RAIRO-Oper. Res. 51 (2017) 607–625
DOI: 10.1051/ro/2017032

RAIRO Operations Research
www.rairo-ro.org

EXACT AND HEURISTIC APPROACHES FOR THE MAXIMUM LIFETIME PROBLEM IN SENSOR NETWORKS WITH COVERAGE AND CONNECTIVITY CONSTRAINTS

Francesco Carrabs[1], Raffaele Cerulli[1], Ciriaco D'Ambrosio[1] and Andrea Raiconi[1]

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.

Mathematics Subject Classification. 90C05, 90C06, 90C59.

Received January 26, 2017. Accepted April 25, 2017.

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 micro-electro-mechanical systems (MEMS) and wireless communications allowed them do be adaptable to diverse scenarios, including environmental and traffic monitoring, healthcare applications, and recent trends such as Internet of Things among others (refer for instance to [2, 4, 6, 31]). 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, or R_S.

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

Keywords. Maximum lifetime, wireless sensor network, column generation, genetic algorithm, steiner tree, partial coverage.

[1] Department of Mathematics, University of Salerno, via Giovanni Paolo II, 132, 84084 Fisciano, Italy.
`[fcarrabs; raffaele; cdambrosio; araiconi]@unisa.it`

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 [9], 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 [23]. 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 [12, 26, 35], heterogeneous networks [5, 11], sensors with adjustable sensing ranges [10, 19, 20, 24, 32] or with angular, orientable sensing ranges, such as video cameras [1, 7, 33], among others.

A significant amount of research has been also spent on WSN problems that consider connectivity issues [3, 8, 13, 14, 16, 17, 27, 30, 36]. 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, or R_C) is considered for each sensor, defining which other sensors are close enough to communicate directly with it.

In particular, in [30] 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 [17], 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 [23] 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 [30].

In this work we also focus on the α-CMLP problem, presenting a heuristic and an exact approach based on the column generation technique. 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 new ILP formulation and a modification to the resolution 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 have very competitive performances.

The rest of the work has the following structure. A formal definition of the problem is provided in Section 2. The column generation scheme and our proposed ILP subproblem formulation are described in Section 3. Our genetic algorithm and its integration within the column generation framework are presented in Section 4. Computational results are described in Section 5, followed by conclusions and future research perspectives in Section 6.

2. Problems definition and mathematical formulation

Let $T = \{t_1, \dots, t_n\}$ be the set of target points of interest, and let $S = \{s_0, s_1, \dots, s_m\}$ 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 $S \backslash \{s_0\}$ 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 $s_i \in S \backslash \{s_0\}$, 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 $S' \subseteq S \backslash \{s_0\}$, 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 S, 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).

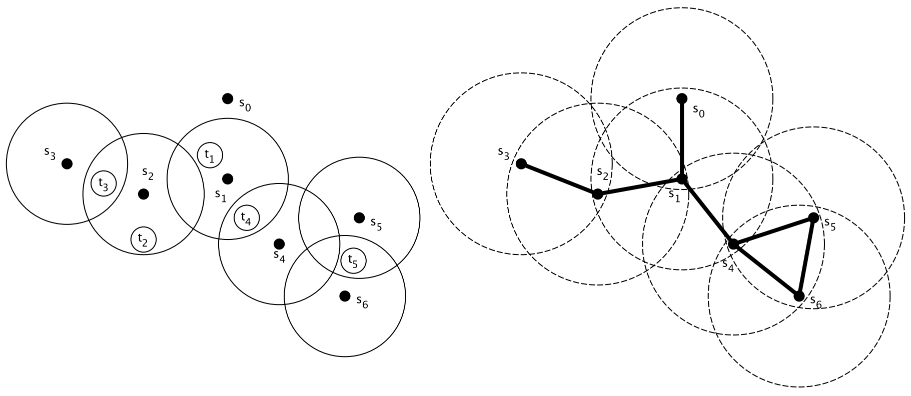

Figure 1. A simple WSN and its connectivity graph.

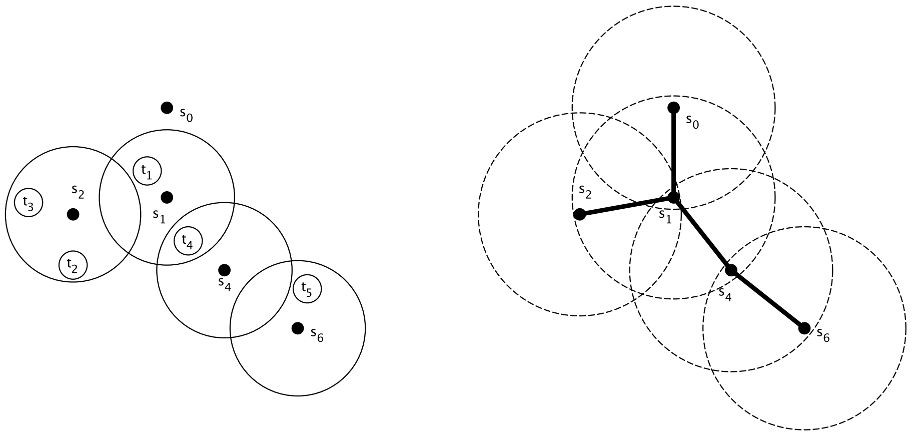

FIGURE 2. A feasible cover and its communication tree.

