University of Salerno
DEPARTMENT OF MATHEMATICS

Exact and heuristic approaches for the maximum lifetime problem in sensor networks with coverage and connectivity constraints

Francesco Carrabs
Department of Mathematics, University of Salerno. E-mail: fcarrabs@unisa.it

Raffaele Cerulli
Department of Mathematics, University of Salerno. E-mail: raffaele@unisa.it

Ciriaco D'Ambrosio
Department of Computer Science, University of Salerno. E-mail: cdambrosio@unisa.it

Andrea Raiconi
Department of Mathematics, University of Salerno. E-mail: araiconi@unisa.it

Technical Report N° 50811
2/9/2015

Noname manuscript No.
(will be inserted by the editor)

Exact and heuristic approaches for the maximum lifetime problem in sensor networks with coverage and connectivity constraints

Francesco Carrabs · Raffaele Cerulli · Ciriaco D'Ambrosio · Andrea Raiconi

Received: date / Accepted: date

Abstract The aim of the Connected Maximum Lifetime Problem is to define a schedule for the activation intervals of the sensors deployed inside a region of interest, such that at all times the activated sensors can monitor a set of interesting target locations and route the collected information to a central base station, while maximizing the total amount of time over which the sensor network can be operational. Complete or partial coverage of the targets are taken into account. To optimally solve the problem, we propose a column generation approach which makes use of an appropriately designed genetic algorithm to overcome the difficulty of solving the subproblem to optimality in each iteration. Moreover, we also devise a heuristic by stopping the column generation procedure as soon as the columns found by the genetic algorithm do not improve the incumbent solution. Comparisons with previous approaches proposed in the literature show our algorithms to be highly competitive, both in terms of solution quality and computational time.

Keywords Maximum Lifetime · Wireless Sensor Network · Column Generation · Genetic Algorithm · Steiner Tree · Partial Coverage

1 Introduction

Wireless Sensor Networks (WSNs) have been applied to several different real-world contexts in the last years. Indeed, technology advancements in fields such as

Francesco Carrabs
Department of Mathematics, University of Salerno.
E-mail: fcarrabs@unisa.it

Raffaele Cerulli
Department of Mathematics, University of Salerno.
E-mail: raffaele@unisa.it

Ciriaco D'Ambrosio (✉)
Department of Mathematics, University of Salerno.
E-mail: cdambrosio@unisa.it

Andrea Raiconi
Department of Mathematics, University of Salerno.
E-mail: araiconi@unisa.it

micro-electro-mechanical systems (MEMS) and wireless communications allowed them do be adaptable to diverse scenarios, including environmental monitoring, healthcare applications, and recent trends such as Internet of Things among others (refer for instance to [2], [4], [25]). Regardless of the considered application, a WSN is usually made of a large amount of devices, called sensors, employed to perform together a monitoring activity. The portion of the space under observation that can be monitored by a given sensor is defined as its sensing range.

A major issue in WSNs is related to the limited amount of activation time that is typically guaranteed by batteries to individual sensing devices. Optimizing the energy consumption of a WSN by appropriately coordinating the use of the sensors that compose it has therefore become an important research field in the last years. In particular, a problem that has been widely studied is related to prolonging for as much as possible the amount of time over which a WSN can monitor a set of interesting target locations located within a geographical area. The problem is usually known as Maximum Network Lifetime Problem (MLP), and several variants of it have been proposed as well, in order to model and take into account characteristics deriving from different real-world applications.

Usually, the solution approaches proposed in the literature for MLP and its variants focus on individuating multiple, not necessarily disjoint sets of sensors (*covers*) which are individually able to monitor the target points. An appropriate activation time has also to be chosen for each cover. Then, the covers can be activated one by one, that is, its sensors can be kept in active state while all the others are turned off, and the network lifetime is given by the sum of all the activation times. It follows that in order to achieve a feasible solution, the sum of the activation times of the covers containing any given sensor has to be bounded by its battery duration. As proven in [8], considering non-disjoint covers can indeed allow to achieve a higher network lifetime. The authors also proved MLP to be NP-Complete, and present an approximation algorithm.

In the last few years, solution approaches based on column generation have been proposed for MLP and variants. These approaches decompose the problem in two parts, namely a subproblem aimed at identifying useful covers and a master problem which assigns activation times to them. Such an approach has been proposed for the classic version of the problem in [17]. Since the subproblem is NP-Hard, the author proposes both an exact ILP formulation and a simple constructive heuristic to solve it, leading to an exact and a heuristic algorithm, respectively. A mixed exact approach combining the two subproblem resolution methods, which makes use of the ILP formulation whenever the heuristic fails, is also presented. Proposed variants of the problem include cases in which only a percentage of targets has to be covered at all times ([11], [20], [28]), heterogeneous networks ([5], [10]), sensors with adjustable sensing ranges ([9], [14], [15], [18], [26]) or with angular, orientable sensing ranges, such as video cameras ([1], [6], [27]), among others.

A significant amount of research has been also spent on WSN problems that consider connectivity issues ([3], [7], [12], [13], [21], [24], [29]). These works take into account sensor-to-sensor communication, in order to gather the collected information and transmit it to a data collecting and processing facility (usually referred to as base station or sink) through single or multi-hop communication. Therefore an additional range (the communication range) is considered for each

sensor, defining which other sensors are close enough to communicate directly with it.

In particular, in [24] the authors propose the Connected Maximum Network Lifetime Problem (CMLP). Consider a communication link existing between each couple of sensors (or a sensor and the base station) if they are within each other's communication range. In CMLP, in addition to the covering request, a path of communication links involving active sensors must exist between each sensor of the cover and the base station. The authors propose two heuristics for its resolution. The first one is a greedy constructive algorithm, while the second one is a GRASP metaheuristic that iteratively uses a randomized version of the greedy approach to produce a different starting solution, which is then improved through a local search step. The GRASP algorithm is also used by the authors to speed up the convergence of an exact column generation approach; this objective is fulfilled by using the set of covers corresponding to the best solution to initialize the master problem.

In [13], the authors extend the problem to consider the case in which only a subset of the targets may require coverage at all times (α-CMLP), enabling to decide trade-offs between achievable network lifetime and required quality of service. Conceptually similarly to the algorithms presented in [17] for the classical MLP, the authors propose two metaheuristics to solve the column generation subproblem (a GRASP and a VNS) and use them to develop three heuristics (named CG-GRASP, CG-VNS and CG-MULTI) and an exact approach (CG-EXACT). While CG-GRASP and CG-VNS use the related metaheuristic to solve the subproblem, CG-MULTI combines both of them, invoking in each iteration GRASP first, and VNS then if GRASP fails. Finally, the CG-EXACT algorithm provides exact solutions by solving an exact ILP formulation to optimality whenever both heuristics fail. The authors proved experimentally that their algorithms perform better than the ones proposed in [24].

In this work we also focus on the α-CMLP problem, presenting a heuristic and an exact approach based on column generation and called HCG and ECG, respectively. In our algorithms, new ideas for the resolution of the subproblem are proposed. In more detail, in order to solve the subproblem heuristically, we propose a highly efficient, appropriately designed genetic algorithm which embeds a Steiner Tree heuristic to satisfy the connectivity requirement. Regarding the exact subproblem resolution, we propose a ILP formulation that reduces the number of required integer variables with respect to the model proposed in [13]; furthermore, we also propose a modification to the exact scheme that interrupts the ILP resolution as soon as a feasible profitable cover is found. These new ideas lead to algorithms that are proven experimentally to outperform the ones proposed in [13].

The rest of the work has the following structure. A formal definition of the problem is provided in Section 2. The previously introduced column generation schemes for CMLP and α-CMLP, as well as our proposed subproblem reformulation, are described in Section 3. Our genetic algorithm and its integration within the column generation framework are presented in Section 4. Our computational results are described in Section 5, followed by conclusions and future research perspectives in Section 6.

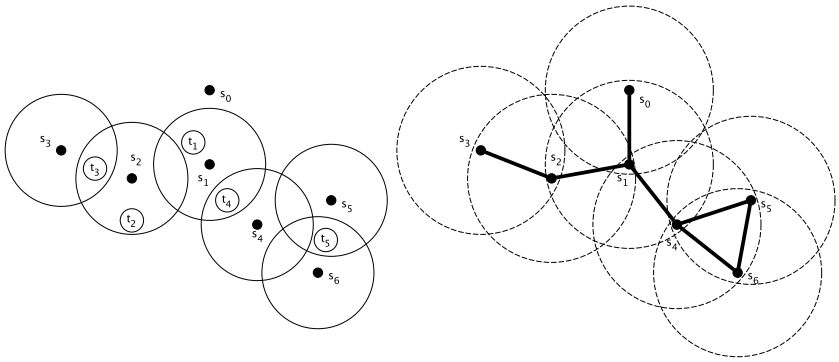


Fig. 1 A simple WSN and its connectivity graph

2 Problems Definition and Mathematical Formulation

Let $T = \{t_1, \ldots, t_n\}$ be the set of the target points of interest, and let $S = \{s_0, s_1, \ldots, s_m\}$ be the set of the sensors that compose the network, as well as the base station s_0. Each sensor is assumed to have a given sensing range as well as a communication range, defining which targets can be monitored by the sensor and which elements of S can directly communicate with it, respectively. Since the base station does not have covering purposes, it is assumed to only have a communication range.

Each sensor belonging to $S \setminus \{s_0\}$ is powered by a battery, which allows it to be in the operational state only for a limited amount of time. No battery concerns are assumed with respect to the base station, since it is supposed to be operational for the whole monitoring process.

