https://doi.org/10.1016/j.cor.2015.02.013

Maximizing Lifetime in Wireless Sensor Networks with Multiple Sensor Families

Francesco Carrabs[1], Raffaele Cerulli[1], Ciriaco D'Ambrosio[2], Monica Gentili[1], Andrea Raiconi[1,*]

Abstract

Wireless sensor networks are generally composed of a large number of hardware devices of the same type, deployed over a region of interest in order to perform a monitoring activity on a set of target points. Nowadays, several different types of sensor devices exist, which are able to monitor different aspects of the region of interest (including sound, vibrations, proximity, chemical contaminants, among others) and may be deployed together in a heterogeneous network. In this work, we face the problem of maximizing the amount of time during which such a network can remain operational, while maintaining at all times a minimum coverage guarantee for all the different sensor types. Some global regularity conditions in order to guarantee a fair level of coverage for each sensor type to each target are also taken into account in a second variant of the proposed problem. For both problem variants we developed an exact approach, which is based on a column generation algorithm whose subproblem is either solved heuristically by means of a genetic algorithm or optimally by an appropriate ILP formulation. In our computational tests the proposed genetic algorithm is shown to be able to dramatically speed up the procedure, enabling the resolution of large-scale instances within reasonable computational times.

Keywords: Wireless Sensor Networks, Multiple Families, Maximum Lifetime Problem, Column Generation, Genetic Algorithm

*Corresponding author

Email addresses: fcarrabs@unisa.it (Francesco Carrabs), raffaele@unisa.it (Raffaele Cerulli), cdambrosio@unisa.it (Ciriaco D'Ambrosio), mgentili@unisa.it (Monica Gentili), araiconi@unisa.it (Andrea Raiconi)

[1]Department of Mathematics, University of Salerno, Via Giovanni Paolo II 132, 84084 Fisciano (SA), Italy

[2]Department of Computer Science, University of Salerno, Via Giovanni Paolo II 132, 84084 Fisciano (SA), Italy

1. Introduction

Due to technological advances which enabled their deployment in relevant and diverse scenarios, Wireless Sensor Networks (WSNs) have been studied extensively in the last years. Possible application contexts include environmental monitoring, traffic control, patient monitoring in healthcare and intrusion detection, among others (see, for example, [1], [2], [3]). The general structure of a WSN is composed of several hardware devices (*sensors*) deployed over a given region of interest. Each sensor can collect information or measure physical quantities for a subregion of the space around it (its *sensing area*), or monitor specific points of interest in the area (*targets*). The targets inside the sensing area of a given sensor are defined as *covered* by it.

Individual sensors are usually powered by batteries which make it possible to keep them functional for a limited time interval, with obvious constraints related to cost and weight factors. Using a network of such devices in a dynamic and coordinated fashion makes it possible to overcome the limitations in terms of range extension and battery duration which characterize each individual sensor, enabling elaborate monitoring of large regions of interest. Prolonging the amount of time over which such monitoring activity can be carried out has therefore emerged as an issue of paramount relevance. This problem, generally known as Maximum Lifetime Problem (MLP), has been widely approached in the literature by proposing methods to determine several possibly overlapping subsets of sensors which are independently able to provide coverage for the target points (*covers*), and by activating them one by one for appropriate amounts of time such that battery constraints are not violated. It should be noted that while sensors could be considered as belonging to different states during their usage in the intended application (such as receiving, transmitting, or idle) in this context two essential states can be identified. That is, each sensor may currently be active (i.e. used in the current cover, and consuming its battery) or not. Activating a cover refers therefore to switching all its sensors to the active state, while switching off all the other ones.

Many works have been proposed in the literature to address MLP and several problem variations. The problem was shown to be NP-Complete in [4]. Earlier works such as [5] and [4] presented approximation and heuristic algorithms to

solve it. The proposed variants of the problem include, among others, cases where a certain percentage of targets may be left uncovered by each cover ([6], [7], [8]) or where the sensing ranges can be adjusted in order to provide optimal trade-offs among coverage and energy consumption ([9], [10], [11], [12]). In works such as [13], [14] and [15], connectivity issues are taken into account in order to route the collected information to a central processing facility. In [16], the authors assume this data collecting node (which they call *sink*) to be able to move to different positions during the monitoring phase, and present two MLP variants; in the first one, the information collected by each node must be forwarded to the sink at all times, while in the second one, nodes may decide to locally store information to forward it to the sink when it moves to a more favorable location.

Moreover, while the sensing range of each device is typically only limited by a certain threshold distance (i.e. they provide coverage on 360 degrees around them), some authors also investigated the case in which the sensing activity is limited to an adjustable restricted angle ([17], [18], [19]), as in the case of video cameras or ultrasonic sensors. Among the proposed resolution methods for MLP variants, Column Generation algorithms have recently proved to be effective methods to solve reasonably large instances to optimality ([6], [10], [11], [12], [13], [14], [16], [19]).

Most of the above presented works take into account homogeneous networks, that is, networks whose sensing devices are perfectly equal and therefore have the same capabilities. This assumption makes sense in many scenarios where a large number of devices based on the same hardware is deployed. However sensor heterogeneity in this contest has been studied as well, in terms of different metrics. In [20], [21], [22], [23], [24] a subset of sensors is provided with larger batteries, and in some cases has longer transmission ranges and better processing capabilities, often in relation to clustering schemes where such sensors serve as cluster heads (sometimes called supernodes). Other works consider heterogeneity in a non-hierarchical context, allowing individually different sensors. For example, sensors with possibly variable battery durations are discussed in [25] and [19], while heterogeneous sensing ranges were analyzed in [26] and [27].

Fewer research efforts have been devoted to the case of networks composed of distinct categories of sensors, each fulfilling a different purpose. Indeed, it could be of interest to monitor several aspects of the same region of interest. For

example, while monitoring a certain geographical area for environmental control purposes, different types of sensors could be employed to monitor pollution levels, temperatures, vibrations, as well as for intrusion detection and other relevant properties. This interpretation of heterogeneity was discussed in [28], where the authors propose a hardware and software testbed for wireless sensor network applications, including sensors with auxiliary energy sources based on solar cells and modular sensor headers.

In this work, we study WSNs where sensors belong to different types, from now on defined as *families*, and present two variants of MLP, namely the Maximum Lifetime with Multiple Families Problem (MLMFP) and the Regular Maximum Lifetime with Multiple Families Problem (MLMFP-R).

Note that, if each target needs to be covered by every family where the WSN is activate, then finding a solution would merely reduce to solving MLP separately for each family, with an objective function value equal to the minimum among such maximum lifetimes. In fact, the covers could be activated in parallel, and the monitoring activity would continue until one of the families has no covers available. However, such a hard requirement could be too restrictive for many real-world cases. It could be reasonable for a portion of the targets to be left uncovered by each family in each cover, as long as some minimum family-dependent threshold is met, and coverage of all the targets is provided by at least one of the families at all times. Consider, for instance, a fire detection scenario which makes use of different types of sensors to monitor heat, humidity and smoke levels. While perfect knowledge using all types of sensors for all target points would be ideal, detections with a high level of accuracy may still be possible if each target is covered by only one or two types of sensors, and the information gathered by sensors monitoring a subset of targets located in the same portion of the area suggests that a fire event is indeed happening. Some sensor types may be more relevant for the detection of the phenomenon of interest (for example, heat or smoke); therefore, a balance between network lifetime and detection accuracy may be obtained by choosing a percentage of the targets that should be covered by such families at all times, which represents the above mentioned threshold.

The regular version of the problem (MLMFP-R) also takes into account some regularity constraints where the aim is to maximize the minimum amount of time for which each target is covered by each family in the solution.

For both problem variants, an exact approach based on Column Generation (CG) is developed and presented, as well as a genetic algorithm which is embedded within the CG to improve its performances.

The rest of the paper is organized as follows. In Section 2 we formally introduce the two problems. The Column Generation exact approach is described in Section 3. In Section 4 we present our genetic algorithm as well as its integration within the CG framework. Section 5 presents the results of our computational experiments. Finally, Section 6 contains our final remarks.

2. Notation and Problems Definition

Consider a wireless network (S, T, F), where $S = \{s_1, \ldots, s_m\}$ is the set of the sensors, $T = \{t_1, \ldots, t_n\}$ is the set of the targets, and $F = \{f_1, \ldots, f_z\}$ is the set of the sensor families. As previously introduced, each sensor is assigned to a family and is able to monitor a subset of targets defined by its sensing range. For each $t_k \in T$ and $s_i \in S$, let γ_{ki} be a binary parameter equal to 1 if t_k is covered by s_i, 0 otherwise. Furthermore, let $\{S_1, \ldots, S_z\}$ be a partition of S, such that $s_i \in S_a$ if the family of sensor s_i is f_a, $\forall a \in \{1, \ldots, z\}$.

A cover $C_j \subseteq S$ is defined in the classical MLP problem as a subset of sensors such that each target of T is covered by at least one sensor in C_j, that is, $\sum_{s_i \in C_j} \gamma_{ki} \geq 1, \forall t_k \in T$. For a cover to be feasible, we consider an additional condition which imposes a minimal coverage threshold to be satisfied by each family. That is, given the *coverage requirement* $0 \leq \tau_a \leq n_a$ associated with f_a, where n_a is the number of targets covered by the sensors in S_a, C_j is feasible if and only if the sensors in $C_j \cap S_a$ cover at least τ_a different targets.

The MLMFP problem consists of finding a set of feasible covers $C_1, \ldots, C_u$ and of assigning a positive activation time $w_1, \ldots, w_u$ with each of them, such that the overall network lifetime is maximized and the battery duration constraint for each sensor is not violated.

Let us assume that we can compute in advance the complete set of feasible covers $\mathcal{C} = \{C_1, \ldots, C_\ell\}$. For each $s_i \in S$ and $C_j \in \mathcal{C}$, let ϕ_{ij} be a binary parameter equal to 1 if s_i belongs to C_j and 0 otherwise. Let us assume each battery duration is normalized to 1 time unit. Then, MLMFP can be described

by the following linear programming formulation:

$$[\mathbf{P}] \max \sum_{C_j \in \mathcal{C}} w_j \tag{1}$$

s.t.

$$\sum_{C_j \in \mathcal{C}} \phi_{ij} w_j \leq 1 \qquad \forall s_i \in S \tag{2}$$

$$w_j \geq 0 \qquad \forall C_j \in \mathcal{C} \tag{3}$$

The objective function (1) maximizes the total network lifetime. Constraints (2) ensure that, for each sensor, the sum of the activation times of the covers in which it is contained does not exceed its normalized battery duration.

Let us consider a solution of MLMFP composed of a set of feasible covers and the related activation times. Additionally, for each target t_k and each family f_a, let w_{ka} be the amount of time t_k is covered by sensors belonging to S_a in the solution. We define the solution to be *regular* if $w_{min} = \min\{w_{ka} | t_k \in T, f_a \in F\}$ is maximized. The regular version of the problem (i.e., MLMFP-R) consists of finding a regular solution which maximizes the network lifetime. The motivation for seeking a regular solution is to avoid any target to be neglected by any family for as much as possible, by making sure that each coverage time w_{ka} is at least equal to this achievable threshold w_{min}.

Let us consider the full set of feasible covers $\mathcal{C} = \{C_1, \ldots, C_\ell\}$. For each $t_k \in T$, $f_a \in F$ and $C_j \in \mathcal{C}$, let ψ_{kaj} be a binary parameter equal to 1 if a sensor in S_a belongs to C_j and covers t_k, 0 otherwise. The problem is then defined as follows:

$$[\mathbf{P2}] \max (W + \varepsilon) w_{min} + \sum_{C_j \in \mathcal{C}} w_j \tag{4}$$

s.t.

(2), (3)

$$(\sum_{C_j \in \mathcal{C}} \psi_{kaj} w_j) - w_{min} \geq 0 \qquad \forall t_k \in T, \forall f_a \in F \tag{5}$$

$$w_{min} \geq 0 \tag{6}$$

Constraints (5) ensure, for each $t_k \in T$ and $f_a \in F$, the quantity w_{min} to be not greater than w_{ka} (that is, the sum of the activation times w_j for each $C_j \in \mathcal{C}$ such that $\psi_{kaj} = 1$). In the objective function (4) the W parameter represents an upper bound on the maximum lifetime $\sum_{C_j \in \mathcal{C}} w_j$ and ε is a small positive coefficient, such that the weighting ensures a regular solution to be sought as primary objective.

It should be noted that while MLMFP and MLMFP-R have different objective functions and the latter introduces additional constraints, each individual cover which may be part of a solution has to satisfy the same conditions in order to be feasible, and therefore the set $\mathcal{C}$ is the same for both problem variants.

The provided formulations cannot be used to solve real world instances of MLMFP or MLMFP-R, since the cardinality of the set of feasible covers $\{C_1, \ldots, C_u\}$ is potentially exponential. For this reason, we developed Column Generation algorithms to solve both the problems, as described in Section 3. Section 2.1 to follow discusses how to adapt model formulations [P] and [P2] when hardware differences among the sensors are taken into account. Section 2.2 discusses the issue of redundant covers in the feasible region of the two problems.

2.1 Modeling hardware differences

The above presented models work under the assumption that all sensor families have the same battery durations. When this is not the case both the models can be easily adapted as follows.

For each $f_a \in F$, let $\Delta_a \geq 1$ be its *consumption ratio*, that is, a parameter such that the battery duration of the sensors belonging to family f_a is normalized to $1/\Delta_a$ time units. Given the family $f_b \in F$ with the longest battery duration, we consider $\Delta_b = 1$. Therefore, for example, if sensors of family f_a consume their batteries twice as fast as sensors of f_b, then $\Delta_a = 2$ and they can be activated for 0.5 time units.

Furthermore, sensors may individually have an initial charge level which is different from the maximum for their family (for example, if the sensor was previously employed for different activities). For a given sensor s, let $0 < charge_s \leq 1$ be its initial charge percentage. Again, let $s \in S_a$ with $\Delta_a = 2$, and let $charge_s = 0.5$. Then, sensor s can be used for $charge_s/\Delta_a = 0.25$ units of time. For both problems, constraints (2) can then be modeled in the following more general form:

$$\sum_{C_j \in \mathcal{C}} \phi_{ij} w_j \leq charge_{s_i}/\Delta_a \qquad \forall f_a \in F, s_i \in S_a \tag{7}$$

2.2 MLMFP, MLMFP-R and cover redundancy

Given a feasible cover C_1, we define it *redundant* if it contains another feasible cover C_2 as a proper subset. It is straightforward to observe that if an optimal solution for MLMFP contains C_1, then an alternative one where C_2 replaces C_1 can be found. Therefore, when looking for optimal solutions for MLMFP, in the methods described in Sections 3 and 4 we focus on individuating non-redundant covers, in order to reduce the search space and speed-up the convergence of our Column Generation algorithm.

