# B{GOP}: An Adaptive Algorithm for Coverage Problems in Wireless Sensor Networks

Dimitrios Zorbas, Dimitris Glynos, Panayiotis Kotzanikolaou, Christos Douligeris
Department Of Informatics, University Of Piraeus,
Karaoli & Dimitriou 80, Piraeus 18535, Greece
email: {dzorbas, daglyn, pkotzani, cdoulig}@unipi.gr

***Abstract*— To achieve power-efficient monitoring of targets in a terrain covered by a sensor network, it is sensible to divide the sensors into *cover sets* and make each of these sets responsible for covering the targets for a certain period of time. Generating the maximum number of such cover sets, has been proved to be an NP-complete problem, and thus algorithms producing sub-optimal solutions have been proposed. In this paper we propose a centralised heuristic algorithm, that efficiently generates cover sets, each one capable of monitoring all targets. Our simulation findings demonstrate an improvement against existing methods, with results close to the optimal solution.**

## I. Introduction

Wireless Sensor Networks (WSNs) find extensive use in many domains such as military applications [1], environment monitoring [2], target surveillance [3] and disaster prevention [4]. Each WSN node is equipped with at least one sensing module and a communication module, which enable the collection of monitoring data and transmission to a base station, situated in close proximity to the WSN. The base station may be a desktop computer or a laptop.

The node deployment and placement can be realised either in a predefined way or randomly, depending on the application and the environment. In hostile environments for example, sensors may be dropped from an aeroplane. With such random placement however, 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 the base station). In the *sleep* mode, a sensor cannot monitor or communicate. However, the node can change to the active mode, any time it receives the appropriate signal (either from another sensor or the base station). Obviously, in the sleep mode a sensor consumes much less energy than in active mode. Finally, in the *off* mode, the nodes are completely inactive. This happens either when the battery of a sensor is exhausted or when the sensor is turned off completely.

Sensors have a limited battery life. If all sensors in a terrain were to be activated at the same time, then the lifetime of the network would be equal to the lifetime of a single sensor (supposing that all sensors have almost equal battery life), say $k$ hours. Clearly, if the uptime of each sensor could be programmed in an efficient and practical way, the lifetime of the WSN would be extended. In order to achieve this, the nodes must be divided into a number of subsets, called *cover sets*. Sensors belonging to the currently scheduled cover set are in active mode, while others are in sleep mode. Each cover set is capable of covering all monitored targets[1]. If the cover sets are node-disjoint sets, then each sensor will be allowed to participate only in one cover set, and thus each set will have a maximum lifetime of $k$. By periodically switching between sets of coverage sensors, we can extend the target coverage time to $s \cdot k$, where $s$ is the number of available sets.

WSN coverage problems are divided into *point coverage* and *area coverage* scenarios. The objective in *point coverage* is to cover a set of points with a subset of the randomly deployed sensor nodes. Every point is monitored by at least one sensor. In the *area coverage* problem, the monitored space is divided into smaller areas called *fields* [5]. Each field is uniquely identified by a set of covering sensors that completely cover (monitor) this field. The goal in area coverage is to produce sets of sensors, that completely cover all fields and consequently the entire area. As described in [6], the area coverage problem is closely linked to the point coverage problem, since the fields are equivalent to the points mentioned previously. As far as the coverage algorithms are concerned, the area coverage problem provides an algorithm with much more targets rather than sensors, while the opposite is true for the point coverage case.

In this paper, we propose a novel centralised algorithm for the generation of node-disjoint cover sets[2]. Each sensor set is capable of monitoring independently all registered targets. The algorithm provides efficient solutions for both point and area coverage problems. Furthermore, we simulate the proposed algorithm and compare its results to that of [5], with regard to the total time of execution and number of generated sets.

## II. Related Work

Cardei and Wu provide in [7] a taxonomy of sensor coverage algorithms according to several design criteria, such as: (i) the coverage *objective*, *i.e.* maximise the lifetime of the network or minimise the required number of deployed sensors, (ii) the *node deployment method*, which may be random or deterministic, (iii) the *homogeneous or heterogeneous nature* of the nodes, *i.e.* whether all nodes have a common sensing

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

[2]This work has been partially supported by the EU through the IST project SWEB.

or communication range, (iv) the degree of *centralisation*, *i.e.* centralised vs distributed algorithms, (v) *additional requirements* for energy efficiency and connectivity. An additional criterion that can be added to the above list is *cover set independence*, *i.e.* whether a node appears in exactly one of the generated sets (node-disjoint coverage algorithms) or not (non-disjoint coverage algorithms).