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, 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. Any spanning tree of this subgraph, rooted at s_0, is called *communication tree* and defines, for each sensor s_i in C, a path of communication links between s_i and s_0 in G'. 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 as *simple*.

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.

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

$$[\mathbf{P}] \max \sum_{C_p \in \{C_1,\ldots,C_M\}} w_p \tag{2.1}$$

s.t.

$$\sum_{C_p \in \{C_1,\ldots,C_M\}} a_{ip} w_p \leq b_i \qquad \forall s_i \in S \setminus \{s_0\} \tag{2.2}$$

$$w_p \geq 0 \qquad \forall p = 1,\ldots,M. \tag{2.3}$$

In constraints (2.2), for each sensor $s_i \in S \setminus \{s_0\}$ 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 s_i. Therefore the constraints (2.2) enforce the respect of the battery life limitations, while the objective function (2.1) maximizes the sum of the activation times and thus the network lifetime.

In practice, solving formulation $[\mathbf{P}]$ directly is not possible due to the difficulty of explicitly enumerating all feasible covers, whose number grows 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, in this work we propose a column generation scheme, embedding an efficient and effective genetic algorithm.

The features of our algorithm are presented in Sections 3 and 4.

3. Column generation for 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. Hence, we define a column to be attractive if the related variable is currently nonbasic and has a negative reduced cost. If such a column can be found, the algorithm iterates by adding it to the previous 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]**, each column represents a feasible cover, therefore we will also refer by extension to attractive covers. In more detail, let π_i, $\forall s_i \in S \backslash \{s_0\}$, be the dual prices associated with each constraint in the master problem (that is, to each sensor); a given cover C_p will be attractive if $\sum_{s_i \in C_p \backslash \{s_0\}} \pi_i - c_p < 0$, where c_p is the cost coefficient of w_p in the objective function of **[P]**. Note that the c_p coefficients are all equal to 1; hence, in our subproblem, we chose to find the cover C_p that minimizes $\sum_{s_i \in C_p \backslash \{s_0\}} \pi_i$. If such a quantity is greater or equal than 1, 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. For further insights on the use of Column Generation for linear and integer programming, the reader can refer to [22].

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 $(s_i, s_j) \in E$. We consider the following formulation for the subproblem:

$$[\mathbf{SP}] \qquad \min \sum_{s_i \in S \backslash \{s_0\}} \pi_i y_i \tag{3.1}$$

s.t.

$$\sum_{(s_0, s_i) \in E^d} f_{0i} = \sum_{s_i \in S \backslash \{s_0\}} y_i \tag{3.2}$$

$$\sum_{(s_i, s_j) \in E^d} f_{ij} - \sum_{(s_j, s_i) \in E^d} f_{ji} = y_j \qquad \forall s_j \in S \backslash \{s_0\} \tag{3.3}$$

$$y_i \le \sum_{(s_j, s_i) \in E} f_{ji} \le (|S| - 1) y_i \qquad \forall s_i \in S \backslash \{s_0\} \tag{3.4}$$

$$\sum_{s_i \in S \backslash \{s_0\}} \delta_{ki} y_i \ge z_k \qquad \forall t_k \in T \tag{3.5}$$

$$\sum_{t_k \in T} z_k \ge T_\alpha \tag{3.6}$$

$$f_{ij} \in \mathbb{Z}^+ \cup \{0\} \qquad \forall (s_i, s_j) \in E^d \tag{3.7}$$

$$y_i \in \{0, 1\} \qquad \forall s_i \in S \backslash \{s_0\} \tag{3.8}$$

$$z_k \in \{0, 1\} \qquad \forall t_k \in T. \tag{3.9}$$

The model uses flow constraints to make sure that the subgraph induced by the produced cover contains at least a communication tree rooted at the base station, and therefore that is is connected. Indeed, each edge $(s_i, s_j) \in E^d$ traversed by positive flow f_{ij} in the solution belongs to the communication tree. Binary variables y_i, $\forall s_i \in S \backslash \{s_0\}$, and z_k, $\forall t_k \in T$, state whether each sensor s_i belongs to the cover and whether each target t_k is covered by it, respectively.

The objective function (3.1) aims at finding the nonbasic cover with minimum reduced cost, as discussed. Constraint (3.2) imposes the amount of flow produced by the base station to be equal to the number of activated sensors. Constraints (3.3) are flow conservation constraints stating that, for each sensor $s_j \in S \backslash \{s_0\}$, the difference between ingoing and outgoing flow is equal to 1 if the sensors belongs the new cover, and 0 otherwise. Constraints (3.4) impose all sensors with ingoing positive flow to be activated in the cover. Finally, constraints (3.5) bind the y_i and the z_k variables, while constraints (3.6) ensure that the new cover provides coverage for the requested number of targets.

In [17], the authors proposed a similar model for the subproblem, using an additional set of binary variables to select arcs belonging to the communication tree; we were able to avoid using these variables by introducing constraints (3.4).

