the second is responsible for executing the desired algorithms on the generated topologies.

5.1.1 Topology Generation

Our terrain generation software is capable of producing 2-dimensional, 3-dimensional as well as k-dimensional topologies[3]. The small degree of freedom offered by 2d terrains makes them good candidates for testing coverage algorithms in deployment scenarios where the terrain need not be modelled with high accuracy. Obviously, the 3d terrain generation software can simulate more realistic deployment scenarios. The user provides the 2d and 3d terrain generation scripts with the number of sensors, targets and the size of the area they will be scattered in, thus allowing the script to control the density of node deployment. Each simulated sensor has a communication radius, R_c, of $50m$ and a sensing radius, R_s, of $10m$. The node placement strategy is described in the following steps:

1. Initially, sensors and targets are scattered randomly in the user–selected field, following a uniform spatial distribution. A base station is also introduced at a fixed position in the field $[0, y/2]$ in 2d terrains and $[0, y/2, z/2]$ in 3d terrains.

2. Targets not covered by any sensor are ignored.

3. Sensors not covering any target are ignored.

4. Each target defines an *area* with the target location being the centre of the area. A sensor *belongs* to an area if the area's centre (i.e., target location) lies within its sensing range R_s. It is possible for a sensor to belong to more than one areas. In order for any two sensors (possibly covering different areas) to be able to exchange data, the Euclidean distance between the

[3]We will be focusing on 2d and 3d terrains, since k-dimensional topologies are mainly useful for testing cover set generation algorithms in application domains other than WSN, where topological restrictions do not apply.

centres of the observed areas must be smaller than $(R_c - 2 \cdot R_s)$. *Solitary areas* that cannot communicate directly with any of the other areas in the terrain (or the base station) are discarded.

5. A graph is constructed with vertices representing the remaining areas and edges connecting the areas that communicate directly. The base station is also introduced as a vertex in the graph along with edges connecting it with the appropriate areas. The transitive closure of the graph is computed and any area that has no path leading to the base station is discarded.

6. Finally, the collection of sensors and targets belonging to the remaining areas form the generated topology.

Steps 2 and 3 ensure that the output of the topology generation scripts is compatible with the coverage algorithm input described in Section 3. That is, all targets must be covered by at least one sensor and any sensor should cover at least one target (i.e., be part of a "neighbour-sensor" set). The connectivity check in step 5 ensures that there exists a path between any sensor in the generated terrain and the base station. By moving the connectivity check to the terrain generation stage, we allow all coverage algorithms to benefit from this; the cover sets produced will have guaranteed connectivity with the base station. It is also possible for the coordinates of a real deployment scenario to be used in step 1. In this case, the above process plays the role of a filter, generating the subset of the sensor network that is capable of communicating with the base station.

5.1.2 Implementation Notes on the Simulated Coverage Algorithms

In our simulations, we compare Static–CCF and Dynamic–CCF to the following algorithms: the disjoint set algorithm by Slijepcevic et al. [14], the Greedy-MSC heuristic algorithm by Cardei et al. [6], the B{GOP} algorithm [19] and a randomised variation of B{GOP}, called B{GOP}–random. For our simulation purposes, we have implemented all of these algorithms in Perl, with the exception of the disjoint–set algorithm proposed by Slijepcevic et al., which we have kept in its original form in the Java programming language. Since this algorithm is implemented in a different programming language we will be assessing its relative rather than its absolute execution time in our simulations.

In the algorithm proposed by Cardei et al. in [6], once a critical target has been found, a sensor that covers the target must be selected. If more than one sensors exist, then the sensor most suitable for selection will be the one with the highest *contribution*. In our implementation of the Cardei non–disjoint algorithm, the following *contribution* formula $f(s)$ was used:

$$f(s) = freq(P_s, T_{cur}) + L_s,$$

where $freq(P_s, T_{cur})$ is the number of uncovered targets that sensor s covers and L_s is the sensor's remaining lifetime.