In the next subsections we present algorithms that maximise the lifetime of the WSN and can be used in random sensor deployment scenarios with homogeneous device characteristics.

### A. Centralised Node-Disjoint Coverage Algorithms

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

Slijepcevic and Potkonjak [5] propose such an 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 from the intersection of coverage limits of sensors and/or the physical limits of the whole area. Every sensor covers one or more fields and one field is covered by at least one sensor. First, the algorithm covers the more sparsely populated fields (the fields that are covered by the smallest number of sensors). These 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. In this way it is assured (during the construction of a cover set) that only one node covering a particular critical field shall be selected. The complexity of the algorithm is $O(n^2)$, where $n$ is the total number of sensors.

Cardei *et al* [8] propose an algorithm to solve the same problem. They construct an undirected graph $G = (V, E)$, where $V$ is the set of sensors and $E$ the set of edges, such that $u\ v \in E$ if and only if $u, v$ 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 [9], despite the production of more sets than [5], 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 [6] propose an algorithm in order to solve the random target coverage problem. This problem can be 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, first introduced by Slijepcevic and Potkonjak [5], as a generalisation of the 3-SAT problem [10]. The authors 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 [6] show a slight improvement in the number of produced sets in comparison to [5], while the complexity of the algorithm depends on the complexity of the mixed integer programming with the results exhibiting a substantial delay.

Abrams *et al* [11] propose a centralised algorithm utilising randomisation to solve the coverage problem. Each generated cover set does not cover all targets and all cover sets must be scheduled successively in order to achieve an 80% coverage of the monitored areas. The complexity of this algorithm is $O(nk|S_{max}|)$, where $n$ is the number of sensors available, $k$ the number of generated cover sets and $|S_{max}|$ is the maximum number of fields a sensor covers.

### B. Centralised Non-disjoint Coverage 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.

Cardei *et al* [12] propose a *Linear Programming* (LP) solution to the target coverage problem for non-disjoint sets. Although the LP algorithm presents high complexity $O(p^3n^3)$, where $p$ is the number of covers and $n$ the number of sensors), the authors also propose a greedy algorithm, with a smaller complexity ($O(dm^2n)$, where $d$ is the number of sensors that cover the targets with the minimum cardinality and $m$ is the number of targets).

Another LP technique is proposed by Berman *et al* [13]. It is based on the $(1+\epsilon)$-approximation of the Garg and Könemann algorithm [14].

Non-disjoint scheduling policies exhibit decreased resiliency and dependability in the presence of faulty or corrupted nodes, since the same node may have to participate in more than one sets. Moreover, non-disjoint algorithms may generate more cover sets than node-disjoint ones, but the generating algorithm incurs a higher order of complexity.

### C. Distributed Coverage Algorithms

In *distributed* coverage algorithms a number of sensor nodes perform the required calculations cooperatively and they then disseminate the scheduling information to the rest of the sensors. This scheme may require some processing from the sensors involved, but it scales better to accommodate larger networks.

Several distributed algorithms have been proposed in the literature [15], [16]. These approaches target to produce not only an efficient coverage algorithm, but also a distributed and localised scheduling scheme, in order to rapidly disseminate the scheduling information throughout the network.

Since the algorithm presented in this paper belongs to the *centralised* category, we will not provide further details on distributed schemes, for sake of brevity.

## III. Problem Description

In this section we describe the characteristics of a centralised algorithm that generates node-disjoint sensor sets, each capable of independently monitoring all targets.

Let $T_0 = \{t_1, t_2, \ldots, t_k\}$ be the set of target points and $S_0 = \{s_1, s_2, \ldots, s_n\}$ the set of sensor nodes. Each target point 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 $r$. We assume that any target lying within the circle defined by the location of a sensor (circle centre) and range $r$ (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.* $\forall\ s_j \in N_i\ :\ |t_i - s_j| \leq r,\ N_i \subseteq S_0,\ t_i \in T_0$, 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 $T_0$. A specific sensor may appear in more than one "neighbour–sensor" sets.