For any given target $t_k \in T$ and sensor $s_i \in S \setminus \{s_0\}$, let δ_{ki} be a binary parameter which assumes value 1 if t_k is located within the sensing range of s_i, 0 otherwise. By extension, given a subset $S' \subseteq S \setminus \{s_0\}$, let $\Delta_{kS'}$ be equal to 1 if $\delta_{ki} = 1$ for at least one sensor $s_i \in S'$, and 0 otherwise. If $\delta_{ki} = 1$ or $\Delta_{kS'} = 1$, target t_k is said to be *covered* by s_i or S', respectively. Furthermore, for any two elements s_i, s_j of the S set, let us define a binary parameter ϕ_{ij} which is equal to 1 if they are close enough to be within each other's communication range, and 0 otherwise. Note that by definition $\phi_{ij} = \phi_{ji}$. Now, consider an undirected graph $G = (S, E)$, such that there exists the *communication link* $(s_i, s_j) \in E$ if and only if $\phi_{ij} = 1$; let us call G the *connectivity graph* of the network.

Figure 1 illustrates the concepts introduced so far, by showing a simple WSN and its connectivity graph. The network contains the base station s_0, a set of 6 sensors, namely $\{s_1, s_2, s_3, s_4, s_5, s_6\}$ and 5 targets, $\{t_1, t_2, t_3, t_4, t_5\}$. Sensing and communication ranges are represented with continuous and dashed circles, respectively. We may note that, for instance, sensor s_4 covers t_4 and can communicate with s_1, s_5 and s_6, thus the connectivity graph includes (s_4, s_1), (s_4, s_5) and (s_4, s_6).

Given a value $\alpha \in (0, 1]$, we define $C \subseteq S$ to be a *feasible cover* (or simply a cover) for the network if the following three conditions hold: (i) $s_0 \in C$; (ii) the sensors in C can provide coverage for at least $T_\alpha = \alpha \times n$ targets,

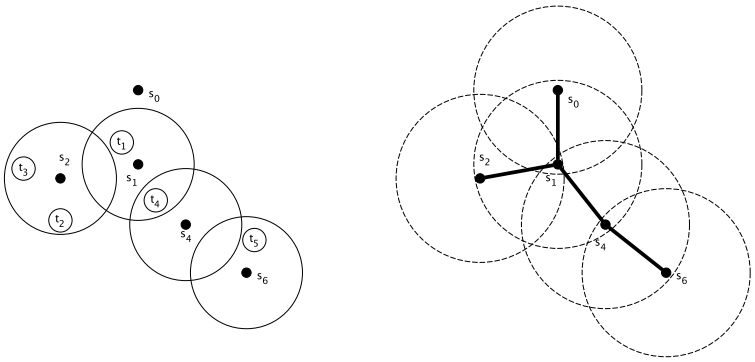


Fig. 2 A feasible cover and its communication tree

that is, $\sum_{t_k \in T} \Delta_{kC} \geq T_\alpha$; (iii) given the connectivity graph G, its subgraph $G' = (C, E(C))$ induced by C is connected. It follows that for each sensor s_i, in a feasible cover C, there exists a path of communication links from s_0 to s_i in G'. These paths from s_0 to all the sensors in C define a *communication tree* rooted at the base station. Each sensor activated in the cover is either used to sense information, as a relay to transmit it to s_0, or for both roles together. Figure 2 shows a feasible cover for the network in Figure 1 and the induced connectivity subgraph, which also corresponds to its only communication tree. Note that since all targets are covered, it is a feasible cover for any considered value of α. It can be observed that if s_4 is removed, the remaining elements would not constitute a feasible cover since Condition (iii) would be violated, meaning that the sensor is needed in the cover for relay purposes. If $T_\alpha \leq 3$, by removing s_2 a different feasible cover would be obtained instead. A cover that does not contain another cover as a proper subset is defined *non-redundant*.

The α-Connected Maximum Lifetime Problem (α-CMLP) consists in finding a collection of pairs (C_p, w_p) where each $C_p \subseteq S$ is a feasible cover and each $w_p \geq 0$ is an amount of time for which C_p is activated, such that each individual sensor is in the active state for an amount of time that does not exceed its battery lifetime, and the sum of the activation times is maximized. With Connected Maximum Lifetime Problem (CMLP) we refer instead to the special case in which full coverage of the set of targets is required (that is, $\alpha = 1$).

Let $C_1, \ldots, C_M$ be a collection of all feasible covers. The following linear model represents α-CMLP:

$$[\mathbf{P}] \max \sum_{p=1}^{M} w_p \tag{1}$$

s.t.

$$\sum_{p=1}^{M} a_{ip} w_p \leq b_i \qquad \forall s_i \in S \setminus \{s_0\} \tag{2}$$

$$w_p \geq 0 \qquad \forall p = 1, \ldots, M \tag{3}$$

In Constraints (2), for each sensor $s_i \in S \backslash \{s_0\}$ and each cover C_p the parameter a_{ip} is equal to 1 if s_i is part of C_p, and 0 otherwise, while b_i is a value representing the battery life of the sensor; for instance, assuming that all sensors have the same characteristics and are equipped with identical batteries, it can be normalized to 1 for all of them. Therefore Constraints (2) enforce the respect of the battery life limitations, while Objective Function (1) maximizes the sum of the activation times and thus the network lifetime.

In practice, solving formulation **[P]** directly is not possible due to the difficulty of explicitly enumerating all feasible covers, whose number grow exponentially with the number of sensors. For this reason, better strategies to focus on useful covers, while implicitly discarding all the others, are required in order to be able to solve the problem. To this end, the authors in [13] and [24] proposed column generation approaches; furthermore, they also developed heuristics aimed at speeding up the convergence of their procedures. As mentioned in the introduction, in our work we propose some modifications to the ILP subproblem formulation and to how it is solved, as well as an effective genetic algorithm that we embed within the column generation framework. These features of our algorithms are presented in Sections 3 and and 4, respectively.

3 Column Generation Approaches for CMLP and α-CMLP

Consider a linear programming formulation with a large number of variables. The column generation (CG) framework operates by dividing the problem in two steps, which are iteratively executed until a proven optimal solution is found. In the first step, the algorithm considers a variant of the original LP formulation (called *restricted master problem*) which is limited to only a subset of its original variables (columns), and solves it. Then, a CG algorithm considers an auxiliary problem (the *separation problem*, or *subproblem*) aimed at building a new *attractive column*, that is, a column that may improve the current (*incumbent*) solution if introduced in the restricted master problem. If such a column can be found, the algorithm iterates by adding it to the set and solving the restricted master problem again; otherwise, the separation problem certifies that the incumbent solution is optimal for the original problem as well. In the case of **[P]**, it follows that each column represents a feasible cover.

In the above given description, an attractive column refers to a currently nonbasic variable with a negative reduced cost. It is straightforward to observe that at least one of such columns is required to exist in order to improve the incumbent solution. The aim of the subproblem is therefore to build a new column (in our case, a feasible cover) while minimizing its reduced cost. If such a reduced cost is positive, it means that all the nonbasic columns which have not been generated so far can be implicitly discarded, and thus the incumbent solution is certified to be optimal; otherwise, the new column is introduced in the restricted master problem and the procedure iterates, as previously described.

Let $G^d = (S, E^d)$ be the directed version of the connectivity graph $G = (S, E)$, where E^d contains both (s_i, s_j) and (s_j, s_i) for each communication link $(s_i, s_j) \in E$. In [24], the authors propose the following single-commodity flow formulation for the CMLP subproblem:

$$\textbf{[SP]} \qquad \min \sum_{s_i \in S \setminus \{s_0\}} \pi_i y_i \tag{4}$$

s.t.

$$\sum_{s_i \in S \setminus \{s_0\}} \delta_{ki} y_i \geq 1 \qquad \forall t_k \in T \tag{5}$$

$$\sum_{(s_0, s_i) \in E^d} f_{0i} = \sum_{s_i \in S} y_i \tag{6}$$

$$\sum_{(s_i, s_j) \in E^d} f_{ij} - \sum_{(s_j, s_i) \in E^d} f_{ji} = y_i \qquad \forall s_i \in S \setminus \{s_0\} \tag{7}$$

$$\sum_{(s_i, s_j) \in E^d} x_{ij} = y_j \qquad \forall s_j \in S \setminus \{s_0\} \tag{8}$$

$$x_{ij} \leq f_{ij} \leq |S| x_{ij} \qquad \forall (s_i, s_j) \in E^d \tag{9}$$

$$f_{ij} \in \mathbb{Z}^+ \cup \{0\} \qquad \forall (s_i, s_j) \in E^d \tag{10}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (s_i, s_j) \in E^d \tag{11}$$

$$y_i \in \{0, 1\} \qquad \forall s_i \in S \setminus \{s_0\} \tag{12}$$

In the objective function (4), parameters $\pi_i \ \forall s_i \in S \setminus \{s_0\}$ correspond to the dual prices associated to Constraints (2) in the incumbent solution, while each binary variable y_i is equal to 1 if the related sensor is chosen to be part of the new cover. Since in the objective function of **[P]** the coefficient of each cover is 1, it follows that the new cover built by **[SP]** will be attractive if its objective function value is less than 1.