Conversely, it can be shown that an optimal solution for MLMFP-R may involve redundant coverage. To illustrate this, consider a simple network with $T = \{t_1, t_2\}$, $S = \{s_1, s_2, s_3\}$, $F = \{f_1, f_2\}$, $S_1 = \{s_1, s_2\}$, $S_2 = \{s_3\}$, $\tau_1 = \tau_2 = \Delta_1 = \Delta_2 = charge_{s_1} = charge_{s_2} = charge_{s_3} = 1$. Furthermore, let s_1 and s_2 cover t_1 and t_2, respectively, while s_3 covers both of them. This network is shown in Figure 1C, where sensors belonging to S_1 and S_2 are represented by dotted and dashed lines, respectively.

The only two feasible non-redundant covers in this network are $C_1 = \{s_1, s_3\}$ and $C_2 = \{s_2, s_3\}$, shown in Figures 1A-1B. Indeed, due to τ_2 being nonzero, s_3 needs to be in each feasible cover, and since it already covers both targets either s_1 or s_2 can be used to also satisfy the τ_1 requirement. Using this set of covers, the maximum achievable w_{min} value is 0.5, obtained when both C_1 and C_2 are activated for such amount of time. This is easy to verify, since the sum of the activation times of the two covers cannot be higher than the lifetime of s_3 which is 1, and any other feasible activation time choice (e.g., 0.6 for C_1 and 0.4 for C_2) would bring a reduction to the amount of time for which either t_1 or t_2 are covered by sensors belonging to family f_1. Conversely, by activating the redundant cover $C_3 = \{s_1, s_2, s_3\}$ for a full time unit, both w_{min} and the network lifetime are equal to 1.

3. Column Generation Approach

Delayed Column Generation (CG) is a widely used linear programming approach for LP problems with a large number of variables. The approach initially

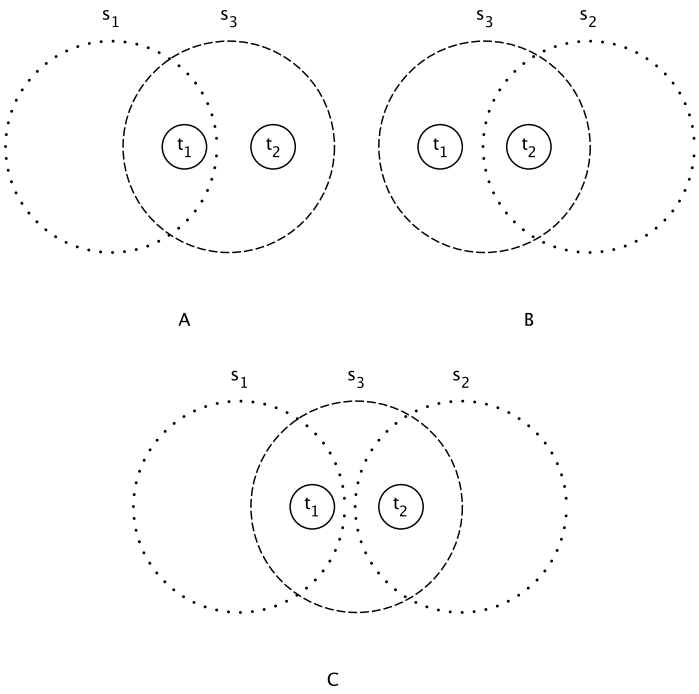


Figure 1: *Sample network. A-B: Feasible non-redundant covers C_1, C_2. C: Complete network and feasible redundant cover C_3.*

considers the original LP formulation (in our case formulations [**P**] and [**P2**]), called the *Master Problem*, restricted to a subset of variables, and optimally solves it. CG then considers a specific optimization problem (called the *Separation Problem*) which either identifies a new attractive variable to be entered in the problem or certifies the optimality of the last solution found. If a new variable is identified, it is included in the Master Problem and the procedure iterates until the optimality test is satisfied. The solution of the Separation Problem therefore avoids the enumerative assessment of all the (potentially exponential) variables that will be nonbasic in the final solution.

Consider the MLMFP problem first and let us call [**SP**] its Separation Problem. Given the last iteration of the master problem, let π_i be the dual prices associated with its constraints, that is, with each sensor. The current solution is optimal if and only if the reduced costs associated with all nonbasic variables are non negative, i.e. $\sum_{i:s_i \in C_j} \pi_i - c_j \geq 0$ for each nonbasic C_j. In our case c_j

are the coefficients of the objective function (1) and are all equal to 1; therefore, the optimality condition reduces to $\sum_{i:\phi_{ij}=1} \pi_i \geq 1$ for each nonbasic C_j. The **[SP]** objective function minimizes the sum of the dual prices of the sensors selected to be part of the new cover, and the optimality test is satisfied if the optimum value of **[SP]** is greater than or equal to 1. Constraints in **[SP]** define the construction of a feasible cover.

Let x_i, $i = 1, \ldots, m$ and y_{ka}, $k = 1, \ldots, n$, $a = 1, \ldots, z$ be two sets of binary variables. Each variable x_i represents the choice of whether or not to include the related sensor s_i in the new cover, while each variable y_{ka} will be set to 1 if target t_k is covered by a sensor belonging to f_a in the cover, 0 otherwise. The separation problem is as follows:

$$[\mathbf{SP}] \min \sum_{s_i \in S} \pi_i x_i \tag{8}$$

s.t.

$$\sum_{s_i \in S_a} \gamma_{ki} x_i \geq y_{ka} \qquad \forall f_a \in F, t_k \in T \tag{9}$$

$$y_{ka} \geq \gamma_{ki} x_i \qquad \forall f_a \in F, s_i \in S_a, t_k \in T \tag{10}$$

$$\sum_{t_k \in T} y_{ka} \geq \tau_a \qquad \forall f_a \in F \tag{11}$$

$$\sum_{f_a \in F} y_{ka} \geq 1 \qquad \forall t_k \in T \tag{12}$$

$$x_i \in \{0, 1\} \qquad \forall s_i \in S \tag{13}$$

$$y_{ka} \in \{0, 1\} \qquad \forall f_a \in F, t_k \in T \tag{14}$$

The objective function (8) makes sure that the reduced cost of the newly generated column is minimized. Constraints (9)-(10) bind the two sets of variables, by letting y_{ka} be equal to 1 if and only if at least one sensor s_i that belongs to f_a and covers t_k, is selected.

Constraints (11) ensure that the coverage requirement for each family is respected. Finally, Constraints (12) impose that all the targets are covered by at least one family (and therefore by at least one sensor).

Note that new redundant columns may be introduced when dual prices are equal to zero. This may occur in particular in the first iterations of the CG

procedure. To avoid this, we modify objective function (8) by adding a small positive costant ε to each dual price, as follows:

$$\min \sum_{s_i \in S} (\pi_i + \varepsilon) x_i \tag{15}$$

By assigning a positive weight to each x_i variable, the new objective function (15) ensures that each sensor added to the new column is needed. Note that the value of the original objective function (8) still has to be evaluated after each **[SP]** iteration in order to determine whether the optimality test is satisfied.

We now define the **[SP2]** subproblem for the MLMFP-R problem. Let π_i and q_{ka} be the dual prices related to Constraints (2) (or their generalized form (7)) and (5) for the last iteration of the **[P2]** master problem; the current solution is optimal if $\sum_{i:\phi ij=1} \pi_i + \sum_{k,a:\psi_{kaj}=1} q_{ka} \geq 1$ for each nonbasic C_j. Therefore, **[SP2]** can be expressed as follows:

$$\textbf{[SP2]} \min \sum_{s_i \in S} \pi_i x_i + \sum_{t_k \in T} \sum_{f_a \in F} q_{ka} y_{ka} \tag{16}$$

$$\text{s.t.}$$

$$(9)\text{-}(14)$$

Finally, for both **[SP]** and **[SP2]** we consider the following set of valid inequalities, which limits for each family the number of selected sensors to be equal to the cardinality of the set of targets at most:

$$\sum_{s_i \in S_a} x_i \leq T \qquad \forall f_a \in F \tag{17}$$

The main drawback of the CG approach presented above is that the subproblems are NP-Hard combinatorial optimization problems, being specializations of the set covering problem. For this reason, in the next section we introduce a genetic algorithm able to quickly compute good feasible solutions for the subproblems. We embedded this genetic algorithm in our CG approach to improve its performance.

4. Genetic Algorithm

As discussed in Section 3, the subproblems are NP-Hard and therefore it is preferable to solve them heuristically, especially for instances of considerable size. We addressed this problem by developing a genetic algorithm (GA) able to return new *attractive covers*, i.e. covers with an objective value lower than 1. The procedure generates feasible solutions for both the problems and evaluates the associated objective function value according to (8) for MFMLP and (16) for MFMLP-R. Furthermore, the GA for MFMLP always removes redundancy, while redundant covers may be generated by the GA for MLMFP-R, due to the motivations provided in Section 2.2.

The GA works within the CG framework as follows. After each iteration of the master problem, the GA is called to solve the subproblem; if it can find attractive covers, then they are added to the master problem, and the procedure iterates. Otherwise, the separation problem, i.e. either **[SP]** or **[SP2]**, is solved, such that either an attractive cover is found or the current solution is proved to be optimal.

The genetic algorithm has the advantage of considering several solutions at once. This approach can find more than a single attractive cover, potentially making it possible to reduce the number of required CG iterations and thus further reducing the computational effort.

The GA is a well-known and widely used meta-heuristic technique for optimization problems. The GA algorithms emulate the biological evolution process based on *chromosomes*, which represent solutions (e.g. feasible covers in our case) for the considered problem. Step by step, the GA produces solutions which are typically better adapted to the environment, encoded by the *fitness function*, used to rank each chromosome. This is achieved through two mechanisms, named *crossover* and *mutation*. The crossover operator combines, in a probabilistic manner, two or more selected individuals (*parent* solutions). The mutation operator, instead, randomly modifies a child chromosome derived from the crossover in order to increase diversity. The overall process is repeated until some desired stop conditions are reached. For a complete and detailed description of the genetic algorithms and their characteristics the reader can refer to [29].

The remaining part of this section describes in detail our genetic algorithm.

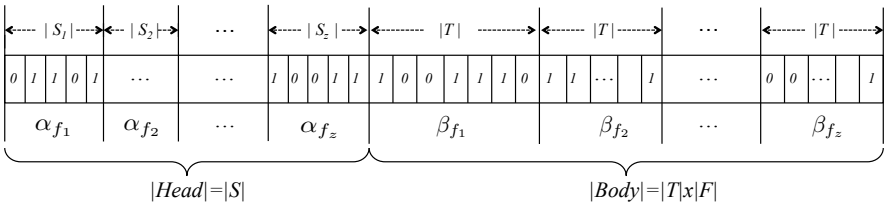


Figure 2: *Chromosome structure.*

4.1 Chromosome representation and fitness function

The chromosome representation is based on the binary encoding represented in Figure 2. It stores the set of sensors activated in a given candidate cover, as well as the related covered targets for each sensor family.

The structure is composed of two distinct components, named *Head* and *Body* respectively.

The *Head* component is a binary vector of length $|S|$. Each position is related to a sensor, and is equal to 1 if it belongs to the cover, 0 otherwise. Moreover, the sensors are sorted by family so that the first $|S_1|$ positions (defined section α_{f_1}) of the Head contain the binary values related to the sensors of family f_1, the subsequent $|S_2|$ positions (α_{f_2}) refer to the sensors of family f_2, and so on for all subsequent families. For instance, the α_{f_1} segment in Figure 2 consists of 5 positions (meaning that $|S_1| = 5$ in the network), and three of them (the second, third and fifth) are currently activated in the chromosome.

The *Body* component represents which targets are covered by each family. This component is partitioned in $|F|$ segments of size $|T|$, sorted by family. More in detail, the i-th position in β_{f_a} is equal to 1 if there is at least one sensor of f_a that covers target t_i and that is currently activated in α_{f_a}, 0 otherwise. For instance, in Figure 2 the segment β_{f_1} consists of the first $|T| = 7$ positions of the Body component, and it shows that the three sensors activated in α_{f_1} cover the targets t_1, t_4, t_5 and t_6.

In the following, we will refer to the sections α_{f_a} and β_{f_a} of a specific chromosome C as $\alpha^C_{f_a}$ and $\beta^C_{f_a}$, respectively. Furthermore, let β^C_{ka} be the position related to target $t_k \in T$ of segment β_{f_a} of chromosome C.

The chromosome representation can be used to check whether it represents a feasible solution. Formally, a given chromosome C is feasible if and only if the following two conditions hold:

$$\sum_{t_k \in T} \beta^C_{ka} \geq \tau_a \qquad \forall f_a \in F \tag{18}$$

$$\sum_{f_a \in F} \beta^C_{ka} \geq 1 \qquad \forall t_k \in T \tag{19}$$

Condition (18) ensures that each family meets its coverage requirement, while condition (19) states that each target must be covered at least once. For instance, in the example in Figure 2, condition (18) is respected for family f_1 if $\tau_1 \leq 4$.

Our GA considers in each iteration a population consisting of only feasible chromosomes. In the case of MFMLP, chromosomes will always be non-redundant as well. The chromosomes are evaluated according to the objective function of [**SP**] for MFMLP and of [**SP2**] for MFMLP-R. That is, in the case of MFMLP, given the vector of dual prices provided by the last Master Problem iteration and sorted by family, the fitness function of a given chromosome is equal to the dot product of its *Head* component and the dual prices vector. For each cover that is found to be attractive at the end of the GA procedure, the *Head* component corresponds to the new column to be included in the restricted columns set of master problem [**P**]. In the case of MFMLP-R, both the *Head* and the *Body* components are used to evaluate the fitness function of a given chromosome, and both components represent the column to be added to [**P2**].

4.2 GA overall structure

In this section we describe the general structure of the GA, whose pseudocode is given in Algorithm 1.

The procedure takes as input the wireless sensor network (S, T, F) and the vector DP of the dual prices provided by the Master Problem. The first step is the generation of an initial population Pop and the identification of the chromosome with the best fitness ($BestFit$). This chromosome is the *incumbent* solution and it will be used for comparisons during the evolution process. The population consists of a predefined number ($Size_{Pop}$) of feasible covers and it is initialized by the procedure described in Section 4.7.