The 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 must cover all targets found in $T_0$. In a node-disjoint algorithm, each sensor is allowed to participate only in one cover set, *i.e.* $C_i \cap C_j = \emptyset$, $\forall\ i, j : 1 \leq i, j \leq m$ and $i \neq j$.

The objective of such an algorithm is to maximise the cardinality $|C|$ of the generated collection $C$, thus producing more sensor sets and extending the lifetime of the sensor network. Obviously, 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 node-disjoint coverage algorithm can produce at most $x$ cover sets, since (i) each cover set must cover all targets and (ii) each of the sensors included in the "neighbour-sensor" set can only be part of one cover set.

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 the sets that can be generated for a given input $I$. As described above, once all sensors from small "neighbour-sensor" sets have been used, 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 (since each sensor can only be part of one cover set). The algorithm will quickly run out of sensors covering this Critical Target and 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 real maximum number of cover sets, without actually trying combinations of cover sets.

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 [5], requires recalculation of the most Critical Targets at the beginning of each loop of the algorithm. Our approach (described in more detail in the next section) tags each sensor with an attribute describing the number of Critical Targets it covers. Sensors can then be sorted by this attribute and the sensor with the minimum Critical Target covers, may be selected. This 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. The attribute value is computed once during the *Setup* phase of the algorithm and depends only on the initial cover relationships between sensors and targets (*i.e.* set $I$).

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. 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 generating more cover sets). Each cover set must therefore contain a minimal amount of sensors, leaving as many sensors as possible for the next cover sets to utilise.

## IV. Algorithm Description

We will now present B{GOP}, a novel centralised coverage algorithm for the generation of node-disjoint cover sets.

### A. Setup

During the *Setup* phase, B{GOP} uses input $I$ to construct the following auxiliary constants:

- The initial target set $T_0$
- The initial sensor set $S_0$
- The "neighbour-sensor" set $N_i$ for each target $t_i \in T_0$
- The "point-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 $B_j$ describing the amount of Critical Targets, sensor $s_j$ covers. The attribute value is computed according to the following formula:

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

where $N_i$ is the "neighbour-sensor" set of the i-th element of $P_j$.

- Maximum *badness*: $max\_badness = \max(B_1, \ldots, B_k)$, $k = |T_0|$.
- The *theoretical* maximum number of possible generated sets, $max\_sets = \min(|N_1|, \ldots, |N_k|),\ k = |T_0|$.

### B. Main Algorithm

B{GOP} starts by initialising $S_{cur}$, a set that keeps track of the available sensors, to $S_0$. The main algorithm is implemented as a greedy heuristic, mapped to three nested loops (see Alg.1): the "Unused Sensor Check" loop, the "Uncovered Target Check" loop and the "Sensor Applicability Check" loop.

The "Unused Sensor 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 initialises the set of currently uncovered targets $T_{cur}$ with the targets found in $T_0$. It then passes control to the "Uncovered Target Check" loop which is responsible for populating the cover set $C_{cur}$ with sensors. Once $C_{cur}$ is ready, it is added to the collection $C$. If the algorithm detects that it has generated the maximum number of possible cover sets ($max\_sets$), it stops searching for further sets and returns the current collection of cover sets $C$.

The "Uncovered Target Check" loop is responsible for selecting a sensor from a set of candidates, provided by the "Sensor Applicability Check" loop. Each candidate belongs to one of four classes (see Fig. 1), depending on the number of targets it covers in $T_{cur}$ and $T_0 - T_{cur}$:

- **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 to cover sets. If no member of the $Best$ class exists, we can try one of the other classes in the order $Good \rightarrow OK \rightarrow Poor$, although this will provide us with results far from the optimum. To clarify this point, one may consider a scenario where it is preferable to select a $Poor$ sensor, covering one already covered target and five uncovered, rather than a $Good$ sensor covering just one uncovered target. Instead of enforcing a static order of preference, the B{GOP} algorithm merges all members of the $Good$, $OK$ and $Poor$ classes into a new class and sorts them according to a weight calculated within the "Sensor Applicability Check" loop.

Once the best qualifying candidate has been selected, it is removed from the current sensor set $S_{cur}$ and added to the current cover set $C_{cur}$. Additionally, all targets monitored by this sensor are removed from $T_{cur}$. 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 to the collection $C$.