Constraints (5)-(12) impose that the chosen sensors constitute a feasible cover for CMLP, that is, they cover all targets and induce a communication tree rooted at the base station s_0 in the communication graph. Binary variables $x_{ij} \ \forall (s_i, s_j) \in E^d$ are used to represent which edges in the connectivity graph are chosen to be part of the tree, while variables f_{ij} are flow variables used to ensure its connectivity. Constraints (5) guarantee that each target is covered by at least one active sensor. Constraint (6) imposes the amount of flow produced by the base station to be equal to the number of activated sensors. Constraints (7) are the flow conservation constraints, while Constraints (8) make sure that exactly one ingoing communication link (s_i, s_j) is chosen for any active sensor s_j. Finally, by effect of Constraints (9) there can only be positive flow on the selected communication links.

In [13], the subproblem formulation was extended to α-CMLP as follows:

$$\textbf{[SP']} \qquad \min \sum_{s_i \in S \setminus \{s_0\}} \pi_i y_i \tag{13}$$

s.t.

(6)-(12)

$$\sum_{s_i \in S \setminus \{s_0\}} \delta_{ki} y_i \geq z_k \qquad \forall t_k \in T \tag{14}$$

$$\sum_{t_k \in T} z_k \geq T_\alpha \tag{15}$$

$$z_k \in \{0,1\} \qquad \forall t_k \in T \tag{16}$$

Since for the α-CMLP problem not all targets are always required to be covered for each cover to be feasible, the additional binary z_k variables are introduced for each $t_k \in T$ to represent the decision for the related target. Constraints (14) state that each z_k variable can be equal to 1 only if at least one of the sensors that can cover it is activated, while Constraints (15) ensure that at least T_α targets are covered.

In this work, we propose a better formulation for the subproblem that, compared to [**SP'**], does not use the x_{ij} variables. Indeed, it can be noted that Constraints (8) and (9) can be substituted with a new set of constraints which imposes all sensors with ingoing positive flow to be activated in the cover. This leads to the following formulation:

$$\textbf{[SP"]} \qquad \min \sum_{s_i \in S \setminus \{s_0\}} \pi_i y_i \tag{17}$$

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

$$(6),(7),(10),(12),(14)\text{-}(16)$$

$$y_i \leq \sum_{(s_j,s_i) \in E} f_{ji} \leq (|S|-1) y_i \qquad \forall s_i \in S \setminus \{s_0\} \tag{18}$$

The main disadvantage of such column generation approaches is that solving the subproblem is NP-Hard (see [13]). As the size of the problem grows, [**SP"**] becomes harder to solve, and the number of iterations required for the column generation procedure to converge is expected to increase as well. It can be noted, however, that the exact subproblem solution is only needed to certify optimality in the final iteration while, in general, any new attractive column could be used to proceed to the next column generation iteration. For this reason, as will be discussed in the next section, we designed a genetic algorithm (GA) to quickly produce attractive columns. Furthermore, even when GA fails and solving the [**SP"**] ILP formulation is indeed needed, we check every incumbent feasible solution found by the solver, and stop the search as soon as a solution with an objective function value which is less than 1 is found. Thanks to the these modifications, the performances of the CG scheme are remarkably improved, as will be analyzed in Section 5.

4 A Genetic Algorithm to Address the [SP"] Subproblem

As previously mentioned, we designed an effective genetic algorithm that we embedded in the column generation framework in order to improve its performances. In this section, we first give a general overview of the algorithm, and then describe all of its different features in detail.

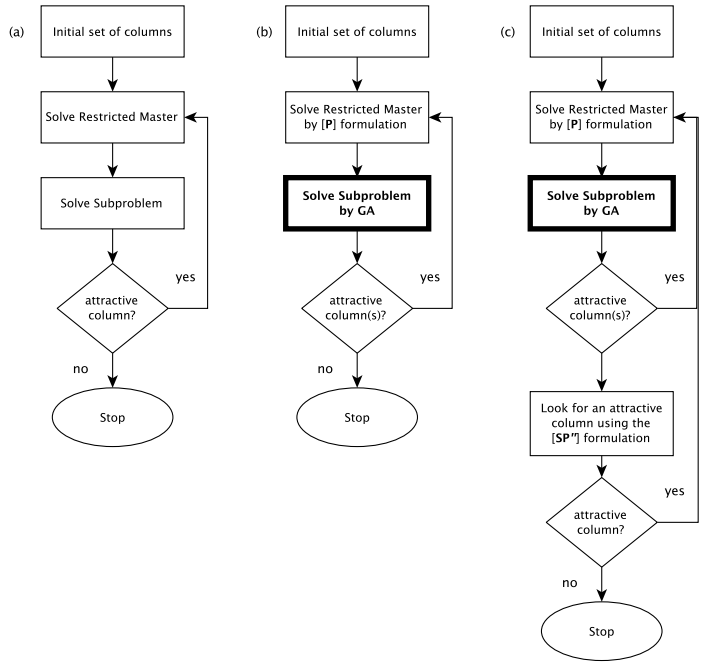


Fig. 3 Classical CG scheme (a). HCG scheme (b). ECG scheme (c).

4.1 GA overall structure

Our genetic algorithm (GA) is used in the column generation scheme to solve [**SP"**] heuristically. As for the mathematical formulation, it is used to build feasible covers and uses the dual prices coming from the latest restricted master problem iteration to weight each sensor.

The GA is used to develop both a heuristic and an exact approach, that we call HCG and ECG, respectively. In HCG, the genetic is used to entirely substitute [**SP"**] in the column generation scheme, and the procedure stops as soon as the GA fails to find an attractive cover. This kind of approach can not certify that the global optimal solution has been reached; however, such a procedure can still prove to be a very effective heuristic, as will be shown in Section 5. Furthermore, we developed ECG by solving [**SP"**] every time that the heuristic for the subproblem fails, in order to either find a new attractive cover which was not found by the GA, or finally certify optimality for the incumbent solution. However, as mentioned in Section 2, in this case the subproblem formulation is not necessarily solved to optimality, since we stop as soon as [**SP"**] finds a column with an objective function which is less than 1. In both HCG and ECG, after each GA iteration we add all the attractive covers contained in its final population, in order to further accelerate the algorithms convergence. The flowcharts in Figures 3(b) and 3(c) illustrate how the GA procedure is integrated within our proposed approaches.

Genetic algorithms are population-based metaheuristics that, as other methods belonging to the class of evolutionary algorithms, use techniques that draw inspira-

Algorithm 1: GA pseudocode

```
Input: WSN = (S, T), α ∈ (0, 1], G = (S, E), DP;
Output: Set of feasible covers;
P ← initP(WSN, G, α, DP);
BestFit ← bestFitness(P, DP);
criteria ← setCriteria(MaxIT, MaxDUP);
SPL ← evaluateShortestPaths(G, DP);
while check(criteria) do
    (Cp1, Cp2) ← tournament(P, DP);
    C ← crossover(Cp1, Cp2);
    C ← mutation(C);
    C ← coverFeasibilityOperator(C, WSN, α);
    C ← connectFeasibilityOperator(C, G, SPL, DP);
    C ← redundancyRemovalOperator(C, WSN, G);
    if C ∉ P then
        P ← insert(C);
        if fitness(C, DP) ≥ BestFit then
            update(criteria);
        else
            BestFit ← fitness(C, DP);
    else
        update(criteria);
Chromos ← chromosomes with fitness < 1;
return Chromos;
```

tion from biological evolution concepts, including natural selection, reproduction and mutation. Given a starting population of individuals representing problem solutions, usually defined *chromosomes*, a GA typically produces new solutions by combining information belonging to two or more *parent* chromosomes, an operation known as the *crossover* operator. Newly generated chromosomes are also often randomly perturbed by means of a *mutation* operator in order to diversify the population. Chromosomes are evaluated and ranked using a *fitness* function, usually connected to the objective function of the considered optimization problem. The overall aim of GAs is to emulate the process of natural selection by iteratively producing better fit individuals, that inherit favorable characteristics from their parents. For an extensive introduction to genetic algorithms, the reader may refer to [16].

The pseudocode of our GA is reported in Algorithm 1. The input data consist of the wireless sensor network $WSN = (S, T)$, its connectivity graph $G = (S, E)$, the α value and the dual prices vector DP obtained from the last iteration of the restricted master problem. GA starts by building a population P of chromosomes, representing feasible covers; each chromosome has the structure described in Section 4.2, and the first population is built as reported in Section 4.7. It also initializes the stopping criteria used by the GA, which will be later described. The final step of the initialization phase involves the evaluation of the shortest paths between each couple of elements of the set S in the connectivity graph G, using a weighting function for the edges of E which depends on the dual prices vector DP. The Floyd-Warshall algorithm (see [23]) is used for this computation. The

shortest paths are used by an operator named *connect feasibility*; the details on this operator, as well as on the considered weighting function, are given in Section 4.5.

After the initialization phase, the procedure builds iteratively new chromosomes, one by one. In more detail, in each iteration two *parent* chromosomes are chosen and combined through the crossover operator (see Section 4.3); the obtained *child* C is then mutated (Section 4.4). Since these operations do not guarantee the feasibility of C, both in terms of coverage and connectivity, two operators are applied in order to eventually transform it into a chromosome which represents a feasible cover (Section 4.5). The final modifications applied to C are made by an operator that checks if some of its sensors can be switched off while preserving feasibility (Section 4.6). The resulting chromosome is introduced in the population, unless an identical one already belongs to it. If the chromosome is added to the population, it replaces an older one which is chosen randomly among the $|P|/2$ chromosomes with worst fitness function. It follows that the population size never changes during the algorithm execution.

The GA ends its execution as soon as one of two stopping criteria is reached, which make use of two parameters, called $MaxIT$ and $MaxDUP$ respectively. The $MaxIT$ parameter refers to a maximum number of iterations without improvements with respect to the best fitness value in P, while $MaxDUP$ is a maximum number of consecutive generated chromosomes which have a duplicate in the population.