The while loop (line 4) iterates until either $MaxIT$ consecutive iterations, without improvements in the incumbent solution fitness $BestFit$, are carried out

Algorithm 1: Genetic Algorithm for MLMFP or MLMFP-R

Input: $DP, (S, T, F)$;
Output: a subset of chromosomes(i.e. columns) for **[P]** or **[P2]**;

```
1  Pop ← InitPopulation();
2  BestFit ← bestFitness(Pop, DP);
3  criteria ← setCriterion(MaxIT, MaxDB);
4  while check(criteria) do
5      (P1, P2) ← tournament(Pop);
6      C ← Crossover(P1, P2);
7      C ← Mutation(C);
8      C ← fixingOperator(C);
9      C ← redundancyOperator(C);
10     if C ∉ Pop then
11         Insert(C, Pop);
12         if fitness(C) ≥ BestFit then
13             update(criteria);
14         else
15             BestFit ← fitness(C);
16     else
17         update(criteria);
18 Chromos ← chromosomes with fitness ≤ fitThreshold;
19 return Chromos;
```

or *MaxDB* consecutive duplicates chromosome are generated. A chromosome is a *duplicate* if it is already present in the population. Forbidding the presence of duplicates in the population makes it possible to avoid looping over a solution space that has almost been exhausted.

Each iteration includes a tournament for the selection of two parent chromosomes (see Section 4.3), a crossover function (Section 4.4) and a mutation function (Section 4.5). Furthermore, two operators, called *fixing* and *redundancy*, are applied. The first one is used to check and eventually restore feasibility for the newly generated chromosome. The redundancy operator always removes eventual redundancy for the MFMLP while, for the MFMLP-R, it may return a redundant cover if it is considered useful to improve the objective function. The two operators are described in Section 4.6.

Each newly generated child chromosome is inserted in the current population *Pop* if and only if it does not already belong to it. If this is the case, it takes the place of one of the $|Pop|/2$ individuals with the worst fitness function value,

selected uniformly at random.

Finally, the chromosomes in the final population whose fitness function is better than a predefined threshold value $fitThreshold$ are returned to the master problem.

4.3 Tournament selection

The selection of the parents is implemented by means of a random binary tournament. In particular, given the current population *Pop*, two individuals are selected at random, and then the one with the best fitness function is chosen as first parent. The process is iterated to select the second parent, making sure that both the chromosomes chosen for the second tournament differ from the first chosen parent.

4.4 Crossover

The crossover function represents the process of coupling between two selected parents. Recall from the chromosome description in Section 4.1 that each family-related segment α_{f_a} in the *Head* component is strongly linked to a specific segment β_{f_a} in the *Body* section, since the former represents the selected sensors for a given family, and the latter the related covered targets. By definition, a feasible chromosome ensures that each couple $(\alpha_{f_a}, \beta_{f_a})$ satisfies the related constraint (18). Therefore, in our genetic algorithm we consider such couples to be genes of the chromosome, which will be used as building blocks for the child chromosome during the crossover. A graphical representation of the gene structure is given in Figure 3.

Given the chromosome structure and the gene definition, the crossover function randomly selects each gene one at a time between the two input parents. In particular, let C_1 and C_2 be the two parents, and $Child$ the child chromosome to be built. For each family $f_a \in F$, the gene $(\alpha_{f_a}^{Child}, \beta_{f_a}^{Child})$ will be equal to $(\alpha_{f_a}^{C_1}, \beta_{f_a}^{C_1})$ with probability 0.5, otherwise it will be equal to $(\alpha_{f_a}^{C_2}, \beta_{f_a}^{C_2})$. The crossover is illustrated in Figure 4.

It is straightforward to observe that this construction ensures that the coverage requirements (18) are satisfied for each family, since by definition both parents are feasible, and therefore each of their genes satisfies the requirement as well.

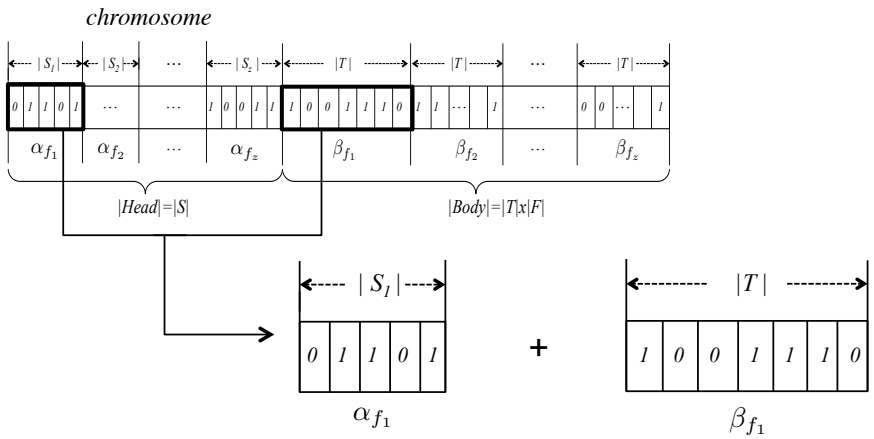


Figure 3: *Gene structure*

4.5 Mutation

The mutation randomly alters the child chromosome produced by the crossover function in order to create diversity during the exploration of the solution space. The mutation operates in two steps. In the first step, it randomly selects a single family f_a, $1 \leq a \leq z$. Successively, it randomly selects a sensor $s_i \in S_a$, and switches its position in the α_{f_a} component either from 1 to 0, or from 0 to 1. The mutation also involves a change in the β_{f_a} segment if deactivating or activating the selected sensor leads to a different target coverage for family f_a.

4.6 Fixing and redundancy operators

As noted in Section 4.4, at the end of the crossover phase, the coverage requirement is respected for all families (that is, condition (18)). However, due to the perturbation brought by the mutation, this may no longer be the case for one of them. Moreover, there is no guarantee that condition (19) is respected; that is, there could be targets that are not covered by any sensor. Therefore, after crossover and mutation, the fixing operator (whose pseudocode is given in Algorithm 2) is applied on each generated chromosome to ensure feasibility. The operator works in two phases, one for each of the two conditions.

If the first condition is not met for some family f_a (Algorithm 2, lines 1-6), let $\hat{S}_a \subseteq S_a$ be the set of sensors of family f_a that are currently not activated in *Child*. Futhermore, let $\hat{T}_a$ be the targets which are currently not covered by the family in the chromosome. The procedure iteratively selects sensors in $\hat{S}_a$

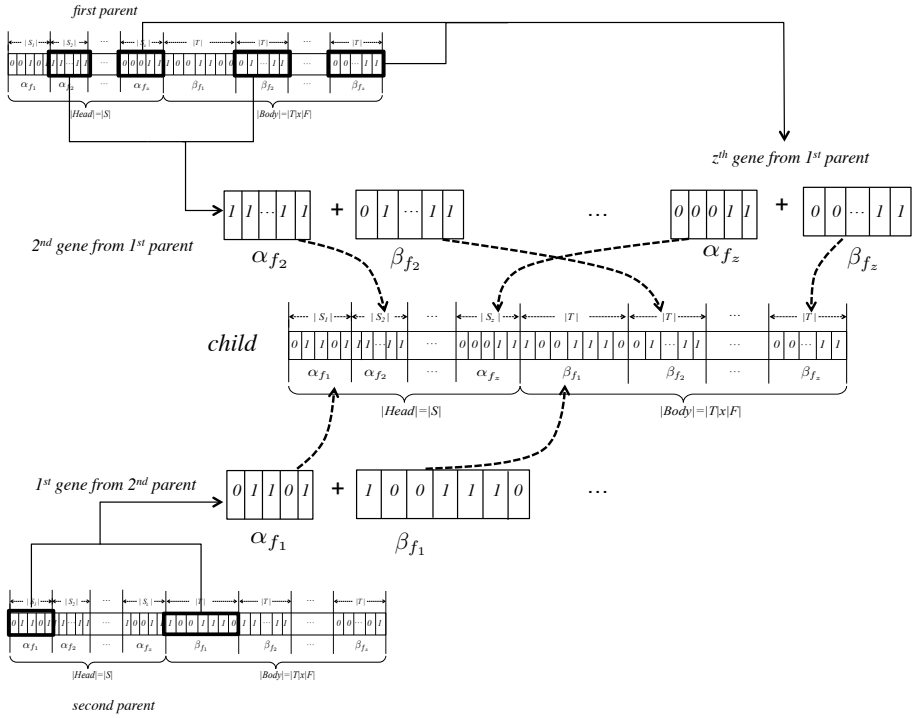


Figure 4: *Crossover*

which cover some elements of $\hat{T}_a$, until τ_a targets are covered by the family. The gene $(\alpha_{f_a}^{Child}, \beta_{f_a}^{Child})$ is updated accordingly at each step.

Regarding the second feasibility condition (Algorithm 2, lines 7-12), the algorithm puts all the globally uncovered targets in $\hat{T}$, if they exist. Then, the procedure iteratively selects a target $t \in \hat{T}$ and a sensor $s \in S$ that can cover t. The gene of $Child$ related to the family of s is updated to include the new sensor, and t is removed from $\hat{T}$, along with any previously uncovered target which is covered by s. The procedure iterates until $\hat{T}$ is empty.

After the application of the fixing operator, the $Child$ chromosome may be redundant. Redundancy is taken into account by two procedures, namely $redundancy_1$ and $redundancy_2$. The $redundancy_1$ procedure first builds a list S_{red} of redundant sensors and then it randomly selects a sensor belonging to it to be switched off. The list of redundant sensors S_{red} is then recomputed, and the process is repeated until S_{red} is equal to the empty set.

The $redundancy_2$ operator builds the S_{red} list of redundant sensors, as well.

Algorithm 2: Fixing Operator

Input: $Child$ chromosome;
Output: fixed $Child$ chromosome;

for $a \leftarrow 1$ **to** z **do**
  while $\sum_{t_k \in T} \beta_{ka}^{Child} < \tau_a$ **do**
    $\hat{S}_a \leftarrow inactiveSensors(Child, f_a)$;
    $\hat{T}_a \leftarrow uncoveredTargets(Child, f_a)$;
    $s \leftarrow randomSelect(\hat{S}_a, \hat{T}_a)$;
    $update(Child, s)$;
$\hat{T} \leftarrow uncoveredTargets(Child)$;
while $|\hat{T}| > 0$ **do**
  $t \leftarrow randomSelect(\hat{T})$;
  $s \leftarrow randomSelect(S, t)$;
  $update(\hat{T})$;
  $update(Child, s)$;
return $Child$;

Then, the procedure checks whether removing a random sensor $s_{r1} \in S_{red}$ from $Child$ would lead to a worse fitness function. If that is the case, a second element $s_{r2} \in S_{red} \setminus \{s_{r1}\}$ is randomly selected and checked for removal. Iteratively, the elements in S_{red} are visited according to a random order; as soon as one can be removed is found, $Child$ is updated and S_{red} is recomputed. The procedure ends when either S_{red} is empty or all its elements have been visited. The $redundancy_2$ operator results to be more computationally intensive than $redundancy_1$.

Redundancy$_1$ operator is used when solving MFMLP. On the other hand, when solving MFMLP-R, $redundancy_1$ is used with a given probability $prob_{red}$, and redundancy$_2$ with probability $1 - prob_{red}$.

4.7 Building the initial population

The procedure for initializing GA builds the initial population Pop, composed of $Size_{Pop}$ random feasible individuals. The population is built iteratively. For each individual, the procedure applies the fixing and redundancy operators, as discussed in Section 4.6, with the only difference that they start from an empty chromosome. If a chromosome is equal to a previously generated one, it is discarded and generated again. If the procedure fails to build a new chromosome for $MaxInitDB$ consecutive iterations, it is interrupted, and

$Size_{Pop}$ is set to the current value of $|Pop|$.

5. Computational Results

This section presents the test scenarios and the results obtained by performing our extensive computational phase. The algorithms were coded in C++ and the tests were performed on a computer with an Intel Xeon 2 GHz processor and 8GB of RAM, equipped with the IBM ILOG CPLEX 12.5.1 solver and the Concert Technology Library for the mathematical formulations. Section 5.1 describes our instances and the test scenarios being considered. The values used for the GA parameters as well as a description of the CG initialization are given in Section 5.2. Finally Section 5.3 presents our computational results organized in tables, along with several comments on them.

5.1 Description of instances and test scenarios

The instances were generated by randomly placing targets and sensors on an area of size 500×500. All the instances can be downloaded from the authors' web site[3]. We assumed the sensing range of each sensor to be equal to 150. We considered instances containing a number of target points $|T|$= 30, 60, 90 or 120, and whose sensors are divided in $|F| = 2$, 4 or 6 sensor families.

For each value of $|F|$, we considered 6 different values for the overall number of sensors $|S|$, corresponding to the cases in which each family has on average 50, 100, 150, 200, 300 or 400 sensors, leading to the values reported in Table 1. However, to better model the heterogeneity which may characterize real-world scenarios, sensors were not evenly distributed among the different families, but rather randomly assigned to them, leading to families with different numbers of sensors. Each family is always guaranteed to cover each target with at least one sensor in order to ensure feasibility for each possible coverage request value, as well as strictly positive w_{min} optimal solution values for MLMFP-R, for the whole set of instances.

For each combination of the above mentioned parameters, we generated 5 different instances. The total number of test instances is therefore equal to 360.

Furthermore, for each instance, two different scenarios were considered, related to the possible values of the coverage request parameters. In the *uniform*

[3] http://www.dmi.unisa.it/people/gentili/www/PublicationM.htm

Avg. sensors per family	Overall sensors		
	$\|F\| = 2$	$\|F\| = 4$	$\|F\| = 6$
50	100	200	300
100	200	400	600
150	300	600	900
200	400	800	1200
300	600	1200	1800
400	800	1600	2400

Table 1: Settings of the $|S|$ parameter.

coverage request scenario, each family f_a is required to provide coverage for $\tau_a = \lfloor |T|/|F| \rfloor$ sensors in each feasible cover. In the *variable* coverage request scenario, we assigned either 2 or 3 different coverage request values to the families, with the lowest one being set to 0. In particular, when $|F| = 2$, one of the families has a coverage request equal to $\lfloor \frac{3}{4}|T| \rfloor$, while the coverage request is equal to 0 for the other family. For $|F| = 4$, the coverage request is set to $\lfloor \frac{3}{8}|T| \rfloor$ for a family, $\lfloor \frac{3}{16}|T| \rfloor$ for 2 of them and 0 for the remaing one. Finally, for instances with 6 families, the three coverage request values are $\lfloor \frac{3}{12}|T| \rfloor$, $\lfloor \frac{3}{24}|T| \rfloor$ and 0, and are assigned to 2 families each. Furthermore, for both scenarios we considered the case in which the consumption ratio of the family with index $i \in \{1, \ldots, |F|\}$ is equal to $(1.0) + (0.1)(i - 1)$. All sensors are always assumed to have fully charged batteries at the beginning of the monitoring phase (that is, $charge_{s_i} = 1 \ \forall s_i \in S$). The coverage request and the consumption ratio values being considered for the two scenarios are summarized in Table 2.

