Università degli Studi di Salerno

Dottorato di Ricerca in Informatica e Ingegneria dell'Informazione
Ciclo 33 – a. a. 2019/2020

TESI DI DOTTORATO / PH.D. THESIS

Coverage in Wireless Sensor Networks with additional operating time constraints: efficient algorithms

ANTONIO IOSSA

SUPERVISOR: **PROF. FRANCESCO PALMIERI**

PHD PROGRAM DIRECTOR: **PROF. PASQUALE CHIACCHIO**

Dipartimento di Ingegneria dell'Informazione ed Elettrica
e Matematica Applicata
Dipartimento di Informatica

To my wife Raffaella, who with her love supports every day of my life and each step of my professional path. Without her patience, her sacrifice and her encouragement, I would not have reached this ambitious goal.

To my sons, Gabriele and Roberta. I hope that the time I have not devoted them due to my PhD commitments, will be compensated by the example I have given them.

Acknowledgements

During the extraordinary experience of my PhD, I had the honour of working with excellent professionals to whom I want to express my gratitude.
I would like to thank my advisor, Professor Francesco Palmieri, for his guidance and his help.
I would also like to express my great gratitude to Professor Raffaele Cerulli, Chair of the Department of Mathematics, for his help, his patience and his precious suggestions, both professional and sometimes personal.
I am infinitely grateful to Dr Ciriaco (Ciro) D'Ambrosio: during our collaboration, I appreciated his professional skills and his intellectual honesty. Ciro shared with me every moment of my research activity, every single step, every reasoning, every hypothesis to verify, every working weekend. His guidance, his encouragement, his patience, his love for details were fundamental for the results achieved in this research work.
Thanks, Ciro!

Abstract

Wireless sensor networks (WSNs) have been a relevant research topic in recent years and, due to technological advances and to the heterogeneity of applicative contexts, today they represent one of the most significant technologies of the 21^{st} century. Environmental monitoring, healthcare, transportation, infrastructure, agriculture are only a subset of all possible application areas of a WSN. WSNs are generally composed of low-cost devices (*sensors*) which collect information about the surrounding space (*sensing area*) that contains specific *targets* of interest. Each sensor monitors targets that are located in its sensing area and the targets in such area are said to be covered by the sensor. Due to weight and size limitations, a sensor is usually powered by a battery which determines a limited operating time. The simultaneous usage of all the sensors, thus, may lead to a faster depletion of the available energy which cause a short *network lifetime*. Extending the network lifetime, i. e. the amount of time during which the WSN is able to perform its monitoring task, represents a very relevant issue. This problem, generally known as Maximum Lifetime Problem (MLP), is a well known and challenging optimization problem which has been addressed successfully with several approaches in the last years. It essentially consists in finding an optimal schedule for sensors activities in a WSN aiming at maximizing the total amount of time during which the WSN is able to perform its monitoring task. The MLP problem can be faced by designing efficient coverage algorithms that exploit the sensor redundancy. Indeed, in WSNs applications involving a huge number of sensors, some of the sensor devices may result to be redundant, that is a target is covered by more than one sensor. A dynamic and coordinated use of the sensors allows to address the limitations imposed by the restricted amount of energy available to the sensors.

A lot of results can be found in literature on the MLP and variants considering classical technical issues (as connectivity and multi-role issues among others) while few research effort has been devoted to investigate specific operational requirements of the sensors. In this thesis we focus on such scenario in which, in order to perform the monitoring activity, each sensor must be active for a predefined period of time defined as *operating time slot*. This context characterizes the *periodic sensing applications* in which a WSN monitors the phenomenon under observation according to a sensitivity cycle which is repeated periodically. A sensitivity cycle consists of a predefined activity time slot, during which the sensors collect information about the targets, followed by an idle period. The idle period is configurable and depends on the application. The activity time slot duration, on the other hand, is fixed a priori and is determined by the sensors operating principle. Examples of periodic sensing applications with fixed activation times can be found in different fields as in structural health monitoring, environmental monitoring, air pollution monitoring, agriculture sensing.

We formally define this problem as Maximum Lifetime Problem with Time Slots (MLPTS). For this new scenario we derive an upper bound on the maximum achievable lifetime and propose a genetic algorithm for finding a near-optimal node activity schedule. The performance evaluation results obtained on numerous benchmark instances, show the effectiveness of the proposed approach. Further, we generalize MLPTS by taking into account the possibility to neglect the coverage of a small percentage of the whole set of targets since, in some applications, the status of the phenomenon under observation, can be estimated or inferred by monitoring even only a subset of all targets. We define such new problem as α_c-MLPTS, where α_c defines the percentage of targets that the WSN has to monitor in each time slot. For this new scenario we propose three approaches: a classical greedy algorithm, a modified version of the genetic algorithm already proposed for MLPTS and a Carousel Greedy algorithm. The comparison of the three approaches is carried out through extensive computational experiments. The computational results show that the Carousel Greedy represents the best trade-off between solutions quality and computational times, and confirm that the network lifetime, also in the case of sen-

sors with operational time constraints, can be considerably improved by omitting the coverage of a modestly percentage of the targets.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Wireless Sensor Network: Motivation

Nowadays there is a growing interest in wireless sensor networks, one of the most significant technologies of the 21^{st} century [1]. A wireless sensor network (WSN) is essentially composed by a large number of sensors whose aim is to detect physical phenomena such as heat, light, air pollution, pressure, etc. Compared to wired networks, WSNs offer simpler deployment and great flexibility of devices. In the context of the Internet of Things [2][3], WSNs are the enabling technology that allows to collect information about the environment in which people live. The numerous application areas in environmental monitoring, automation, healthcare, oil and gas, transportation, infrastructure, insurance services and smart home, among others, have driven the market of WSNs to a staggering scale. Indeed the WSNs market was valued at USD 46.76 billion in 2019 and it is expected to reach USD 123.93 billion by 2025 [4][5]. In addition to the heterogeneity of the WSN applications, market growth has been driven by the increasing adoption of wireless technologies and by the reduced cost of the sensor nodes. Indeed nowadays we are practically surrounded by sensors, ranging from those aboard a simple smartphone to the most specialized sensors such as those that detect air quality, vehicles position, persons heartbeat. Generally speaking, the main goal of a WSN is to acquire knowledge from the environment. The information acquired from the subjects

under monitoring, defined *targets* in literature, can be used, for example, to better study the impact that natural and man-made phenomena can have on the environment, including effects on climate, pollution, safety and many other aspects. There are also contexts where the use of a WSN to acquire information is necessary because the human access is not possible due to the hostility of the environment itself. This is the case, for example, to natural disasters such as floods, earthquakes and so on, where sensors can be deployed and controlled by flying drones [6][7].

While the rapid technological development of sensors has made the WSNs a consolidated technology, the management of the limited energy resources available to the sensors, still represent an open topic for which researchers have felt the need to identify solutions to prolong as much as possible the sensor operation. Since the progress of battery technology in recent years has not shown significant changes, although there are systems that make it possible to acquire part of the energy from the environment [8][9], a targeted and intelligent use of energy resources is still actually a theme to face. Indeed, in order to extend the lifetime of a sensor, it must be taken into account that a sensor is equipped with a limited energy resource which, if exhausted, generally makes the sensor unoperational since it is difficult, and in some cases impossible, replace or recharge its battery. These limits must be taken into account in the design of a WSN because they have a direct impact on the network operating time. The network operating time, i. e. the time for which the network is able to perform the monitoring activity for which it was designed, is usually defined as *network lifetime*. The design of sensors management algorithms that allow to maximize the possible network lifetime has been the subject of intense studies that can essentially be divided into two approaches: power-aware and duty scheduling. In the first case, to maximize the network lifetime, the algorithms act on the network configuration and such approaches can be used when there is a low density of sensors [10][11] and for structured network (see Chapter 2.2.1). Algorithmic approaches based on sensors activity scheduling are used for high density, or unstructured, network scenarios. Unstructured networks are those in which a precise positioning of the sensors is not possible due to the hostil-

ity of the environment in which they are deployed [12][13][14]. For the same reason, obviously, it is not possible to provide additional energy to the sensors. However, since an exact positioning of the sensors is not possible since they are usually scattered from above, a certain sensor redundancy is created which is exploited by the scheduling algorithms. This redundancy makes it possible to identify *cover*, subsets of sensors that are able to autonomously perform the monitoring task for which the network is designed. At any given time, only the sensors belonging to a cover are active, heading off wasting the energy of useless sensors. Since it is usually possible to identify numerous of these subsets in unstructured networks, the maximum lifetime can be obtained by searching and activating them one at a time for an adequate activation time. The identification of these subsets allows to plan the activities of the sensors to ensure the operation of the WSN as much as possible [13][15][16][17] [18][19][20].

The sensor activity scheduling algorithms give as output a collection of couples. Each couple is made by a cover and an activation time for the cover itself. In some cases, the activation time of the cover can be not supported by the technological characteristics of the sensor. Indeed, can be necessary that the sensor is active for a fixed amount of time due to the nature of the phenomenon to observe or due to the sensor functioning principle. So, we will focus on *periodic sensing applications* in which the WSN monitors a phenomenon according to a sensitivity cycle which is repeated periodically. A sensitivity cycle consists of a predefined activity *time slot*, during which the sensors collect information about the targets, followed by an idle period. The idle period is configurable and depends on the application. The activity time slot duration, on the other hand, is fixed a priori and is determined by the sensors operating principle. Examples of periodic sensing applications with fixed activation times can be found in different fields as in structural health monitoring [21][22][23], environmental monitoring [24], air pollution monitoring [25][26][27], agriculture sensing [28]. Given this motivations, in this thesis we study the current literature on wireless sensor networks to design and propose efficient approaches to maximize the network life time in the case of periodic sensing applications, under different coverage constraints.

1.2 Contributions of this thesis

In this thesis we perform an in-depth study on the coverage problems that arise in the design of wireless sensor networks. We will focus on the problem of maximizing the network lifetime for WSN applications where the sensor activity needs to be performed for a predetermined time due to operational constraints and, also, the computational resources available to the WSN are limited. We formally define this problem as Maximum Lifetime Problem with Time Slots (MLPTS). Focusing on the algorithmic aspects of MLPTS, we derive an upper bound on the maximum lifetime and proposed a genetic algorithm to find a near-optimal sensors activity schedule. Then, we perform extensive computational experiments on numerous benchmark instances that show the effectiveness of the proposed approach.

Further, we generalize MLPTS by taking into account the possibility, for each subset of active sensors, to neglect the coverage of a small percentage of the whole set of targets. For this problem, defined α_c-MLPTS, we propose three approaches: a modified version of the genetic algorithm already proposed for MLPTS, a classical greedy algorithm, and a Carousel Greedy algorithm obtained by using the recently developed homonym paradigm, that can be used to improve the performances of a standard greedy algorithm. The comparison of the three approaches is carried out through extensive computational experiments. As will be shown in the analysis of the computational results, the Carousel Greedy represents the best trade-off between the proposed approaches and the network lifetime can be considerably improved by omitting the coverage of a percentage of the targets.

1.3 Thesis organization

This thesis is structured in such a way that the chapters can be read independently of each other. Chapter 1 introduces the reader to the research topic and describes the structure of the thesis. Those who already have a background on WSN can directly read Chapters 4 and 5. Chapter 4 presents the genetic approach adopted for the problem of maximum lifetime with operational constraints and describes our

research published on the journal *Soft Computing*. Chapter 5 presents efficient approaches devised for the maximum lifetime problem with Time Slots and Coverage Constraints and it describes our research accepted for publication on *The Journal of Supercomputing*.

Below is a more detailed description of each chapter of the thesis.

- Chapter 2 introduces some general concepts about WSNs and presents typical coverage problems that arise in the design of a wireless sensor network. This chapter reports also the main results of the literature related to this field.

- Chapter 3 describes the algorithmic approaches adopted to face MLPTS and α_c-MLPTS. In particular, in addition to an overview of genetic algorithms and greedy approaches, the chapter describes the Carousel Greedy approach, which represents a recently designed paradigm that can be used to improve the performances of a standard greedy algorithm in terms of solution quality.

- Chapter 4 presents our research work on the genetic approach to the maximum lifetime problem with additional operating time slot constraints. This chapter formally defines the MLPTS and identifies an upper bound to the value of the optimal solution. The chapter presents in detail the building blocks of the designed genetic algorithm, describes the performed experimentation and presents the analysis of the computational results.

- Chapter 5 presents our research work on the algorithmic approaches to a generalization of the MLPTS problem in which is taken into account the possibility of neglecting the coverage of a small percentage of the entire set of targets. The chapter describes three approaches: a classic greedy algorithm, a Carousel Greedy algorithm and a modified version of the genetic algorithm already proposed for MLPTS in Chapter 4. The chapter also reports the comparison of the three algorithmic approaches by considering the results of a deep phase of computational experiments.

- In Chapter 6 we present our conclusions on the research works and presents some future research directions.