The version of B{GOP} used in our experiments, has been modified slightly to use the badness formula found in the Static–CCF algorithm (see Section 4.1.1). Its randomised variation, B{GOP}–random, introduces a probability–driven selection process for candidates of *Best* and *Good–OK–Poor* classes. Specifically, in the *Best* class, the probability of a particular node being selected is inversely proportional to its badness value, while in the *Good–OK–Poor* class, the same probability is proportional to the result of the CCF cost function for this node. By experimenting with randomisation, we expect to gain some insight into the effectiveness of non-deterministic node selection strategies. Randomised versions of Static–CCF and Dynamic–CCF were also developed, but since their performance was similar to that of B{GOP}–random, they have been omitted from our evaluation for reasons of brevity.

5.2 Simulation

In this section, we evaluate the performance of our algorithms in the production of node disjoint and non-disjoint cover sets, by way of simulation. We use two simulation scenarios, each one characterised by the number of sensors deployed, the number of targets monitored and the dimensions of the terrain (2d or 3d terrain). The first scenario involves the deployment of 350 sensor nodes and 40 targets on a 2-dimensional terrain, while the second senario consists of 500 sensor nodes and 20 targets deployed on a 3-dimensional terrain. These parameters are selected in order to satisfy the topology generation requirements described in Section 5.1.1. More specifically, these parameters ensure that each sensor covers at least one target and each target is covered by at least one deployed sensor.

The above scenarios are used for the simulation of both disjoint and non-disjoint cover set generation algorithms. Although our algorithms are capable of solving both problems in a unified manner (via parameter w), we use this distinction in order to compare our results against the results of algorithms that address only one of the two problems (e.g. the node disjoint algorithm of Slijepcevic et al. [14]). Our primary goal is to measure the impact of deployment density on the algorithm execution time, in simplified (2d) and realistic (3d) topologies for both disjoint and non-disjoint algorithms. Our secondary goal, which concerns only non-disjoint algorithms, is to investigate the relationship between the node participations and the network lifetime offered by the generated cover sets.

For each simulation scenario we generate 20 random topologies, as described in Section 5.1.1. We execute each algorithm 20 times on each topology, in order to calculate the *network lifetime* and the *average time per generated cover set*. Additionally, for the non-disjoint algorithms, we also examine the relation between *network lifetime* and *maximum node participations*. We measure *network lifetime* in battery units, where one battery unit equals the time h that a sensor can operate before running out of battery resources. The *network lifetime* is computed by $\frac{\sum_{i=1}^{20}|C_i|}{20\cdot w}$, where $|C_i|$ denotes the number of generated sets produced by the i-th execution of the algorithm.

As described in Section 4.1.2, the *CCF* formula uses three "tunable" parameters, namely α, β and γ. For the simulations of the Static–CCF algorithm, we have used the values $\alpha = 0.35$, $\beta = 0.02$ and $\gamma = 0.63$. Our simulations of the Dynamic–CCF algorithm use equal values for all three parameters, i.e. $\alpha = \beta = \gamma = \frac{1}{3}$.

All experiments were carried out on a Pentium 4 3.4Ghz host with 1GB of RAM, running the Debian GNU/Linux operating system.

5.2.1 Results using the disjoint sets approach

Figure 6 illustrates the performance of the node disjoint algorithms (Static-CCF, Dynamic-CCF, Slijepcevic et al. [14], B{GOP} [19] and B{GOP}-random) in the 2d scenario. As shown in Figure 6a all of the examined algorithms present similar network lifetime, equal or very close to the theoretical maximum. B{GOP}–random is an exception to this, particularly when dealing with dense sensor deployments. As far as the algorithm execution time is concerned, the Slijepcevic et al. algorithm exhibits an exponential increase in execution time (see Figure 6b) as the terrain becomes smaller and targets receive more sensors in their "neighbour-sensor" sets. Dynamic–CCF also comes with a performance penalty, when compared to Static–CCF, due to the algorithm's higher order of complexity. Figure 7 presents similar results, showing the performance of the algorithms in the 3d scenario.