By considering the two above mentioned coverage request scenarios for each of the 360 instances, it follows that 720 experiments were run for each of our two proposed approaches.

As discussed in Section 5.3, we also ran some tests for a "pure" CG approach which does not embed the GA after the CG initialization, and therefore relies on the **[SP]** formulation to generate new covers. We performed this comparison on a subset of the generated instances for the MFMLP problem, as explained in Section 5.3.

5.2 Parameter setting and CG initialization

Parameter values were chosen after a preliminary tuning phase. The population size $Size_{Pop}$ was chosen to be equal to $50 + \lceil \sqrt{|S|} \rceil$. The two termination criteria, namely the maximum number of iterations without improvements

uniform coverage requests			
$\|T\|$	$\|F\| = 2$	$\|F\| = 4$	$\|F\| = 6$
30	15;15	7;7;7;7	5;5;5;5;5;5
60	30;30	15;15;15;15	10;10;10;10;10;10
90	45;45	22;22;22;22	15;15;15;15;15;15
120	60;60	30;30;30;30	20;20;20;20;20;20
variable coverage requests			
$\|T\|$	$\|F\| = 2$	$\|F\| = 4$	$\|F\| = 6$
30	22;0	11;5;5;0	7;7;3;3;0;0
60	45;0	22;11;11;0	15;15;7;7;0;0
90	67;0	33;16;16;0	22;22;11;11;0;0
120	90;0	45;22;22;0	30;30;15;15;0;0
consumption ratios			
	$\|F\| = 2$	$\|F\| = 4$	$\|F\| = 6$
	1.0;1.1	1.0;1.1;1.2;1.3	1.0;1.1;1.2;1.3;1.4;1.5

Table 2: Coverage request and consumption ratio values.

$MaxIT$ and the maximum number of consecutive duplicates $MaxDB$ were chosen to be equal to 1500 and 100, respectively. During the initialization phase, the limit on the number of consecutive duplicates $MaxInitDB$ was set to 100 as well. The $prob_{red}$ value regulating the redundancy operator choice was set to 0.8. Finally, the value 0.9 was chosen for the fitness threshold value $fitThreshold$ when solving MFMLP, and 0.5 when solving MFMLP-R.

As introduced in Section 3, in order to initialize the CG algorithm, a subset of feasible covers has to be provided for the first iteration of the master problem. We generated these covers using a first run of the GA. As a heuristic criterion, during this GA execution each sensor is given an equal, strictly positive weight, meaning that when fitness function is evaluated covers with fewer sensors are favored. Furthermore, for this iteration the $fitThreshold$ value is unbounded, meaning that the whole set of $Size_{Pop}$ covers is returned and added to the master problem.

During the computational tests performed on the CG algorithm which does not embed the GA to produce new covers, the same heuristic initialization method is still used to identify the starting subset. Hence, the GA is executed once for each of these tests.

5.3 Tests and Results

Let us analyze the impact of embedding the proposed GA within the CG paradigm by comparing our proposed algorithm (referred to as CG+GA) with a pure Column Generation approach (referred to as CGonly). This last approach only uses the genetic algorithm during the initialization phase, as reported in Section 5.2, and generates each sub-sequent attractive cover by solving the mixed integer formulation **[SP]** to optimality.

The results are reported in Table 3 for the basic (i.e. non regular) version of the problem, similar conclusions can be derived for the regular version of the problem whose results are reported in Table 6 in the Appendix. We performed the comparison on the subset of 60 instances corresponding to the lowest values of the $|S|$ parameter, that is $|S| = 100$ for $|F| = 2$, $|S| = 200$ for $|F| = 4$ and $|S| = 300$ for $|F| = 6$. For each of those instances, computational tests were performed for both the coverage request scenarios. As shown in the tables, the performances of the pure approach tend to degrade quickly as the size of the instances grows, therefore, it cannot be expected to find solutions in reasonable time on the largest ones. The table entries contain average values and standard deviations for the 5 tests corresponding to each choice of $|T|$, $|F|$, $|S|$ and coverage requirement scenario type.

The CG+GA approach consistently outperforms the pure CG approach, and the computational times difference between the two procedures increases with the number of sensors. The minimum average speed-up (column speed-up in Table 3) is equal to 7.35 for $|S| = 100$, 31.75 for $|S| = 200$, and 46.43 for $|S| = 300$. Overall, the CG+GA is up to 112.85 times faster than CGonly and required a maximum computational time of 6.83 seconds on average, on a set of 5 instances which required on average 683.68 seconds when solved by CGonly (which is the maximum value for this procedure as well). CG+GA shows a consistent and robust behavior on all the instances with a coefficient of variation (i.e., the ratio between standard deviation and average value) for the speed-up always less than 50% for the uniform coverage request scenarios and less than 42% for the variable requests scenarios (except for two instances for $|S| = 100$ for both the cases).

The good performance of the CG+GA approach is due to multiple good columns that are returned by GA and added to the master problem. The number

Table 3: **Comparison of our approach (CG+GA) and a pure column generation approach (CGonly) when solving MFMLP.**

Each entry reported in the table refers to the same scenario corresponding to different choices of $|T|$, $|F|$, $|S|$ and coverage requirement. Columns *avg.* and *std. dev.* are average and standard deviation values computed among the five different instances generated for each scenario, respectively. Column *solution* contains the average solution value computed among the five different instances of the scenario. Columns *SP it.* and *time* refer to the number of times the subproblem [SP] was solved to optimality and to the computational time in seconds for both the algorithms, respectively. Column *GA it.* refers to the number of times GA is invoked. The *speed-up* heading refers to the ratio between the computational time of CGonly and that of CG+GA.

$\|F\| = 2$, $\|S\| = 100$, uniform coverage requests													
instance		CGonly				CG+GA						speed-up	
		SP it.		time		GA it.		SP it.		time			
$\|T\|$	solution	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev
30	10.35	104.8	67.84	8.48	9.35	6.2	4.97	1.0	0.00	0.61	0.59	12.65	3.58
60	6.22	35.6	24.06	3.60	2.14	2.8	0.84	1.0	0.00	0.35	0.09	10.44	6.01
90	6.73	29.6	25.46	5.35	4.84	2.4	0.55	1.0	0.00	0.46	0.09	11.15	8.43
120	7.15	44.8	40.57	11.50	10.57	2.8	1.30	1.0	0.00	0.66	0.26	15.56	7.54
$\|F\| = 2$, $\|S\| = 100$, variable coverage requests													
instance		CGonly				CG+GA						speed-up	
		SP it.		time		GA it.		SP it.		time			
$\|T\|$	solution	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev
30	10.44	72.2	16.42	3.62	0.81	3.8	1.30	1.0	0.00	0.38	0.14	9.85	2.10
60	6.22	28.6	16.13	2.52	1.39	2.4	0.55	1.0	0.00	0.36	0.07	7.35	4.62
90	6.73	39.6	28.18	6.48	5.30	2.8	1.10	1.0	0.00	0.52	0.16	11.15	6.96
120	7.15	41.4	27.48	8.61	5.47	3.4	1.14	1.0	0.00	0.83	0.25	9.68	4.09

$\|F\| = 4$, $\|S\| = 200$, uniform coverage requests													
instance		CGonly				CG+GA						speed-up	
		SP it.		time		GA it.		SP it.		time			
$\|T\|$	solution	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev
30	17.25	157.8	122.73	31.00	25.54	6.8	5.26	1.0	0.00	0.99	0.89	31.75	6.66
60	13.02	117.0	42.66	53.78	20.91	4.2	1.30	1.0	0.00	0.92	0.42	61.39	18.72
90	14.39	179.6	58.88	131.32	41.83	6.8	3.03	1.0	0.00	2.07	0.72	65.45	11.85
120	15.12	196.6	47.11	206.82	72.04	5.6	2.07	1.0	0.00	2.28	0.84	94.53	39.23
$\|F\| = 4$, $\|S\| = 200$, variable coverage requests													
instance		CGonly				CG+GA						speed-up	
		SP it.		time		GA it.		SP it.		time			
$\|T\|$	solution	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev
30	17.43	180.2	125.79	32.87	22.69	7.0	4.80	1.0	0.00	0.96	0.66	35.65	6.27
60	12.12	88.8	32.58	41.30	20.17	4.4	1.67	1.0	0.00	0.95	0.28	41.69	11.11
90	14.39	149.0	43.05	103.40	33.64	5.4	2.07	1.0	0.00	1.56	0.55	71.39	28.86
120	15.12	174.0	60.45	164.13	62.48	4.8	1.48	1.0	0.00	1.89	0.50	88.01	35.89

$\|F\| = 6$, $\|S\| = 300$, uniform coverage requests													
instance		CGonly				CG+GA						speed-up	
		SP it.		time		GA it.		SP it.		time			
$\|T\|$	solution	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev
30	22.34	239.4	59.18	91.96	16.03	10.4	4.72	2.2	1.64	2.02	1.26	56.99	26.68
60	19.49	267.2	124.98	239.55	128.92	9.2	3.11	1.4	0.89	2.86	1.64	93.68	47.52
90	17.50	296.2	217.06	273.03	153.98	9.0	7.07	1.6	1.34	4.43	4.46	88.39	37.70
120	19.04	351.8	118.54	683.68	246.88	11.6	5.03	1.2	0.45	6.83	3.89	112.85	30.66
$\|F\| = 6$, $\|S\| = 300$, variable coverage requests													
instance		CGonly				CG+GA						speed-up	
		SP it.		time		GA it.		SP it.		time			
$\|T\|$	solution	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev
30	21.18	216.4	53.34	68.24	18.09	7.8	1.92	1.0	0.00	1.49	0.35	46.43	11.57
60	23.22	354.4	194.68	268.46	148.10	12.2	7.05	1.0	0.00	3.64	2.66	78.85	13.71
90	18.99	278.6	126.06	204.70	87.68	8.2	3.77	1.0	0.00	2.78	1.60	79.32	14.01
120	18.24	263.4	129.20	433.61	173.85	9.6	3.13	1.0	0.00	5.25	2.50	85.70	22.64

of subproblem iterations (column *SP it.*) is much lower for CG+GA with respect to CGonly. Note in particular that, for all instances with $|S| = 100, 200$, it is equal to 1, meaning that for all the related tests it was only needed to certify the solution optimality in the last iteration. The maximum number of subproblem iterations on average performed by the CG+GA approach is equal to 2.2. Conversely, for CGonly the average number of needed subproblem iterations varies between a minimum of 28.6 and a maximum of 354.4.

Let us analyze the performance of our approach on the entire set of instances. We report in the paper tables and figures corresponding to the scenarios with $|F| = 4$, the equivalent tables and figures for $|F| = 2$ and $|F| = 6$ are given in Appendix.

The performances of CG+GA scale well when bigger instances are considered and, overall, all tests could be executed within reasonable computational times for both problems versions. In particular, the computational time increases with the size of the instance, as expected, for both the problems and for both the two different coverage requests scenarios.

This trend is evident from Figure 5 and column *time* of Table 4, where the computational times of our algorithm when solving the basic version of the problem (on the left) and the regular version of the problem (on the right), for the uniform coverage request scenario (on the top) and the variable coverage request scenario (on the bottom), with $|F| = 4$, are shown. The same figures and tables for $|F| = 2$ (Figure 7 and Table 7) and $|F| = 6$ (Figure 8 and Table 8) are given in the Appendix. The average computational time when solving the basic version of the problem with uniform coverage request is equal to 5.75 seconds (varying between 0.35 and 29.83) for $|F| = 2$, 27.95 seconds (varying between 0.92 and 120.88) for $|F| = 4$, and 155.06 seconds (varying between 2.02 and 1040.16) for $|F| = 6$. The average computational time when solving the basic version of the problem with variable coverage request is equal to 8.10 seconds (varying between 0.36 and 29.65) for $|F| = 2$, 23.36 seconds (varying between 0.95 and 99.79) for $|F| = 4$, and 94.12 seconds (varying between 1.49 and 510.03) for $|F| = 6$. The average computational time when solving the regular version of the problem with uniform coverage request is equal to 32.65 seconds (varying between 0.51 and 344.57) for $|F| = 2$, 321.01 seconds (varying between

Table 4: **Results of CG+GA for $|F| = 4$ scenarios when solving the basic version of the problem (MLMFP) and the regular version (MLMFP-R)**.

Each entry is an average of five instances. Column *lifetime* contains the average lifetime (which is the same for both the problems). Column *GA it.* contains the number of times GA is invoked. Column *SP it.* contains the number of times the separation problem is solved to optimality. Column *GA columns* reports the average number of columns generated by GA. Column *time* shows the computational time in seconds.