**Algorithm 1** B{GOP} Adaptive Coverage Algorithm

**Require:** $S_0 \neq \emptyset$, $T_0 \neq \emptyset$, $P \neq \emptyset$, $B \neq \emptyset$, $max\_sets > 0$, $max\_badness > 0$

$C = \emptyset$
$S_{cur} = S_0$
**while** $S_{cur} \neq \emptyset$ **do** /* Unused sensor check */
  $C_{cur} = \emptyset$
  $T_{cur} = T_0$
  **while** $T_{cur} \neq \emptyset$ **do** /* Uncovered target check */
    $best := none$
    $other := none$
    $selected := none$
    $min\_badness := max\_badness + 1$
    $max\_benefit := 0$
    **for all** $s \in S_{cur}$ **do** /* Sensor applicability check */
      **if** $freq(P_s, T_{cur}) = 0$ **then**
        ignore this sensor
      **else if** $freq(P_s, T_{cur}) = freq(P_s, T_0)$ **then**
        **if** $B_s < min\_badness$ **then**
          $min\_badness := B_s$
          $best := s$
        **end if**
      **else**
        $in := freq(P_s, T_{cur})$
        $out := freq(P_s, T_0) - in$
        $\alpha := \dfrac{|C|}{max\_sets}$
        $\beta := 1 - \dfrac{B_s}{max\_badness}$
        $benefit := \dfrac{in}{(out+1)^{\alpha}} + \beta$
        **if** $benefit > max\_benefit$ **then**
          $max\_benefit := benefit$
          $other := s$
        **end if**
      **end if**
    **end for** /* Sensor applicability check */
    $selected := best$ **or** $other$
    **if** $selected = none$ **then**
      **return** $C$
    **end if**
    $S_{cur} = S_{cur} - \{selected\}$
    $T_{cur} = T_{cur} - P_{selected}$
    $C_{cur} = C_{cur} \cup \{selected\}$
  **end while** /* Uncovered target check */
  $C = C \cup \{C_{cur}\}$
  **if** $|C| = max\_sets$ **then**
    **return** $C$
  **end if**
**end while** /* Unused sensor check */
**return** $C$

The "Sensor Applicability Check" loop is responsible for classifying candidates according to their coverage status, sorting members of the $Best$ and $merged$ classes and, finally, selecting the top $Best$ and $merged$ class sensors. To test the coverage status of a sensor, the utility function $freq(P_j, T)$ is used, which counts the targets in $T$ that sensor $s_j$ covers, *i.e.*:

$$freq(P_j, T) = |P_j \cap T|$$

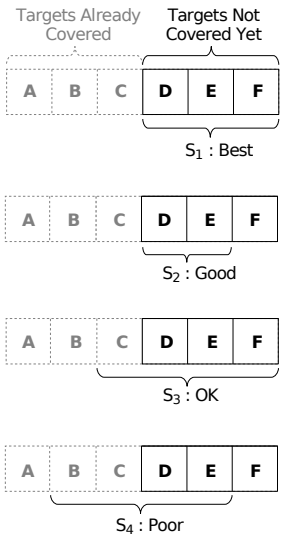


Fig. 1. The four classes of sensor candidates: Best, Good, OK and Poor

If the candidate covers no targets from the current target set $T_{cur}$, it can be safely ignored.

Candidates of the $Best$ class are sorted according to their $badness$ attribute $B_j$ (see *Setup*), while candidates of the $merged$ classes are sorted according to the following $benefit$ attribute:

$$benefit = \frac{in}{(out+1)^{\alpha}} + \beta$$

where $in$ is the number of uncovered targets a candidate covers, $out$ is the number of already covered targets a candidate covers, $\alpha$ is a measure of how close the algorithm is to providing the maximum number of generated sets and $\beta$ is the normalised value of the $badness$ attribute of the sensor. The objective of the $benefit$ attribute is threefold: *a)* to select candidates that cover as many uncovered targets as possible (thus generating smaller cover sets), *b)* to select candidates that cover as few already covered targets as possible (thus minimising the probability of double-covering a *Critical Target*) and *c)* to risk selecting a sensor from the $OK$ or $Poor$ classes when it is of low $badness$ or when the algorithm has still many cover sets to produce. By managing this risk intelligently, the algorithm adapts to the needs of each coverage scenario (sparse/dense network, area/point coverage).