In the last step, GA returns all the chromosomes in P that correspond to attractive covers.

4.2 Chromosome Representation and Fitness Function

Each chromosome C in our GA algorithm is internally represented as a binary vector of length $|S|$. The element in the i-th position of the vector, with $i \in \{0, 1 \ldots, m\}$, is called the i-th *gene* of C and is denoted by $C[i]$. The gene $C[i]$ is associated to $s_i \in S$, and it is equal to 1 if and only if s_i is activated in C. In this case, we say that C contains s_i. By extension, chromosomes corresponding to feasible covers are defined to be feasible as well. Obviously, since each feasible cover has to contain the base station s_0, $C[0]$ must be equal to 1 in each feasible chromosome C. It is also easy to see that, ruling out the $C[0]$ gene, a feasible chromosome is a column of **[P]**. The operators of our GA make sure that only feasible chromosomes are introduced in the population.

The fitness function is equivalent to the objective function (17) of the **[SP"]** formulation, and is easily computed as the dot product of the chromosome and the dual prices vector DP (we assume the dual price of s_0 to be equal to 0). It follows that any feasible chromosome with a fitness value lower than 1 is an attractive column for the restricted master problem.

4.3 Crossover operator

The aim of the crossover operator is to create new individuals from chromosomes in the current population (their parents), which hopefully inherit their good features,

eventually leading to better solutions. The selection of the parents is carried out by using a binary tournament strategy. That is, the crossover randomly selects two chromosomes of P, and the one with the best fitness is designated as first parent C_{p1}. The same procedure is used to select the second parent C_{p2}, making sure that C_{p1} is different from C_{p2}.

After the parents selection, the crossover operator generates the child C by performing a bitwise logical AND operation on the parents, that is, for any given position $i \in \{0, 1, \ldots, m\}$, $C[i] = 1$ if and only if $C_{p1}[i] = 1$ and $C_{p2}[i] = 1$. It is easy to see that since both parent chromosomes are feasible, the child always contains the base station, i.e. $C[0] = 1$. This choice will not be modified by the subsequently applied operators.

4.4 Mutation operator

Mutation operators are commonly applied after the crossover phase as a mean to add diversification to the population by applying some random perturbations to the newly generated chromosomes.

Our mutation operator randomly selects a gene of the child C whose value is identical in the parents, if it exists, and changes its value ($C[0]$ is excluded from the random selection). In this way, at least one gene will differ between the child and its parents. In the unlikely case in which the parents share no common genes except the one corresponding to s_0, the child will contain no sensors, and will be entirely built by the operators described in Sections 4.5 and 4.6.

4.5 Feasibility Fixing Operators

The child chromosome C derived by the crossover and the mutation procedures is not guaranteed to be feasible. Indeed, neither the coverage of T_α targets nor the connectivity of the induced subgraph are guaranteed. For this reason, we introduce two feasibility fixing operators whose aim is to make C feasible.

The two operators are applied in sequence. The first one, *cover feasibility*, starts by checking which targets are covered by the currently activated sensors. If they are fewer than T_α, it randomly selects a currently uncovered target t_k and a sensor s_i among the ones that can cover t_k. Then, the operator activates s_i in C and updates the set of targets covered by the chromosome. The procedure iterates by selecting and activating new sensors, until at least T_α targets are covered.

The chromosome C returned by the cover feasibility operator may still be unfeasible, since the activated sensors cover the required number of targets, but its induced subgraph G' in G may be disconnected. For this reason, we introduce a second feasibility fixing operator, named *connect feasibility*, whose aim is to activate new sensors in C to connect G'. To this end, we formulate this issue as a Steiner Tree. Given an undirected and edge weighted graph $G = (V, E)$, and given a subset of vertices $V' \subseteq V$, the Steiner Tree problem consists in finding a minimum cost subtree of G which covers all the vertices in V'. The tree may include elements of $V \setminus V'$. The vertices in V' are named *basic*, while those in $V \setminus V'$ are the *steiner* vertices.

In our case, we mark the base station and the sensors activated in C as basic, and all other sensors as steiner. Furthermore, with the aim of facilitating the selection of sensors with low dual price values, we define a function which assigns to each edge of E a weight equal to the sum of the dual prices of its endpoints (recall that s_0 is assumed to have a dual price equal to 0).

The Steiner Tree problem is well-known to be NP-hard [22] therefore it is not reasonable to optimally solve it every time that a chromosome is built. Several heuristics have been proposed in the literature to solve the Steiner Tree problem. A survey on these heuristics can be found in [19]. In this work, we use a fast and effective construction heuristic that the authors call CHINS (Cheapest Insertion) and that works as follows:

CHINS Heuristic
Input: Weighted graph G, basic vertices list;
Output: Steiner Tree $\mathcal{T}$;

1. Initialize the solution $\mathcal{T}$ with a single arbitrary basic vertex i;
2. **Repeat:**
   (a) Find the shortest path $\mathcal{P}$ in G between any basic vertex j not in $\mathcal{T}$ and any vertex in $\mathcal{T}$;
   (b) Add all edges and vertices of $\mathcal{P}$ to $\mathcal{T}$;
3. **Until** $\mathcal{T}$ contains all basic vertices.

An improvement of CHINS called CHINS-Q iterates the procedure $|V'|$ times, selecting every time a different basic vertex for the starting choice i, and finally returning the best encountered solution.

Our connect feasibility operator implements CHINS-Q. For each element s_i contained in C, an iteration is performed. In each iteration, we first build a vector C' with $C'[i] = 1$ and all other genes equal to 0. Then, among all the shortest paths between an s_j contained in C that is not is C' and an s_p in C', the procedure individuates the one with lowest weight and activates in C' all the elements belonging to this shortest path. This operation is repeated until C' contains all the elements of C and thus it is a feasible chromosome.

Finally, the best solution found is returned.

4.6 Redundancy Removal Operator

The C chromosome resulting from the feasibility operators is guaranteed to be feasible. However, since such operators may have added redundant coverage, we use a final operator to try to deactivate some of its sensors without compromising feasibility. In more detail, the operator first considers the tree individuated by the connect feasibility operator. Then, it builds a list of all the leaves of the tree whose deactivation would not compromise the coverage of T_α targets (eventually excluding s_0). If the list is not empty, one of its sensors is randomly chosen and deactivated. The list is then updated, and the procedure iterates until no more sensors can be deactivated.

4.7 Building the Starting Population and CG initialization

Each individual belonging to the starting population P is built by applying the two feasibility fixing and the redundancy removal operators on a chromosome which initially only contains the base station s_0. Each chromosome built by applying these steps is added to the population unless an identical one already belongs to it. The procedure iterates until either a predefined number $Size_P$ of chromosomes have been built, or a threshold $maxInitDUP$, representing a maximum number of consecutive duplicate chromosomes, is reached. In the latter case, $Size_P$ is updated to the resulting value of $|P|$. As previously mentioned, the population size remains constant throughout the GA execution.

At the beginning of both our heuristic and exact approaches, in order to initialize the restricted master problem with a set of feasible columns, we use a preliminary run of our GA, using random values for the dual prices. The whole set of $Size_P$ chromosomes belonging to the final population is used to produce the starting set of columns.

5 Computational Results

This section presents the results of our extensive computational test phase, on two groups of benchmark instances proposed in [24] for the case $\alpha = 1$ and in [13] for the general case, respectively. We only perform a comparison with the algorithms described in [13], since the authors experimentally proved that these algorithms outperform the ones proposed in [24].

5.1 Instances description

The computational test are carried out on the benchmark instances used in [13]. These instances are divided into two groups. The instances that we call group 1 were introduced for the first time in [13], while group 2 is composed by older instances originally proposed in [24].

Instances in group 1 have a number of sensors $|S \setminus \{s_0\}|$ varying in the set $\{100, 200, 300, 400, 500\}$ and a number of targets $|T|$ equal to either 15 or 30. Different coverage levels are considered, represented by the α value which varies in the set $\{0.7, 0.85, 1\}$. The communication range R_C is fixed and equal to 125 for all nodes in S, while the sensing range value R_S varies in the set $\{100, 125\}$. For each combination of parameters four different instances were generated, that together represent a *scenario*. Therefore, there are in total 240 group 1 test instances, that compose 60 scenarios. Instances belonging to group 2 have 15 targets and a number of sensors varying in the set $\{50, 75, 100, 150, 200\}$. For each value of $|S \setminus \{s_0\}|$, 5 different instances were generated, for a total of 25 instances. The communication range is assumed to be equal to the sensing range and full coverage for the whole set of targets is required. In the instances of both groups, sensors, targets and the base station are randomly disposed over a bidimensional region and all sensors can be activated for at most 1 time unit. More details on the instances can be found in [13] and [24].

5.2 Testing environment and parameters settings

Our algorithms have been coded in C++, and the tests were performed using a machine running under the OSX Lion operating system, with an Intel Core i5 2.5 GHz processor and 4GB of RAM (single thread mode). Our approaches make use of the IBM ILOG CPLEX 12.6.1 solver and the Concert Technology Library to solve the mathematical formulations.

The tests in [13] ran on a similar setup, using a machine equipped with an Intel Core i5 1.6 Ghz processor with 2GB of RAM using the same operating system. Their algorithms were similarly coded in C++, in conjunction with the Gurobi optimization engine for the mathematical models.