The main disadvantage of such column generation approaches is that the subproblem is NP-Hard (see [17]) and therefore, 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, we interrupt the exact resolution of the **[SP]** ILP formulation as soon as an attractive cover is found. These features allowed us to develop an effective CG approach, as will be shown in Section 5.

4. A genetic algorithm to address the 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.

4.1. GA overall structure

Our genetic algorithm (GA) is used in the column generation scheme to solve the subproblem 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 algorithm 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 3b and 3c illustrate how the GA procedure is integrated within HCG and ECG, respectively.

Genetic algorithms are population-based metaheuristics that, as other methods belonging to the class of evolutionary algorithms, use techniques that draw inspiration from biological evolution concepts, including

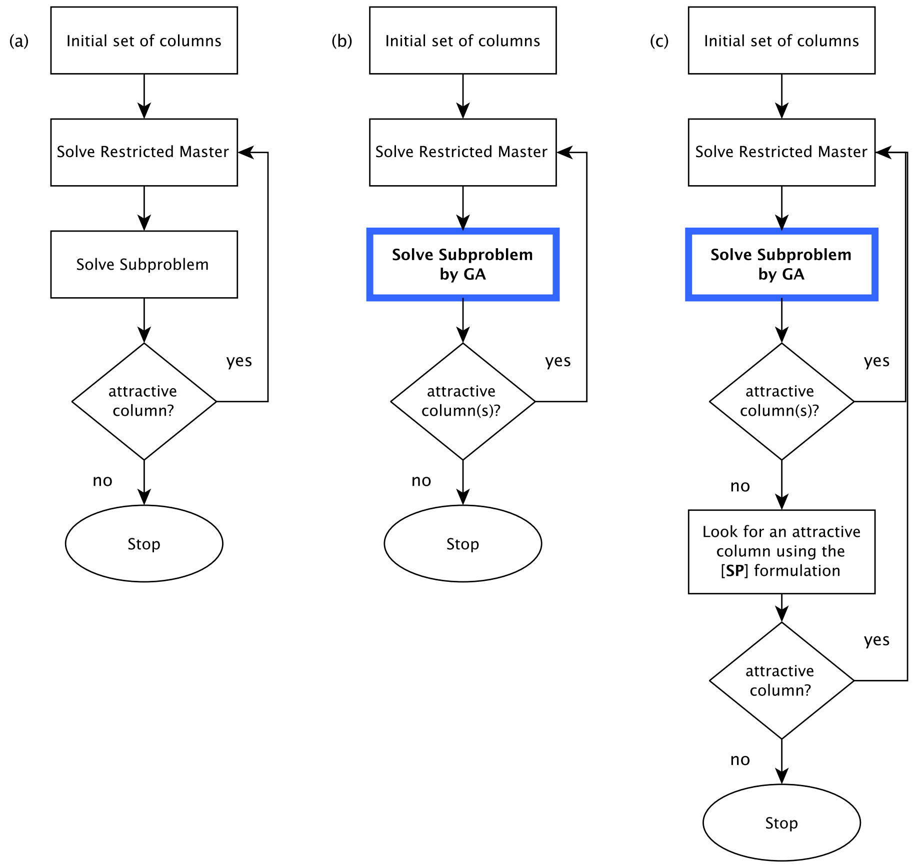

FIGURE 3. Classical CG scheme (a). HCG scheme (b). ECG scheme (c).

natural selection, reproduction and mutation. Given a starting population of individuals representing problem solutions, usually defined as *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 [21].

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 [29]) 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.

Algorithm 1: GA pseudocode.

```
Input: $WSN = (S,T)$, $\alpha \in (0,1]$, $G = (S,E)$, $DP$;
Output: Set of feasible covers;
1  $P \leftarrow initP(WSN, G, \alpha, DP)$;
2  $BestFit \leftarrow bestFitness(P, DP)$;
3  $criteria \leftarrow setCriteria(MaxIT, MaxDUP)$;
4  $SPL \leftarrow evaluateShortestPaths(G, DP)$;
5  while $check(criteria)$ do
6      $(\zeta_{p1}, \zeta_{p2}) \leftarrow tournament(P, DP)$;
7      $\zeta \leftarrow crossover(\zeta_{p1}, \zeta_{p2})$;
8      $\zeta \leftarrow mutation(\zeta)$;
9      $\zeta \leftarrow coverFeasibilityOperator(\zeta, WSN, \alpha)$;
10     $\zeta \leftarrow connectFeasibilityOperator(\zeta, G, SPL, DP)$;
11     $\zeta \leftarrow redundancyRemovalOperator(\zeta, WSN, G)$;
12     if $\zeta \notin P$ then
13         $P \leftarrow insert(\zeta)$;
14         if $fitness(\zeta, DP) \geq BestFit$ then
15             $update(criteria)$;
16         else
17             $BestFit \leftarrow fitness(\zeta, DP)$;
18     else
19         $update(criteria)$;
20 $Chromos \leftarrow$ chromosomes with fitness $< 1$;
21 return $Chromos$;
```