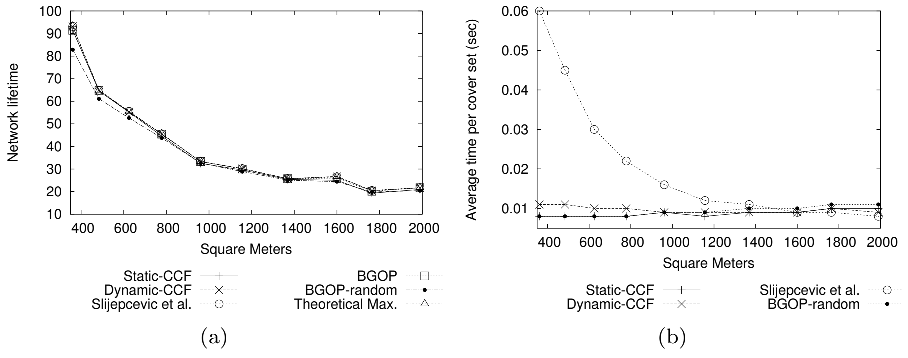


Figure 6: Performance of node disjoint algorithms in a 2d environment with a variable terrain size

5.2.2 Results using the non-disjoint sets approach

For the algorithms generating non-disjoint sets (Static-CCF, Dynamic-CCF, Cardei et al. Greedy-MSC heuristic [6]), we conduct three experiments. In the first experiment, we allow sensors to participate in ten produced sets (at

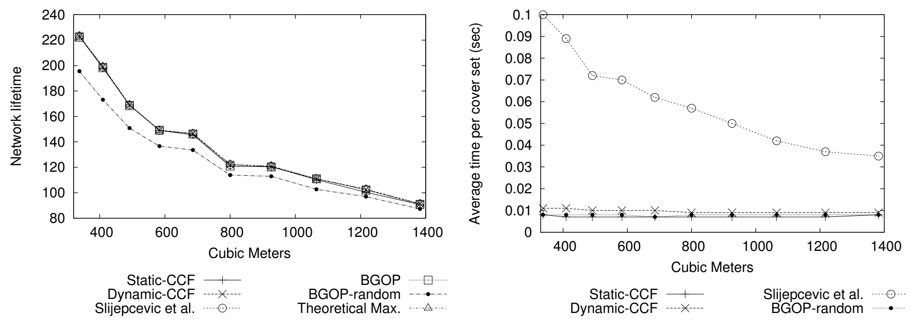


Figure 7: Performance of node disjoint algorithms in a 3d environment with a variable terrain size

maximum) and we record the achieved network lifetime and required execution time per generated set. Table 3 shows the results of using the non-disjoint algorithms in the 3d scenario. The results of two disjoint-set algorithms (Static–CCF and B{GOP}–random) are also displayed in the table, in order to assist us in our comparison.

Table 3: Network Lifetime (NL) and Average execution Time per Set (ATS) for disjoint and non-disjoint algorithms in a 3d environment with variable Terrain Size (TS)

	Static–CCF disjoint		**B{GOP}–random disjoint**		**Static–CCF non disjoint**		**Dynamic–CCF non disjoint**		**Cardei-Greedy-MSC non disjoint**	
TS (m^3)	**NL**	**ATS** (sec)	**NL**	**ATS** (sec)	**NL**	**ATS** (sec)	**NL**	**ATS** (sec)	**NL**	**ATS** (sec)
337.5	213.40	0.005	190.20	0.006	213.40	0.078	213.40	0.099	202.20	0.027
409.6	168.00	0.005	149.40	0.005	168.00	0.077	168.00	0.095	164.00	0.025
491.3	164.00	0.005	145.20	0.005	164.00	0.078	164.00	0.093	159.00	0.025
583.2	151.40	0.005	137.00	0.005	150.80	0.078	151.40	0.093	146.80	0.024
685.9	147.00	0.005	129.80	0.005	147.00	0.073	147.00	0.084	138.00	0.023
800.0	120.00	0.005	115.20	0.005	120.00	0.077	120.00	0.088	115.00	0.019
926.1	137.00	0.005	125.00	0.005	137.00	0.074	137.00	0.084	133.00	0.022
1064.8	126.00	0.005	110.40	0.005	128.00	0.076	129.00	0.084	124.00	0.022
1216.7	90.00	0.005	86.40	0.006	90.00	0.086	90.00	0.090	89.00	0.018