After a parameter tuning phase, we determined the values of the parameters used by our GA algorithm for all the tests. The population size $Size_P$ was chosen to be equal to 100. We recall that this parameter also controls the maximum number of new columns which is returned to the restricted master problem after each GA iteration, since each attractive cover found in the final population is used to produce one of them. The $maxInitDUP$ threshold used during the initialization (Section 4.7) was chosen to be equal to 100, while the chosen values for the two parameters $MaxIT$ and $MaxDUP$, regulating the termination criteria (Section 4.1) are 2000 and 100, respectively. Similarly to [13], a 3600 seconds time limit is considered for each test, and the best solution found is reported whenever the time limit is reached. All results are rounded to two decimal places.

5.3 Impact of the premature subproblem interruptions

In this work we introduce two different methods to overcome the difficulty of solving **[SP"]** to optimality, namely our GA algorithm and the invocation of a CPLEX callback function to prematurely stop the subproblem resolution as soon as it finds an attractive cover. The underlying idea is that the latter method is used to produce a new cover that can both improve the current incumbent solution and help GA to escape the local optimum in which it was trapped, hopefully regaining effectivity in its next invocation, with a computational expense that can be significantly smaller than solving **[SP"]** to optimality.

While for the mentioned reasons we believe that these methods are most effective when used in conjunction, in order to evaluate the impact of the callback invocations, without being affected by the GA performances, we carried out a first test phase in which we compared the traditional column generation approach (i.e. the one represented in Figure 3(a)) with and without the callback invocation during the subproblem resolution. In the following the two procedures are referred to as CG-Call and CG-Std, respectively. For both these algorithms, the GA is used only once to produce the initial restricted set of columns, as described in Section 4.7. The results of this comparison are presented in Table 1.

We performed these tests on the group 1 instances with at most 200 sensors. Results for instances with $R_S = 100$ are shown in the top half, while those with $R_S = 125$ are reported in the bottom half. Each line in the table represents a scenario composed of four instances with the same characteristics, and the results reported in each line report the average values on these four instances. The

				CG-Std				CG-Call				% Gap	
	$\|S \setminus \{s_0\}\|$	$\|T\|$	α	LT	Time	S-Inv	#Opt	LT	Time	S-Inv	#Opt	LT	Time
$R_S = 100$	100	15	0.70	6.83	956.03	108.75	3	6.88	65.90	232.25	4	**0.73**	**93.11**
			0.85	6.19	2704.51	96.00	1	6.64	1067.56	389.75	3	**7.27**	**60.53**
			1.00	4.00	1.76	26.50	4	4.00	2.18	37.50	4	0.00	-23.86
	100	30	0.70	6.99	1954.41	124.25	3	7.00	136.70	298.50	4	**0.14**	**93.01**
			0.85	6.13	1951.51	126.75	2	6.58	1850.06	598.25	2	**7.34**	**5.20**
			1.00	3.90	905.36	45.50	3	4.00	15.52	98.75	4	**2.56**	**98.29**
	200	15	0.70	15.00	1150.73	175.75	3	15.89	934.58	653.25	3	**5.93**	**18.78**
			0.85	14.43	2729.80	255.50	1	15.21	1145.94	1025.50	3	**5.41**	**58.02**
			1.00	10.25	141.86	174.75	4	10.25	54.47	216.25	4	0.00	**61.60**
	200	30	0.70	15.44	2054.10	239.75	2	15.83	1000.75	1112.25	3	**2.53**	**51.28**
			0.85	13.24	2748.47	233.00	1	14.69	2070.51	1857.25	3	**10.95**	**24.67**
			1.00	8.75	163.89	129.50	4	8.75	39.78	142.00	4	0.00	**75.73**
$R_S = 125$	100	15	0.70	7.00	88.39	62.25	4	7.00	10.71	95.50	4	0.00	**87.88**
			0.85	6.73	1822.44	98.75	2	6.88	117.09	278.75	4	**2.23**	**93.58**
			1.00	4.67	926.98	53.25	3	4.75	29.04	102.00	4	**1.71**	**96.87**
	100	30	0.70	7.00	37.88	65.25	4	7.00	13.76	86.75	4	0.00	**63.67**
			0.85	6.65	1886.03	119.50	2	6.79	227.41	344.75	4	**2.11**	**87.94**
			1.00	4.37	1804.60	49.75	2	4.75	76.04	143.50	4	**8.70**	**95.79**
	200	15	0.70	15.55	1034.71	184.75	3	16.25	342.70	415.75	4	**4.50**	**66.88**
			0.85	15.28	1053.57	208.00	3	15.75	558.08	589.75	4	**3.08**	**47.03**
			1.00	12.48	1833.12	212.25	2	13.00	480.36	595.00	4	**4.17**	**73.80**
	200	30	0.70	16.25	477.56	203.50	4	16.25	667.70	694.00	4	0.00	-39.81
			0.85	14.92	2219.76	231.00	2	15.50	1198.77	1272.00	3	**3.89**	**46.00**
			1.00	11.26	989.82	196.25	3	11.75	734.31	590.25	4	**4.35**	**25.81**

Table 1 Comparison between the CG-Std and CG-Call approaches.

$|S\setminus\{s_0\}|$, T and α columns report the instances characteristics. The LT column reports the lifetime values expressed in time units found within the time limit, while the column *Time* shows the computational time in seconds. The *S-Inv* and *#Opt* columns report the average number of subproblem invocations and the number of optimal solutions found in the scenario, respectively. The last two columns, under the *% Gap* heading, report the percentage gap of the lifetimes and of the computational times, respectively. Let LT(Alg) and $Time$(Alg) be the average lifetime value and computational time reported by a given Alg procedure on a considered scenario. The LT gaps are computed as

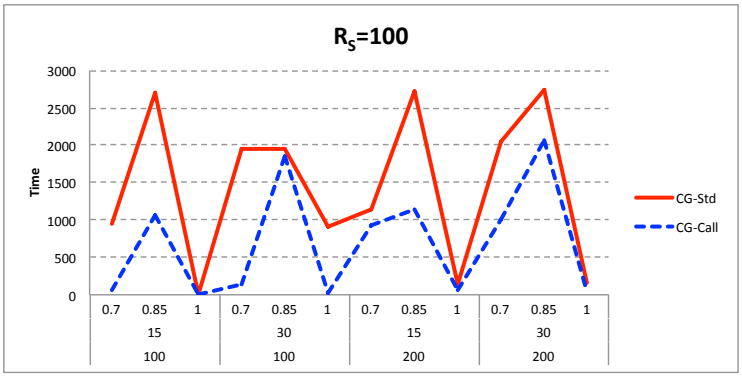
$$100 \times \frac{LT(\text{CG-Call}) - LT(\text{CG-Std})}{LT(\text{CG-Std})}$$

meaning that if a gap is positive CG-Call found on average better solutions than CG-Std on the considered scenario. The Time gaps are computed instead as

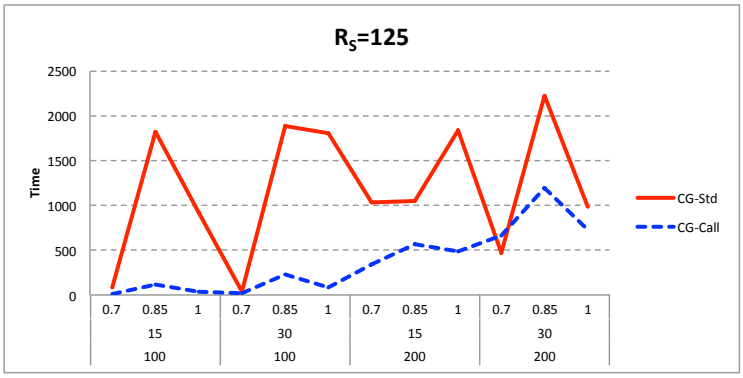
$$100 \times \frac{Time(\text{CG-Std}) - Time(\text{CG-Call})}{Time(\text{CG-Std})}$$

and therefore a positive gap represents a scenario in which CG-Call required on average less computational time than CG-Std. Whenever a gap is positive, it is marked in bold to highlight that CG-Call performs better than CG-Std.

The results of Table 1 show that CG-Call is on average faster than CG-Std in 22 out of 24 scenarios, with a time gap that grows up to 98.29%. In many cases, the computational time of CG-Call is less than half the computational time of CG-Std, and in 6 cases CG-Call is one order of magnitude faster. Note that in one of the two scenarios in which CG-Std is faster than CG-Call, the difference is less than 0.5 seconds. Finally, the time limit is reached by CG-Call for only 8 out of 96 instances, as opposed to 31 time limits reached by CG-Std.



(a)



(b)

Fig. 4 Computational times comparisons for the CG-Std and CG-Call approaches on group 1 instances with $R_S = 100$ (a) and $R_S = 125$ (b).

Regarding the effectiveness of the two approaches, CG-Call finds better solutions than CG-Std in 18 out of 24 scenarios and, on the remaining 6 scenarios, the solutions are the same. On the other hand, the number of subproblem invocations for CG-Call is always higher than the one for CG-Std. Indeed, the peak of the S-Inv value for CG-Call is equal to 1857.25 for $R_S = 100$ and to 1272 for $R_S = 125$, while for CG-Std it is equal to 255.50 and 231, respectively. While always solving the subproblem to optimality can predictably reduce the number of required iterations, the results clearly show that in most cases the low S-Inv values for CG-Std are due to the high amounts of time spent by the solver to find and certify the optimal solution at each invocation, which limits the number of iterations that can be performed within the time limit and significantly affects the final solution quality.

To further highlight the performance improvements brought by the callback usage, the computational times of two algorithms are plotted in Figure 4(a) for $R_S = 100$ and Figure 4(b) for $R_S = 125$, respectively.