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 Sect. 4.3); the obtained *child* ζ is then mutated (Sect. 4.4). Since these operations do not guarantee the feasibility of ζ, 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 (Sect. 4.5). The final modifications applied to ζ are made by an operator that checks if some of its sensors can be switched off while preserving feasibility (Sect. 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 ζ in our GA algorithm is internally represented as a binary vector of length $|S|$. The element in the ith position of the vector, with $i \in \{0, 1 \ldots, m\}$, is called the ith *gene* of ζ and is denoted by $\zeta[i]$. The gene $\zeta[i]$ is associated to $s_i \in S$, and it is equal to 1 if and only if s_i is activated in ζ. In this case, we say that ζ 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, $\zeta[0]$ must be equal to 1 in each feasible chromosome ζ. It is also easy to see that, ruling out the $\zeta[0]$ 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 (3.1) 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 ζ_{p1}. The same procedure is used to select the second parent ζ_{p2}, making sure that ζ_{p1} is different from ζ_{p2}.

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

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 ζ whose value is identical in the parents, if it exists, and changes its value ($\zeta[0]$ 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 ζ 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 ζ 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 ζ 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 ζ 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 ζ 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 *terminals*, while those in $V \setminus V'$ are the *steiner* vertices.

In our case, we mark the base station and the sensors activated in ζ as terminals, 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). By solving the Steiner tree problem, we intend to identify some additional steiner sensors that, once added to ζ, guarantee its induced subgraph to contain at least a tree and therefore to be connected.

The Steiner Tree problem is well-known to be NP-hard [28] 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 [25]. In this work, we use a fast and effective construction heuristic that the authors call CHINS (Cheapest Insertion), originally proposed in [34], that works as follows:

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

1. Initialize the solution $\mathcal{T}$ with a single arbitrary terminal $i$;
2. Repeat:
   (a) Find the shortest path $\mathcal{P}$ in $G$ between any terminal $j$ 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 terminals.
```

An improvement of CHINS called CHINS-Q iterates the procedure $|V'|$ times, selecting every time a different terminal 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 ζ, an iteration is performed. In each iteration, we first build a vector ζ' with $\zeta'[i] = 1$ and all other genes equal to 0. Then, among all the shortest paths between an s_j contained in ζ that is not is ζ' and an s_p in ζ', the procedure individuates the one with lowest weight and activates in ζ' all the elements belonging to this shortest path. This operation is repeated until ζ' contains all the elements of ζ and thus it is a feasible chromosome.

Finally, the best solution found is returned.

4.6. Redundancy removal operator

The ζ chromosome resulting from the feasibility operators is guaranteed to be feasible. However, since such operators may cause the cover to be non-simple, 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 has 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 benchmark instances proposed in [17]. These instances 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 $\{100, 125\}$. A sensor s_i can cover a target t_j if their euclidean distance is not greater than R_S, and there exists a communication link between two sensors s_i and s_j if their euclidean distance is not greater than R_C. Furthermore, all sensors have the same battery life, normalized to 1 time unit. For each combination of parameters four different instances were generated, that together represent a *scenario*. Therefore, there are in total 240 test instances, that compose 60 scenarios.

5.1. 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.

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 chromosomes. 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 (Sect. 4.7) was chosen to be equal to 100, while the chosen values for the two parameters $MaxIT$ and $MaxDUP$, regulating the termination criteria (Sect. 4.1) are 2000 and 100, respectively. Similarly to [17], a 3600 s 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.

5.2. Impact of the premature subproblem interruptions

As introduced in Section 3, in this work we modified the column generation scheme by interrupting the [**SP**] resolution as soon as an attractive cover is found. This operation is carried out by invoking an appropriate CPLEX callback.

In order to evaluate the impact of the premature subproblem interruption, we performed a preliminary test phase, in which we compared the traditional column generation approach (*i.e.* the one represented in Fig. 3a) with and without the callback invocation during the [**SP**] resolution. Note that these approaches do not make use of the genetic algorithm. In the following, the two procedures are referred to as CG-Call and CG-Std, respectively. To guarantee a fair comparison, both algorithms were initialized with the same restricted set of columns. The results of this comparison are presented in Table 1.

We performed these tests on the instances with at most 200 sensors, since CG-Std often reaches the time limit on larger instances. 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 $|S\backslash\{s_0\}|$, $|T|$ and α columns report the instances characteristics. The LT column reports the lifetime values expressed in time units found within the time limit, the column *Time* shows the computational time in seconds and the *#Opt* column reports the number of optimal solutions found for each scenario. The last column, under the *% Gap* heading, reports the percentage gap between lifetimes. In more detail, let LT(Alg) be the average lifetime value reported by a given *Alg* procedure on a considered scenario. The LT gaps are computed as

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

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

	$\|S\backslash\{s_0\}\|$	$\|T\|$	α	CG-Std LT	CG-Std Time	CG-Std #Opt	CG-Call LT	CG-Call Time	CG-Call #Opt	%GAP LT
$R_S = 100$	100	15	0.70	6.81	929.37	3	6.88	43.94	4	**1.03**
			0.85	6.43	1868.93	2	6.64	385.85	4	**3.27**
			1.00	4.00	1.79	4	4.00	1.92	4	0.00
	100	30	0.70	6.94	1207.31	3	7.00	107.10	4	**0.86**
			0.85	6.13	2007.75	2	6.57	1830.36	2	**7.18**
			1.00	3.99	904.07	3	4.00	8.17	4	**0.25**
	200	15	0.70	15.44	1123.95	3	15.99	929.94	3	**3.56**
			0.85	14.42	2442.94	2	15.28	1137.25	3	**5.96**
			1.00	10.25	95.54	4	10.25	29.37	4	0.00
	200	30	0.70	15.37	1187.56	3	15.77	1002.00	3	**2.60**
			0.85	13.39	2727.10	1	14.65	2111.24	2	**9.41**
			1.00	8.75	73.95	4	8.75	30.81	4	0.00
$R_S = 125$	100	15	0.70	7.00	21.64	4	7.00	5.61	4	0.00
			0.85	6.78	1838.72	2	6.88	65.46	4	**1.47**
			1.00	4.75	577.59	4	4.75	13.70	4	0.00
	100	30	0.70	7.00	12.54	4	7.00	13.66	4	0.00
			0.85	6.69	1832.81	2	6.79	862.44	4	**1.49**
			1.00	4.47	1798.70	2	4.75	40.84	4	**6.26**
	200	15	0.70	16.17	972.32	3	16.25	309.44	4	**0.49**
			0.85	15.36	1037.80	3	15.75	509.25	4	**2.54**
			1.00	12.57	1797.85	2	13.00	335.90	4	**3.42**
	200	30	0.70	16.25	964.08	4	16.25	413.68	4	0.00
			0.85	14.77	2192.41	2	15.54	1192.11	3	**5.21**
			1.00	11.23	930.28	3	11.75	678.70	4	**4.63**

When the LT gap is positive, it means that CG-Call found a better solution than CG-Std on the considered scenario, and the value is marked in bold into the table.