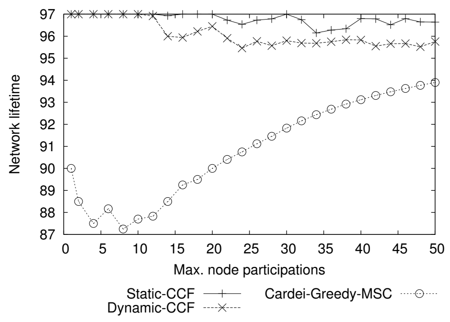
The disjoint algorithms provide results very close to the optimum solution (theoretical maximum), in most of the examined scenarios. Note that the non-disjoint algorithms also provide results very close to the optimum solution, as in the disjoint approach. However, they exhibit ten to fifteen times longer execution times when compared to the disjoint approaches. This is due to the fact that the maximum allowed participations parameter w increases the complexity of the algorithm in the non-disjoint case. Both algorithms however, provide nearly optimum results.

In the second experiment (see Figure 8) we wish to test the impact of node participations on the achieved network lifetime. To this end, we use two topologies, one based on the 2d scenario and one based on the 3d scenario. As shown in Figures 8a and 8b, the CCF algorithms produce a large number of cover sets even when a small number of maximum node participations has been used. On

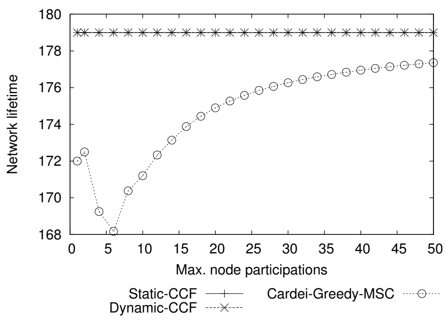
Table 4: Node Participations and Average execution Time per Set (ATS) when producing the theoretical maximum number of cover sets (3d space, variable terrain size)

	Static–CCF		**Dynamic–CCF**		**Cardei-Greedy-MSC**	
Terrain Size (m^3)	**Max. node participations**	**ATS** (sec)	**Max. node participations**	**ATS** (sec)	**Max. node participations**	**ATS** (sec)
337.5	1	1.07	1	1.71	56	54.17
409.6	1	0.84	1	1.18	28	18.40
491.3	1	0.82	1	1.15	26	16.80
583.2	1	0.76	1	1.06	26	14.97
685.9	1	0.74	1	0.88	32	16.64
800.0	1	0.60	1	0.72	26	9.23
926.1	1	0.69	1	0.82	24	11.53
1064.8	3	2.69	1	0.77	40	12.32
1216.7	1	0.45	1	0.54	7	1.07

the contrary, the Greedy-MSC requires the participation of a node in more than fifty sets in order to reach a satisfactory result, which comes at a cost in total execution time.



(a)



(b)

Figure 8: Network lifetime in relation to maximum node participations in 2d (left) and 3d (right) environments

Finally, in our third experiment we wish to find the minimum number of node participations required, in order to produce cover sets that offer a certain network lifetime. Table 4 presents the performance of the examined algorithms in a 3d scenario with variable node deployment density, where the algorithms must produce the theoretical maximum number of possible cover sets. Both flavours of the CCF algorithm achieve this goal in a short amount of time, using only one node participation (in most topologies). On the other hand, the Cardei et al. Greedy-MSC heuristic requires a node to be a member of more than 50 sets in order to satisfy the above requirement in dense sensor deployment scenarios.