Uniform coverage requests										
instance			MLMFP				MLMFP-R			
$\|T\|$	$\|S\|$	lifetime	GA it.	SP it.	GA columns	time	GA it.	SP it.	GA columns	time
30	200	17.25	6.8	1.0	368.4	0.99	12.0	1.2	648.0	2.71
60	200	13.02	4.2	1.0	205.8	0.92	6.6	1.0	356.0	1.62
90	200	14.39	6.8	1.0	371.2	2.07	8.0	1.0	439.8	2.94
120	200	15.12	5.6	1.0	295.0	2.28	9.0	1.0	500.6	3.93
30	400	25.85	6.6	1.0	385.2	1.42	12.2	1.0	770.6	4.18
60	400	28.00	8.4	1.0	509.8	2.57	17.0	1.0	1098.8	8.41
90	400	33.24	13.4	1.0	860.4	6.52	24.2	1.0	1565.6	24.93
120	400	28.64	10.2	1.0	637.4	6.03	16.0	1.2	1005.6	12.64
30	600	46.17	11.8	1.0	797.2	4.70	25.2	1.0	1729.6	14.53
60	600	38.42	8.2	1.0	534.6	3.66	20.6	1.0	1438.8	14.74
90	600	40.47	10.2	1.0	682.6	6.40	23.0	1.0	1612.8	20.40
120	600	40.42	12.6	1.0	862.2	11.56	26.0	1.0	1830.6	37.60
30	800	63.49	13.0	1.2	922.0	6.60	39.0	1.0	2956.4	38.65
60	800	59.14	15.0	1.0	1099.6	10.87	39.6	1.0	3002.0	58.11
90	800	55.31	14.4	1.0	1045.2	13.64	33.2	1.0	2484.6	60.36
120	800	55.30	16.0	1.0	1174.4	24.19	37.0	1.0	2780.8	139.75
30	1200	130.22	35.8	1.0	2935.0	59.64	98.4	1.0	8048.2	704.44
60	1200	99.19	36.2	1.0	2970.8	120.88	84.2	4.8	6433.6	1555.31
90	1200	84.25	23.8	1.0	1918.8	46.09	61.2	1.0	5005.6	477.89
120	1200	75.51	18.6	1.0	1485.4	37.36	56.4	1.0	4640.4	480.58
30	1600	149.97	33.4	1.0	2888.2	101.12	94.6	1.2	8225.4	978.93
60	1600	116.75	21.0	1.0	1782.8	49.63	79.0	1.2	6843.2	729.54
90	1600	108.17	25.0	1.2	2123.6	83.00	82.0	1.0	7150.4	1590.86
120	1600	99.55	21.0	1.0	1784.0	68.66	67.0	1.4	5796.6	741.10
Variable coverage requests										
instance			MLMFP				MLMFP-R			
$\|T\|$	$\|S\|$	lifetime	GA it.	SP it.	GA columns	time	GA it.	SP it.	GA columns	time
30	200	17.43	7.0	1.0	378.4	0.96	11.6	1.2	647.0	1.95
60	200	12.12	4.4	1.0	217.4	0.95	7.8	2.4	341.8	2.50
90	200	14.39	5.4	1.0	283.0	1.56	7.2	1.0	391.2	2.07
120	200	15.12	4.8	1.0	245.2	1.89	9.6	1.0	541.8	3.85
30	400	24.69	6.2	1.0	359.4	1.37	12.0	1.2	734.4	3.20
60	400	28.00	7.2	1.0	432.0	2.24	18.0	1.0	1165.2	7.43
90	400	33.24	12.2	1.0	779.6	5.47	22.0	1.0	1436.0	15.72
120	400	28.64	9.8	1.0	609.4	5.39	16.4	1.0	1053.0	10.92
30	600	46.17	9.6	1.0	636.2	3.10	19.0	1.0	1309.0	7.99
60	600	38.42	8.8	1.0	575.8	4.20	21.8	1.0	1531.4	14.75
90	600	40.47	9.2	1.2	591.8	5.99	24.8	1.0	1748.0	23.44
120	600	40.42	11.6	1.0	789.0	9.93	27.0	1.4	1876.0	38.97
30	800	63.49	13.4	1.2	951.0	6.66	38.8	1.2	2927.2	39.19
60	800	59.14	12.0	1.2	844.2	8.07	38.6	1.0	2919.8	49.34
90	800	55.31	13.8	1.0	1001.0	13.39	36.4	1.0	2749.6	60.72
120	800	55.30	15.0	1.2	1078.4	19.00	36.8	1.0	2771.4	107.64
30	1200	130.22	33.2	1.0	2717.8	46.27	96.8	1.0	7991.0	525.88
60	1200	99.19	32.4	1.0	2641.2	99.79	75.2	3.0	5905.8	820.74
90	1200	84.25	21.2	1.0	1698.0	34.33	55.6	1.4	4511.8	337.06
120	1200	75.51	18.2	1.0	1452.6	34.28	53.4	1.0	4380.8	385.51
30	1600	149.97	33.6	1.4	2871.0	85.19	94.6	1.8	8157.8	1185.02
60	1600	116.75	23.2	1.2	1963.0	48.70	75.4	1.0	6584.2	669.72
90	1600	108.17	25.2	1.2	2145.2	75.31	81.8	1.4	7122.2	1381.18
120	1600	99.55	18.0	1.0	1516.6	46.67	63.0	1.0	5504.6	615.62

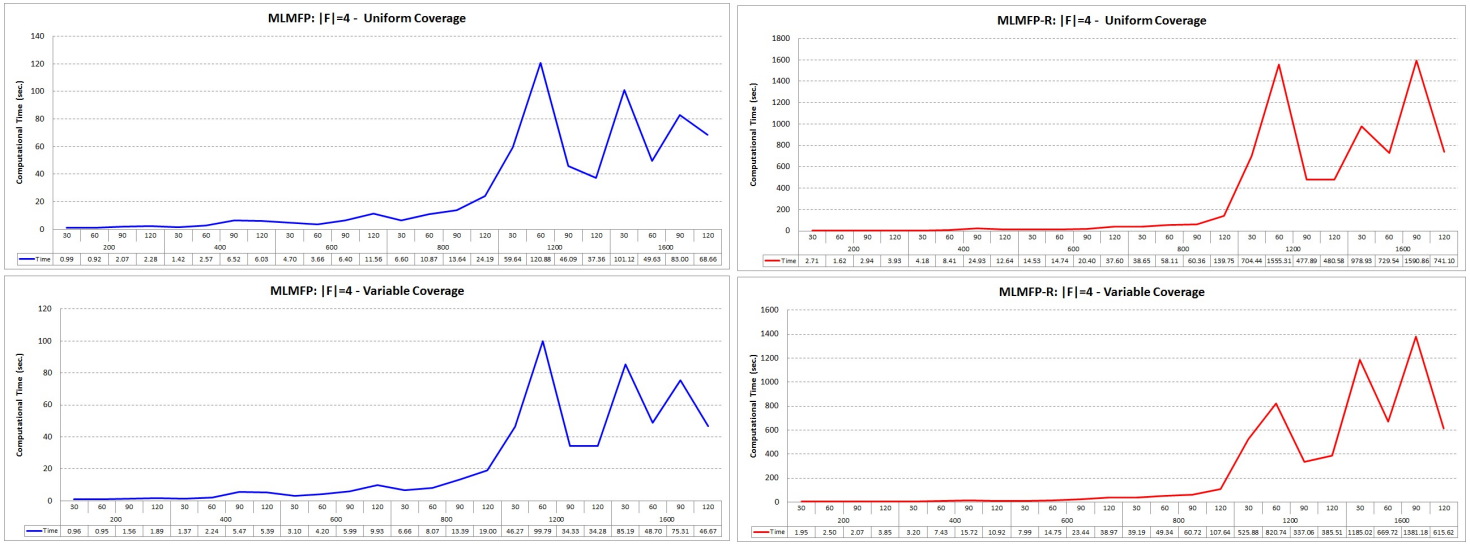


Figure 5: *Computational time of CG+GA when solving the basic version of the problem (on the left) and the regular version of the problem (on the right), for the uniform coverage request scenario (on the top) and the variable coverage request scenario (on the bottom), with* $|F| = 4$.

1.62 and 1590.86) for $|F| = 4$, and 1100.89 seconds (varying between 4.01 and 5219.41) for $|F| = 6$, while it is equal to 37.46 seconds (varying between 0.42 and 142.02) for $|F| = 2$, 262.93 seconds (varying between 1.95 and 1381.18) for $|F| = 4$, and 836.97 seconds (varying between 2.18 and 3563.90) for $|F| = 6$, on scenarios with variable coverage requirement.

The requested average time to solve the regular version of the problem is generally higher than the average time required to solve the basic version of the problem. Indeed, the GA requires more iterations when solving the MLMFP-R as can be observed in our results (column *GA it.* in Table 4 for $|F| = 4$, and Table 7 for $|F| = 2$ and Table 8 for $|F| = 6$ in the Appendix) where the number of GA iterations is higher when solving the regular version of the problem and the total number of returned columns is much greater. Solving the regular version of the problem, with uniform coverage requests, requires on average 129.94% more GA iterations with respect to the solution of the basic version of the problem for $|F| = 2$, 134.72% more iterations for $|F| = 4$, and 101.17% more iterations for $|F| = 6$. Solving the regular version of the problem, with variable coverage requests, requires on average 101.87% more iterations with respect to the solution of the basic version of the problem for $|F| = 2$, 145.68% more iterations for $|F| = 4$, and 117.80% more iterations for $|F| = 6$. We believe that this could be due to the different objective function of the regular problem which forces the GA to explore more deeply the solution space to find the right combination of covers to satisfy the regularity condition. The number of subproblem iterations (column *SP it.* in the tables) keeps being low for both problem variants, witnessing the effectiveness of the GA algorithm. More in particular, for MLMFP and uniform coverage requests, the subproblem is solved on average 1.43, 1.02 and 1.42 times for $|F| = 2$, $|F| = 4$ and $|F| = 6$, respectively, while for variable coverage requests it is solved on average 2.39, 1.07 and 1.10 times. For MLMFP-R, the correspondent numbers of subproblem invocations are 6.78, 1.21 and 2.84 for uniform coverage requests and 6.38, 1.25 and 1.72 for variable coverage requests.

When comparing the quality of the solutions returned by the two problems we can observe that *the maximum lifetime is the same on all the instances both for the original version of the problem and for the regular version.* This is a counterintuitive result since, by enforcing the individuation of a regular solu-

tion, one would expect a deterioration in terms of lifetime for MLMFP-R with respect to MLMFP. Furthermore, as will be discussed later, MLMFP-R is indeed able to improve significantly the value of w_{min} in the returned solution, in particular for the variable scenario. Therefore, we believe that this result is due to the existence of several alternative solutions corresponding to the same optimal lifetime value in the feasible region on each instance. Hence, a regular solution could always be found for the considered set of instances without compromising the maximum lifetime value which can be obtained when regularity is not enforced.

The maximum lifetime increases, as expected, with the size of the instance for both the problem variants, as we can observe in Figure 6 for $|F| = 4$ (Figure 9, 10 in Appendix for $|F| = 2$ and $|F| = 6$, respectively). This was a predictable result since a larger number of sensors allows more covers to exist.

For a given version of the problem, the network lifetime is usually the same for the two coverage requirement scenarios, except for instances with fewer number of sensors. For example, identical solutions values were found in 21 out of 24 cases for $|F| = 4$ except for two cases with $|S| = 200$, and one case with $|S| = 400$. Overall, in these three cases the difference between average solutions is always less than 7.43%. On these datasets, one could expect a fewer number of feasible covers to exist. Therefore, if some of the covers are feasible for a given type of coverage requirement and not for the other, on bigger instances a larger set of alternative covers may be available and help to converge to the same optimal lifetime. This result is significant, since it suggests that when a particular robustness level is needed for a given application in terms of coverage request for a subset of particularly relevant sensor families, it can be expected to be obtained with reasonable trade-offs in terms of solution quality, especially if many sensors are available in the network.

We can also compare the quality of the optimal solutions of the two problems with respect to the level of regularity by comparing the value of the variable w_{min}, which, we recall, is the minimum amount of time, among all the families, for which a target is covered. Refer to Table 5 for $|F| = 4$, and Tables 9 and 10 in the Appendix for $|F| = 2$ and $|F| = 6$, respectively.

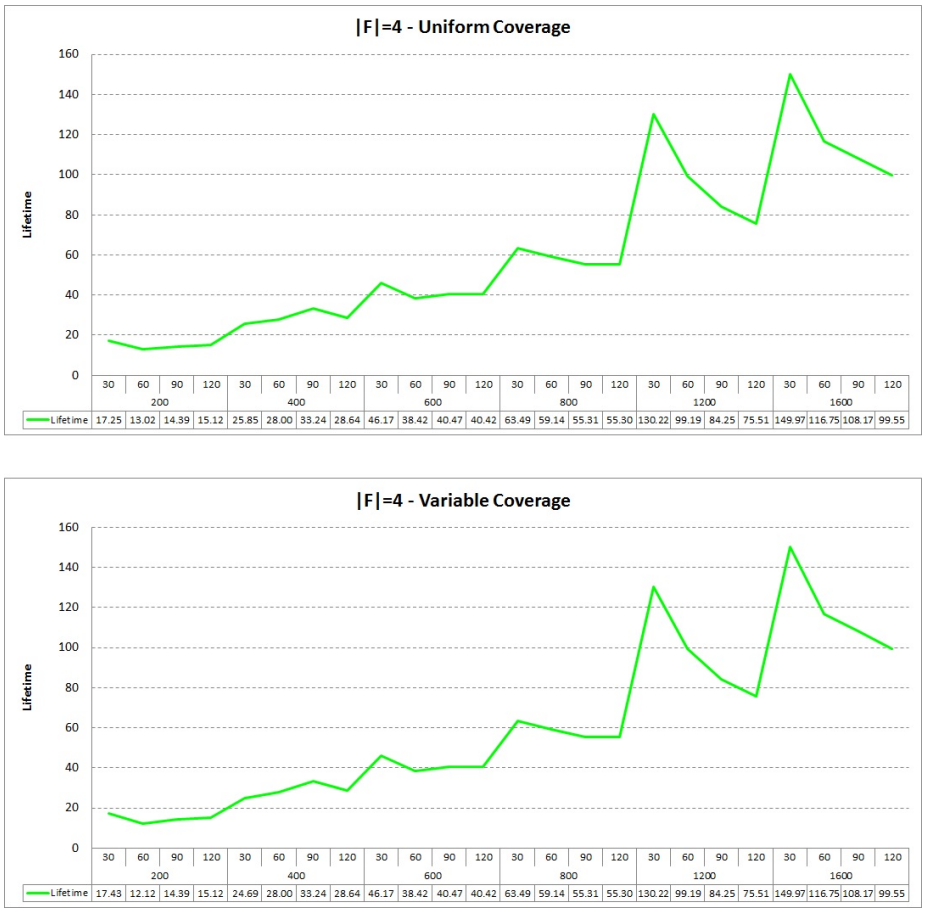


Figure 6: *Lifetime values when solving the two problems for the uniform coverage request scenario (on the top) and the variable coverage request scenario (on the bottom), with* $|F| = 4$.

Solving the regular version of the problem improves the value of w_{min}. This is an expected result, since, when solving the regular version of the problem, we look for solutions such that w_{min} is maximized, while there are no requirements for w_{min} in the basic version of the problem. Hence, alternative optimal solutions with the same lifetime but lower value of w_{min} can be generally selected when solving the basic variant of the problem. In particular, the average percentage difference between the optimum w_{min} obtained when solving MLMFP-R, and the value of w_{min}, obtained when solving MLMFP, with uniform coverage requests is equal to 11.89% for $|F| = 2$, 11.66% for $|F| = 4$, and

Table 5: **Values of w_{min} for $|F| = 4$ scenarios**.