Results in Table 1 show that CG-Call is faster than CG-Std in 22 out of 24 scenarios, with the difference being less than 2 s in the 2 remaining scenarios. Indeed, 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. Finally, CG-Call does not find certified optimal solutions for 8 out of 96 instances, as opposed to CG-Std that does not find 27 of them.

Regarding the effectiveness of the two approaches, CG-Call finds better solutions than CG-Std in 17 out of 24 scenarios, with an LT gap that grows up to 9.41%. On the remaining 7 scenarios, the solutions are the same.

Due to these results, in all the remaining experiments we always used the premature interruption technique.

5.3. Exact algorithms comparison

Tables 2 and 3 contain the comparisons between our exact algorithm ECG and the CG-EXACT algorithm, introduced in [17], 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. In addition to lifetime values, we also compute gaps for the computational times of the two methods. Let $Time(Alg)$ be the average computational time spent by a procedure Alg to solve a given scenario; the Time

TABLE 2. Comparison between ECG and CG-EXACT algorithms on instances with $R_S = 100$.

$R_S = 100$										
				ECG			CG-EXACT		% GAP	
$\|S\backslash\{s_0\}\|$	$\|T\|$	α	LT	Time	#Opt	LT	Time	#Opt	LT	Time
100	15	0.70	6.88	1.17	4	6.88	8.78	4	0.00	**86.67**
		0.85	6.64	623.86	4	6.64	921.47	4	0.00	**32.30**
		1.00	4.00	0.78	4	4.00	1.80	4	0.00	**56.67**
100	30	0.70	7.00	1.77	4	7.00	6.54	4	0.00	**72.94**
		0.85	6.60	1805.78	2	6.57	1922.59	2	**0.46**	**6.08**
		1.00	4.00	1.67	4	4.00	4.06	4	0.00	**58.87**
200	15	0.70	16.25	5.00	4	16.25	414.98	4	0.00	**98.80**
		0.85	15.51	905.52	3	15.42	941.60	3	**0.58**	**3.83**
		1.00	10.25	3.08	4	10.25	12.69	4	0.00	**75.73**
200	30	0.70	16.25	4.95	4	16.25	128.92	4	0.00	**96.16**
		0.85	15.42	909.91	3	15.35	1514.62	3	**0.46**	**39.92**
		1.00	8.75	3.33	4	8.75	19.80	4	0.00	**83.18**
300	15	0.70	18.25	7.80	4	18.25	34.72	4	0.00	**77.53**
		0.85	18.25	8.66	4	18.25	91.10	4	0.00	**90.49**
		1.00	15.00	7.65	4	15.00	86.26	4	0.00	**91.13**
300	30	0.70	18.25	8.51	4	18.25	47.93	4	0.00	**82.24**
		0.85	18.25	10.39	4	18.25	104.34	4	0.00	**90.04**
		1.00	13.25	7.64	4	13.25	48.34	4	0.00	**84.20**
400	15	0.70	31.90	910.45	3	30.68	999.22	3	**3.98**	**8.88**
		0.85	29.64	917.58	3	28.66	1151.16	3	**3.42**	**20.29**
		1.00	18.25	10.99	4	18.25	95.70	4	0.00	**88.52**
400	30	0.70	30.74	924.52	3	29.55	1007.00	3	**4.03**	**8.19**
		0.85	27.99	953.23	3	26.90	1907.09	2	**4.05**	**50.02**
		1.00	18.00	18.09	4	18.00	125.38	4	0.00	**85.57**
500	15	0.70	48.67	1827.31	2	45.04	2554.30	2	**8.06**	**28.46**
		0.85	43.26	2713.20	1	39.20	2742.29	1	**10.36**	**1.06**
		1.00	29.00	30.68	4	29.00	335.83	4	0.00	**90.86**
500	30	0.70	48.39	1834.39	2	44.83	2748.99	1	**7.94**	**33.27**
		0.85	41.04	2721.35	1	37.24	2768.34	1	**10.20**	**1.70**
		1.00	26.25	32.69	4	26.25	308.25	4	0.00	**89.39**
#Opt Found					102			100		

gaps are computed as

$$100 \times \frac{Time(\text{CG-EXACT}) - Time(\text{ECG})}{Time(\text{CG-EXACT})}.$$

Again, the cases in which ECG performs better than CG-EXACT are highlighted in bold. The final 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 0.46% to 10.36%, while on the remaining scenarios the solution is the same. Moreover, ECG solves to optimality 2 instances more than CG-EXACT. Regarding the computational time performances, ECG is always faster than CG-EXACT and, in the 19 scenarios in which both algorithms find

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

$R_S = 125$										
			ECG			CG-EXACT			% GAP	
$\lvert S\backslash\{s_0\}\rvert$	$\lvert T\rvert$	α	LT	Time	#Opt	LT	Time	#Opt	LT	Time
100	15	0.70	7.00	0.91	4	7.00	1.79	4	0.00	**49.16**
		0.85	6.88	1.27	4	6.88	8.37	4	0.00	**84.83**
		1.00	4.75	0.95	4	4.75	3.62	4	0.00	**73.76**
100	30	0.70	7.00	1.02	4	7.00	3.05	4	0.00	**66.56**
		0.85	6.79	1.98	4	6.79	327.39	4	0.00	**99.40**
		1.00	4.75	1.45	4	4.75	91.41	4	0.00	**98.41**
200	15	0.70	16.25	3.59	4	16.25	19.56	4	0.00	**81.65**
		0.85	15.75	3.71	4	15.75	31.46	4	0.00	**88.21**
		1.00	13.00	3.96	4	13.00	46.56	4	0.00	**91.49**
200	30	0.70	16.25	3.15	4	16.25	19.34	4	0.00	**83.71**
		0.85	16.25	36.44	4	16.09	925.98	3	**0.99**	**96.06**
		1.00	11.75	4.38	4	11.75	96.81	4	0.00	**95.48**
300	15	0.70	18.25	8.80	4	18.25	27.97	4	0.00	**68.54**
		0.85	18.25	7.32	4	18.25	40.07	4	0.00	**81.73**
		1.00	16.75	6.19	4	16.75	52.10	4	0.00	**88.12**
300	30	0.70	18.25	7.51	4	18.25	29.23	4	0.00	**74.31**
		0.85	18.25	8.57	4	18.25	50.03	4	0.00	**82.87**
		1.00	16.00	8.64	4	16.00	82.83	4	0.00	**89.57**
400	15	0.70	34.27	911.88	3	33.36	991.85	3	**2.73**	**8.06**
		0.85	32.23	913.99	3	31.06	998.19	3	**3.77**	**8.44**
		1.00	24.88	16.79	4	24.88	298.96	4	0.00	**94.38**
400	30	0.70	33.60	914.87	3	32.08	956.19	3	**4.74**	**4.32**
		0.85	30.32	921.68	3	29.08	1021.31	3	**4.26**	**9.76**
		1.00	22.38	28.44	4	22.38	245.22	4	0.00	**88.40**
500	15	0.70	54.62	1819.65	2	50.51	1957.14	2	**8.14**	**7.03**
		0.85	48.99	1824.38	2	45.58	2624.69	2	**7.48**	**30.49**
		1.00	37.75	51.36	4	36.82	1937.38	2	**2.53**	**97.35**
500	30	0.70	54.85	1818.40	2	51.13	1947.69	2	**7.28**	**6.64**
		0.85	47.12	1856.93	2	43.07	2741.61	1	**9.40**	**32.27**
		1.00	35.50	87.43	4	34.47	1938.50	2	**2.99**	**95.49**
	#Opt Found				108			102		