6 Conclusions

In this paper, we have presented a centralised greedy algorithm for the efficient production of both node disjoint and non-disjoint cover sets.

Table 5: A comparison between the proposed algorithm and the algorithms of [14, 6]

	Static–CCF	**Slijepcevic et al.**	**Cardei et al.**
Type of sets produced	Disjoint and non-disjoint	Disjoint	Disjoint and non-disjoint
Critical node handling	Prioritise nodes that cover targets with low criticality	Starts cover set with a critical node. Other nodes covering the same critical targets are ignored.	Starts cover set with a critical node. It does not implement any critical node avoidance strategy.
Candidate node selection criteria	a) # of uncovered targets vs. # of already covered b) # of available targets c) association with poorly monitored targets d) remaining battery life	# of already covered targets the candidate covers	a) # of uncovered targets the candidate covers b) remaining battery life
Complexity	$O(wn^2k)$	$O(n^2)$	$O(dk^2n)$
Dense node deployment incurs significant penalty in execution time	No	Yes	Yes, due to increased number of participations required for optimal solution

To evaluate the efficiency of the proposed algorithm, we have measured its performance against that of other approaches found in the literature. Our test cases included various deployment scenarios, in 2-dimensional and 3-dimensional terrains. We have found that the Static–CCF version of the proposed algorithm outperforms other similar algorithms in terms of execution time, while it exhibits comparable and near optimal results in terms of generated coverage sets. Moreover, we have found that the Dynamic–CCF version of the algorithm, produces marginally better results but exhibits longer execution times.

When producing non-disjoint cover sets, both Static–CCF and Dynamic–CCF provide cover sets offering longer network lifetime than those produced by [6]. Also, they require a smaller number of node participations in order to achieve these results.

Table 5 summarises the improved characteristics of the Static–CCF algorithm, in comparison to other existing approaches. First, our algorithm is flexible enough to effectively avoid the "double-covering" of critical fields, *i.e.* a situation where a critical field is covered by more than one nodes in a given cover set. This is achieved by ranking sensors according to the CCF function, that is based on several factors such as the coverage status of candidates, their association with poorly monitored targets and their available battery resources. Moreover, the proposed algorithm shows a relatively low complexity, allowing for the efficient production of cover sets even in dense deployment scenarios.

Finally, by comparing the results gathered from the node disjoint and non-disjoint algorithms we observe that it is possible to generate the optimal number of cover sets with a node disjoint approach to CCF, rather than investing on a more time-consuming non-disjoint algorithm.

References

[1] Z. Abrams, A. Goel, and S. Plotkin. Set k-cover algorithms for energy efficient monitoring in wireless sensor networks. In *Proc. of Third Inter-*

national Symposium on Information Processing in Sensor Networks, pages 424–432. ACM, 2004.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[3] P. Berman, G. Calinescu, C. Shah, and A. Zelikovsky. Power efficient monitoring management in sensor networks. In *Proc. of Wireless Communications and Networking Conference*, volume 4, pages 2329–2334. IEEE, March 2004.

[4] M. Cardei and D-Z. Du. Improving wireless sensor network lifetime through power aware organization. *ACM Wireless Networks*, 11(3):333–340, 2005.

[5] M. Cardei, D. MacCallum, M. X. Cheng, M. Min, X. Jia, D. Li, and D.-Z. Du. Wireless sensor networks with energy efficient organization. *Journal of Interconnection Networks*, 3(3-4):213–229, 2002.

[6] M. Cardei, M. Thai, Y. Li, and W. Wu. Energy-efficient target coverage in wireless sensor networks. In *Proc. of INFOCOM 05*, volume 3, pages 1976–1984. IEEE, March 2005.