In both figures, the x-axis reports the characteristics of the instances, while the y-axis reports the computational time in seconds. The results for CG-Call and CG-Std are represented by dashed and a continuous lines, respectively. The

figures show that the performance gap between the two algorithms increases along with the R_S value. Indeed on the scenarios with $R_S = 100$, despite significant performance gaps, similar trends can be noticed. On the scenarios with $R_S = 125$, instead, the behavior of the two procedures is completely different. In 7 out of 12 scenarios, CG-Std spends more than 1000 seconds, and in 5 cases more than 1800 seconds. CG-Call, instead, requires about 1200 seconds once, and less than 750 seconds in all other cases.

Given the presented analysis, the method which makes use of the callback function was considered to be the best trade-off, and was used in all the remaining experiments.

5.4 Comparisons on group 1 dataset

Tables 2 and 3 contain the comparisons between ECG and CG-EXACT, introduced in [13], for the cases $R_S = 100$ and $R_S = 125$, respectively. As for the previous table, each entry is an average over the 4 instances of the related scenario, and all column headings have the same meaning. Gaps are similarly computed, and the cases in which ECG performs better than CG-EXACT are highlighted in bold. The last row (*#Opt Found*) reports the overall number of optimal solutions found by each approach.

The results of Table 2 show that ECG is more effective than CG-EXACT, as well as significantly more efficient. Indeed, for 11 scenarios the average solution found by ECG is better than the one found by CG-EXACT, with a solution gap that ranges from 0.30% to 10.46%, while on the remaining scenarios the solution is the same. Moreover, the values in the #Opt column show that ECG finds the optimal solution for 102 out of 120 instances, as opposed to the 100 found by CG-EXACT. Regarding the computational time performances, ECG is always faster than CG-EXACT and, in the 19 scenarios in which both algorithms find all the optimal solutions (#Opt=4), the performance gap ranges from 39.44% to 98.87%. In 6 scenarios, ECG is an order of magnitude faster than CG-EXACT.

The results reported in Table 3 show that similar behaviors can be observed for the case $R_S = 125$. Indeed, for 11 scenarios the average solution found by ECG is better than the one found by CG-EXACT, with a solution gap that ranges from 0.99% to 9.12%. Moreover, ECG optimally solves all the scenarios with up to 300 sensors and, in general, it finds 108 out of 120 optimal solutions, as opposed to the 102 found by CG-EXACT. Again, ECG is always faster than CG-EXACT, and in the 19 scenarios where both procedures find all the optimal solutions, the performance gap ranges from 45.81% to 99.38%. In 10 scenarios, ECG is an order of magnitude faster than CG-EXACT.

Despite the slightly different hardware configuration and the different solver used in our tests and in those reported in [13], we believe that the previous comparisons proved ECG to outperform CG-EXACT, given the consistently better results and the gap in terms of computational time required, which is often huge. However, in our opinion, even stronger evidence is provided when the heuristic approaches are compared. Indeed, in this case the solvers are only used to solve the restricted master problem, and therefore their impact on the performances of the two procedures is much lower. Furthermore, it will be shown that with respect to the best heuristic presented in [13], HCG is able to find better solutions in less

Group 1: $R_S = 100$											
			ECG			CG-EXACT			% GAP	
$\lvert S \setminus \{s_0\}\rvert$	$\lvert T\rvert$	α	LT	Time	#Opt	LT	Time	#Opt	LT	Time
100	15	0.70	6.88	1.14	4	6.88	8.78	4	0.00	**87.02**
		0.85	6.64	47.03	4	6.64	921.47	4	0.00	**94.90**
		1.00	4.00	1.09	4	4.00	1.80	4	0.00	**39.44**
100	30	0.70	7.00	1.61	4	7.00	6.54	4	0.00	**75.38**
		0.85	6.59	1805.37	2	6.57	1922.59	2	**0.30**	**6.10**
		1.00	4.00	1.89	4	4.00	4.06	4	0.00	**53.45**
200	15	0.70	16.25	4.70	4	16.25	414.98	4	0.00	**98.87**
		0.85	15.52	905.61	3	15.42	941.60	3	**0.65**	**3.82**
		1.00	10.25	2.95	4	10.25	12.69	4	0.00	**76.75**
200	30	0.70	16.25	4.85	4	16.25	128.92	4	0.00	**96.24**
		0.85	15.43	911.05	3	15.35	1514.62	3	**0.52**	**39.85**
		1.00	8.75	3.60	4	8.75	19.80	4	0.00	**81.82**
300	15	0.70	18.25	6.84	4	18.25	34.72	4	0.00	**80.30**
		0.85	18.25	8.19	4	18.25	91.10	4	0.00	**91.01**
		1.00	15.00	7.48	4	15.00	86.26	4	0.00	**91.33**
300	30	0.70	18.25	8.86	4	18.25	47.93	4	0.00	**81.51**
		0.85	18.25	11.38	4	18.25	104.34	4	0.00	**89.09**
		1.00	13.25	7.97	4	13.25	48.34	4	0.00	**83.51**
400	15	0.70	31.97	913.13	3	30.68	999.22	3	**4.20**	**8.62**
		0.85	29.62	916.11	3	28.66	1151.16	3	**3.35**	**20.42**
		1.00	18.25	13.07	4	18.25	95.70	4	0.00	**86.34**
400	30	0.70	30.63	918.90	3	29.55	1007.00	3	**3.65**	**8.75**
		0.85	28.02	943.67	3	26.90	1907.09	2	**4.16**	**50.52**
		1.00	18.00	23.57	4	18.00	125.38	4	0.00	**81.20**
500	15	0.70	48.68	1825.62	2	45.04	2554.30	2	**8.08**	**28.53**
		0.85	43.30	2713.48	1	39.20	2742.29	1	**10.46**	**1.05**
		1.00	29.00	31.97	4	29.00	335.83	4	0.00	**90.48**
500	30	0.70	48.41	1822.66	2	44.83	2748.99	1	**7.99**	**33.70**
		0.85	41.02	2725.31	1	37.24	2768.34	1	**10.15**	**1.55**
		1.00	26.25	34.22	4	26.25	308.25	4	0.00	**88.90**
#Opt Found					102			100		

Table 2 Comparison between ECG and CG-EXACT algorithms on group 1 instances with $R_S = 100$

time and to individuate a significantly higher number of optimal solutions, proving to be the most effective heuristic known to date for the problem.

In Tables 4 and 5, HCG is compared with the overall best-performing heuristic approach presented in [13], namely CG-MULTI. In order to determine whether these algorithms found optimal solutions, we compare their solution values with the known optimal values found by ECG. The #Opt Found values reported in the last row of Table 4 clearly show that, with 102 optimal solutions found, HCG is much more effective than CG-MULTI, which only finds 85 optimal solutions for $R_S = 100$. Overall, in 21 out of 30 scenarios the solutions found by HCG are better than the solutions found by CG-MULTI, with a solution gap that ranges from 0.59% to 11.01%. Moreover, CG-MULTI never finds better solutions than HCG. It is interesting to observe that in the scenarios with up to 300 sensors, only in 3 cases (corresponding lines 5, 8 and 11) HCG does not find all the optimal solutions. In more detail, on the 72 instances associated to the above mentioned scenarios, only 4 instances are not solved to optimality by HCG. On the other hand, on the same 72 instances, CG-MULTI does not find the optimal solution 19 times and, in particular, it finds all 4 optimal solutions only in scenarios corresponding to $\alpha = 1$,

Group 1: $R_S = 125$											
			ECG			CG-EXACT			% GAP	
$\lvert S \setminus \{s_0\}\rvert$	$\lvert T\rvert$	α	LT	Time	#Opt	LT	Time	#Opt	LT	Time
100	15	0.70	7.00	0.97	4	7.00	1.79	4	0.00	**45.81**
		0.85	6.88	1.40	4	6.88	8.37	4	0.00	**83.27**
		1.00	4.75	1.02	4	4.75	3.62	4	0.00	**71.82**
100	30	0.70	7.00	1.03	4	7.00	3.05	4	0.00	**66.23**
		0.85	6.79	2.03	4	6.79	327.39	4	0.00	**99.38**
		1.00	4.75	1.59	4	4.75	91.41	4	0.00	**98.26**
200	15	0.70	16.25	3.76	4	16.25	19.56	4	0.00	**80.78**
		0.85	15.75	3.83	4	15.75	31.46	4	0.00	**87.83**
		1.00	13.00	3.58	4	13.00	46.56	4	0.00	**92.31**
200	30	0.70	16.25	3.96	4	16.25	19.34	4	0.00	**79.52**
		0.85	16.25	68.63	4	16.09	925.98	3	**0.99**	**92.59**
		1.00	11.75	4.52	4	11.75	96.81	4	0.00	**95.33**
300	15	0.70	18.25	7.08	4	18.25	27.97	4	0.00	**74.69**
		0.85	18.25	7.49	4	18.25	40.07	4	0.00	**81.31**
		1.00	16.75	7.22	4	16.75	52.10	4	0.00	**86.14**
300	30	0.70	18.25	7.48	4	18.25	29.23	4	0.00	**74.41**
		0.85	18.25	8.57	4	18.25	50.03	4	0.00	**82.87**
		1.00	16.00	9.16	4	16.00	82.83	4	0.00	**88.94**
400	15	0.70	34.24	906.84	3	33.36	991.85	3	**2.64**	**8.57**
		0.85	32.21	917.02	3	31.06	998.19	3	**3.70**	**8.13**
		1.00	24.88	18.50	4	24.88	298.96	4	0.00	**93.81**
400	30	0.70	33.57	914.10	3	32.08	956.19	3	**4.64**	**4.40**
		0.85	30.24	930.29	3	29.08	1021.31	3	**3.99**	**8.91**
		1.00	22.38	19.19	4	22.38	245.22	4	0.00	**92.17**
500	15	0.70	54.49	1819.71	2	50.51	1957.14	2	**7.88**	**7.02**
		0.85	48.97	1822.00	2	45.58	2624.69	2	**7.44**	**30.58**
		1.00	37.75	54.86	4	36.82	1937.38	2	**2.53**	**97.17**
500	30	0.70	54.89	1816.55	2	51.13	1947.69	2	**7.35**	**6.73**
		0.85	47.00	1835.68	2	43.07	2741.61	1	**9.12**	**33.04**
		1.00	35.50	86.53	4	34.47	1938.50	2	**2.99**	**95.54**
#Opt Found					108			102		