### C. Algorithm Analysis

In order to select a single sensor for inclusion to 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 selected sensor has left uncovered. Once a generated set is complete (*i.e.* covers all targets), the algorithm will start building a new one and 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 III the *theoretical* maximum number of generated sets is sometimes impossible to achieve, since the algorithm exits prematurely, having 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 B{GOP} would have included all available sensors in the generated sets. In that case, the time taken to produce the generated sets from $n$ sensors and $k$ targets would have been proportional to the following sum:

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

We can thus deduce that the total running time of B{GOP} is $O(n^2k)$.

We will now prove that B{GOP} is capable of generating at least one cover set, if one exists. 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 targets in $T_0$ with sensors from $S_0$) but B{GOP} has failed to produce a cover set. This effectively 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 $T_{cur}$ :

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

The set $C_{cur}$ contains the sensors already selected for inclusion to 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 $C_{cur}$:

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

Any 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, the B{GOP} algorithm will have a sufficient number of sensors to generate at least one cover set.

### D. 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 set 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[3]. By minimising the number of unnecessary calls to $freq(P,T)$, the total execution time of the algorithm can be reduced.

The initial coverage of a sensor $freq(P_s,T_0)$ 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 coverage $freq(P_s,T_0)$ whenever the current target set is initialised to the original target set (*i.e.* $T_{cur}=T_0$).
- As soon as a sensor has been selected for the generated set, the targets the 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 the next section.

## V. Simulation

To test the efficiency of the algorithm presented in this paper, we developed a prototype implementation in the Perl programming language. A set of Perl scripts were responsible for generating two types of virtual terrains: (i) a 2-dimensional $1km^2$ terrain with a user-defined amount of sensors, targets and occupied terrain space, each sensor having a communication radius of $50m$ and sensing radius of $2m$, placed randomly on the terrain following the uniform distribution, and (ii) random sensor-to-target assignment, following the uniform distribution, with user-defined number of sensors, targets and maximum number of targets per sensor.

We examine three different simulation scenarios. We run each simulation scenario 5 times, with random target and sensor deployment and we compute for each scenario the average results of these 5 runs. Moreover, we run each one of these 5 runs (per simulation scenario) 5 times in order to compute the minimum time values. The value "average time per set" is reported as the average of the minimum time values, divided by the number of produced sets. For all of the examined scenarios, we compare our algorithm's results to those of the Slijepcevic and Potkonjak [5] algorithm. All experiments were carried out on a Pentium III 1Ghz host with 512MB of RAM, running the Debian GNU/Linux operating system.

[3]should a hash-table be employed for set member lookups

| | B{GOP} | | | Slijepcevic | | |
|---|---|---|---|---|---|---|
| Max Covers per Sensor | Sets produced | Total time | Avg time per set | Sets produced | Total time | Avg time per set |
| 5 | 18.6 | 2.836s | 0.155s | 18.6 | 0.313s | 0.017s |
| 10 | 38.8 | 4.215s | 0.109s | 37.8 | 0.571s | 0.015s |
| 15 | 53.2 | 5.049s | 0.095s | 51.2 | 1.043s | 0.020s |
| 20 | 69.8 | 5.999s | 0.086s | 65.0 | 1.874s | 0.029s |
| 25 | 82.4 | 6.902s | 0.084s | 77.0 | 3.067s | 0.040s |
| 30 | 95.8 | 7.956s | 0.083s | 89.0 | 4.861s | 0.055s |
| 35 | 108.6 | 8.996s | 0.083s | 101.2 | 7.197s | 0.071s |
| 40 | 121.8 | 10.245s | 0.084s | 113.0 | 10.485s | 0.093s |
| 45 | 135.6 | 11.331s | 0.084s | 126.2 | 14.671s | 0.116s |
| 50 | 147.2 | 12.711s | 0.086s | 138.4 | 19.506s | 0.141s |
| 55 | 159.6 | 14.074s | 0.088s | 148.2 | 24.302s | 0.164s |
| 60 | 172.2 | 15.647s | 0.091s | 161.2 | 31.718s | 0.197s |
| 65 | 186.0 | 17.526s | 0.094s | 174.8 | 40.611s | 0.232s |
| 70 | 202.6 | 19.868s | 0.098s | 189.6 | 51.852s | 0.273s |
| 75 | 220.2 | 22.370s | 0.103s | 206.4 | 67.242s | 0.326s |
| 80 | 232.4 | 24.196s | 0.104s | 219.6 | 81.481s | 0.371s |