[7] M. Cardei and J. Wu. Energy efficient coverage problems in wireless ad hoc sensor networks. *Computer Communications*, 29(4):413–420, 2006.

[8] Antoine Gallais, Jean Carle, David Simplot-Ryl, and Ivan Stojmenović. Localized sensor area coverage with low communication overhead. *IEEE Transactions on Mobile Computing*, 7(5):661–672, 2008.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[10] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proc. of 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 300–309. IEEE, November 1998.

[11] A.J. Goldsmith and S.B. Wicker. Design challenges for energy-constrained ad hoc wireless networks. *Wireless Communications, IEEE*, 9(4):8–27, August 2002.

[12] Liu Hai, P. Wan, C.-W. Yi, Jia Xiaohua, S. Makki, and N. Pissinou. Maximal lifetime scheduling in sensor surveillance networks. In *Proc. of INFOCOM 05*, volume 4, pages 2482–2491. IEEE, March 2005.

[13] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. of International Workshop on Wireless Sensor Networks and Applications*, pages 88–97. ACM, September 2002.

[14] S. Slijepcevic and M. Potkonjak. Power efficient organization of wireless sensor networks. In *Proc. of International Conference on Communications (ICC'01)*, pages 472–476. IEEE, June 2001.

[15] My T. Thai, Feng Wang, Hongwei Du, and Xiaohua Jia. Coverage problems in wireless sensor networks: Designs and analysis. *International Journal of Sensor Networks, Special issue on coverage problems*, 3:191–200, 2008.

[16] D. Tian and N. D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *Proc. of 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 32–41. ACM Press, September 2002.

[17] H. Zhang and J. Hou. Maintaining sensing coverage and connectivity in large sensor networks. *Ad Hoc & Sensor Wireless Networks*, 1:89–123, 2005.

[18] Fan Ye Zhong, G. Songwu Lu, and Lixia Zhang. Peas: a robust energy conserving protocol for long-lived sensor networks. In *Proc. of 10th IEEE International Conference on Network Protocols*, pages 200–201, 2002.

[19] D. Zorbas, D. Glynos, P. Kotzanikolaou, and C. Douligeris. B{GOP}: An adaptive coverage algorithm for wireless sensor networks. In *Proc. of the 13th European Wireless Conference*, Paris, April 2007.

Algorithm 1 The Static-CCF Algorithm

```
Require: $S_0 \neq \emptyset$, $T_0 \neq \emptyset$, $P \neq \emptyset$, $max\_sets > 0$, $B \neq \emptyset$, $B_{max} > 0$, $w > 0$, $\alpha, \beta, \gamma \in (0,1)$
  $C = \emptyset$
  $S_{avail} = S_0$
  for all $s \in S_{avail}$ do
    $L_s := w$
  end for
  while $S_{avail} \neq \emptyset$ do {Sensor availability check}
    $S_{cur} = S_{avail}$
    $T_{cur} = T_0$
    $C_{cur} = \emptyset$
    while $T_{cur} \neq \emptyset$ do {Uncovered target check}
      $selected := none$
      $max\_CCF := 0$
      for all $s \in S_{cur}$ do {Sensor applicability check}
        if $freq\,(P_s, T_{cur}) \neq 0$ then
          $uncovered := freq\,(P_s, T_{cur})$
          $covered := freq\,(P_s, T_0) - uncovered$
          $r := 1 - \frac{|T_{cur}|}{|T_0|}$
          $coverage := \frac{uncovered}{(covered + 1)^r}$
          $CCF := \alpha \cdot \frac{coverage}{|T_{cur}|} + \beta \cdot \left(1 - \frac{B_s}{B_{max}}\right) + \gamma \cdot \frac{L_s}{w}$
          if $CCF > max\_CCF$ then
            $max\_CCF := CCF$
            $selected := s$
          end if
        end if
      end for {Sensor applicability check}
      if $selected = none$ then
        return $C$
      end if
      $T_{cur} = T_{cur} - P_{selected}$
      $S_{cur} = S_{cur} - \{selected\}$
      $L_{selected} := L_{selected} - 1$
      if $L_{selected} = 0$ then
        $S_{avail} = S_{avail} - \{selected\}$
      end if
      $C_{cur} = C_{cur} \cup \{selected\}$
    end while {Uncovered target check}
    $C = C \cup \{C_{cur}\}$
    if $|C| = max\_sets$ then
      return $C$
    end if
  end while {Sensor availability check}
  return $C$
```