Chapter 2

Coverage Optimization in Wireless Sensor Networks

2.1 Introduction

Wireless sensor networks represent a sophisticated technology that allows monitoring an area through the use of sensors that detect information from the environment in which they are installed [29][30][31], and make such information available externally to such an environment for further processing. The technological advances of the last years in wireless communications and micro-electro-mechanical systems, in conjunction with the miniaturization of computing and sensing devices, have permitted to the WSN technology to reach an advanced level of maturity. A further element that has stimulated the researchers and allowed the development of these types of networks is represented by the numerous fields of applications. Nowadays, there are WSN installations that monitor infrastructures such as power grids, water pipes, bridges, or even systems that monitor entire buildings to collect information on the health of the structure itself. Air quality monitoring [25][26], precision agriculture [28] and optimization of plant growth [32], detection of floods [33], monitoring of vehicular traffic [34] are further examples of WSN applications, among others. Many other applications are described and examined in [35][36][37][38]. A relevant topic in the design of a WSN is represented by the *network coverage*

problems on which we focus our attention. Generally speaking, the concept of coverage essentially expresses *how well* an area of interest is monitored by a sensor network [17]. Of course, different applications may have different coverage requirements. For example, if a WSN is used for monitoring the environmental pollution of an urban area, it may be sufficient to cover a certain percentage of the area to obtain a meaningful measure of the pollution status. Therefore, the network coverage is an application requirement to be considered when designing a WSN. This requirement affects the power consumption of the WSN which in turn determines the *network lifetime*, that is essentially the time interval for which the network is operative.

More precisely, we define network lifetime as the time interval in which the network is able to meet the coverage requirements of the application for which it was designed [39]. This definition implies that if the network is unable to meet the coverage requirements it is considered inoperative.

The network operativity time is strongly influenced by the structural constraints of the sensors which, as explained in the next section, are of limited size and weight and therefore have limited energy resources. Furthermore, in some contexts, it is not possible to supply additional energy to the sensors due to adverse environmental conditions. For this reason, in the design of WSNs, it is of fundamental importance to identify techniques that allow energy to be used efficiently in order to extend the network lifetime. In this chapter we will provide a general introduction to WSNs with a focus on designing networks that are energy efficient.

2.2 Wireless Sensor Networks overview

Generally speaking, a WSN is a network of sensor nodes that cooperatively monitor the environment in which they are installed allowing interaction between people or computers and the surrounding environment. A WSN is integrated in the environment for which it was designed, in the sense that the sensor nodes register the phenomenon in which they are immersed, monitoring the target points in surrounding

space. Also, the sensors itself communicate with each others, transmitting detailed information on the environment under monitoring. The sensors node are ones of the most important parts of a WSN. From an architectural point of view, a sensor node consists of the following functional components: a *sensor*, a *microcontroller*, a *wireless transceiver*, a *power supply* and a *power management* module. The *power management* module makes the power available for the operation of the entire system. The *sensor* represents the boundary of the sensor node and it is able to detect the status of a phenomenon that occurs in the environment under monitoring. Such sensor, or transducer, is a device that converts physical phenomenon into electrical signals. The sensor therefore has the task of transforming physical quantities such as light, concentration of polluting particles in the air, vibrations, chemical quantities into electrical signals. The *microcontroller* manages the operation of the entire system. It deals with the interaction of the sensor node with the other nodes of the WSN, processes the electrical signals received by the sensor and allows to perform scheduling activities of the node. The sensor node is equipped with a *wireless transceiver* that allows a connection to the other nodes. The adoption of the miniaturization technology based on microelectromechanical systems (MEMS), allows to realize sensor nodes of small size and limited power. Nowadays, there are numerous MEMS sensors available on the market that can be used to measure physical quantities such as velocity, acceleration, atmospheric pressure, temperature, humidity, sound, etc. Thanks to this miniaturization technology, all the functional components described above as the miniature sensing device, the associated power supply and so on, can packaged as a miniature MEMS sensor. All these components are subject to constraints such as low energy consumption, low production costs, operational capabilities in hostile environments and, also, to strict dimensional constraints. In some cases a sensor node is contained in a box of one cubic centimeter [40][41]. Given these constraints, it is easy to understand that the lifetime a WSN strongly depends on the battery life of the sensors that compose it. In summary, a *sensor node* is the sensing unit which includes tools such as battery, wireless transceiver, microcontroller and have the main objective of preliminary processing and reporting of the

information collected, while the *sensor* refers only to the hardware device which is able to perceive the state changes of phenomenon. In this thesis we will use the term *sensor node* and *sensor* interchangeably.

Intending with the term *sensing* all the activities of measurement and control of changes in the state of the phenomenon or environment under monitoring, it is possible to identify different *sensing models*. A sensing model expresses a measure of detection capability and its quality by evaluating the relationship between environment, sensors and target. As described in [42] and reported in [43] and other works, generally speaking , the sensing quality of a sensor node decreases with increasing distance. Typical sensing models are based on the Euclidean distance between sensors and target points in a two-dimensional space, and as in most of the works we refer, they consider the concept of coverage, which we now introduce. Let $S = \{s_1, s_2, ..., s_n\}$ be the set of WSN sensors distributed in the area under monitoring and let z be any point in that area. Let (z_x, z_y) and (s_x, s_y) be the Cartesian coordinates, respectively, of the point z and of a generic sensor s. The Euclidean distance between z and s is given by the following relation:

$$d(z,s) = \sqrt{(z_x - s_x)^2 + (z_y - s_y)^2} \tag{2.1}$$

In this thesis we refer to the most studied and simplified sensing model, the *binary coverage model* which considers the following formulation as a coverage function:

$$f(z,s) = \begin{cases} 1, & \text{if } d(z,s) \leq R_s \\ 0, & otherwise \end{cases} \tag{2.2}$$

where R_s is named *sensing radius* or *sensing range*. The coverage function $f(z,s)$ holds 1 when the point z is in the sensig range of the sensor s, while hold 0 otherwise. The sensing range depends on the sensor technology and it defines the sensor detection area, i.e. the area within which the sensor is able to sense the phenomenon for which it was designed.

This model is an *omnidirectional coverage model* and does not consider the angle φ between the targets and the sensors with $0 \leq \varphi(s,z) < 2\pi$. However, we can adopt this model without any loss of

generality since, as reported in [44], by means of simple modifications this model can be also used in three-dimensional contexts.

2.2.1 Coverage problems

Coverage problems can be of two types: *target coverage problems* and *area coverage problems* [45]. In the first type, the objective is to monitor only a set of distinct points which can be specific targets within the environment, while in the second type the main objective is to monitor an entire area of interest. Figure 2.1-*a* shows a WSN where the circles represent sensor sensing ranges and the sensors completely cover the irregular area. In the second case, the sensors cover the targets, represented by small rectangles, leaving some zones of the area uncovered. In [46] it is shown that a target coverage problem instance can be transformed into an area coverage problem instance, so we can refer to both indifferently.

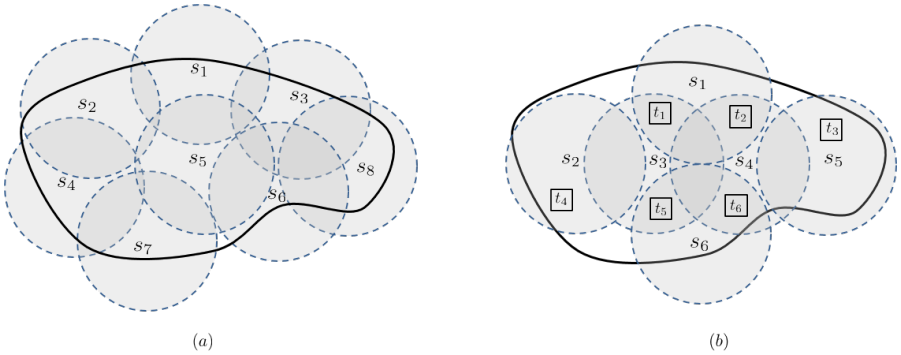


Figure 2.1: Coverage problems examples: (a) area coverage - (b) targets coverage

Generally speaking, in the design of a WSN there are further aspects to consider [31][44] of which the main ones are listed below:

- *Coverage type*: as previously mentioned a first issue arise from the type of the subject under monitoring. If the subject is a continuous area then we refers to an *area coverage type*. If the sub-

ject is a set of targets points (discrete points in an area) the we refers to an *target coverage type*.

- *Deployment type*: there are two types of WSN deployments that determine network topology: *structured* (or deterministic) and *unstructured* (or random)[47]. In the first one, the monitored area is easily accessible and the positioning of the sensors in the surrounding environment can be planned in advance. The positioning can be planned in such a way to minimize the number of sensors with a consequent reduction in management costs. Generally speaking an unstructured deployment is preferable in cases of hostile and inaccessible environments, such as in the case of natural disasters, and when the deployment involves the use of a high number of sensors.

- *Coverage ratio*: this aspect indicates what percentage of the area, or how many points in the area the WSN must cover to meet the coverage requirements. We generally refer to *full coverage* when the WSN covers the entire set of targets or the entire area. Instead, we will specify *partial coverage* when the WSN needs to cover only a subset of points to meet coverage requirements.

- *Network type*: when a WSN is composed of sensors that communicate with a single node that collects information, called *sink*, the network is defined *simple*. A WSN is instead *layered* when in addition to the sensor nodes, there is more than one nodes that collects the information detected by the sensor nodes.

- *Activity scheduling*: it refers to the ability of the WSN to change the operating state of the sensor nodes, allowing any redundant sensors to enter an energy saving state. This operating mode allows energy savings which can have a positive impact on the network lifetime. The activity scheduling algorithms can be centralized or distributed. Distributed algorithms allow each node to decide its own state based on information distributed across the network. This process generally reduces the energy due the communication between nodes but increases the consumption of

processing energy. In centralized scheduling algorithms, a single node makes global decisions on the operating states of all sensors. This type of algorithm significantly reduces the consumption of processing energy and allows to extend the network lifetime. More details on planning activities can be found in the following sections.

We can recall that sensors are small resource constrained devices. They can monitor a given target for an limited amount of time before depleting all their energy. Hostile scenarios can make impossible to supply other energy resources to these sensors. Therefore developing energy efficient algorithms and operational scheme to prolonging the network lifetime is one of the most important requirements common to all previous design aspects.

2.3 Network Lifetime and Coverage Optimization

A first definition of WSN network lifetime is available in [39]. In this work, the network lifetime is defined as the amount of time between the instant in which the WSN is activated and the instant of time in which the first sensor failure occurs. In the case of unstructured high-density networks, however, the network can continue to operate even if one of the sensors is damaged or has run out of all its energy. A different definition, more relevant to our research works (see Chapters 4 and 5), is those according to which the network lifetime is the time interval for which the network is able to satisfy the specific coverage requirements of the application. We choose this type of definition because it is a basic, better adapted to the coverage problems. Thanks to this definition, whatever are the objectives under monitoring, the network lifetime identifies the time interval for which the region of interest is monitored by sensors. In [39] the reader can find a complete list of the main definitions.

2.3.1 Covers Scheduling on WSN

A crucial aspect to consider in order to extend the lifetime of the network as much as possible is the optimization of the use of the limited energy resource of a sensor. There are several approaches in the literature to address energy-efficient coverage to extend the lifetime of the network. A first approach is based on adjusting the sensing range of hte sensors to save energy. A second approach is to design an efficient coverage distribution plan, but it is not always feasible, especially in cases of adverse conditions of the WSN deploy environment. A third approach, on which our research work is based, is to define a sensor activity plan that leaves some sensors in an *active* state while the others are in a state of *sleep* that does not consume energy. Such approach requires to identify *covers*, i. e. subsets of sensors, that if activated are able to achieve autonomously the coverage requirements of the network [17][48][49][50][51][52]. Indeed, usually the deployed sensors provide redundant coverage so that keeping them all simultaneously in an active state causes only a waste of energy without real benefits. So, the identification of an efficient scheduling of their operational states (*idle* or *active*) could help in overcoming the limitations in terms of battery duration which characterizes each individual sensor. It is straightforward, indeed, to note that the identification of such *covers* and their activation times can extend the amount of time over which a WSN is able to perform its monitoring activity. The problem to find an optimal schedule for sensor activities in a wireless sensor network aiming to maximize the total amount of time during which the WSN is able to perform its monitoring task, is known in literature as the Maximum Network Lifetime Problem (MLP). It essentially consist in finding non-necessarily disjoint clusters of sensors, which are autonomously able to surveil specific locations (*targets*) in an area of interest, and activating each of them one at time in order to guarantee the network activity for as long as possible. The pioneering work [17] shows that the MLP is NP-complete and presents an approximation algorithm able to overcome the performances of previous approaches [52] by simply building not necessarily disjoint clusters of sensors (covers). In [53] the authors builds a linear programming (LP)