TABLE I

1ST SCENARIO (1000 SENSORS, 100 TARGETS): NUMBER OF PRODUCED SETS IN RELATION TO THE INCREASE IN SENSOR COVERAGE CAPACITY

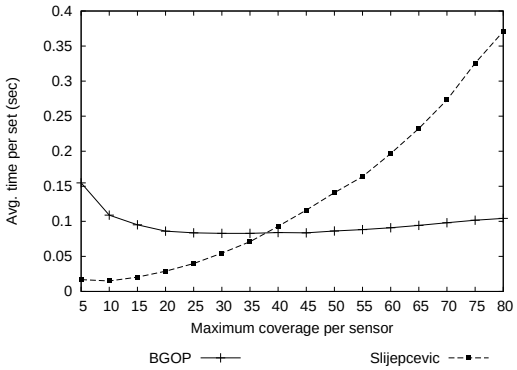


Fig. 2. 1st scenario (1000 sensors, 100 targets): Average time per produced set

The first scenario involves the deployment of 1000 sensor nodes and 100 targets. We examine the effect of an increase in the maximum number of targets that a sensor node can cover. This is equivalent to an increase in the sensor coverage range. As shown in Table I, B{GOP} produces more cover sets than the algorithm of [5]. The improvement of B{GOP} becomes more intense, as we move towards a more dense deployment. An interesting point, as shown in Fig. 2, is that the B{GOP} algorithm exhibits almost constant performance. The Slijepcevic and Potkonjak algorithm starts with better runtimes, but these grow exponentially as the number of targets-per-sensor increases.

In the 2nd scenario, the network consists of 100 sensors, while the number of targets ranges from 10 to 50. A sensor node may cover all targets. Fig. 3 compares the average number of covers computed by the B{GOP} and Slijepcevic and Potkonjak algorithms. This graph also illustrates the *theoretical* maximum number of possible sets, for each simulated case.

Finally, the 3rd scenario examines the performance of B{GOP} in the area coverage domain. In this case we use a constant number of 1000 randomly dropped sensor nodes,