Algorithm 2 The Dynamic-CCF Algorithm

```
Require: $S_0 \neq \emptyset,\ T_0 \neq \emptyset,\ P \neq \emptyset,\ N \neq \emptyset,\ max\_sets > 0,\ w > 0,\ \alpha, \beta, \gamma \in (0,1)$
  $C = \emptyset$
  $S_{avail} = S_0$
  for all $s \in S_{avail}$ do
    $L_s := w$
  end for
  while $S_{avail} \neq \emptyset$ do {Sensor availability check}
    $S_{cur} = S_{avail}$
    $T_{cur} = T_0$
    $C_{cur} = \emptyset$
    $Critical\_Targets = \emptyset$
    $N' = recompute\_neighbour\_sets(T_0, N, S_{avail})$
    $min\_cardinality = recompute\_min\_cardinality(N')$
    for all $t_i \in T_{cur}$ do
      if $|N'_i| = min\_cardinality$ then
        $Critical\_Targets = Critical\_Targets\ \cup\ \{t_i\}$
      end if
    end for
    for all $s \in S_{cur}$ do
      $H_s := 1$
    end for
    while $T_{cur} \neq \emptyset$ do {Uncovered target check}
      $selected := none$
      $max\_CCF := 0$
      for all $s \in S_{cur}$ do {Sensor applicability check}
        if $freq\,(P_s, T_{cur}) \neq 0$ then
          $uncovered := freq\,(P_s, T_{cur})$
          $covered := freq\,(P_s, T_0) - uncovered$

          $r := 1 - \frac{|S_{avail}|}{|S_0|}$
          $coverage := \frac{uncovered}{(covered + 1)^r}$
          $CCF := \alpha \cdot \frac{coverage}{|T_{cur}|} + \beta \cdot H_s + \gamma \cdot \frac{L_s}{w}$
          if $CCF > max\_CCF$ then
            $max\_CCF := CCF$
            $selected := s$
          end if
        end if
      end for {Sensor applicability check}
      if $selected = none$ then
        return $C$
      end if
      for all $t_i \in P_{selected}$ do
        if $t_i \in Critical\_Targets$ then
          for all $s \in N'_i$ do
            $H_s := 0$
          end for
        end if
      end for
      $T_{cur} = T_{cur} - P_{selected}$
      $S_{cur} = S_{cur} - \{selected\}$
      $L_{selected} := L_{selected} - 1$
      if $L_{selected} = 0$ then
        $S_{avail} = S_{avail} - \{selected\}$
      end if
      $C_{cur} = C_{cur}\ \cup\ \{selected\}$
    end while {Uncovered target check}
    $C = C \cup \{C_{cur}\}$
    if $|C| = max\_sets$ then
      return $C$
    end if
  end while {Sensor availability check}

  return $C$
```

Function 3 *recompute_neighbour_sets*

```
Require: $T_0,\ N,\ S_{avail}$
    for all $t_i \in T_0$ do
        for all $s_j \in N_i$ do
            if $s_j \in S_{avail}$ then
                $N'_i = N'_i \cup \{s_j\}$
            end if
        end for
    end for

    return $N'$
```

Function 4 *recompute_min_cardinality*

```
Require: $N'$
    $min\_cardinality = \infty$
    for all $N'_i \in N'$ do
        if $|N'_i| < min\_cardinality$ then
            $min\_cardinality = |N'_i|$
        end if
    end for

    return $min\_cardinality$
```