model for the MLP and they face it for the first time with a column generation technique, a well-known and widely practiced techniques for solving large scale linear programs. However, we find several MLP variants in the literature, each variant facing a different issue. In [48] the authors present an extension of [17] and they propose to use sensors with an adjustable sensing range in order to reduce interferences at the MAC layer. They propose a mathematical model, two heuristics and a model-based algorithm to find the maximum number of covers and a specific sensing range for each sensor to guarantee the coverage of all the targets. In [54] the authors consider WSNs where each sensor can be activated with several power levels. For each power level they consider different sensing ranges and power consumptions. They also present some heuristic approaches and an exact approach based on the column generation technique. The possible failure of the sensors with a consequent uncovering of some targets is considered in several works and it is generally treated as a variant of the MLP named k-MLP. In k-MLP, the objective is to look for covers such that each target is covered by at least k different sensors belonging to the cover. In [55] the authors face the problem to minimize the possibility of uncovered targets and maximizing the network lifetime. They studied and proposed an LP-based algorithm and a greedy heuristic that represents a tradeoff between the two problems. In [56] the authors consider that sensors could be subject to unpredictable failures and they propose an alternative strategy to k-MLP problem by considering wireless networks composed of sensors with adjustable sensing ranges. The Maximum Network α-Lifetime Problem (α-MLP) is another interesting variant of the classical MLP which was proposed in [49] and further investigated in [57], [58] and [59]. In such a variant, a predefined portion of the overall number of targets is allowed to be neglected in each cover. In [49] the authors presented both an heuristic algorithm and an exact one, showing that generally large improvements in terms of overall network lifetime can be already achieved by neglecting a small percentage of targets within each cover. In [57] the authors proposed a hybrid exact approach for the α-MLP problem which combines a column generation approach with a genetic algorithm. Computational tests proved the high performances of the proposed hybrid approach

in terms of requested computational time with respect to the previous algorithms presented in literature for MLP and for α-MLP. In [59] the authors considered α-MLP when sensors may assume more than one role. Furthermore, the impact of connectivity issues has been addressed in several papers, [13][16][58][60], while interference constraints among sensors has been considered in [61] and [62]. In this last work, the authors apply, for the first time in this context, a Carousel Greedy approach [63], in order to speedup the column generation-based algorithm presented in the manuscript. Carousel Greedy is a generalized and promising meta-heuristic approach that we adopted in our algorithm presented in Section 5.4. More classical heuristic approaches based on local search techniques are adopted in [64][65] and in [66]. In [67] the authors apply simulated annealing techniques [68][69] while in the paper [70] the authors consider the possibility of reconfiguring the network to maximize its duration. In [15] the authors face the MLP by considering the scenario in which the monitoring area can be discretized into sub-areas called zones and only recently, in [71] the authors extended the MLP problem by taking into account the issue of charging the sensor batteries in harvesting scenarios. In the next section we describe the scenario on which we focused on, where in order to monitor a target, each sensor needs to be active for a fixed amount of time (operating *time slot*).

2.3.2 Application scenario and operating time slot constraints

In our thesis we consider high-density unstructured WSN with a random deployment of sensors (see Section 2.2.1). The sensors are scattered from above in the area of interest and they monitor the targets placed in the area itself. The sensors collect information on the targets and cooperate to deliver them to a node defined as central and located near the WSN, which also has the function of coordinating the activity of the sensors. We assume that a sensor can have the following operating states: *receive*, *idle* and *transmit*. As reported in previous works [17][72], the energy consumption of a sensor node is similar when the node is in the transmit and receive states, while it is lower when the

sensor is in the state of inactivity. So, as in other works [49][54] we consider for simplicity only two operating states, which we define *idle* (or *sleep*) and *active*, which indicate when the sensor node consumes battery power or not. Furthermore in this thesis we considered the context where, in order to monitor a target, each sensor needs to be active for a fixed amount of time (operating *time slot*). Such a context characterizes periodic sensing applications, where the WSN monitors the phenomenon under observation according to a periodic working schedule that depends on the phenomenon itself. In such application context, the WSN alternates periods of activity equal to a time slot, or multiples of it, with idle periods [24][27][73][74]. The applications described in these works are characterized by a sensitivity cycle which is repeated periodically. A sensitivity cycle consists of a predefined activity time slot, during which the sensors collect information about the targets, followed by an idle period. The idle period is configurable and depends on the application. The default activity time slot, on the other hand, is fixed a priori and is determined by the sensors operating principle. Examples of periodic sensing applications in environmental monitoring context can be found in [27] and in [24]. In more detail, in [27] the default activity time slot is equal to 30 seconds, during which the sensor detects the concentration of particulate matter in the air based on the laser scattering principle, and to do this it is necessary a predefined amount of time, while the idle period is configured equal to 5 minutes. In [24], the authors studied the long-term deployment of a WSN to explore the status and trends of soil moisture and transpiration within a watershed. In such a case the activity time slot was set equal to 50 milliseconds, while the idle period was configured equal to 1 second. Other examples of periodic sensing applications can be found in several context as in structural health monitoring [21][22][23][75][76][77], in vivo glucose sensing [78] and agriculture sensing [28].

2.4 MLPTS problem definition

Let $N = \{S, T\}$ be a WSN in which $S = \{s_1, \ldots, s_m\}$ is the set of sensors and $T = \{t_1, \ldots, t_k\}$ is the set of targets. We consider that all the sensors have the same sensing range and the same amount of *battery lifetime* (lt_i), normalized to 1 time unit. Also, as reported in the previous section, we assume that the sensors can be in two different operation modes, active and sleep. When a sensor s_i is in active mode, it collects information about the targets that are in its sensing range. Each sensor s_i can perform its monitoring tasks for a fixed amount of time, named *time slot* (τ_s) with $0 < \tau_s \leq 1$. Whenever a sensor is in sleep mode, it does not perform any operation and the power consumption is assumed to be equal to 0 while, when a sensor s_i is in active mode, its battery lifetime is decreased by τ_s time units. A subset $C \subseteq S$ is defined to be a *feasible cover* if the sensors $s_i \in C$, all together, are able to monitor, for a τ_s amount of time, all the targets when they are in active mode. Given T, S and τ_s, the maximum lifetime problem with time slot (MLPTS) consists in finding a collection of feasible covers $\Gamma = \{C_1, \ldots, C_l\}$, such that the network lifetime is maximized and each sensor is in an active mode for a total amount of time that does not exceed its battery lifetime.

In Figure 2.2-a, a WSN with sensors $s_1, \ldots, s_5$, targets $t_1, \ldots, t_4$ and sensing ranges represented by circles is shown. We can see that the sensor s_1 covers the targets t_1 an t_2, the sensor s_2 covers t_3, and so on. Let's assume a time slot $\tau_s = 0.3$. We can observe that, if we use all the sensors in each time slot τ_s, we achieve a maximum lifetime equal to 0.9 time units. Instead, it is possible to achieve a network lifetime of 1.5 time units by activating, individually, the covers shown in Figure 2.2, that is $\{s_1, s_2, s_4\}$, $\{s_3, s_4\}$, $\{s_1, s_2, s_5\}$, $\{s_1, s_3\}$, $\{s_1, s_2, s_5\}$. Figure 2.3 shows the activation of the sensors in each time slot and next to each sensor label is shown its remaining lifetime (lt_i).

We observe that each of the above subsets covers all targets and therefore meets the coverage requirements. Maximizing the number of covers for a given WSN thus maximizes the lifetime of the network. This is particularly true for unstructured networks where the targets are covered in a redundant manner due to the high density of the sensors.

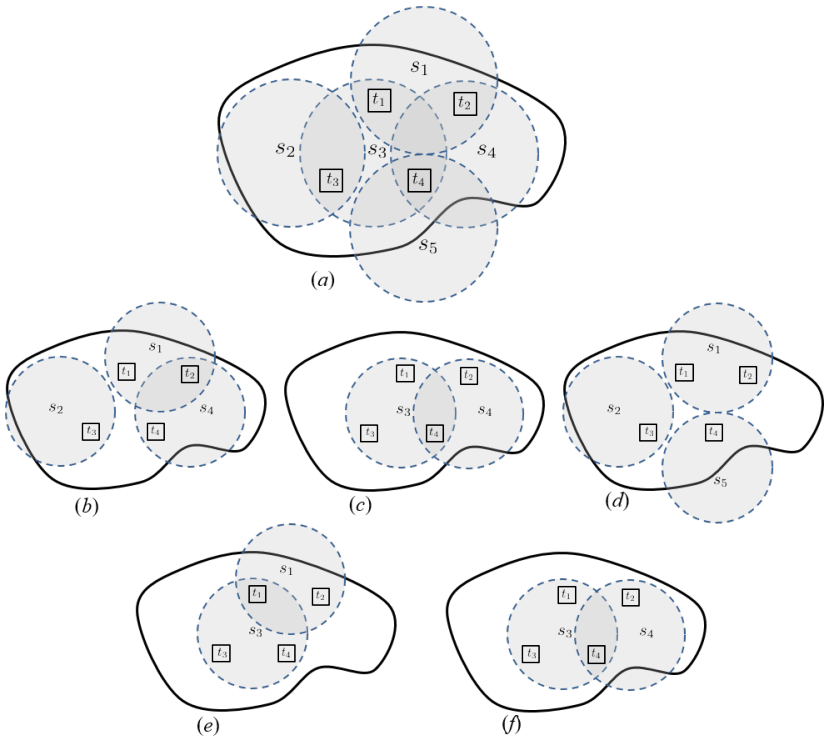


Figure 2.2: Wireless Sensor Network and covers examples

In the next chapter we report a general description about the methods that we will then use to face the MLPTS. In particular, in Chapter 4 we apply a genetic approach to MLPTS, while in Chapter 5 we face a more general version of MLPTS, named α_c-MLPTS, with a classic greedy algorithm, a Carousel Greedy algorithm and a modified version of the genetic algorithm already proposed for MLPTS in Chapter 4.

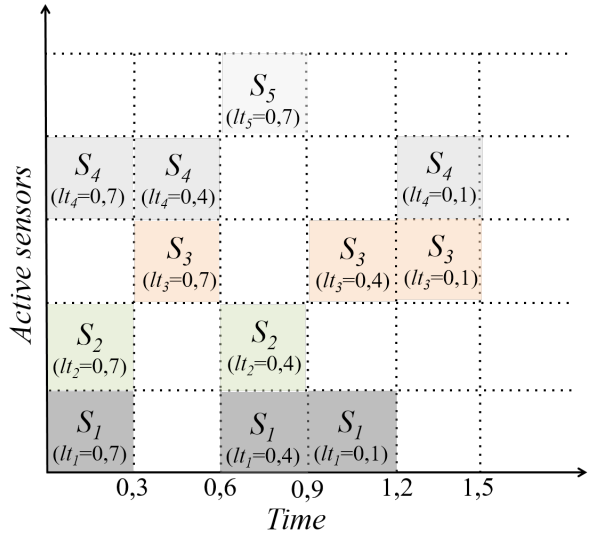


Figure 2.3: Sensor activity scheduling example

Chapter 3

Maximum Lifetime Problem with Time Slots: algorithms

This chapter reports the algorithmic approaches used to face MLPTS. In addition to an overview of genetic algorithms the chapter describes the Carousel Greedy approach, which represents a recently designed paradigm that can be used to improve the performances of a standard greedy algorithm in terms of solution quality.

3.1 Genetic algorithms

A genetic algorithm (GA) emulates biological evolution and natural selection and is a well-known and widely used meta-heuristic technique for optimization problems. Historically, the first algorithm inspired by the Darwin's theory on natural evolution was developed by Holland in 1957 in his work *Adaption in Natural Artificial Systems* [79]. Holland's ideas have been intensively studied and now genetic algorithms represent a mature technique to solve optimization problems. Genetic algorithms can solve many problems in different fields [80]. Scheduling, timetabling, data mining, pattern recognition among others, are just some examples of genetic algorithms applications.

A GA emulates the *natural evolution* that is the process that alters the genetic characteristics of individuals from generation to generation so that they adapt better to their environment. The evolutionary

process of a GA is based on *chromosomes*. In nature, a chromosome represents the structure of a living individual while for a GA a chromosome represents a possible solution to an optimization problem. In nature, the *fitness* of an individual expresses how well the individual is adapted to the environment in which he lives. Similarly, in the GA each chromosome is evaluated through a *fitness function* that expresses a measure of the quality of the solution, represented by the chromosome itself, to an optimization problem. In nature, the evolution of individuals is linked to the reproduction process: 1) during their life, individuals (parents) mate producing new individuals (children) whose genetic heritage is a combination of that of their parents; 2) once generated, the children undergo mutations with respect to the genetic heritage inherited from their parents as a result of environmental influences. Analogously, in a GA the evolution of individuals is achieved through the *crossover*, *selection* and *mutation* operators. The crossover operator probabilistically combines the genetic material of two or more individuals. Natural selection is the process by which the GA ensures that new solutions are typically, step by step, better adapted to the environment. The mutation operator randomly changes the value of one or little pieces (*genes*) of a chromosome derived from the crossover phase. The goal of the mutation operator is to increase the diversity of individuals which corresponds to a better exploration of the research space. The whole process is repeated until a stop condition is reached which may be a lack of improvement in the fitness function of the best individual, the achievement of a maximum number of generations, or other conditions related to the specific optimization problem.