Each entry is an average of five instances. Results are reported for both the problems and for both the coverage requirements. Column w_{min} for MLMFP is evaluated by checking the minimum amount of time for which, among all the families, a target is covered in the optimum solution. Column w_{min} for MLMFP-R is the optimum value of the related variable obtained when solving this problem variant. Column *% Gap* reports the percentage difference between the optimum w_{min} obtained when solving MLMFP-R and the value of w_{min} obtained when solving MLMFP.

	$\|F\|$=4	Uniform coverage requests			Variable coverage requests		
Instance		MLMFP	MLMFP-R	% Gap	MLMFP	MLMFP-R	% Gap
$\|T\|$	$\|S\|$	w_{min}	w_{min}		w_{min}	w_{min}	
30	200	1.74	2.11	21.26	1.41	2.11	49.65
60	200	1.12	1.40	25.00	1.09	1.40	28.44
90	200	0.59	1.07	81.36	0.80	1.07	33.75
120	200	1.26	1.34	6.35	0.81	1.34	65.43
30	400	3.95	3.96	0.25	2.87	3.96	37.98
60	400	3.34	4.19	25.45	3.06	4.19	36.93
90	400	4.21	4.41	4.75	4.27	4.41	3.28
120	400	3.36	3.72	10.71	3.03	3.72	22.77
30	600	6.34	6.34	0.00	5.54	6.34	14.44
60	600	4.78	6.02	25.94	3.57	6.02	68.63
90	600	6.24	6.58	5.45	5.02	6.58	31.08
120	600	6.02	6.20	2.99	5.64	6.20	9.93
30	800	11.04	12.19	10.42	7.61	12.19	60.18
60	800	9.33	10.11	8.36	7.55	10.11	33.91
90	800	8.96	9.44	5.36	8.43	9.44	11.98
120	800	8.74	9.11	4.23	8.10	9.11	12.47
30	1200	24.01	24.35	1.42	21.88	24.35	11.29
60	1200	16.81	17.78	5.77	15.85	17.78	12.18
90	1200	15.45	15.54	0.58	12.90	15.54	20.47
120	1200	13.49	14.91	10.53	13.42	14.91	11.10
30	1600	28.60	29.45	2.97	22.94	29.45	28.38
60	1600	21.50	22.31	3.77	18.22	22.31	22.45
90	1600	18.02	20.27	12.49	15.39	20.27	31.71
120	1600	16.98	17.72	4.36	15.26	17.72	16.12

4.60% for $|F| = 6$. The average percentage difference when solving the problems with variable coverage requests is equal to 86.03% for $|F| = 2$, 28.11% for $|F| = 4$, and 52.43% for $|F| = 6$.

695 *The value of w_{min}, when solving the regular version of the problem are the same for both the uniform and the variable request scenarios.* This is due to the fact that the w_{min} value depends on target-family combinations of the most

unfortunate coverage situations for each instance. In order to investigate this aspect, we checked for each solution provided by MLMFP-R, which target-family combinations corresponded to the w_{min} coverage level. We found that, for any given input instance at least one of such unfortunate combinations was always found to be common to the uniform and variable scenarios.

The value of w_{min}, when solving the basic version of the problem, is higher for the uniform coverage request scenario. This result can be explained by the fact that in the uniform request scenarios, targets are roughly uniformly divided between the families in each cover. This may bring naturally to a fair level of coverage between targets and families, which brings the minimal coverage level to approach the optimum value of w_{min}. On the other hand, in the case of variable coverage requests there are unconstrained families which could bring to more unpredictable coverage levels.

6. Conclusions

In this work we addressed the maximum network lifetime problem on heterogeneous networks composed of different sensor families, as well as a variant of this problem which takes into account some regularity conditions in order to guarantee a fair level of coverage for each sensor family to each target. These problems are of great interest, since nowadays many different sensor types exist and they may be needed to work in a coordinated fashion in many relevant application contexts. To the best of our knowledge, the problems have not been previously studied in the literature. For each problem we developed an exact approach based on column generation, in which the subproblem is solved either heuristically by means of a genetic algorithm or optimally using an appropriate ILP formulation. The proposed genetic algorithm is proven to be able to significantly reduce the computational time of the procedure. Indeed, in our extensive computational phase we were able to optimally solve instances of significant size within reasonable computational times. With respect to future research directions, we intend to further study the use of multiple sensor families, due to its great relevance for real world scenarios. In particular, we intend to explore issues such as connectivity, routing and resistance to failures, which may arise in specific applications.

Acknowledgements

We would like to thank the three anonymous referees whose insightful comments helped improving the quality and readability of the paper. This work was partially supported by the Laboratory of Mathematical Models and Methods for Applications at the Department of Mathematics of the University of Salerno.

References

[1] H. Alemdar, C. Ersoy, Wireless sensor networks for healthcare: a survey, Computer Networks 54 (15) (2010) 2688–2710.

[2] V. L. Boginski, C. W. Commander, P. M. Pardalos, Y. Ye (Eds.), Sensors: theory, algorithms, and applications, Vol. 61 of Springer Optimization and Its Applications, Springer-Verlag, New York, 2011.

[3] S. K. Das, G. Ghidini, A. Navarra, C. M. Pinotti, Localization and scheduling protocols for actor-centric sensor networks, Networks 59 (3) (2012) 299–319.

[4] M. Cardei, M. T. Thai, Y. Li, W. Wu, Energy-efficient target coverage in wireless sensor networks, in: Proceedings of the 24th conference of the IEEE Communications Society, Vol. 3, 2005, pp. 1976–1984.

[5] P. Berman, G. Calinescu, C. Shah, A. Zelikovsky, Power efficient monitoring management in sensor networks, in: Proceedings of the Wireless Communications and Networking Conference, Vol. 4, 2004, pp. 2329 – 2334.

[6] M. Gentili, A. Raiconi, $\alpha-$coverage to extend network lifetime on wireless sensor networks, Optimization Letters 7 (1) (2013) 157–172.

[7] A. Rossi, A. Singh, M. Sevaux, Column generation algorithm for sensor coverage scheduling under bandwidth constraints, Networks 60 (3) (2012) 141–154.

[8] C. Wang, M. T. Thai, Y. Li, F. Wang, W. Wu, Minimum coverage breach and maximum network lifetime in wireless sensor networks, in: Proceedings of the IEEE Global Telecommunications Conference, 2007, pp. 1118–1123.

[9] M. Cardei, J. Wu, M. Lu, Improving network lifetime using sensors with adjustable sensing ranges, International Journal of Sensor Networks 1 (1-2) (2006) 41–49.

[10] R. Cerulli, R. De Donato, A. Raiconi, Exact and heuristic methods to maximize network lifetime in wireless sensor networks with adjustable sensing ranges, European Journal of Operational Research 220 (1) (2012) 58–66.

[11] A. Rossi, A. Singh, M. Sevaux, An exact approach for maximizing the lifetime of sensor networks with adjustable sensing ranges, Computers & Operations Research 39 (12) (2012) 3166–3176.

[12] R. Cerulli, M. Gentili, A. Raiconi, Maximizing lifetime and handling reliability in wireless sensor networks, Networks 64 (4) (2014) 321–338.

[13] A. Alfieri, A. Bianco, P. Brandimarte, C. F. Chiasserini, Maximizing system lifetime in wireless sensor networks, European Journal of Operational Research 181 (1) (2007) 390–402.

[14] A. Raiconi, M. Gentili, Exact and metaheuristic approaches to extend lifetime and maintain connectivity in wireless sensors networks, in: J. Pahl, T. Reiners, S. Voss (Eds.), Network Optimization, Vol. 6701 of Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, 2011, pp. 607–619.

[15] Q. Zhao, M. Gurusamy, Lifetime maximization for connected target coverage in wireless sensor networks, IEEE/ACM Transactions on Networking 16 (6) (2008) 1378–1391.

[16] B. Behdani, Y. S. Yun, J. Cole Smith, Y. Xia, Decomposition algorithms for maximizing the lifetime of wireless sensor networks with mobile sinks, Computers & Operations Research 39 (5) (2012) 1054–1061.

[17] J. Ai, A. A. Abouzeid, Coverage by directional sensors in randomly deployed wireless sensor networks, Journal of Combinatorial Optimization 11 (1) (2006) 21–41.

[18] Y. Cai, W. Lou, M. Li, X.-Y. Li, Energy efficient target-oriented scheduling in directional sensor networks, IEEE Transactions on Computers 58 (9) (2009) 1259–1274.

[19] A. Rossi, A. Singh, M. Sevaux, Lifetime maximization in wireless directional sensor network, European Journal of Operational Research 231 (1) (2013) 229–241.

[20] E. J. Duarte-Melo, M. Liu, Analysis of energy consumption and lifetime of heterogeneous wireless sensor networks, in: Proceedings of the IEEE Global Telecommunications Conference, Vol. 1, 2002, pp. 21–25.

[21] J.-J. Lee, B. Krishnamachari, C.-C. J. Kuo, Impact of heterogeneous deployment on lifetime sensing coverage in sensor networks, in: Proceedings of the Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004, pp. 367–376.

[22] V. P. Mhatre, C. Rosenberg, D. Kofman, R. Mazumdar, N. Shroff, A minimum cost heterogeneous sensor network with a lifetime constraint, IEEE Transactions on Mobile Computing 4 (1) (2005) 4–15.

[23] S. Soro, W. B. Heinzelman, Prolonging the lifetime of wireless sensor networks via unequal clustering, in: Proceedings of the IEEE International Parallel and Distributed Processing Symposium, 2005.

[24] W. Awada, M. Cardei, Energy-efficient data gathering in heterogeneous wireless sensor networks, in: Proceedings of the IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, 2006, pp. 53–60.

[25] H. Liu, P. Wan, C.-W. Yi, X. Jia, S. Makki, N. Pissinou, Maximal lifetime scheduling in sensor surveillance networks, in: Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 4, 2005, pp. 2482–2491.

[26] L. Lazos, R. Poovendran, Stochastic coverage in heterogeneous sensor networks, ACM Transactions on Sensor Networks 2 (3) (2006) 325–358.

[27] H. Zhang, J. C. Hou, Maintaining sensing coverage and connectivity in large sensor networks, Ad Hoc & Sensor Wireless Networks 1 (1-2) (2005) 89–124.

[28] E. Welsh, W. Fish, J. P. Frantz, GNOMES: a testbed for low power heterogeneous wireless sensor networks, in: Proceedings of the International Symposium on Circuits and Systems, Vol. 4, 2003, pp. 836–839.

[29] L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.

Appendix: Additional Figures and Tables

Table 6: **Comparison of our approach (CG+GA) and a pure column generation approach (CGonly) when solving MFMLP-R.**

Each entry reported in the table refers to the same scenario corresponding to different choices of $|T|$, $|F|$, $|S|$ and coverage requirement. Columns *avg.* and *std. dev.* are average and standard deviation values computed among the five different instances generated for each scenario, respectively. Column *solution* contains the average solution value computed among the five different instances of the scenario. Columns *SP it.* and *time* refer to the number of times the subproblem [SP] was solved to optimality and to the computational time in seconds for both the algorithms, respectively. Column *GA it.* refers to the number of times GA is invoked. The *speed-up* heading refers to the ratio between the computational time of CGonly and that of CG+GA.

$\|F\| = 2$, $\|S\| = 100$, uniform coverage requests													
instance		CGonly				CG+GA						speed-up	
		SP it.		time		GA it.		SP it.		time			
$\|T\|$	solution	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev
30	10.35	120.6	63.33	9.39	9.43	12.6	17.01	7.6	14.76	2.21	3.89	9.33	5.53
60	6.22	33.6	22.10	3.31	1.95	3.8	1.64	1.0	0.00	0.51	0.20	6.54	3.07
90	6.73	42.0	35.33	6.82	5.83	3.8	1.64	1.0	0.00	0.69	0.20	9.08	5.15
120	7.15	51.0	36.28	12.32	9.00	4.0	2.35	1.0	0.00	0.95	0.55	12.45	3.69
$\|F\| = 2$, $\|S\| = 100$, variable coverage requests													
instance		CGonly				CG+GA						speed-up	
		SP it.		time		GA it.		SP it.		time			
$\|T\|$	solution	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev
30	10.44	111.2	51.83	5.72	2.92	5.8	1.64	1.0	0.00	0.50	0.15	11.55	4.36
60	6.22	36.8	15.94	3.18	1.38	3.4	1.14	1.0	0.00	0.42	0.10	7.54	3.31
90	6.73	47.6	36.47	6.87	5.50	4.0	2.00	1.0	0.00	0.67	0.27	9.19	4.47
120	7.15	57.0	26.01	11.78	4.89	4.2	1.79	1.0	0.00	0.92	0.39	12.77	2.72

$\|F\| = 4$, $\|S\| = 200$, uniform coverage requests													
instance		CGonly				CG+GA						speed-up	
		SP it.		time		GA it.		SP it.		time			
$\|T\|$	solution	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev
30	17.25	239.4	259.96	44.88	57.63	12.0	10.17	1.2	0.45	2.71	3.64	16.64	4.78
60	13.02	131.4	64.37	58.04	29.55	6.6	3.65	1.0	0.00	1.62	1.04	38.16	12.99
90	14.39	174.6	66.21	126.81	50.65	8.0	2.65	1.0	0.00	2.94	1.40	44.13	3.86
120	15.12	201.2	49.50	197.79	66.78	9.0	3.08	1.0	0.00	3.93	1.73	54.76	25.05
$\|F\| = 4$, $\|S\| = 200$, variable coverage requests													
instance		CGonly				CG+GA						speed-up	
		SP it.		time		GA it.		SP it.		time			
$\|T\|$	solution	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev
30	17.43	225.8	158.32	37.19	28.53	11.6	7.83	1.2	0.45	1.95	1.87	22.28	5.37
60	12.12	109.4	51.03	47.42	26.65	7.8	2.77	2.4	3.13	2.50	2.69	25.89	11.38
90	14.39	148.8	57.70	98.46	42.23	7.2	2.68	1.0	0.00	2.07	0.80	47.69	12
120	15.12	167.4	60.59	147.08	55.89	9.6	5.13	1.0	0.00	3.85	2.70	45.37	23.31