Table 3 Comparison between ECG and CG-EXACT algorithms on group 1 instances with $R_S = 125$

in which a lower number of feasible covers is likely to exist. These results highlight the higher versatility of our heuristic, which is often able to find optimal solutions regardless of the considered type of instance. Regarding the computational time efficiency, CG-MULTI results to be faster than HCG only once (see the scenario corresponding to line 3), however the time gap is lower than 0.15 seconds and can be considered negligible. In the other 29 scenarios, HCG is up to 98.05% faster than CG-MULTI (see line 7), in 22 of them the time gap is greater than 70%, and 12 times HCG is one order of magnitude faster.

The results reported in Table 5 show that both the heuristics are more effective when $R_S = 125$. Indeed, the number of optimal solutions found grows to 107 for HCG and to 91 for CG-MULTI. In 18 out of 30 scenarios, the solutions found by HCG are better than the ones found by CG-MULTI, with a solution gap that ranges from 0.15% to 8.82%. In the remaining scenarios, the two algorithms report the same solutions.

Looking at the first 18 scenarios, it can be seen that the optimal solution is not found only once by HCG and 10 times by CG-MULTI. The results show a lower influence of the α parameter with respect to the results reported in Table 4. For

Group 1: $R_S = 100$										
			HCG			CG-MULTI			% GAP	
$\lvert S \setminus \{s_0\}\rvert$	$\lvert T\rvert$	α	LT	Time	#Opt	LT	Time	#Opt	LT	Time
100	15	0.70	6.88	0.95	4	6.63	4.08	3	**3.77**	**76.72**
		0.85	6.64	1.92	4	6.04	10.16	1	**9.93**	**81.10**
		1.00	4.00	0.97	4	4.00	0.84	4	0.00	-15.48
100	30	0.70	7.00	1.44	4	6.75	4.75	3	**3.70**	**69.68**
		0.85	6.54	4.31	2	6.49	44.82	2	**0.77**	**90.38**
		1.00	4.00	1.72	4	4.00	2.75	4	0.00	**37.45**
200	15	0.70	16.25	4.13	4	15.96	212.26	2	**1.82**	**98.05**
		0.85	15.46	7.13	3	15.37	253.00	3	**0.59**	**97.18**
		1.00	10.25	2.56	4	10.25	8.64	4	0.00	**70.37**
200	30	0.70	16.25	4.20	4	16.00	100.17	3	**1.56**	**95.81**
		0.85	15.34	15.21	3	14.69	715.21	1	**4.42**	**97.87**
		1.00	8.75	3.29	4	8.50	3.29	3	**2.94**	0.00
300	15	0.70	18.25	6.62	4	18.00	9.00	3	**1.39**	**26.44**
		0.85	18.25	7.58	4	18.00	28.88	3	**1.39**	**73.75**
		1.00	15.00	6.80	4	15.00	54.26	4	0.00	**87.47**
300	30	0.70	18.25	6.95	4	17.00	9.26	3	**7.35**	**24.95**
		0.85	18.25	10.52	4	16.44	39.28	3	**11.01**	**73.22**
		1.00	13.25	7.73	4	13.25	29.72	4	0.00	**73.99**
400	15	0.70	31.79	41.90	3	30.56	693.82	3	**4.02**	**93.96**
		0.85	29.51	44.19	3	28.63	1077.91	3	**3.07**	**95.90**
		1.00	18.25	9.48	4	18.25	28.67	4	0.00	**66.93**
400	30	0.70	30.69	80.98	3	29.55	947.06	3	**3.86**	**91.45**
		0.85	27.80	67.41	3	26.90	1877.53	2	**3.35**	**96.41**
		1.00	18.00	14.81	4	18.00	46.64	4	0.00	**68.25**
500	15	0.70	48.25	158.90	2	45.04	2498.70	2	**7.13**	**93.64**
		0.85	42.91	267.60	1	39.20	2714.57	1	**9.46**	**90.14**
		1.00	29.00	25.86	4	29.00	227.92	4	0.00	**88.65**
500	30	0.70	48.00	187.61	2	44.83	2692.89	1	**7.07**	**93.03**
		0.85	40.63	323.42	1	37.24	2740.50	1	**9.10**	**88.20**
		1.00	26.25	32.27	4	26.25	198.10	4	0.00	**83.71**
#Opt Found					102			85		

Table 4 Comparison between HCG and CG-MULTI algorithms on group 1 instances with $R_S = 100$

these scenarios, HCG results again to be usually faster than CG-MULTI. Indeed, CG-MULTI is faster in 2 out of 30 scenarios (see lines 1 and 4), however in these cases the time gap is lower than 0.3 seconds. In the other 28 scenarios, HCG is up to 97.32% faster than CG-MULTI (see line 27), in 20 of them the time gap is greater than 70%, and 13 times HCG is one order of magnitude faster.

5.5 Comparisons on group 2 dataset

In the last part of this analysis, we briefly compare ECG and HCG with the CG-EXACT and CG-MULTI algorithms presented by [13] on the group 2 dataset. Results for all the procedures are reported in Table 6, where each line refers to one of the 25 instances in this dataset. We recall that for all of these instances $|T| = 15$ and $\alpha = 1$.

For this group of instances, the optimal solution is always found by each of the 4 algorithms. The *Opt LT* column reports the optimal solution vales for each instance. The requested time in seconds for each algorithm is reported in the

Group 1: $R_S = 125$										
			HCG			CG-MULTI			% GAP	
$\|S \setminus \{s_0\}\|$	$\|T\|$	α	LT	Time	#Opt	LT	Time	#Opt	LT	Time
100	15	0.70	7.00	0.89	4	7.00	0.84	4	0.00	-5.95
		0.85	6.88	1.04	4	6.88	4.31	4	0.00	**75.87**
		1.00	4.75	0.82	4	4.75	2.07	4	0.00	**60.39**
100	30	0.70	7.00	0.91	4	6.75	0.68	3	**3.70**	-33.82
		0.85	6.79	1.52	4	6.78	9.56	3	**0.15**	**84.10**
		1.00	4.75	1.52	4	4.71	4.94	3	**0.85**	**69.23**
200	15	0.70	16.25	2.92	4	16.00	8.88	3	**1.56**	**67.12**
		0.85	15.75	3.46	4	15.00	19.70	2	**5.00**	**82.44**
		1.00	13.00	3.33	4	12.50	36.06	3	**4.00**	**90.77**
200	30	0.70	16.25	3.01	4	16.25	6.61	4	0.00	**54.46**
		0.85	16.25	13.18	3	15.74	210.15	2	**3.24**	**93.73**
		1.00	11.75	3.85	4	11.75	90.65	4	0.00	**95.75**
300	15	0.70	18.25	5.89	4	18.25	6.22	4	0.00	**5.31**
		0.85	18.25	5.93	4	18.25	13.22	4	0.00	**55.14**
		1.00	16.75	5.86	4	16.75	23.10	4	0.00	**74.63**
300	30	0.70	18.25	6.41	4	18.25	6.59	4	0.00	**2.73**
		0.85	18.25	6.83	4	18.25	18.23	4	0.00	**62.53**
		1.00	16.00	7.71	4	15.75	27.25	3	**1.59**	**71.71**
400	15	0.70	34.22	32.31	3	32.86	908.98	2	**4.14**	**96.45**
		0.85	32.20	42.09	3	31.06	939.45	3	**3.67**	**95.52**
		1.00	24.88	14.09	4	24.88	208.49	4	0.00	**93.24**
400	30	0.70	33.62	65.79	3	31.76	748.26	2	**5.86**	**91.21**
		0.85	30.22	67.50	3	29.08	961.71	3	**3.92**	**92.98**
		1.00	22.38	16.81	4	22.38	149.90	4	0.00	**88.79**
500	15	0.70	54.35	139.91	2	50.51	1901.83	2	**7.60**	**92.64**
		0.85	48.89	173.34	2	45.58	2569.32	2	**7.26**	**93.25**
		1.00	37.75	50.51	4	36.82	1881.92	2	**2.53**	**97.32**
500	30	0.70	54.67	197.51	2	50.51	1892.06	2	**8.24**	**89.56**
		0.85	46.87	225.80	2	43.07	2713.86	1	**8.82**	**91.68**
		1.00	35.50	86.84	4	34.47	1883.25	2	**2.99**	**95.39**
#Opt Found					107			91		