Solving an optimization problem means looking for the best solution in the search space, i. e. the space of feasible solutions. Genetic algorithms exploit the evolutionary concepts mentioned above to find the best solution, that is the individual who has the best fitness value, trying to avoid or escape the local optima. GA achieves this goal by making individuals evolve from generation to generation, obtaining individuals who gradually have better fitness, i.e. a better solution to the optimization problem. There are two key aspects upon which genetic algorithms are based, namely randomness and actual population (i.e. a set of feasible solutions). GAs are nondeterministic algorithms, which

means that randomness is used within the selection, crossover and mutation operators to better simulate the evolutionary process. They also does not work on a single solution, but rather work on a large set of solutions at the same time, which allows genetic procedures to consider a significant amount of diversity at each iteration.

Despite their good characteristics, genetic algorithms also have some drawbacks. Indeed, they represent a ”big picture”, a general scheme that normally works without knowing specific notions about the optimization problem, and this generality leaves them the opportunity to be applicable to a broad set of problems. Obviously, specialized algorithms designed for a given problem can outperform a more generic GA in terms of computational effort and solution accuracy. However, given their characteristics, genetic algorithms are often useful to better investigate a large and complicated research space, or to hybridize other approaches, exact or not.

3.1.1 GA general scheme

A genetic algorithm is composed of a series of steps that follow the paradigm of the evolution of the species. The starting step for the application of a GA is the choice of the representation of individuals, that is the design of a structure that effectively represents a solution to our problem. This structure is known as *chromosome*. GAs work with a *initial population* of individuals of limited size. The initial population of solutions can evolve towards better individuals (hence better solutions for the optimization problem) through the combination of some of their information and that of other individuals, allowing new solutions to inherit better characteristics than previous ones. Therefore the initial search space is reduced to this set of solutions. Given the current population and the details of the optimization problem we would like to solve, a GA uses specific operators that can combine two or more solutions to obtain new ones. The *fitness function*, as reported above, is used to give a measure of the quality of the solutions. Typically the fitness function corresponds to the objective function of the optimization problem. As in nature, the *natural selection* process favors the best individuals, and genetic algorithms implement a *selection process*

on the current population in order to select only the best individuals, based on the fitness value. However, during the evolutionary stages, elements with worse fitness values could also be recombined with others as a means of increasing diversity and exploring new portion of the research space. This diversification is further guaranteed by the mutation operator which randomly alters part of the information belonging to the newly produced chromosomes. The following steps summarize the general structure of a GA.

Genetic Algorithm general scheme:

1. Build an *initial population* randomly: after defined the chromosome structure, the GA generate an initial population randomly;

2. Compute the *fitness* value for each individual: once defined the fitness function that typically corresponds to the objective function of the optimization problem, the GA can rank the quality of each individual of the initial population;

3. Create a *new population* that replaces the old one by means of the following steps:

   (a) *Select* two chromosomes according to their fitness: here two or more individuals are selected by means of specific procedures, i.e. Roulette Wheel Selection, Random Selection, Rank Selection, Tournament Selection [81];

   (b) Apply *Crossover*: the GA combines the genetic material of the selected individuals to generate one or more offspring individuals. Typically the selection and the crossover operator are applied on good individuals belonging to the population since the hope is that by combining them even better solutions could be obtained. There are several crossover techniques such as Single point, Two Point, Multipoint, etc [81];

   (c) Apply *Mutation*: with a predefined probability the algorithm applies a mutation operator on the generated offspring. This operator modifies some information in the newly generated offspring in order to vary the solution and

improve the search to avoid getting stuck in the local optimal;

(d) *Acceptance*: each newly produced offspring, will be inserted in the new population that will replace the older population. In the described scheme, known as generational GA, the new offspring will be used only in the new generation. Another option is represented by the *steady-state* GA in which newly produced offspring is immediately added to the current population and replaces an outgoing older chromosome which is selected according to various criteria, e.g. it might be the older one, the one with the worst fitness or it might be selected through probabilistic tournaments.

4. If any *Stop condition* is reached, the algorithm terminates, otherwise a new iteration is carried out starting from step 2.

The above points represent the steps of a genetic algorithm in its standard definition. In Chapter 4, we will apply this algorithm to the MLPTS problem. In particular, in the Section 4.3.1 we will describe the representation of the solution, while in Section 4.3.2 we will describe the operators used by the algorithm. Section 4.4 reports and analyzes the results of the performed experimentation.

3.2 Greedy algorithms

Greedy algorithms have a long history and many applications throughout computer science, and this is due to their intuitive appeal and conceptual clarity. It is not easy to define exactly a greedy algorithm. Generally speaking, the optimization algorithms work through a series of steps and make a choice at each step. An algorithm for an optimization problem follows a greedy approach if the choice that it makes is the one that seems locally optimal. That is, the greedy algorithm makes an optimal choice locally in the hope that this choice will lead to a global optimal solution. The greedy method is powerful enough and works well for a wide variety of problems. In the context of graph

algorithms, the well known Kruskal's algorithm [82] and Prim's algorithm [83] for the Minimum Spanning Tree problem provide classic examples of the greedy method. Surveys of approaches to the Minimum Spanning Tree Problem, together with historical background, can be found in the seminal works [84] [85]. Further examples of greedy approaches are represented by the Dijkstra's algorithm for the shortest paths from a single source [86] and by the Chvátal greedy heuristic for the set covering problem [87]. Interval Scheduling and the problem of scheduling to minimize the maximum lateness are two of a range of basic scheduling problems for which a simple greedy algorithm can be shown to produce an optimal solution. A collection of related problems can be found in the survey in [88]. Another well-known example of a greedy algorithm is the Huffman algorithm that provides a method for optimal prefix codes [89].

The Section 5.3 describes our greedy approach to face MLPTS. The experimental results of the greedy implementation are reported instead in Section 5.5. As we will see, our greedy approach to MPLTS has excellent execution times, but a lower solution quality than the genetic algorithm presented in the same Chapter 5 which, however, has higher execution times. In the next section, we present an recently presented approach, named Carousel Greedy, that improves the solution quality of a greedy algorithm with slightly higher execution times. We describe the application of such approach in the Section 5.4.

3.3 Carousel Greedy

Greedy algorithms are often used to solve optimization problems because they are easy to describe and implement and usually have very fast execution times. However, greedy algorithms offer a level of solution quality that is not always acceptable. On the other hand, it is not always possible to apply exact approaches when the size of the problem is large. When the size of the problem is such that it does not allow an exact approach, it is possible to apply the so-called metaheuristics (genetic algorithms, tabu search, etc.) which, however, can be complex to code and whose execution times grow rapidly with the size of the

problem. Recently, in [63], the authors proposes the *Carousel Greedy* paradigm which allows to obtain solution algorithms that have the accuracy of metaheuristics with execution time comparable to greedy algorithms. Carousel Greedy (CG) is essentially an enhanced greedy algorithm that, compared to a greedy algorithm, examines a larger space of possible solutions with a small and predictable increase in computational effort. More precisely, the CG is a generalized paradigm that can be used to improve the performances of a standard greedy algorithm in terms of solution quality. In [63], the authors demonstrated that CG can be successfully applied to numerous classical optimization problems such as the minimum vertex cover problem, the maximum indipendent-set problem, the minimum weight vertex cover and the minimum label spanning tree problem. The authors also demonstrate that CG can be easily implemented through a simple and fast procedure as a classic greedy algorithm. CG has been used in many other works [15][61][90] showing the effectiveness of the proposed method. In most cases the authors have shown that CG is able to obtain results comparable to metaheuristics, while being much faster than them. In [61], it can be seen that the CG performs even better than the previous genetic algorithm used in [62].

Below is a high level scheme of the CG paradigm:

1) create a partial solution using a greedy algorithm;

2) modify the partial solution in a deterministic way using the same greedy algorithm;

3) apply the greedy algorithm to produce a complete and feasible solution.

Unlike metaheuristics, the CG does not improve the solution quality by passing from a feasible solution to another better solution. The algorithm during its iterations constructs a single feasible solution starting from the partial solution provided by the greedy. In fact, a requirement for the application of CG is the existence, for the problem to be address, of constructive greedy algorithm, i. e. an algorithm that builds the solution incrementally.

It should be noted that the notion of a generalized greedy algorithm is not new. Duin and Voss introduced the Pilot method in 1999 and applied it to Steiner's problem in graphs [91]. The basic idea behind the Pilot method is to repeatedly apply a greedy algorithm, each time from a different starting point. There are also several other generalized greedy algorithms, such as randomized Greedy [92], iterated greedy [93] which are all easy to implement and which all generate a number of feasible solutions. The CG algorithm is also easy to implement. Since it generates a single feasible solution it is faster than other generic greedy algorithms.

The underlying intuition of the CG is that during the execution of a greedy algorithm, *the early decisions taken to construct a solution are likely to be less informed and valid that the later ones*. During the first steps of the construction of the solution, indeed, the greedy choices could be even arbitrary or not very effective, since many choices may lead to similar solutions. So, the overall quality of the solution can be compromised by inadequate early choices. Given this intuition, the CG approach operates in the following main steps, to extend a generic greedy algorithm:

1. an initial solution is built, using the greedy algorithm;
2. a partial solution is obtained from the initial one. The partial solution is built by discarding a given percentage of the last choices made to build the initial solution;
3. the partial solution is then modified for a given number of iterations. In each iteration the oldest choice is replaced with a new one.
4. in the last step, the partial solution is completed to produce a feasible solution by using the greedy heuristic.

Figure 3.1 illustrates how CG works. Each row is a step in the *CG* algorithm. We will now explain the figure, row by row. *CG* requires that two parameters, α and β, be specified, where α is an integer and β is a percentage. We assume that $\alpha = 1$ and $\beta = 0,4$ and we will describe later these parameters. We suppose that I is a generic input

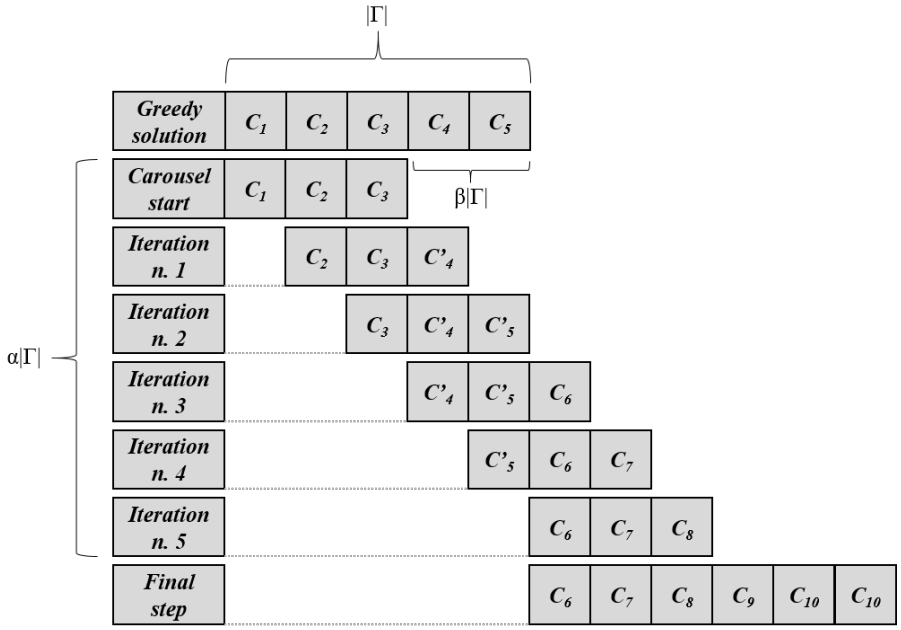


Figure 3.1: Carousel greedy illustration for $|\Gamma| = 5$, $\alpha = 1$, and $\beta = 40\%$

instance of a MLPTS problem and G is a greedy algorithm that gives as output a feasible solution Γ. In the first row, the CG begins executing G and obtaining a solution Γ that is composed by the covers C_1, C_2, C_3, C_4, C_5 , indicated in sequence of selection. In the row 2, the *CG* delete the last $\beta = 40\%$ of the covers contained in Γ, leaving C_1, C_2, and C_3, to postpone the last 40% of selections up to the very end. The algorithm maintains thus a set of $(1-\beta)|\Gamma|$ covers until the last row. In the third row, *CG* eliminates the cover C_1, the first selected in row 1 (*an early decision*), and performs a single iteration of algorithm I which selects C'_4, obtaining the partial solution C_2, C_3, C'_4. In the next row 4, C_2 is removed and the cover C'_5 is selected yielding C_3, C'_4, C'_5. The algorithm proceeds in this way up to line number 7, the one before the final step. We observe that at this point the partial solution is C_6, C_7, C_8 and we have completed $\alpha|\Gamma| = 5$ iterations from the beginning of the *CG* algorithm. In the last line, *CG* applies algo-

rithm G to the partial solution from line 7, until a feasible solution is obtained, then stops.