$\|F\| = 6$, $\|S\| = 300$, uniform coverage requests													
instance		CGonly				CG+GA						speed-up	
		SP it.		time		GA it.		SP it.		time			
$\|T\|$	solution	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev
30	22.34	298.0	78.04	99.81	18.78	14.8	4.76	3.0	3.08	4.01	2.30	32.07	18.78
60	19.49	282.6	115.50	244.68	121.84	12.6	1.34	1.0	0.00	4.49	0.33	54.86	27.58
90	17.50	282.2	213.34	277.24	182.21	12.8	7.50	1.8	1.79	7.66	6.89	44.52	13.65
120	19.04	386.4	154.33	708.71	311.89	18.0	7.31	3.2	2.68	17.77	14.60	55.28	24.82
$\|F\| = 6$, $\|S\| = 300$, variable coverage requests													
instance		CGonly				CG+GA						speed-up	
		SP it.		time		GA it.		SP it.		time			
$\|T\|$	solution	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev	avg	std dev
30	21.18	278.8	42.41	74.74	16.02	11.2	2.28	1.0	0.00	2.18	0.65	36.73	12.88
60	23.22	408.2	232.85	326.88	203.10	18.0	8.69	1.6	1.34	8.96	9.29	44.72	12.05
90	18.99	235.2	122.47	185.30	99.77	11.8	4.44	1.0	0.00	4.56	2.63	42.01	14.21
120	18.24	287.4	160.52	452.93	220.10	15.2	6.06	1.6	1.34	10.83	7.40	50.09	22.58

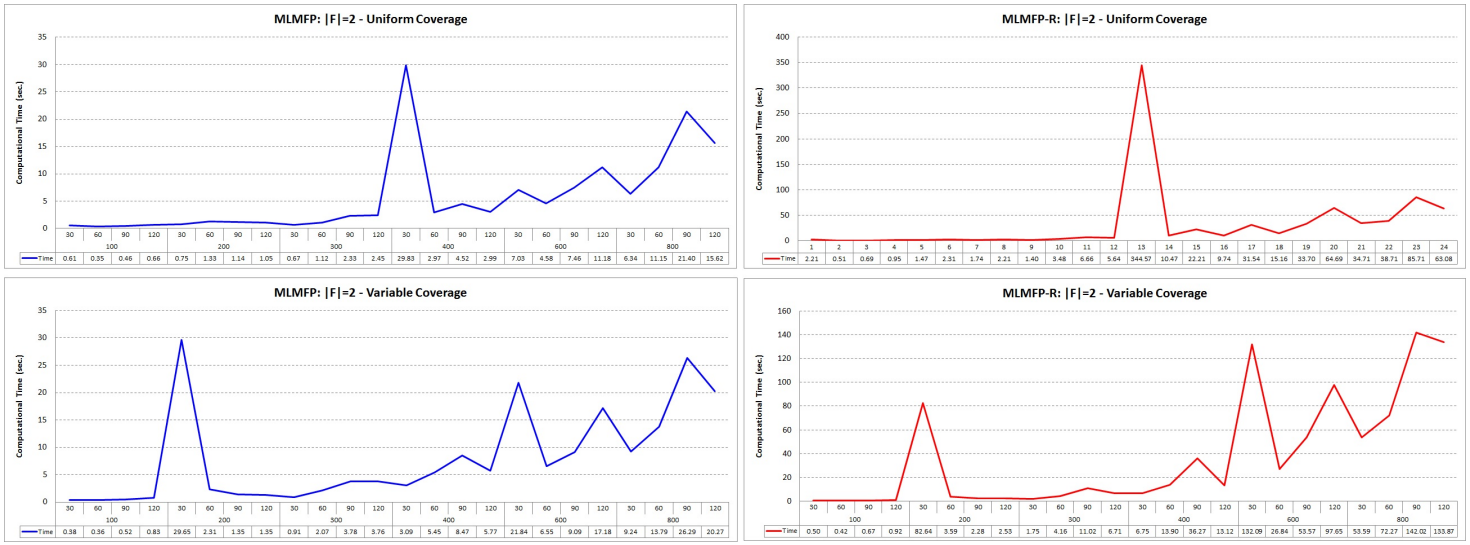


Figure 7: *Computational time of CG+GA when solving the basic version of the problem (on the left) and the regular version of the problem (on the right), for the uniform coverage request scenario (on the top) and the variable coverage request scenario (on the bottom), with* $|F| = 2$.

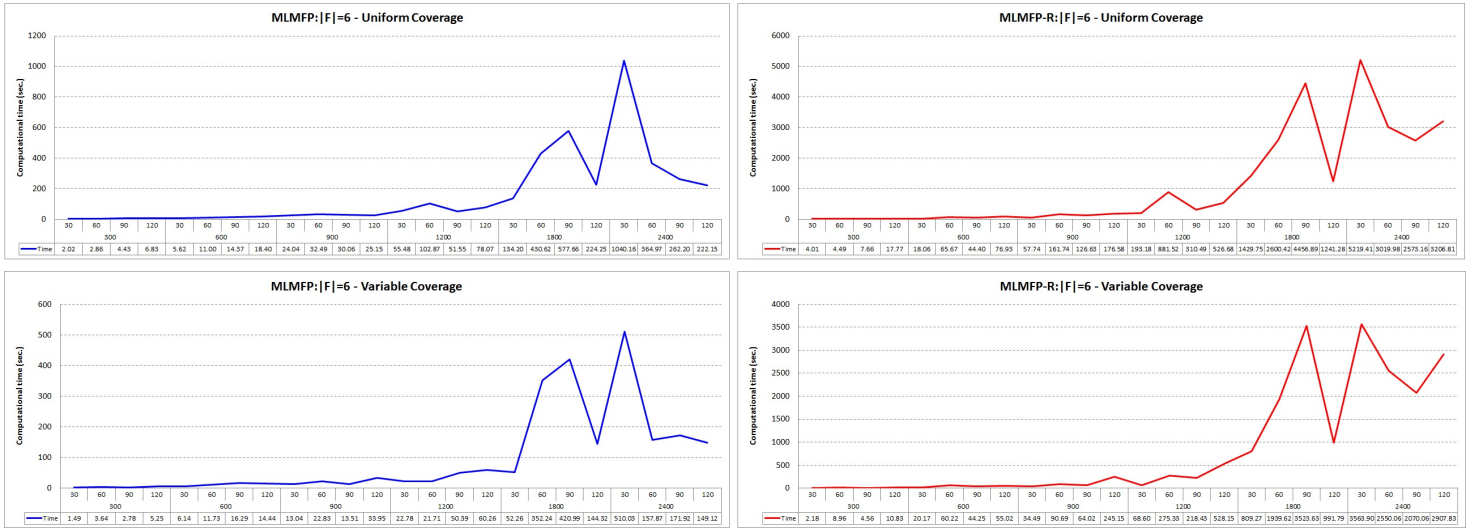


Figure 8: *Computational time of CG+GA when solving the basic version of the problem (on the left) and the regular version of the problem (on the right), for the uniform coverage request scenario (on the top) and the variable coverage request scenario (on the bottom), with* $|F| = 6$.

Table 7: **Results of CG+GA for $|F| = 2$ scenarios when solving the basic version of the problem (MLMFP) and the regular version (MLMFP-R)**.

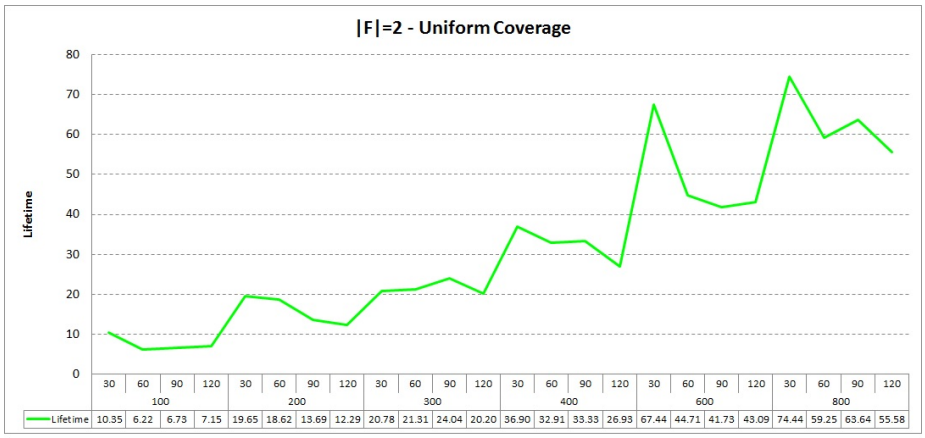
Each entry is an average of five instances. Column *lifetime* contains the average llifetime (which is the same for both the problems). Column *GA it.* contains the number of times GA is invoked. Column *SP it.* contains the number of times the separation problem is solved to optimality. Column *GA columns* reports the average number of columns generated by GA. Column *time* shows the computational time in seconds.

Uniform coverage requests										
instance			MLMFP				MLMFP-R			
$\|T\|$	$\|S\|$	lifetime	GA it.	SP it.	GA columns	time	GA it.	SP it.	GA columns	time
30	100	10.35	6.2	1.0	273.4	0.61	12.6	7.6	279.2	2.21
60	100	6.22	2.8	1.0	104.2	0.35	3.8	1.0	165.2	0.51
90	100	6.73	2.4	1.0	82.0	0.46	3.8	1.0	166.2	0.69
120	100	7.15	2.8	1.0	104.8	0.66	4.0	1.0	176.2	0.95
30	200	19.65	5.6	1.0	294.4	0.75	9.2	1.4	475.6	1.47
60	200	18.62	6.8	1.0	370.6	1.33	10.0	1.0	569.6	2.31
90	200	13.69	4.4	1.0	218.2	1.14	6.0	1.0	319.6	1.74
120	200	12.29	3.6	1.0	165.8	1.05	6.2	1.4	305.8	2.21
30	300	20.78	3.8	1.0	187.8	0.67	7.2	1.0	412.6	1.40
60	300	21.31	4.4	1.0	227.8	1.12	10.8	1.0	657.8	3.48
90	300	24.04	7.2	1.0	417.2	2.33	14.2	1.0	883.8	6.66
120	300	20.20	6.0	1.0	338.2	2.45	10.4	1.0	627.0	5.64
30	400	36.90	27.8	11.0	1030.2	29.83	149.8	131.4	1200.6	344.57
60	400	32.91	8.4	1.0	510.2	2.97	19.2	1.0	1237.8	10.47
90	400	33.33	10.2	1.0	636.8	4.52	26.4	1.0	1756.2	22.21
120	400	26.93	5.8	1.0	333.8	2.99	15.2	1.0	978.8	9.74
30	600	67.44	16.0	1.0	1111.8	7.03	42.2	1.4	3002.8	31.54
60	600	44.71	8.8	1.0	576.6	4.58	23.2	1.0	1633.8	15.16
90	600	41.73	11.0	1.2	727.4	7.46	25.0	1.0	1776.8	33.70
120	600	43.09	12.0	1.0	817.2	11.18	32.6	1.2	2316.4	64.69
30	800	74.44	12.0	1.2	837.8	6.34	39.4	1.2	2966.4	34.71
60	800	59.25	13.6	1.0	983.2	11.15	34.6	1.0	2609.8	38.71
90	800	63.64	16.6	1.0	1224.6	21.40	39.6	1.0	2975.2	85.71
120	800	55.58	9.4	1.0	657.2	15.62	29.8	1.0	2243.8	63.08
Variable coverage requests										
instance			MLMFP				MLMFP-R			
$\|T\|$	$\|S\|$	lifetime	GA it.	SP it.	GA columns	time	GA it.	SP it.	GA columns	time
30	100	10.44	3.8	1.0	165.0	0.38	5.8	1.0	280.6	0.50
60	100	6.22	2.4	1.0	80.4	0.36	3.4	1.0	140.6	0.42
90	100	6.73	2.8	1.0	106.0	0.52	4.0	1.0	178.2	0.67
120	100	7.15	3.4	1.0	142.6	0.83	4.2	1.0	189.8	0.92
30	200	18.74	45.6	34.0	660.4	29.65	128.2	118.6	611.0	82.64
60	200	18.62	10.4	1.0	606.4	2.31	14.4	1.0	854.2	3.59
90	200	13.69	5.0	1.0	258.8	1.35	8.2	1.0	462.0	2.28
120	200	12.29	4.2	1.0	204.4	1.35	7.8	1.0	437.6	2.53
30	300	20.78	4.6	1.0	243.2	0.91	9.8	1.0	592.2	1.75
60	300	21.31	7.4	1.0	428.4	2.07	13.4	1.0	833.0	4.16
90	300	24.04	9.8	1.0	592.8	3.78	20.8	1.0	1326.2	11.02
120	300	20.20	8.0	1.0	473.4	3.76	12.8	1.4	760.8	6.71
30	400	38.35	10.4	1.0	647.8	3.09	19.2	1.0	1246.0	6.75
60	400	32.91	13.4	1.0	858.6	5.45	23.2	1.0	1518.8	13.90
90	400	33.33	16.0	1.0	1044.0	8.47	32.6	1.0	2173.8	36.27
120	400	26.93	9.8	1.0	613.4	5.77	17.6	1.0	1145.2	13.12
30	600	67.44	28.6	1.0	2047.4	21.84	60.8	11.4	3569.4	132.09
60	600	44.71	11.8	1.0	802.8	6.55	31.0	1.0	2215.4	26.84
90	600	41.73	11.6	1.0	786.2	9.09	31.8	1.2	2268.8	53.57
120	600	43.09	15.2	1.0	1056.8	17.18	37.4	1.2	2664.6	97.65
30	800	74.44	15.8	1.0	1155.4	9.24	46.6	1.2	3542.8	53.59
60	800	59.25	17.4	1.4	1255.4	13.79	40.6	1.0	3081.0	72.27
90	800	63.64	21.8	1.0	1633.6	26.29	45.0	1.0	3416.4	142.02
120	800	55.58	14.0	1.0	1021.2	20.27	38.8	1.0	2952.0	133.87

Table 8: **Results of CG+GA for $|F| = 6$ scenarios when solving the basic version of the problem (MLMFP) and the regular version (MLMFP-R)**.

Each entry is an average of five instances. Column *lifetime* contains the average llifetime (which is the same for both the problems). Column *GA it.* contains the number of times GA is invoked. Column *SP it.* contains the number of times the separation problem is solved to optimality. Column *GA columns* reports the average number of columns generated by GA. Column *time* shows the computational time in seconds.