Table 5 Comparison between HCG and CG-MULTI algorithms on group 1 instances with $R_S = 125$

related column. The last row contains the average computational time for each of the 4 approaches. The results clearly show that the group 2 instances are easier to solve with respect to those belonging to group 1. It has to be noted that for one of the instances with 50 sensors, namely the one corresponding to line 5 in the table, the authors of [13] reported to have found an optimal solution with value 4.66 instead of 4.67. The same also happened in the work that originally presented this set of instances, namely [24]. By having access to the original data collected for the experiments reported in [24], we were able to confirm that for this work the reported value was indeed wrong, and due to a rounding error or a typographical mistake. Hence we assumed the same to be also true for [13]. It follows that the comparison on the group 2 instances is essentially focused on the performance of the algorithms. Regarding the exact algorithms, ECG is faster than CG-EXACT in 24 out of 25 instances, requiring a computational time that is always lower than 10 seconds. On the other hand, CG-EXACT requires up to 95 seconds. The average computational time of ECG is equal to 1.58 seconds, while for CG-EXACT it is equal to 12.7 seconds. The performances of the two heuristic algorithms appear to be more similar. Indeed, CG-MULTI results faster

		Exact Approaches		Heuristic Approaches	
$\|S \setminus \{s_0\}\|$	**Opt LT**	**ECG Time**	**CG-EXACT Time**	**HCG Time**	**CG-MULTI Time**
50	2.00	**0.25**	0.51	0.31	**0.04**
	2.50	**0.40**	0.73	0.55	**0.19**
	3.00	**0.30**	0.45	0.27	**0.05**
	4.00	**0.28**	0.42	0.39	**0.03**
	4.67	1.04	**0.71**	0.62	**0.37**
75	7.00	**1.17**	1.61	0.83	**0.64**
	4.00	**0.39**	0.88	0.42	**0.07**
	3.00	**0.44**	1.15	**0.39**	0.51
	7.00	**0.58**	1.27	**0.44**	0.70
	7.00	**0.62**	1.37	**0.88**	1.01
100	10.00	**1.21**	6.31	**1.23**	2.59
	7.00	**0.65**	2.13	0.66	**0.27**
	7.00	**0.69**	1.72	0.45	**0.12**
	9.00	**0.81**	2.42	0.58	**0.31**
	8.00	**0.62**	3.45	0.63	**0.26**
150	17.00	**2.95**	87.84	**3.25**	54.83
	14.00	**1.37**	15.12	**1.62**	8.38
	16.00	**2.25**	14.45	**2.23**	5.77
	13.00	**1.40**	5.70	1.25	**0.57**
	14.00	**1.35**	5.84	1.26	**0.76**
200	25.00	**9.02**	95.04	**7.99**	78.00
	20.00	**3.03**	11.54	**2.91**	10.97
	19.00	**3.96**	17.12	**3.28**	15.24
	18.00	**2.62**	32.68	2.39	**2.08**
	19.00	**2.13**	7.00	**1.88**	2.96
Avg. Time		1.58	12.70	1.47	7.47

Table 6 Comparison of exact and heuristic approaches on the group 2 dataset

than HCG for 14 out of 25 instances. However in the worst case, that occurs for both heuristics when solving one of the instances with 200 sensors (see line 21), CG-MULTI requires 78 seconds, while HCG only requires 7.99 seconds. Moreover, the average computational time over the whole group of instances is equal to 1.47 seconds for HCG and 7.46 seconds for CG-MULTI. Actually, the computational times for the first 15 instances can be considered negligible, since they are almost always lower then 1 second for both heuristics. By restricting the comparison to the largest instances with at least 150 sensors, where the computational times are mostly higher than 1 second, we can see that HCG is faster than CG-MULTI for 7 out of 10 instances, with computational time gaps between 36.49% and 94.07%.

6 Conclusions

In this work we faced the Connected Maximum Lifetime Problem with either complete or partial target coverage requirements. We developed an exact and a heuristic algorithm, both based on column generation. The main contribution of our work is the proposal of new ideas for the resolution of the subproblem. Namely,

we developed an efficient genetic algorithm embedding a Steiner Tree heuristic, a novel ILP formulation with a reduced number of integer variables, and a modification to the column generation scheme that involves the premature interruption of the subproblem resolution as soon as a profitable column is found. The algorithms developed using these ideas were proven experimentally to outperform previous approaches presented in the literature for the problem.

Future research efforts will focus on adapting our techniques to other problems in the same research field, such as the Maximum Lifetime Problem in multi-role sensor networks, in which different energy consumption rates are considered depending on the role adopted by the sensors during the monitoring activity.

References

1. J. Ai and A. A. Abouzeid. Coverage by directional sensors in randomly deployed wireless sensor networks. *Journal of Combinatorial Optimization*, 11(1):21–41, 2006.
2. H. Alemdar and C. Ersoy. Wireless sensor networks for healthcare: a survey. *Computer Networks*, 54(15):2688–2710, 2010.
3. A. Alfieri, A. Bianco, P. Brandimarte, and C. F. Chiasserini. Maximizing system lifetime in wireless sensor networks. *European Journal of Operational Research*, 181(1):390–402, 2007.
4. Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.
5. W. Awada and 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*, pages 53–60, 2006.
6. Y. Cai, W. Lou, M. Li, and X.-Y. Li. Energy efficient target-oriented scheduling in directional sensor networks. *IEEE Transactions on Computers*, 58(9):1259–1274, 2009.
7. I. Cardei and M. Cardei. Energy-efficient connected-coverage in wireless sensor networks. *International Journal of Sensor Networks*, 3(3):201–210, 2008.
8. M. Cardei, M. T. Thai, Y. Li, and W. Wu. Energy-efficient target coverage in wireless sensor networks. In *Proceedings of the 24th conference of the IEEE Communications Society*, volume 3, pages 1976–1984, 2005.
9. M. Cardei, J. Wu, and M. Lu. Improving network lifetime using sensors with adjustable sensing ranges. *International Journal of Sensor Networks*, 1(1-2):41–49, 2006.
10. F. Carrabs, R. Cerulli, C. D'Ambrosio, M. Gentili, and A. Raiconi. Maximizing lifetime in wireless sensor networks with multiple sensor families. *Computers & Operations Research*, 60:121–137, 2015.
11. F. Carrabs, R. Cerulli, C. D'Ambrosio, and A. Raiconi. A hybrid exact approach for maximizing lifetime in sensor networks with complete and partial coverage constraints. To appear in Journal of Network and Computer Applications, 2015.
12. F. Castaño, E. Bourreau, N. Velasco, A. Rossi, and M. Sevaux. Exact approaches for lifetime maximization in connectivity constrained wireless multi-role sensor networks. *European Journal of Operational Research*, 241(1):28–38, 2015.
13. F. Castaño, A. Rossi, M. Sevaux, and N. Velasco. A column generation approach to extend lifetime in wireless sensor networks with coverage and connectivity constraints. *Computers & Operations Research*, 52(B):220–230, 2014.
14. R. Cerulli, R. De Donato, and 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):58–66, 2012.
15. R. Cerulli, M. Gentili, and A. Raiconi. Maximizing lifetime and handling reliability in wireless sensor networks. *Networks*, 64(4):321–338, 2014.
16. L. Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.
17. K. Deschinkel. A column generation based heuristic for maximum lifetime coverage in wireless sensor networks. In *SENSORCOMM 11, 5th Int. Conf. on Sensor Technologies and Applications*, volume 4, pages 209 – 214, 2011.

18. A. Dhawan, C. T. Vu, A. Zelikovsky, Y. Li, and S. K. Prasad. Maximum lifetime of sensor networks with adjustable sensing range. In *Proceedings of the Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, pages 285 – 289, 2006.
19. C. Duin and S. Voss. Steiner tree heuristics - a survey. In H. Dyckhoff, U. Derigs, M. Salomon, and H. C. Tijms, editors, *Operations Research Proceedings 1993. Papers of the 22nd Annual Meeting of DGOR in Cooperation with NSOR*, pages 485–496. Springer-Verlag, 1994.
20. M. Gentili and A. Raiconi. $\alpha-$coverage to extend network lifetime on wireless sensor networks. *Optimization Letters*, 7(1):157–172, 2013.
21. Y. Gu, Y. Ji, and B. Zhao. Maximize lifetime of heterogeneous wireless sensor networks with joint coverage and connectivity requirement. In *EmbeddedCom-09, 8th International Conference on Embedded Computing*, pages 226–231, 2009.
22. F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*. North-Holland, Amsterdam, 1992.
23. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.
24. A. Raiconi and M. Gentili. Exact and metaheuristic approaches to extend lifetime and maintain connectivity in wireless sensors networks. In J. Pahl, T. Reiners, and S. Voss, editors, *Network Optimization*, volume 6701 of *Lecture Notes in Computer Science*, pages 607–619. Springer, Berlin/Heidelberg, 2011.
25. P. Rawat, K. D. Singh, H. Chaouchi, and J. M. Bonnin. Wireless sensor networks: a survey on recent developments and potential synergies. *The Journal of Supercomputing*, 68(1):1–48, 2014.
26. A. Rossi, A. Singh, and M. Sevaux. An exact approach for maximizing the lifetime of sensor networks with adjustable sensing ranges. *Computers & Operations Research*, 39(12):3166–3176, 2012.
27. A. Rossi, A. Singh, and M. Sevaux. Lifetime maximization in wireless directional sensor network. *European Journal of Operational Research*, 231(1):229–241, 2013.
28. C. Wang, M. T. Thai, Y. Li, F. Wang, and W. Wu. Minimum coverage breach and maximum network lifetime in wireless sensor networks. In *Proceedings of the IEEE Global Telecommunications Conference*, pages 1118–1123, 2007.
29. Q. Zhao and M. Gurusamy. Lifetime maximization for connected target coverage in wireless sensor networks. *IEEE/ACM Transactions on Networking*, 16(6):1378–1391, 2008.