We observe some important aspects. As evident, CG maintains only a single feasible partial solution and generates a complete one only at the end. Furthermore, as the CG replace the covers that were initially selected by the greedy algorithm G, it is clear that the eventually first errors, due to early decisions, can be corrected. However, the CG is only slightly more complicated than the original greedy algorithm. Regarding the parameters α and β, it is evident that increasing α means increasing the number of basic iterations (rows in Figure 3.1) and, therefore, the running time. The parameter β is related to each partial solution whose size of $(1-\beta)|\Gamma|$ is maintained throughout α turns of the carousel within CG. In [63] the authors conducted various experiments to identify the interaction between the two parameters and the effects on the algorithm performance. Based on their experiments, they observe that for alpha values that increase starting from 1, the running time increases and so does the quality of the solution, until it stabilizes. The β parameter is instead linked to the quality of the solution. Finally, based on the experiments results, they propose to set α to a value between 5 and 20 and β between 1% and 10%.

We reported below a CG pseudocode description, assuming that I is a generic input instance of a problem and G is a greedy algorithm that produces a solution Γ. Note that in rows 5 and 7, using G, means we select new elements to add to the partial solution R according to the greedy algorithm G.

```
    Input: I, G. (I is a generic input instance, G is a greedy algorithm)
1   Let Γ ← the solution produced by G
    (ordered by sequence of selection, the last selected elements are in the head)
2   R ← the partial solution produced by removing from head of Γ, β|Γ| elements
3   for α|Γ| iterations
4       remove from tail of R an element
5       using G, add an element to head of R
6   end for
7   using G, add elements to R as long as the solution remains feasible
8   return R.
```

About the computational complexity, in [63] the authors shown that given a generic greedy algorithm that has complexity $O(G)$, the complexity of a *CG* approach is equal to $(1+\alpha)\cdot O(G)$.

In Chapter 5, we apply the *CG* to the MLPTS problem. In more detail, in Section 5.3 we describe a generic greedy algorithm for the MLPTS problem. This algorithm will be used as a basic algorithm by the *CG* in the Section 5.4. The two approaches will be compared, in terms of execution times and quality of the solution, in the 5.5 section.

Chapter 4

A Genetic approach for the Maximum Network Lifetime Problem with additional operating Time Slot constraints

4.1 Introduction

Maximum Network Lifetime Problem (MLP) is a well known and challenging optimization problem which has been addressed successfully with several approaches in the last years. It essentially consists in finding an optimal schedule for sensors activities in a wireless sensor network (WSN) aiming at maximizing the total amount of time during which the WSN is able to perform its monitoring task. In this chapter we consider a new scenario in which, in order to monitor some locations in a geographical area, the sensors need to be active for a fixed amount of time, defined as operating *time slot*. For this new scenario we derive an upper bound on the maximum lifetime and propose a genetic algorithm for finding a near-optimal node activity schedule. The performance evaluation results obtained on numerous benchmark instances, show the effectiveness of the proposed approach.

Wireless Sensor Networks (WSNs) represent nowadays one of the most advanced technologies able to collect and process information in

heterogeneous contexts [17][94][95]. WSNs are generally composed of low-cost devices (*sensors*) which collect information about the surrounding space (*sensing area*) that usually contains specific targets of interest. While advancements in wireless communications and micro-electro-mechanical systems allowed the adoption of sensor networks in many scenarios, battery technologies experienced much smaller improvements over time. Indeed, energy consumption is still one of the most important issues that has generated a great research interest, especially in the last years due to the diffusion of Internet of Things (IoT) applications and cyber physical systems. In more detail, one of the most important aspects considered to face such issue concerns scheduling sensors activities. The sensors are generally powered by batteries that keep them fully functional only for a limited amount of time. Given a WSN deployed with such sensors, the determination of an efficient scheduling of their operational states (*idle* or *active*) could help in overcoming the limitations in terms of battery duration which characterizes each individual sensor. Usually the deployed sensors provide redundant coverage so that keeping them all simultaneously in an active state causes only a waste of energy without real benefits. In this specific context the main aim is to find non-necessarily disjoint subsets of sensors (*covers*) which are able to provide coverage for all the targets, while keeping all the other sensors in idle state. It is straightforward to note that the identification of such *covers* and their activation times can extend the amount of time over which a WSN is able to perform its monitoring activity. This problem is known in literature as the Maximum Network Lifetime Problem (MLP) and its variants have been addressed with different approaches in the last two decades. In [17], the authors show that the MLP is NP-complete. They also propose an approximation algorithm that generates not necessarily disjoint subsets of sensors that improve the lifetime obtained by previous approaches [52] based on disjoint subset of sensors. In [53] the author proposes a ILP formulation and a column generation-based heuristic to solve the MLP. In the literature there are different variants of the problem in which the MLP occurs different applications. Reliability issues are considered in [56], while the experiences presented in [48][54] take into account sensors with adjustable sensing ranges.

Furthermore, [13][16][60] consider the combination of coverage and connectivity issues. In [61] the authors investigated interference issues among close sensors, while in [15] the authors considered the context in which the monitoring activity is related to zones. In this chapter we considered the context where, in order to monitor a target, each sensor needs to be active for a fixed amount of time (operating *time slot*). So we aim to solve the MLP by activating not necessarily disjoint subsets of sensors for fixed time intervals. This new problem has been defined as Maximum Lifetime Problem with Time Slots (MLPTS). Such a context characterizes periodic sensing applications, where the WSN monitors the phenomenon under observation according to a periodic working schedule that depends on the phenomenon itself. In such working schedule the monitoring time slots could be alternated with time periods in which the WSN is idle [73][74]. Examples of periodic sensing applications with fixed activation times can be found in different fields as in structural health monitoring [75][77][76], environmental monitoring [24], in vivo glucose monitoring [78], etc. For MLPTS we derive an upper bound on the maximum lifetime and propose a genetic algorithm that is able to determine a near-optimal sensors activity schedule. The performance evaluationl results obtained on numerous benchmark instances, show the effectiveness of the proposed approach. The remaining of the paper is organized as follow: in Section 4.2 we define formally MLPTS, and show some useful details that we further use in the proposed resolutive approach; Section 4.3 describes the proposed method and Section 4.4 presents the performed computational tests. Finally, in the last section we give some future directions.

4.2 MLPTS problem definition

Let $N = \{T, S\}$ be a WSN where $T = \{t_1, \ldots, t_k\}$ is the set of k targets and $S = \{s_1, \ldots, s_m\}$ is the set of m sensors. We assume that all sensors have the same technical characteristics in terms of *operation modes*, *sensing range* and *battery duration*. In more detail, we assume that each sensor can be in two different operation modes, sleep and ac-

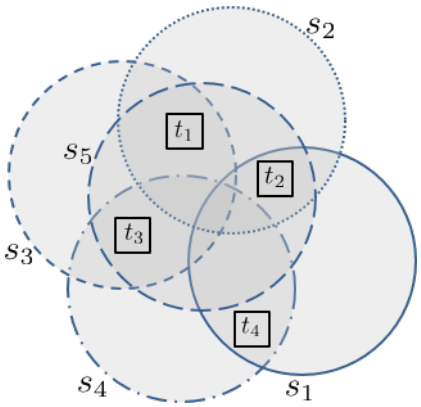


Figure 4.1: a WSN example composed of five sensors s_1, s_2, s_3, s_4, s_5, distinguished by theirs sensing ranges represented by circles and targets t_1, t_2, t_3, t_4.

tive. When a sensor is in sleep mode, it does not perform any operation and its power consumption is assumed to be equal to zero. Conversely, when a sensor is in active mode, it is able to perform its own task, by collecting information about its monitoring targets. When a sensor is active it can perform its monitoring tasks for a fixed amount of time, that is a fixed time slot $\tau_s > 0$, and we assume the power consumption to be proportional to τ_s. The total activation time of a sensor cannot exceed the battery lifetime (lt) normalized to 1 time unit, $lt_i = 1$ for each s_i. Let $T(s_i) \subseteq T$ be the subset of the targets within the sensing range of a sensor s_i. A subset of activated sensors $C \subseteq S$ is defined to be a *cover* if $\bigcup_{s_i \in C} T(s_i) = T$, that is, all the targets are kept under monitoring by the sensors belonging to C. Given T, S and τ_s, the MLPTS consists in finding a collection of feasible covers $\mathcal{C} = \{C_1, \ldots, C_l\}$, that maximizes the network lifetime $\varphi = l \cdot \tau_s$, and such that each sensor is in active mode for a total amount of time that does not exceed its battery lifetime. Figure 4.1 shows a WSN with sensors $s_1, \ldots, s_5$, targets $t_1, \ldots, t_4$ and sensing ranges represented by circles.

If we assume a $\tau_s = 0.5$ time units and all sensors to be active at once, the maximum network lifetime achievable is equal to 1 time unit, by activating all sensors twice. However it is straightforward to note

that it is possible to achieve a network lifetime of 1.5 time units by activating individually the following covers, $\{s_1, s_2, s_4\}$, $\{s_1, s_5\}$, $\{s_4, s_5\}$, for a $\tau_s = 0.5$ each. The optimal lifetime of 2 time units is achieved by activating the cover in the collection $\mathcal{C}^*$=$\{\{s_1, s_3\}, \{s_1, s_5\}, \{s_2, s_3, s_4\}, \{s_4, s_5\}\}$. Observe that an exhaustive research of feasible covers is not applicable due to the high (potentially exponential) number of covers (see [17]).

4.2.1 Network lifetime upper bound in MLPTS

Let $\bar{\varphi}_{\tau_s}$ be the maximum lifetime of a WSN when a time slot of τ_s time units is given and let us denote with $S(t) \subseteq S$ the subset of sensors covering a given target $t \in T$. We denote with t_{lc}, the least covered target, that is t_{lc}=$argmin_{t \in T}\{|S(t)|\}$. It is easy to see that, given the coverage constraint of the MLPTS each feasible cover must contain at least a sensor belonging to $S(t_{lc})$ and hence the following upper bound on the maximum lifetime holds:

$$\bar{\varphi_{\tau_s}} \leq t_s \cdot \sum_{i \in S(t_{lc})} \lfloor \frac{lt_i}{\tau_s} \rfloor \tag{4.1}$$

Furthermore, given a solution $\mathcal{C} = \{C_1, \ldots, C_l\}$, it is easy to derive that the number l of feasible covers is bounded by the following relation:

$$l \leq \frac{\bar{\varphi_{\tau_s}}}{\tau_s} = \bar{l} \tag{4.2}$$

Finally, we can observe that both $\bar{\varphi_{\tau_s}}$ and $\bar{l}$ can be evaluated in polynomial time by finding the least covered target $S(t_{lc})$.

4.3 A genetic algorithm for solving MLPTS

A genetic algorithm (GA) is an iterative probabilistic algorithm inspired by biological evolution that it is generally used for finding good solution for complex optimization problems through solution space exploration. GA iteratively emulates the typical steps of the biological evolution, that is *natural selection*, *crossover* and *mutation*. The GA

considers the evolution process based on chromosomes (i. e. individuals), elements that represent feasible solutions for the optimization problem. Starting from an initial set of random individuals, a typical GA produces new solutions by applying the crossover operator, that combines the genetic information of *selected* individuals. A *mutation* operator is then applied in order to guarantee a proper diversification of the genetic material of the new generated individuals. A *fitness* function, that usually corresponds to the objective function of the optimization problem, is used to rank the quality of each single individual, in order to perform selection of the best solutions determined so far. The natural selection together with the fitness function generally guarantee that the new generated individuals have better characteristics. The overall process is generally repeated until one or more stop conditions are reached, in order to perform a satisfactory exploration of the solution space. The stopping conditions can be of different nature, for example a specified amount of elapsed time, a maximum number of iterations of the evolutive process, a lack of improvements in the fitness value of the best individual or other criteria connected to the characteristics of the optimization problem. A detailed description about genetic algorithms can be found in [96].

The main steps of our GA are listed in Algorithm 1. The GA takes as input a WSN $\{T,S\}$ where T is a set of k targets and S is a set of m sensors, a time slot value τ_s, a population size *pSize*, a crossover and a mutation probability, *cProb* and *mProb* respectively, and finally a maximum iterations parameter *maxIt* as one of the stopping criteria. Line 1 estimates the network lifetime upper bound $\bar{\varphi}_{\tau_s}$, as described in Section 4.2.1. In lines 2 and 3 the GA generates an initial population P of individuals and identifies the best initial solution value. The subroutine $InitPopulation$ for the generation of the initial population is described in Section 4.3.1. The while loop of lines 5-14 iteratively applies $Select$, $Crossover$, and $Mutation$ operators described in Section 4.3.2. The while loop iterates until either the incumbent *BestLT* reaches its upper bound $\bar{\varphi}_{\tau_s}$, that is the optimal value of lifetime has been found by the GA, or *maxIt* consecutive iterations without improvements of the incumbent value *BestLT* are performed. Finally the chromosome with the best fitness value is returned as output of the

input : $\{T,S\}$, τ_s, *pSize*, *cProb*, *mProb*, *maxIt*
output: A chromosome p

```
1  φ̄_τs ← GetLifetimeUpperBound (T, S, τ_s);
2  P ← InitPopulation (pSize);
3  BestLT ← BestLifetime (P);
4  It ← 0 ;
5  while BestLT < φ̄_τs and It < maxIt do
6  |   P ← Select (P);
7  |   P ← Crossover (P, cProb);
8  |   P ← Mutate (P, mProb);
9  |   It ← It + 1 ;
10 |   if BestLifetime (P) > BestLT then
11 |   |   BestLT ← BestLifetime (P);
12 |   |   It ← 0 ;
13 |   end
14 end
15 p ← BestChromosome (P);
16 return p;
```

Algorithm 1: Genetic algorithm

GA.