Uniform coverage requests										
instance			MLMFP				MLMFP-R			
$\|T\|$	$\|S\|$	lifetime	GA it.	SP it.	GA columns	time	GA it.	SP it.	GA columns	time
30	300	22.34	10.4	2.2	545.4	2.02	14.8	3.0	774.0	4.01
60	300	19.49	9.2	1.4	518.8	2.86	12.6	1.0	767.0	4.49
90	300	17.50	9.0	1.6	495.6	4.43	12.8	1.8	721.6	7.66
120	300	19.04	11.6	1.2	693.4	6.83	18.0	3.2	969.8	17.77
30	600	43.18	14.8	1.0	1022.8	5.62	26.4	2.0	1781.0	18.06
60	600	46.43	19.0	1.2	1327.0	11.00	38.2	2.2	2611.4	65.67
90	600	38.08	16.4	1.4	1109.8	14.37	27.6	1.4	1904.8	44.40
120	600	35.75	16.8	1.0	1172.2	18.40	28.6	2.0	1928.6	76.93
30	900	66.35	24.4	3.0	1681.6	24.04	35.6	2.8	2539.0	57.74
60	900	62.70	27.2	1.6	2021.4	32.49	48.0	4.6	3346.2	161.74
90	900	55.85	19.4	1.8	1391.8	30.06	37.2	2.6	2691.8	126.63
120	900	56.06	18.4	1.0	1381.6	25.15	39.0	1.0	2973.0	176.58
30	1200	85.99	28.8	1.4	2312.4	55.48	53.8	6.0	3929.2	193.18
60	1200	85.29	30.8	1.0	2521.4	102.87	69.8	3.8	5421.8	881.52
90	1200	74.84	23.2	1.0	1867.0	51.55	46.4	2.6	3605.0	310.49
120	1200	82.27	29.0	1.0	2367.8	78.07	60.0	1.2	4882.4	526.68
30	1800	141.09	34.2	1.0	3072.4	134.20	102.0	1.0	9226.8	1429.75
60	1800	148.89	58.6	3.2	5095.2	430.62	112.0	7.8	9397.6	2600.42
90	1800	129.12	51.4	1.0	4640.8	577.66	109.2	6.8	9085.8	4456.89
120	1800	109.60	34.2	1.2	3047.4	224.25	68.8	1.2	6142.8	1241.28
30	2400	206.46	74.4	1.6	7126.8	1040.16	160.2	5.8	14837.0	5219.41
60	2400	143.24	33.8	1.0	3224.4	364.97	94.2	1.2	9013.6	3019.98
90	2400	144.00	34.8	1.2	3305.8	262.20	90.6	2.0	8576.8	2573.16
120	2400	142.28	29.6	1.0	2814.6	222.15	97.6	1.2	9381.8	3206.81
Variable coverage requests										
instance			MLMFP				MLMFP-R			
$\|T\|$	$\|S\|$	lifetime	GA it.	SP it.	GA columns	time	GA it.	SP it.	GA columns	time
30	300	21.18	7.8	1.0	452.6	1.49	11.2	1.0	673.4	2.18
60	300	23.22	12.2	1.0	755.6	3.64	18.0	1.6	1073.8	8.96
90	300	18.99	8.2	1.0	484.2	2.78	11.8	1.0	713.2	4.56
120	300	18.24	9.6	1.0	577.0	5.25	15.2	1.6	901.8	10.83
30	600	45.86	14.4	1.0	997.0	6.14	28.2	1.0	1975.4	20.17
60	600	47.45	20.4	1.0	1442.4	11.73	39.0	3.2	2587.4	60.22
90	600	37.25	18.4	1.4	1249.8	16.29	28.8	2.2	1938.4	44.25
120	600	35.65	15.2	1.0	1051.6	14.44	28.0	1.2	1965.8	55.02
30	900	68.27	19.4	1.6	1408.8	13.04	33.4	2.2	2425.4	34.49
60	900	67.55	23.0	1.0	1746.2	22.83	42.8	1.0	3252.8	90.69
90	900	56.07	14.0	1.0	1032.2	13.51	33.6	1.0	2559.6	64.02
120	900	55.74	18.8	1.0	1409.8	33.95	41.6	2.8	3031.4	245.15
30	1200	86.45	20.8	1.0	1665.8	22.78	44.2	1.0	3575.8	68.60
60	1200	79.42	18.8	1.0	1499.8	21.71	55.4	3.2	4357.0	275.33
90	1200	76.24	21.6	1.0	1737.2	50.39	42.4	1.4	3395.6	218.43
120	1200	82.27	27.8	1.0	2264.0	60.26	62.4	1.0	5104.2	528.15
30	1800	141.09	27.0	1.2	2376.6	52.26	92.4	1.4	8336.8	809.27
60	1800	149.74	58.4	2.0	5186.2	352.24	108.0	3.6	9365.6	1939.62
90	1800	134.89	52.0	1.0	4714.6	420.99	110.2	3.6	9470.6	3523.63
120	1800	109.60	31.6	1.0	2829.2	144.32	68.8	1.0	6164.2	991.79
30	2400	215.23	59.6	1.2	5736.8	510.03	151.8	1.4	14514.8	3563.90
60	2400	143.24	25.8	1.0	2417.0	157.87	86.8	1.4	8307.0	2550.06
90	2400	144.00	32.0	1.0	3040.2	171.92	85.8	1.4	8194.0	2070.06
120	2400	142.28	26.8	1.0	2537.2	149.12	91.8	1.0	8861.0	2907.83



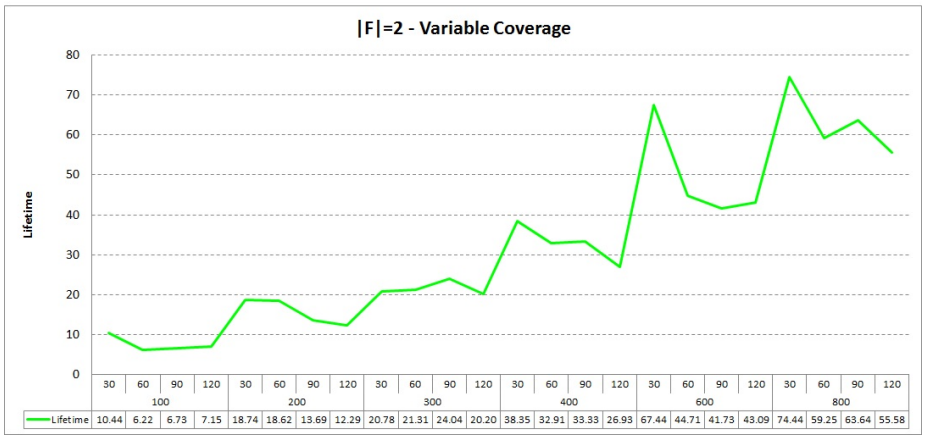


Figure 9: *Lifetime values when solving the two problems for the uniform coverage request scenario (on the top) and the variable coverage request scenario (on the bottom), with* $|F| = 2$.

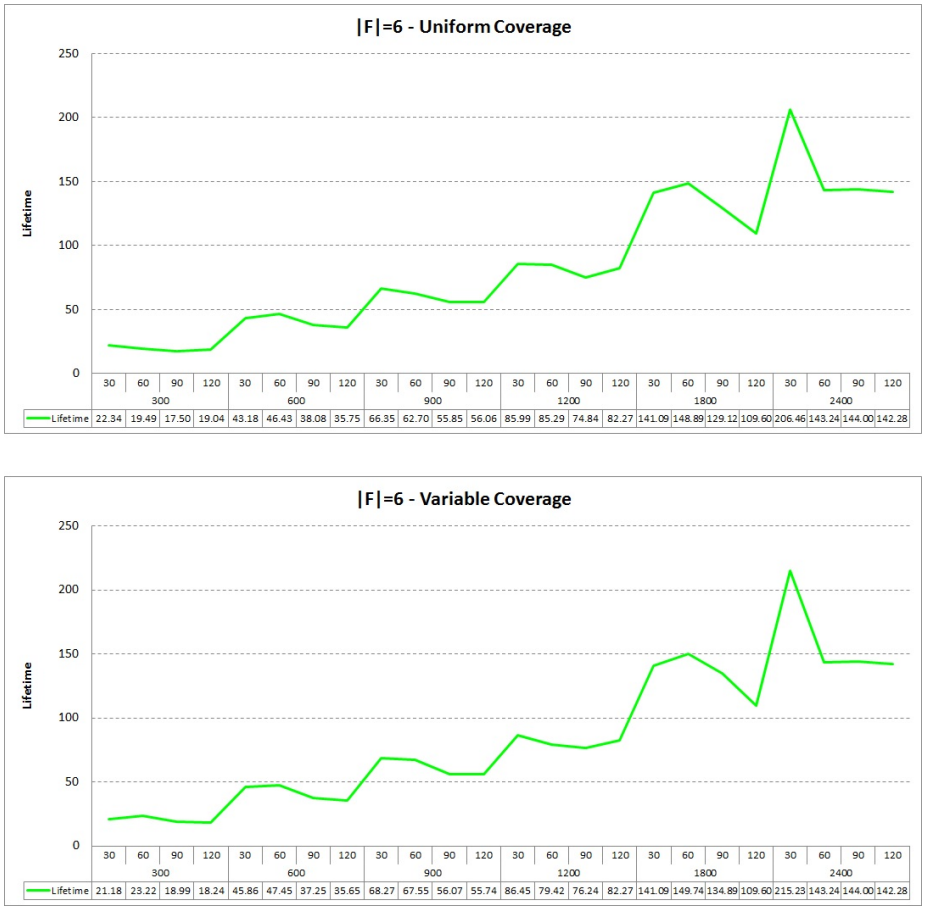


Figure 10: *Lifetime values when solving the two problems for the uniform coverage request scenario (on the top) and the variable coverage request scenario (on the bottom), with* $|F| = 6$.

Table 9: **Values of w_{min} for $|F| = 2$ scenarios**.

Each entry is an average of five instances. Results are reported for both the problems and for both the coverage requirements. Column w_{min} for MLMFP is evaluated by checking the minimum amount of time for which, among all the families, a target is covered in the optimum solution. Column w_{min} for MLMFP-R is the optimum value of the related variable obtained when solving this problem variant. Column *% Gap* reports the percentage difference between the optimum w_{min} obtained when solving MLMFP-R and the value of w_{min} obtained when solving MLMFP.

$\|F\|$=2		Uniform coverage requests			Variable coverage requests		
Instance		MLMFP	MLMFP-R	% Gap	MLMFP	MLMFP-R	% Gap
$\|T\|$	$\|S\|$	w_{min}	w_{min}		w_{min}	w_{min}	
30	100	2.42	2.78	14.88	1.23	2.78	126.02
60	100	1.13	1.27	12.39	0.41	1.27	209.76
90	100	1.47	2.07	40.82	0.65	2.07	218.46
120	100	1.22	1.65	35.25	0.67	1.65	146.27
30	200	6.63	7.18	8.30	4.96	7.18	44.76
60	200	5.20	6.07	16.73	4.31	6.07	40.84
90	200	4.60	5.40	17.39	2.96	5.40	82.43
120	200	3.36	4.25	26.49	2.12	4.25	100.47
30	300	7.23	7.89	9.13	3.31	7.89	138.37
60	300	6.65	7.64	14.89	4.61	7.64	65.73
90	300	7.95	9.25	16.35	4.80	9.25	92.71
120	300	6.45	7.09	9.92	3.99	7.09	77.69
30	400	13.00	13.31	2.38	5.94	13.31	124.07
60	400	11.77	12.22	3.82	9.43	12.22	29.59
90	400	13.65	14.45	5.86	9.06	14.45	59.49
120	400	9.39	10.33	10.01	6.42	10.33	60.90
30	600	28.74	29.25	1.77	20.32	29.25	43.95
60	600	18.26	19.22	5.26	11.02	19.22	74.41
90	600	16.41	17.55	6.95	10.19	17.55	72.23
120	600	17.95	20.04	11.64	13.20	20.04	51.82
30	800	31.42	32.87	4.61	19.57	32.87	67.96
60	800	24.04	25.02	4.08	15.78	25.02	58.56
90	800	22.75	23.58	3.65	18.96	23.58	24.37
120	800	23.41	24.09	2.90	15.65	24.09	53.93

Table 10: **Values of w_{min} for $|F| = 6$ scenarios**.

Each entry is an average of five instances. Results are reported for both the problems and for both the coverage requirements. Column w_{min} for MLMFP is evaluated by checking the minimum amount of time for which, among all the families, a target is covered in the optimum solution. Column w_{min} for MLMFP-R is the optimum value of the related variable obtained when solving this problem variant. Column *% Gap* reports the percentage difference between the optimum w_{min} obtained when solving MLMFP-R and the value of w_{min} obtained when solving MLMFP.

$\|F\|$=6		Uniform coverage requests			Variable coverage requests		
Instance		MLMFP	MLMFP-R	% Gap	MLMFP	MLMFP-R	% Gap
$\|T\|$	$\|S\|$	w_{min}	w_{min}		w_{min}	w_{min}	
30	300	1.21	1.48	22.31	0.36	1.48	311.11
60	300	0.92	1.15	25.00	0.71	1.15	61.97
90	300	0.70	0.78	11.43	0.55	0.78	41.82
120	300	1.09	1.09	0.00	0.38	1.09	186.84
30	600	3.38	3.50	3.55	2.57	3.50	36.19
60	600	4.17	4.36	4.56	3.38	4.36	28.99
90	600	3.14	3.31	5.41	2.34	3.31	41.45
120	600	2.96	2.96	0.00	1.73	2.96	71.10
30	900	5.21	5.21	0.00	4.39	5.21	18.68
60	900	5.48	5.63	2.74	4.29	5.63	31.24
90	900	5.20	5.21	0.19	3.42	5.21	52.34
120	900	5.36	5.36	0.00	4.27	5.36	25.53
30	1200	7.22	7.63	5.68	5.94	7.63	28.45
60	1200	8.85	9.05	2.26	6.34	9.05	42.74
90	1200	6.40	6.65	3.91	5.82	6.65	14.26
120	1200	7.60	7.88	3.68	7.22	7.88	9.14
30	1800	17.55	18.27	4.10	11.37	18.27	60.69
60	1800	14.91	14.99	0.54	13.84	14.99	8.31
90	1800	14.15	14.17	0.14	12.59	14.17	12.55
120	1800	10.33	10.33	0.00	9.02	10.33	14.52
30	2400	22.94	23.53	2.57	16.80	23.53	40.06
60	2400	16.31	16.85	3.31	10.91	16.85	54.45
90	2400	15.01	15.09	0.53	11.89	15.09	26.91
120	2400	15.88	17.24	8.56	12.40	17.24	39.03