all the optimal solutions (*i.e.* #Opt=4), the performance gap ranges from 32.30% to 98.80%. 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 9.40%. 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 49.16% to 99.40%. In 8 scenarios, ECG is an order of magnitude faster than CG-EXACT.

It has to be pointed out that the tests in [17] were run on a machine equipped with an Intel Core i5 1.6 GHz processor with 2GB of RAM (their algorithms were coded in C++, similarly to ours). While the test environment differences do not allow a completely accurate comparison, we believe that the remarkable performance gaps provide solid evidence about the competitiveness of our approach. In our opinion, this is due to the effectiveness and efficiency of our GA algorithm. Jointly with the premature **[SP]** interruptions, it is able to produce quickly most of the needed covers, avoiding whenever possible the expensive exact resolution of the subproblem. Further evidence about the good performances of the GA is provided in Section 5.4, in which we present the results of our HCG heuristic.

With respect to the considered set of instances, we can note that, for most scenarios and both algorithms, the instances with $\alpha = 1$ are the easiest to solve. It is easy to understand why requiring full coverage of the set of targets is a simplifying factor; indeed, it can reduce significantly the number of existing feasible covers, and eliminates the need to chose which targets should be covered (all z_k variables are forced to 1 by formulation **[SP]**).

Counter-intuitively, the instances with $\alpha = 0.7$ generally required less time to be solved than those with $\alpha = 0.85$, while theoretically allowing more feasible covers to exist. We believe that, in this case, the dominating factor responsible for the complexity of these instances is the need, for $\alpha = 0.85$, to cover more targets with respect to $\alpha = 0.7$. This leads to generally larger connected covers (and therefore to a greater number of choices that have to be made) whenever the subproblem has to be solved. In accordance with this intuition, we note that both algorithms find a larger number of optimal solutions in the case $R_S = 125$, in which sensors are likely to be able to individually cover more targets with respect to the case $R_S = 100$.

5.4. Heuristic algorithms comparison

In Tables 4 and 5, HCG is compared with the overall best-performing heuristic approach presented in [17], 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 $R_S = 100$. This is a very significant result since it shows that, for these scenarios, HCG was able to find all the optimal solutions certified by ECG, emphasizing the GA effectiveness. 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 (100 sensors, 30 targets, $\alpha = 0.85$; 200 sensors, 15 targets, $\alpha = 0.85$; 200 sensors, 15 targets, $\alpha = 0.85$) 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 $\alpha = 1$, in which, as previously mentioned, 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 100 sensors, 15 targets, $\alpha = 1$), however the time gap is lower than 0.15 s and can be considered negligible. In the other 29 scenarios, HCG is up to 98.05% faster than CG-MULTI (see the scenario with 200 sensors, 15 target and $\alpha = 0.7$), 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 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 for 1 instance out of 72 by HCG, and in 10 cases by CG-MULTI. The results show a lower influence of the α parameter with respect to the results reported in Table 4. For these scenarios, HCG results again to be usually faster

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

$R_S = 100$										
			HCG			CG-MULTI			% GAP	
$\|S\backslash\{s_0\}\|$	$\|T\|$	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		

than CG-MULTI. Indeed, CG-MULTI is faster in 2 out of 30 scenarios (100 sensors, 15 targets $\alpha = 0.7$; 100 sensors, 30 targets, $\alpha = 0.7$), however in these cases the time gap is lower than 0.3 s. In the other 28 scenarios, HCG is up to 97.32% faster than CG-MULTI (see the scenario with 500 sensors, 15 targets and $\alpha = 1$), in 20 of them the time gap is greater than 70%, and 13 times HCG is one order of magnitude faster.

6. Conclusions

In this work we faced the α-CMLP problem. 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

TABLE 5. Comparison between HCG and CG-MULTI algorithms on instances with $R_S = 125$.

$R_S = 125$										
			HCG			CG-MULTI			% GAP	
$\lvert S\backslash\{s_0\}\rvert$	$\lvert T\rvert$	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		

of the subproblem. Namely, we developed an efficient genetic algorithm embedding a Steiner Tree heuristic, a new ILP formulation for the subproblem, and a modification to the column generation scheme that involves the premature interruption of the exact subproblem resolution as soon as an attractive 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 with redundant connected coverage for fault-tolerant applications, or inducing bounded-degree connectivity trees (see [15, 18]) to avoid network congestion.

References

[1] J. Ai and A.A. Abouzeid, Coverage by directional sensors in randomly deployed wireless sensor networks. *J. Comb. Optim.* **11** (2006) 21–41.

[2] H. Alemdar and C. Ersoy, Wireless sensor networks for healthcare: a survey. *Comput. Netw.* **54** (2010) 2688–2710.

[3] A. Alfieri, A. Bianco, P. Brandimarte and C.F. Chiasserini, Maximizing system lifetime in wireless sensor networks. *Eur. J. Oper. Res.* **181** (2007) 390–402.