The following sections describe the details of our GA. Section 4.3.1 illustrates the chromosome representation and the fitness function while Section 4.3.2 describes the generation of the initial population and the operators used by the GA.

4.3.1 Solution representation and fitness function

As shown in Section 4.2.1 a MLPTS feasible solution cannot contain more than $\bar{l}$ covers (4.2). Hence we designed a chromosome composed of at most $\bar{l}$ distinct components $C_1,\ldots,C_{\bar{l}}$. Each component C_l corresponds to a gene g_l and represents a feasible cover for the MLPTS. So as shown in Figure 4.2, each gene is a sequence of m binary values each of them associated to a sensor of the network. In more detail, each position i of a gene g_l is equal to 1 ($g_l{}^i = 1$) if the sensor s_i is

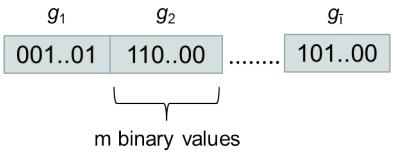
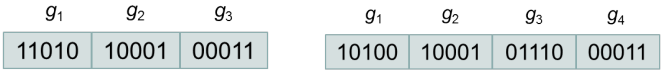


Figure 4.2: Generic chromosome structure



(a) Chromosome p_1 (b) Chromosome p_2

Figure 4.3: Solutions representation

active in the cover C_l, and 0 otherwise. By construction, the length of a chromosome p is at most equal to $|p| = m \cdot \bar{l}$. We observe that, given a chromosome p composed by l_p genes, the network lifetime $\varphi(p)$ achieved by p, is equal to $\tau_s \cdot l_p$. Let us now consider the WSN reported in Figure 4.1 and suppose a time slot $\tau_s = 0.5$ time units. Since the least covered target is t_4 and $S(t_4) = \{s_1, s_4\}$, by the relation 4.1 the maximum network lifetime $\bar{\varphi}$ cannot be greater than 2 units of time. Figure 4.3 shows two chromosomes, p_1 and p_2, the former relative to the solution $\mathcal{C}_1 = \{\{s_1, s_2, s_4\}, \{s_1, s_5\}, \{s_4, s_5\}\}$, the latter relative to the solution $\mathcal{C}_2 = \{\{s_1, s_3\}, \{s_1, s_5\}, \{s_2, s_3, s_4\}, \{s_4, s_5\}\}$.

It is easy to see that $l_{p_1} = 3$ and $\varphi(p_1) = 1.5$ time units, while $l_{p_2} = 4$ and $\varphi(p_2) = 2$ time units.

The fitness function has two components $\varphi(p)$ and $\rho(p)$ combined by using a weighted sum approach [97] :

$$f(p) = w_1 \cdot \varphi(p) + w_2 \cdot \rho(p) \tag{4.3}$$

where $w_1 + w_2 = 1$ and $0 \leq w_1, w_2 \leq 1$. The first component $\varphi(p)$, corresponds to the network lifetime achievable by p, while $\rho(p)$ is an ad-hoc designed component that lets the GA to distinguish among solution with same lifetime value but different residual battery lifetime distribution.

Given p, we define $rlt_i(p)$ as the residual battery lifetime of a sensor s_i, that is:

$$rlt_i(p) = lt_i - \sum_{l=1}^{l_p} {g_l}^i \tau_s \quad i = 1, \ldots, m \tag{4.4}$$

and we denote with $rlt(p)$ the vector of the residual battery lifetime, in which the $i-th$ entry is the residual battery lifetime of s_i. Let us now consider the example chromosome shown in Figure 4.3 (a) relative to the WSN reported in Figure 4.1, where we assume a τ_s = 0.5 time units. We note that, $rlt(p_1) = \{0.0, 0.5, 1.0, 0.0, 0.0\}$ that is, the residual battery lifetime of sensors s_1, s_4 and s_5 is equal to 0, while the residual lifetime of s_2 and s_3 is equal to 0.5 and 1, respectively.

The second component of the fitness function estimates the difference between the mean value $E[rlt(p)]$ and the variance value $\sigma^2[rlt(p)]$, and it is equal to:

$$\rho(p) = E[rlt(p)] - \sigma^2[rlt(p)] \tag{4.5}$$

We observe that since $0 \leq rlt_i(p) \leq 1$ $i = 1, \ldots, m$ the following relations hold, where the 4.7 derives from the Popoviciu's inequality on variances [98]:

$$0 \leq E[rlt(p)] \leq 1 \tag{4.6}$$

$$0 \leq \sigma^2[rlt(p)] \leq \frac{1}{4} \tag{4.7}$$

Indeed $\rho(p)$ corresponds to a measure about uniformly distributed residual battery lifetime, among two chromosomes with the same network lifetime $\varphi(p)$, the bigger is its value the better are distributed the residual battery lifetime values. The motivation of such distinction is related to the crossover operator and it is described in detail in Section 4.3.1. Furthermore we choose w_1 and w_2 in such a way that $\varphi(p)$ has a greater relevance than $\rho(p)$, as stated in the following proposition.

Let p_1 and p_2 be two chromosomes and let f be the fitness function defined by equation (4.3) where $w_1 = 1 - \frac{1}{4} \cdot \tau_s$ and $w_2 = \frac{1}{4} \cdot \tau_s$. Then

$f(p_1) > f(p_2)$ if and only if one of the following conditions holds:

$$l_{p_1} > l_{p_2} \ \ (\varphi(p_1) > \varphi(p_2)) \tag{4.8}$$

$$l_{p_1} = l_{p_2} \text{ and } \rho(p_1) > \rho(p_2) \tag{4.9}$$

4.3.2 Initialization and operators

The initial population P is composed of *pSize* chromosomes. Each chromosome p is built by selecting uniformly at random $|p|$ binary values. In order to guarantee the feasibility of p a *makeFeasible* operator is then applied on p. The Algorithm 2 shows the pseudocode of the *makeFeasible* operator. The operator builds a feasible chromosome p' starting from a not necessarily feasible one p. The while loop of the lines 3-9 tries to add to p' randomly, all the feasible genes of p by checking the residual lifetime of the sensors at each insertion. If the operator is not able to add at least a feasible gene to p', lines 10-16 build and add a feasible gene to p'. Then line 18 tries to switch off active sensors of p' by preserving its feasibility. The procedure iterates until *pSize* feasible chromosomes are generated.

Given an initial population of individuals, a roulette wheel *selection* step [99] is then applied in order to randomly choose $pSize/2$ couples of chromosomes on which the GA applies the *crossover* operator defined next. The *crossover* operator generates new chromosomes by combining the genetic material of the selected chromosomes. Our crossover is a variation of the two-points crossover [99]. It takes as input two chromosomes p and q, composed by l_p and l_q genes respectively, and builds two offspring chromosomes of_1 and of_2 as shown in Figure 4.4. In more details, once extracted two random integer crossover points cp_1 and cp_2, the first offspring of_1 is obtained combining the firsts $cp_1 - 1$ genes of p with the lasts $l_q - (cp_2 - 1)$ genes of q. The offspring of_2 is obtained combining p and q analogously. In order to guarantee the feasibility of the generated individuals *makeFeasible* is then applied on each offspring. The two chromosomes with the highest fitness among the offsprings and their parents, are then inserted in the current population P.

input : A chromosome $p = \{g_1, g_2, \ldots g_{l_p}\}$ not necessarily feasible
output: A feasible chromosome p'

$p' \leftarrow \emptyset$;
$L \leftarrow \{1, ..., l_p\}$;
while $L \neq \emptyset$ **do**
  $i \leftarrow$ `Random`(L);
  $L \leftarrow L \setminus \{i\}$;
  if g_i *is feasible and* $rlt(p' \bigcup \{g_i\}) \geq 0$ **then**
    $p' \leftarrow p' \bigcup \{g_i\}$;
  end
end
if $p' = \emptyset$ **then**
  while g_1 *is not feasible* **do**
    $i \leftarrow$ `Random`(S);
    $S \leftarrow S \setminus \{i\}$;
    ${g_1}^i \leftarrow 1$;
  end
  $p' \leftarrow p' \bigcup \{g_1\}$;
end
$p' \leftarrow$ `CheckRedundancy`(p');
return p';

Algorithm 2: MakeFeasible operator

The *mutation* operator is then applied on the chromosomes in order to guarantee a properly perturbation of the genetic material and hence a better and wider exploration of the solution space. The chromosomes to be mutated are randomly chosen accordingly to a mutation probability value *mProb* that the algorithm takes as input. Once extracted to be mutate, a single random binary value of each of the genes of the selected chromosome is randomly changed in the opposite.

Finally, the GA gives as its output the chromosome with the highest fitness value obtained by calling the procedure *BestChromosome*.

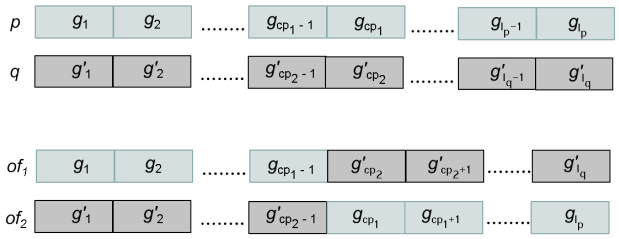


Figure 4.4: Crossover operation: generation of offspring of_1 of_2 starting from parents p and q

4.4 Performance evaluation results

We performed an extensive performance analysis by working on instances proposed in literature by [53] and by [49] for the classical Maximum Lifetime Problem. Technical details about the instances can be found in the related papers. Our GA was coded in C++ and the tests were performed on a macOS platform running on an Intel Core i5 2.5 GHz processor with 8GB RAM. We carried out a preliminary tuning test phase in order to estimate the best values for the GA parameters. The *pSize* parameter was set equal to 50 while the stopping criterion *maxIt* was set to 10. Finally the crossover and mutation probabilities, *cProb* and *mProb* were set to 0.3 and 0.05 respectively. All the tests were performed by considering time slot τ_s values equal to 0.1 and 0.3. Table 4.1 reports the results obtained by our GA on benchmark instances proposed in [53]. Each line in the Table 4.1 corresponds to a test scenario composed of 10 different instances (different topologies) with the same number of sensors and targets and shows the average computational time and average lifetime obtained on these 10 instances. The first three columns report the time slot value, the number of sensors and targets considered in the scenarios. The column *UB* reports the average upper bound for the scenarios while column *Lifetime* and *Time* report respectively the solution values measured in time units and the CPU runtime in seconds. Finally the GAP column shows the average percentage gap between the solution found by GA and the upper bound value. The percentage GAP is estimated as $100 \times (UB - GA)/UB$. First of all we can observe from the Time

Table 4.1: Deshinkel results

			GA			
	|S|	**|T|**	**UB**	**Lifetime**	**Time**	**GAP**
$\tau_s = 0.1$	50	30	3.80	3.73	0.05	1.84%
		60	3.00	2.98	0.08	0.67%
		90	2.80	2.75	0.10	1.79%
		120	2.70	2.64	0.14	2.22%
	100	30	8.70	8.62	0.29	0.92%
		60	7.20	7.09	0.39	1.53%
		90	6.90	6.77	0.53	1.88%
		120	6.70	6.59	0.71	1.64%
	150	30	14.70	14.64	0.95	0.41%
		60	12.30	12.07	1.40	1.87%
		90	11.80	11.52	1.84	2.37%
		120	11.30	11.11	2.33	1.68%
	200	30	19.60	19.52	1.84	0.41%
		60	17.30	17.16	2.87	0.81%
		90	16.60	16.39	3.15	1.27%
		120	15.50	15.20	3.46	1.94%
$\tau_s = 0.3$	50	30	3.42	3.33	0.02	2.63%
		60	2.70	2.64	0.02	2.22%
		90	2.52	2.46	0.03	2.38%
		120	2.43	2.40	0.04	1.23%
	100	30	7.83	7.65	0.09	2.30%
		60	6.48	6.36	0.12	1.85%
		90	6.21	6.10	0.17	1.77%
		120	6.03	5.91	0.23	1.99%
	150	30	13.23	13.23	0.31	0.00%
		60	11.07	10.86	0.44	1.90%
		90	10.62	10.47	0.60	1.41%
		120	10.17	9.96	0.75	2.06%
	200	30	17.64	17.52	0.46	0.68%
		60	15.57	15.39	0.90	1.16%
		90	14.94	14.82	1.22	0.80%
		120	13.95	13.86	1.40	0.65%

columns that GA is very fast on all the instances proposed by [53] with a running time lower than 0.95 seconds on 23 test scenarios out of 32. In more details the running time grows up to 3.46 seconds on the instances when $\tau_s = 0.1$, and up to 1.4 when $\tau_s = 0.3$. Indeed having a $\tau_s = 0.1$ let a larger quantity of energy to be used and and so a wider solution space that leads to a higher computational effort. Furthermore we can note that on 25 test scenarios the percentage GAP is lower than 1.99%. While on 9 test scenarios it is lower than 0.92%. This clearly demonstrates the effectiveness of the approach that produces very good solutions on this data set in at most 3.46 seconds.