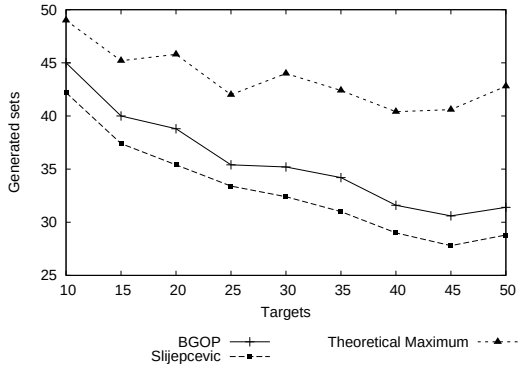


Fig. 3. 2nd scenario (100 sensors, 10-50 targets): Number of produced sets

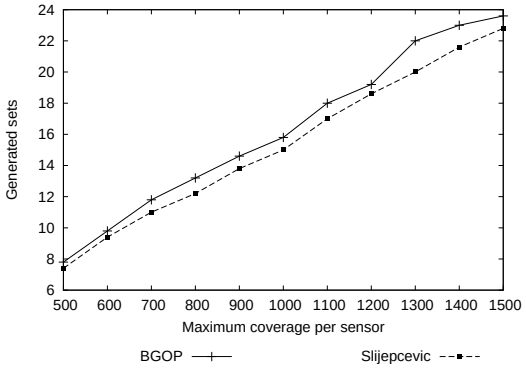


Fig. 4. 3rd scenario (1000 sensors, 5000 fields): Number of produced sets

| | B{GOP} | | | Slijepcevic | | |
|---|---|---|---|---|---|---|
| Max Covers per Sensor | Sets produced | Total time | Avg time per set | Sets produced | Total time | Avg time per set |
| 500 | 7.8 | 8.270s | 1.061s | 7.4 | 4.377s | 0.592s |
| 600 | 9.8 | 10.228s | 1.043s | 9.4 | 6.672s | 0.710s |
| 700 | 11.8 | 12.352s | 1.047s | 11.0 | 9.161s | 0.833s |
| 800 | 13.2 | 14.136s | 1.071s | 12.2 | 11.893s | 0.974s |
| 900 | 14.6 | 16.238s | 1.112s | 13.8 | 15.318s | 1.110s |
| 1000 | 15.8 | 18.116s | 1.147s | 15.0 | 19.103s | 1.273s |
| 1100 | 18.0 | 21.650s | 1.203s | 17.0 | 25.738s | 1.514s |
| 1200 | 19.2 | 23.922s | 1.246s | 18.6 | 30.786s | 1.653s |
| 1300 | 22.0 | 28.905s | 1.314s | 20.0 | 40.862s | 2.043s |
| 1400 | 23.0 | 31.205s | 1.357s | 21.6 | 47.564s | 2.202s |
| 1500 | 23.6 | 33.269s | 1.410s | 22.8 | 54.853s | 2.405s |

TABLE II

3RD SCENARIO (1000 SENSORS, 5000 FIELDS): SIMULATION RESULTS FOR VARYING MAX. NUMBER OF COVERED FIELDS PER SENSOR.

5000 fields and a varying maximum number of fields that a sensor may cover. Fig. 4 illustrates the number of produced sets of both algorithms. As shown in Table II, the B{GOP} algorithm produces more cover sets with a satisfactory growth rate in execution time.

## VI. CONCLUSIONS

In this paper, we have presented B{GOP}, a centralised coverage algorithm for WSN. Our work contributes to the field of sensor coverage in the following ways:

- Modelling of greedy coverage algorithms for node-disjoint sets.
- Introduction of a novel efficient algorithm for both point and area coverage scenarios.
- Introduction of sensor candidate categorisation, with 4 classes of sensors, depending on their coverage status.
- Flexible avoidance of double-covering a critical field, by ranking sensors according to the coverage status of the fields they cover.
- Adaptive sensor selection policy based on: class, coverage of critical fields and number of available sensors.

Our simulations have shown that B{GOP} outperforms algorithms such as [5] and that it is suitable for use in any area or point coverage scenario.

## REFERENCES

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor network: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[2] 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*. ACM, September 2002, pp. 88–97.

[3] L. Hai, P. Wan, C.-W. Yi, J. Xiaohua, S. Makki, and N. Pissinou, "Maximal lifetime scheduling in sensor surveillance networks," in *Proc. of INFOCOM 05*, vol. 4. IEEE, March 2005, pp. 2482–2491.

[4] A. Goldsmith and S. Wicker, "Design challenges for energy-constrained ad hoc wireless networks," *Wireless Communications, IEEE*, vol. 9, no. 4, pp. 8–27, August 2002.

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

[6] M. Cardei and D.-Z. Du, "Improving wireless sensor network lifetime through power aware organization," *ACM Wireless Networks*, vol. 11, no. 3, pp. 333–340, 2005.

[7] M. Cardei and J. Wu, "Energy efficient coverage problems in wireless ad hoc sensor networks," *Computer Communications*, vol. 29, no. 4, pp. 413–420, 2006.

[8] 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*, vol. 3, no. 3-4, pp. 213–229, 2002.

[9] M. T. Thai, F. Wang, H. Du, and X. Jia, "Coverage problems in wireless sensor networks: Designs and analysis," *International Journal of Sensor Networks*, to appear, 2006.

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

[11] Z. Abrams, A. Goel, and S. Plotkin, "Set k-cover algorithms for energy efficient monitoring in wireless sensor networks," in *Proc. of Third International Symposium on Information Processing in Sensor Networks*. ACM, 2004, pp. 424–432.

[12] M. Cardei, M. Thai, Y. Li, and W. Wu, "Energy-efficient target coverage in wireless sensor," in *Proc. of INFOCOM 05*, vol. 3. IEEE, March 2005, pp. 1976–1984.

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

[14] N. Garg and J. Könemann, "Faster and simpler algorithms for multi-commodity flow and other fractional packing problems," in *Proc. of 39th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, November 1998, pp. 300–309.

[15] 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*. ACM Press, September 2002, pp. 32–41.

[16] F. Y. Zhong, G. S. Lu, and L. Zhang, "Peas: a robust energy conserving protocol for long-lived sensor networks," in *Proc. of 10th IEEE International Conference on Network Protocols*, 2002, pp. 200–201.