[4] L. Atzori, A. Iera and G. Morabito, The internet of things: A survey. *Comput. Netw.* **54** (2010) 2787–2805.

[5] W. Awada and M. Cardei, Energy-efficient data gathering in heterogeneous wireless sensor networks, in *Proc. of the IEEE International Conference on Wireless and Mobile Computing, Networking and Communications* (2006) 53–60.

[6] L. Bianco, C. Cerrone, R. Cerulli and M. Gentili, Locating sensors to observe network arc flows: Exact and heuristic approaches. *Comput. Oper. Res.* **46** (2014) 12–22.

[7] Y. Cai, W. Lou, M. Li and X.-Y. Li, Energy efficient target-oriented scheduling in directional sensor networks. *IEEE Trans. Comput.* **58** (2009) 1259–1274.

[8] I. Cardei and M. Cardei, Energy-efficient connected-coverage in wireless sensor networks. *Int. J. Sensor Networks* **3** (2008) 201–210.

[9] M. Cardei, M.T. Thai, Y. Li and W. Wu, Energy-efficient target coverage in wireless sensor networks. In Vol. 3 of *Proc. of the 24th conference of the IEEE Communications Society* (2005) 1976–1984.

[10] M. Cardei, J. Wu and M. Lu, Improving network lifetime using sensors with adjustable sensing ranges. *Int. J. Sensor Networks* **1** (2006) 41–49.

[11] F. Carrabs, R. Cerulli, C. D'Ambrosio, M. Gentili and A. Raiconi, Maximizing lifetime in wireless sensor networks with multiple sensor families. *Comput. Oper. Res.* **60** (2015) 121–137.

[12] 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. *J. Network Comput. Appl.* **58** (2015) 12–22.

[13] F. Carrabs, R. Cerulli, C. D'Ambrosio and A. Raiconi, An exact algorithm to extend lifetime through roles allocation in sensor networks with connectivity constraints. To published in: *Optim. Lett.* (2016). DOI: 10.1007/s11590-016-1072-y.

[14] F. Carrabs, R. Cerulli, C. D'Ambrosio and A. Raiconi, Extending lifetime through partial coverage and roles allocation in connectivity-constrained sensor networks. *IFAC-PapersOnLine* **49** (2016) 973–978.

[15] F. Carrabs, R. Cerulli, M. Gaudioso and M. Gentili, Lower and upper bounds for the spanning tree with minimum branch vertices. *Comput. Optim. Appl.* **56** (2013) 405–438.

[16] F. Castaño, E. Bourreau, N. Velasco, A. Rossi and M. Sevaux, Exact approaches for lifetime maximization in connectivity constrained wireless multi-role sensor networks. *Eur. J. Oper. Res.* **241** (2015) 28–38.

[17] 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. *Comput. Oper. Res.* **52** (2014) 220–230.

[18] C. Cerrone, R. Cerulli and A. Raiconi, Relations, models and a memetic approach for three degree-dependent spanning tree problems. *Eur. J. Oper. Res.* **232** (2014) 442–453.

[19] R. Cerulli, R. De Donato and A. Raiconi, Exact and heuristic methods to maximize network lifetime in wireless sensor networks with adjustable sensing ranges. *Eur. J. Oper. Res.* **220** (2012) 58–66.

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

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

[22] G. Desaulniers, J. Desrosiers and M. Solomon, *Column Generation*. Springer US (2005).

[23] K. Deschinkel, A column generation based heuristic for maximum lifetime coverage in wireless sensor networks, in Vol. 4 of *SENSORCOMM 11, 5th Int. Conf. on Sensor Technologies and Applications* (2011) 209–214.

[24] A. Dhawan, C.T. Vu, A. Zelikovsky, Y. Li and S.K. Prasad, Maximum lifetime of sensor networks with adjustable sensing range, in *Proc. of the Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing* (2006) 285 – 289.

[25] C. Duin and S. Voss, Steiner tree heuristics – a survey, in *Operations Research Proceedings 1993. Papers of the 22nd Annual Meeting of DGOR in Cooperation with NSOR*, edited by H. Dyckhoff, U. Derigs, M. Salomon and H.C. Tijms. Springer-Verlag (1994) 485–496.

[26] M. Gentili and A. Raiconi, $\alpha-$coverage to extend network lifetime on wireless sensor networks. *Optim. Lett.* **7** (2013) 157–172.

[27] 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* (2009) 226–231.

[28] F.K. Hwang, D.S. Richards and P. Winter, The Steiner Tree Problem. North-Holland, Amsterdam (1992).

[29] C.H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Englewood Cliffs, New Jersey (1982).

[30] A. Raiconi and M. Gentili, Exact and metaheuristic approaches to extend lifetime and maintain connectivity in wireless sensors networks, in Network Optimization, edited by J. Pahl, T. Reiners and S. Voss. Vol. 6701 of *Lect. Notes Comput. Sci.* Springer, Berlin/Heidelberg (2011) 607–619.

[31] P. Rawat, K.D. Singh, H. Chaouchi and J.M. Bonnin, Wireless sensor networks: a survey on recent developments and potential synergies. *J. Supercomput.* **68** (2014) 1–48.

[32] A. Rossi, A. Singh and M. Sevaux, An exact approach for maximizing the lifetime of sensor networks with adjustable sensing ranges. *Comput. Oper. Res.* **39** (2012) 3166–3176.

[33] A. Rossi, A. Singh and M. Sevaux, Lifetime maximization in wireless directional sensor network. *Eur. J. Oper. Res.* **231** (2013) 229–241.

[34] H. Takahashi and A. Matsuyama, An approximate solution for the steiner problem in graphs. *Math. Japonica* **24** (1980) 573–577.

[35] C. Wang, M.T. Thai, Y. Li, F. Wang and W. Wu, Minimum coverage breach and maximum network lifetime in wireless sensor networks. In *Proc. of the IEEE Global Telecommunications Conference* (2007) 1118–1123.

[36] Q. Zhao and M. Gurusamy, Lifetime maximization for connected target coverage in wireless sensor networks. *IEEE/ACM Trans. Netw.* **16** (2008) 1378–1391.