We also analyzed the results obtained by our GA on the Group 1 and Group 2 sets of benchmark instances proposed in [49]. Table 4.2 shows the results of our GA when used to solve the $MLPTS$ on the Group 1 instances. The description of the Table 4.2 heading is the same of Table 4.1. However, in this set, each instance contains 15 targets. Furthermore each line in the tables shows the average values obtained over 5 different test scenarios. This set of instances is the easiest to solve. The running time is lower than 0.38 seconds on 7 out of 8 test scenarios. In the worst case the average running time grows up to 1.22 seconds. Even in this case the average computational effort is bigger when the GA is used to solve the problems with $\tau_s = 0.1$ rather when $\tau_s = 0.3$. If we take a look to the GAP column we observe that on 7 out of 8 test scenarios the percentage gap is lower than 0.43% and this further confirms the effectiveness of the approach. On the other side, the Group 2 set of instances is the hardest to solve. This set contains instances with 100 targets, while the number of sensors is not fixed *a priori*. Because of this the number of the sensors is not reported in the headings of the Table 4.3.

The instances are distinguished by the type of the sensors deployment named *Design* and *Scattering* respectively, whose technical details can be found in [49]. On these two sets even if the running time is always less 0.15 seconds, the GAP column reports a percentage gap that ranges from 1.11%, on the *Scattering* instances, up to 2.67% on the *Design* instances. Among all, Group 2 is the hardest set of instances to solve and we think that this is because of the greater number of sensors considered in this case which leads to a higher number of

Table 4.2: Group 1 results

			GA			
	|S|	**|T|**	**UB**	**Lifetime**	**Time**	**GAP**
$\tau_s = 0.1$	25	15	3.60	3.60	0.01	0.00%
	50		9.40	9.38	0.07	0.21%
	100		15.40	15.38	0.35	0.13%
	150		25.00	24.98	1.22	0.08%
$\tau_s = 0.3$	25	15	3.24	3.24	0.00	0.00%
	50		8.46	8.28	0.02	2.13%
	100		13.86	13.80	0.11	0.43%
	150		22.50	22.44	0.38	0.27%

Table 4.3: Group 2 results

			GA			
	τ_s	**|T|**	**UB**	**Lifetime**	**Time**	**GAP**
Design	0.1	100	3	2.92	0.06	2.67%
	0.3		2.7	2.64	0.02	2.22%
Scattering	0.1	100	3	2.97	0.15	1.11%
	0.3		2.7	2.67	0.05	1.11%

feasible covers. However, we can observe that these results prove that our GA is a fast and effective algorithm for the MLPTS problem, able to solve within 3.46 seconds, at most on average, all the test scenarios of the instances proposed in the literature for the classical MLP.

4.5 Conclusion

In this chapter we considered the maximum lifetime problem in a new scenario in which sensors must be active for a specific period of time, defined as a time slot. For solving such a maximum lifetime problem with additional time slot constraints, we have developed a genetic algo-

rithm and derived an upper bound on the maximum value of lifetime. The performance evaluation results obtained by an extensive experimentation on the numerous benchmark instances existing in literature show the effectiveness of the proposed approach. Future research will focus on the study of heuristic solution approaches to the aforementioned problem, and on the analysis of variants of the problem itself.

Chapter 5

Maximum Network Lifetime Problem with Time Slots and Coverage Constraints: efficient approaches

5.1 Introduction

In Wireless Sensor Networks applications involving a huge number of sensors, some of the sensor devices may result to be redundant. As a consequence, the simultaneous usage of all the sensors may lead to a faster depletion of the available energy and to a shorter network lifetime. In this context, one of the well known and most important problems is the Maximum Network Lifetime Problem (MLP). MLP consists in finding non-necessarily disjoint subsets of sensors (*covers*), which are autonomously able to surveil specific locations (*targets*) in an area of interest, and activating each cover, one at a time, in order to guarantee the network activity for as long as possible. MLP is a challenging optimization problem and several approaches have been proposed to address it in the last years. A recently proposed variant of the MLP is the Maximum Lifetime Problem with Time Slots (MLPTS), where the sensors belonging to a *cover* must be operational for a fixed amount of time, called operating *time slot*, whenever the cover is ac-

tivated. In this chapter, we generalize MLPTS by taking into account the possibility, for each subset of active sensors, to neglect the coverage of a small percentage of the whole set of targets. We define such new problem as α_c-MLPTS, where α_c defines the percentage of targets that each cover has to monitor. For this new scenario we propose three approaches: a classical greedy algorithm, a Carousel Greedy algorithm and a modified version of the genetic algorithm already proposed for MLPTS. The comparison of the three heuristic approaches is carried out through extensive computational experiments. The computational results show that the Carousel Greedy represents the best trade-off between the proposed approaches and confirm that the network lifetime can be considerably improved by omitting the coverage of a percentage of the targets.

Wireless Sensor Networks (WSNs) nowadays represent one of the most advanced technologies for collecting and processing information in heterogeneous contexts such as air quality monitoring [95][94], object tracking [100][101], healthcare monitoring [102][103][104], structural health checking [21][22][23]. In addition, a lot of projects are experiencing an increasing usage of wireless devices in environmentally friendly programs worldwide, from personal to industrial and heterogeneous environments and Internet of Things (IoT) technologies are becoming more and more easily accessible from people all around the globe. A clear example is represented by a project named *luftdaten.info* [27]. In such a project, volunteers, from Germany and now, day by day, all around the world, have deployed thousands of particulate matter sensors in urban areas to check, with specific time intervals (time slots), the air quality. The project is based on a "sensor community" whose aim is the construction of a contributor-driven global sensor network that creates a big dataset of open environmental data. The mission of such a project is giving people access to a platform for the easy collection and sharing of environmental data. In such a context is clear that the use of IoT technologies makes air pollution monitoring less complex and helps in better understanding the environment. Furthermore, it is easy to see that such worldwide distributed application can be easily adopted in industrial environment making the air pollution monitoring more and more detailed.

The rapid and great evolution of IoT technologies has led to the reduction of sensor devices in terms of size, energy consumption, and cost and now they are integrated in almost every object. However battery technology, which is still essentially linked to a chemical reaction, has had a much lower development curve and, it is well known that energy consumption is still one of the most important issues that impacts on the network lifetime of a WSN. As a consequence, the optimization of the usage of the limited energy resources is a crucial aspect to be considered in order to extend the network lifetime [17][52]. Indeed, one of the most important and largely studied problem is the Maximum Network Lifetime Problem (MLP) which consist in finding non-necessarily disjoint clusters of sensors, which are autonomously able to surveil specific locations (*targets*) in an area of interest, and activating each of them one at time in order to guarantee the network activity for as long as possible. However, since the sensors are powered by batteries, especially in harvesting environment, they can be active only for an amount of time given by their battery capacity. Therefore it is fundamental to decide for how long it is possible to schedule the activity of such clusters (*covers*) by avoiding to exceed the battery capacity of each sensor.

One of the pioneering work of the MLP problem is [17]. In this paper the authors propose an Integer Programming (IP) formulation for the MLP. In particular they define the Maximum Set Problem MSC and they show that the MLP is NP-complete [105]. They also design two heuristics that are able to overcome the performances of previous approaches [52] by building not necessarily disjoint clusters of sensors (covers). In more detail, the two proposed heuristics aim to maximize the number of covers, solving MSC problem using a linear programming and a greedy approach respectively. The idea behind the algorithms is to select sensors that have the highest battery life and that cover the maximum number of uncovered targets.

In [53] the authors builds a linear programming model for the MLP and they face it for the first time with a column generation technique, a well-known and widely practiced techniques for solving large scale linear programming problems. However, in the literature are present several MLP variants, each of them facing different issues with differ-

ent approaches. In [48] the authors present an extension of [17] and they propose to use sensors with an adjustable sensing range in order to reduce interferences at the MAC layer. They show that even this variant is a NP-complete problem; furthermore, they propose an Integer Programming (IP) model, two model-based heuristics and a greedy algorithm to find the maximum number of covers and a specific sensing range for each sensor to guarantee the coverage of all the targets. The proposed heuristics are based on a relaxation and rounding mechanism. The IP model is first relaxed into a linear programming problem (LP). The solutions of the LP model are then rounded to obtain a feasible solution for the IP. The other proposed greedy approach, builds a solution incrementally and selects sensors considering two criteria. The first criterion gives priority to sensors that cover multiple targets per unit of energy. Furthermore, since the algorithm must also detect the sensing ranges, a smaller detection range is preferred as long as the target coverage objective is reached, allowing the sensor to be operational longer.

In [54] the authors consider WSNs where each sensor can be activated with several power levels. For each power level they consider different sensing ranges and power consumptions. They also present some heuristic approaches and an exact approach based on the column generation technique.

The possible failure of the sensors with a consequent uncovering of some targets is considered in several works and it is generally treated as a variant of the MLP named k-MLP. In k-MLP, the objective is to look for covers such that each target is covered by at least k different sensors belonging to the cover. In [55], the authors face the problem to minimize the possibility of uncovered targets and maximizing the network lifetime. They studied and proposed an LP-based algorithm and a greedy heuristic that represents a tradeoff between the two problems. In [56], the authors consider that sensors could be subject to unpredictable failures and they propose an alternative strategy to k-MLP problem by considering wireless networks composed of sensors with adjustable sensing ranges able to adapt their sensing capability in response to sensor failures. The authors provide an exact algorithm and perform computational experiments to demonstrate the performances

of the proposed approach.

The Maximum Network α-Lifetime Problem (α-MLP) is another interesting variant of the classical MLP which was proposed in [49] and further investigated in [57], [58] and [16]. In such a variant, a predefined portion of the overall number of targets is allowed to be neglected in each cover. In [49], the authors presented both a heuristic algorithm and an exact one, showing that generally large improvements in terms of overall network lifetime can be already achieved by neglecting a small percentage of targets within each cover. In [57], the authors proposed a hybrid exact approach for the α-MLP problem which combines a column generation approach with a genetic algorithm. Computational tests proved the high performances of the proposed hybrid approach in terms of requested computational time with respect to the previous algorithms presented in literature for MLP and for α-MLP. In [58], the authors considered α-MLP when sensors may assume several roles. They provide a mathematical formulation for the α-MLP and propose a column generation based approach. In [16], the authors consider connectivity issues when a communication link exists between each couple of sensors. The authors developed an exact and a heuristic algorithm, both based on column generation. They used an appropriately designed genetic algorithm to overcome the difficulty of solving the subproblem to optimality in each iteration of the exact approach. Moreover, they devise the heuristic by stopping the exact column generation procedure as soon as the genetic algorithm does not improve the incumbent solution.

Furthermore, the impact of connectivity issues has been addressed in several others papers, [13][58][60], while interference constraints among sensors has been considered in [61] and [62]. In this last work, the authors apply, for the first time in this context, a Carousel Greedy approach [63], in order to speedup the column generation-based algorithm presented in the manuscript. Carousel Greedy is a generalized and promising meta-heuristic approach that it is adopted in our algorithm presented in Section 5.4. More classical heuristic approaches based on local search techniques are adopted in [64][65] and in [66]. In [67], the authors apply simulated annealing techniques [68][69] while in the paper [70] the authors consider the possibility of reconfiguring

the network to maximize its duration. In [15], the authors face the MLP by considering the scenario in which the monitoring area can be discretized into sub-areas called zones and only recently, in [71] the authors extended the MLP problem by taking into account the issue of charging the sensor batteries in harvesting scenarios.

Among the MLP variants, in this chapter it is further investigated the problem presented in [106]. Here the authors addressed the MLP when the activity of each sensor needs to be performed for a fixed amount of time because of the operational constraints. The authors defines formally such problem as Maximum Lifetime Problem with Time Slots (MLPTS). They derived an upper bound on the maximum lifetime and proposed a genetic algorithm to find a near-optimal node activity schedule. In such application context, the WSN alternates periods of activity equal to a time slot, or multiples of it, with idle periods [24][27][73][74]. The applications described in these works are characterized by a sensitivity cycle which is repeated periodically. A sensitivity cycle consists of a predefined activity time slot, during which the sensors collect information about the targets, followed by an idle period. The idle period is configurable and depends on the application. The default activity time slot, on the other hand, is fixed a priori and is determined by the sensors operating principle. Examples of periodic sensing applications in environmental monitoring context can be found in [27] and in [24]. In more detail, in [27] the default activity time slot is equal to 30 seconds, during which the sensor detects the concentration of particulate matter in the air based on the laser scattering principle, while the idle period is configured equal to 5 minutes. In [24], the authors studied the long-term deployment of a WSN to explore the status and trends of soil moisture and transpiration within a watershed. In such a case the activity time slot was set equal to 50 milliseconds, while the idle period was configured equal to 1 second. Other examples of periodic sensing applications can be found in several context as in structural health monitoring [21][22][23][75][76][77], in vivo glucose sensing [78] and agriculture sensing [28].

As can be seen from the literature, a lot of results can be found on the MLP and variants considering classical technical issues (as connectivity and multi-role issues among others) while few research ef-

fort has been devoted to investigate specific operational requirements of the sensors (as the amount of time required by a sensor to perform its sensing task). Given such background and the interest in the periodic sensing applications, we further investigated the Maximum Lifetime Problem with Time Slots. The contribution of this chapter can be summarized as follow:

- we generalize the Maximum Lifetime Problem with Time Slots (MLPTS) presented in [106], taking into account the possibility to neglect a small percentage of targets in each cover and then further prolong the network lifetime, as in [49] and [57] for the classical MLP;

- we formally define such problem as α_c-MLPTS, where α_c is a parameter that defines the percentage of targets that the WSN must monitor in each cover;

- we present a new greedy heuristic and an improved version that implements the recent Carousel Greedy paradigm;

- we modify the genetic algorithm presented in [106], to consider the partial coverage of the targets.

Our computational experiments show that the Carousel Greedy algorithm represents the best tradeoff in terms of computational time and quality of the solution.

The remaining of the chapter is organized as follow: Section 5.2 defines formally the α_c-Maximum Lifetime Problem with Time Slot (α_c-MLPTS), and shows some useful details that are further used in the proposed approach; Section 5.3 describes the proposed greedy algorithm and Section 5.4 shows the Carousel Greedy approach; Section 5.5 presents the performed computational tests. Finally, the last Section presents conclusions and some future research directions.

5.2 MLPTS and α-MLPTS problem definitions

Let $N = \{S, T\}$ be a WSN where $S = \{s_1, \ldots, s_m\}$ is the set of m sensors and $T = \{t_1, \ldots, t_k\}$ is the set of k targets. Each sensor s_i has a *battery lifetime* (lt_i), and a sensing range that determines the targets that can be monitored by the sensor s_i. It is assumed that the sensors have the same amount of battery lifetime, normalized to 1 time unit. Furthermore, it is supposed that the sensors can be in two different operation modes, active and sleep. When a sensor s_i is in active mode, it collects information about the targets that are in its sensing range and this subset of targets is denoted by $T(s_i) \subseteq T$. As defined in [106], each sensor s_i can perform its monitoring tasks for a fixed amount of time, named *time slot* (τ_s) with $0 < \tau_s \leq 1$. Whenever a sensor s_i is in active mode, its battery lifetime is decreased by τ_s time units while, when a sensor is in sleep mode, it does not perform any operation and the power consumption is assumed to be equal to 0. A subset $C \subseteq S$ is defined to be a *feasible cover* if $|\bigcup_{s_i \in C} T(s_i)| = |T|$ that is, the sensors $s_i \in C$, all together, are able to monitor all the targets when they are in active mode. As originally defined in [106], given T, S and τ_s, the maximum lifetime problem with time slot (MLPTS) consists in finding a collection of feasible covers $\Gamma = \{C_1, \ldots, C_l\}$, such that the network lifetime, given by $\varphi = l \cdot \tau_s$, is maximized and each sensor is in an active mode for a total amount of time that does not exceed its battery lifetime.

In Figure 5.1, a WSN with sensors $s_1, \ldots, s_4$, targets $t_1, \ldots, t_5$ and sensing ranges represented by circles is shown. Let's assume a time slot $\tau_s = 0.3$. Since the targets t_1 and t_4 are within the sensing range of s_1, it holds that $T(s_1) = \{t_1, t_4\}$. Similarly $T(s_2) = \{t_1, t_2, t_3, t_4\}$, and so on. The sets $\{s_1, s_3, s_4\}$ and $\{s_2, s_3\}$, are feasible covers as well as the whole set S, as it can be seen in Figure 5.2. However, if all the sensors are used, that is $C = \{s_1, s_2, s_3, s_4\}$, for each time slot τ_s, it is possible obtain a network lifetime equal to 0.9 time units. Instead, it is possible to achieve a network lifetime of 1.2 time units by activating, individually, the covers in the collection $\Gamma^* = \{\{s_1, s_2, s_3\}, \{s_2, s_4\}, \{s_2, s_3\}, \{s_1, s_3, s_4\}\}$. It is easy to note that it is not possible add

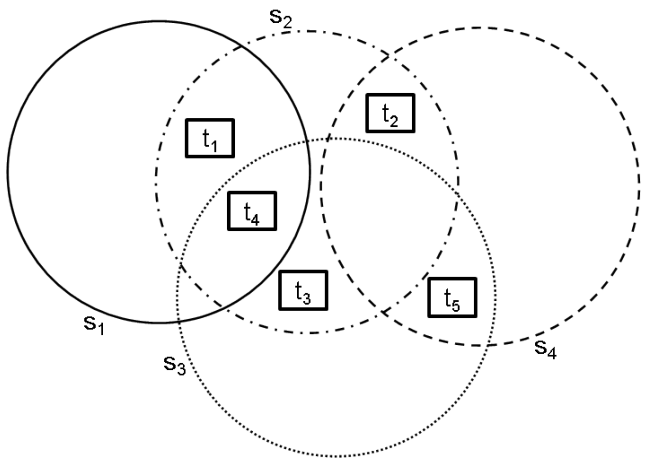


Figure 5.1: a WSN example composed of four sensors s_1, s_2, s_3, s_4, distinguished by theirs sensing ranges represented by circles and targets t_1, t_2, t_3, t_4, t_5.

additional feasible covers to Γ^* without exceeding the sensor lifetimes.

The classical MLPTS requires the coverage of the entire set of targets. However, in some practical cases, like in air monitoring, fire detection and underwater monitoring, it is possible to neglect some targets. Indeed, in these cases, it can be observed that the status of the phenomenon related to the uncovered targets can be estimated or inferred from the covered targets. Let us consider, for example, the air quality monitoring in a civil context. It is easy to see, for example, that the pollution level relative to a specific target point left intentionally uncovered, may be determined by the pollution level of the nearest covered targets. Another interesting case study is represented by fire prevention applications where a WSN generally may use sensors to monitor smoke and/or heat levels. In such a case it would be ideal to gather information for all target points, however, detections with a proper degree of correctness are still possible if some targets are left uncovered. In this way, by choosing a proper amount of targets to be left uncovered it is also possible to achieve a desirable balance between network lifetime and the degree of detection accuracy. Finally, another relevant scenario in which it could be convenient to left some

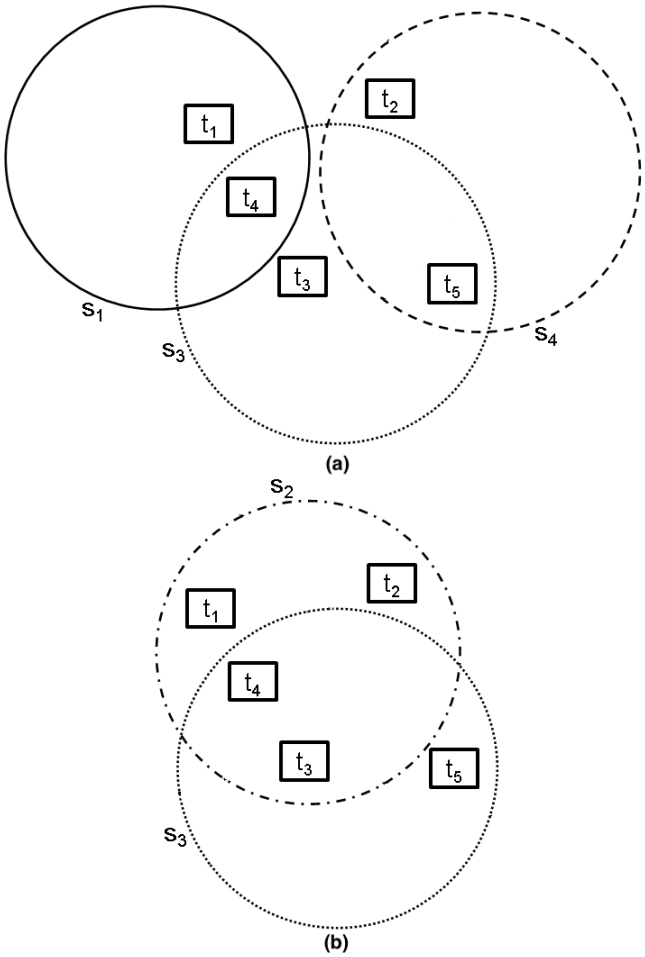


Figure 5.2: two examples of classical feasible covers (a) $\{s_1,\ s_3,\ s_4\}$ and (b) $\{s_2,\ s_3\}$.

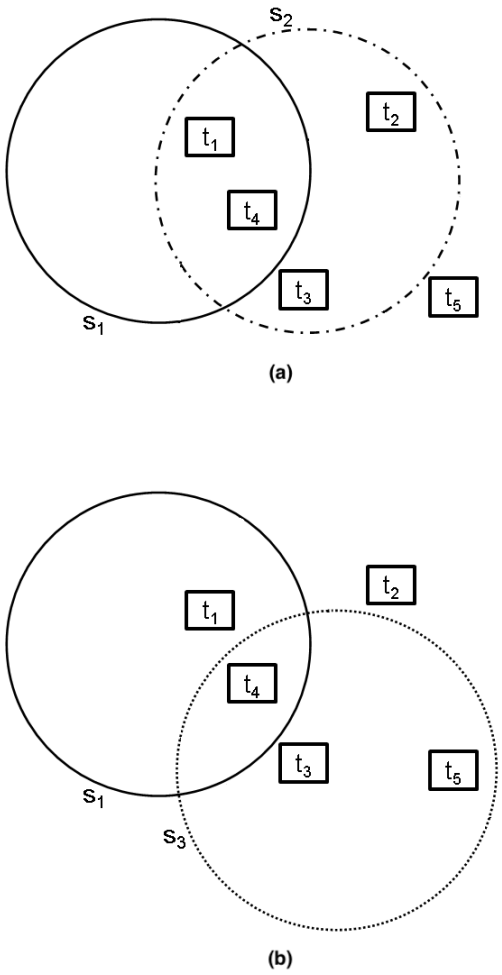


Figure 5.3: two examples of α_c-covers (a) $C_1^{0.8} = \{s_1, s_2\}$ and (b) $C_2^{0.8} = \{s_1, s_3\}$ when $\alpha_c = 0.8$.

targets uncovered in favor of the network lifetime is underwater monitoring. Indeed, in such a case is even more difficult or unpractical to supply energy to a sensor or to replace some damaged sensors in the network. Generally speaking, in many WSN applications, it is acceptable to leave some targets uncovered to find a fair balance between the quality of the sensing mission and the network lifetime.

In [49] and [57], the authors face the classic maximum lifetime problem (MLP) and they show that by neglecting even a small percentage of the targets, a significant improvement of the network lifetime can be obtained. In the following, the MLPTS problem studied in [106] is generalized to applications where some targets can be neglected during the monitoring activity. Given a value $\alpha_c \in (0,1]$, $C^{\alpha_c} \subseteq S$ is said to be a feasible α_c-cover if $|\bigcup_{s_i \in C^{\alpha_c}} T(s_i)| \geq \alpha_c \cdot |T|$, that is, the sensors belonging to C^{α_c} are able to monitor at least $\alpha_c \cdot k$ targets. The maximum α_c-lifetime problem with time slot (α_c-MLPTS) consist then in finding a collection of feasible α_c-covers $\Gamma = \{C_1^{\alpha_c}, \ldots, C_l^{\alpha_c}\}$, such that the network lifetime $\varphi = l \cdot \tau_s$ is maximized and the battery lifetime capacity of each sensor is not exceeded. Let us now consider again the WSN in Figure 5.1, with $\tau_s = 0.3$ and let us assume that $\alpha_c = 0.8$, that is, it is necessary to guarantee the coverage of 4 out of 5 targets with each α_c-cover. The sets $C_1^{0.8} = \{s_1, s_2\}$ and $C_2^{0.8} = \{s_1, s_3\}$ are examples of feasible α_c-covers since, it holds that $|\bigcup_{s_i \in C_j^{0.8}} T(s_i)| \geq 4$, where $j \in \{1,2\}$ as it can be see in Figure 5.3. A feasible solution is represented by the following collection of α_c-covers, $\Gamma_1^{\alpha_c} = \{\{s_1, s_2\}, \{s_1, s_3\}, \{s_2, s_3, s_4\}, \{s_2, s_3\}, \{s_1, s_4\}\}$. Such collection is able to guarantee a lifetime of 1.5 time units. However, it is possible to achieve the optimal lifetime of 2.1 time units by activating individually the α_c-covers in the collection $\Gamma^{\alpha_c^*} = \{\{s_2\}, \{s_3, s_4\}, \{s_2\}, \{s_3, s_4\}, \{s_2\}, \{s_1, s_4\}, \{s_1, s_3\}\}$.

It is easy to observe that the problem α_c-MLPTS is a generalization of MLPTS, furthermore, when $\alpha_c = 1$, α_c-MLPTS coincides with MLPTS. Therefore, from now on, the chapter focuses on the α_c-MLPTS that considers the MLPTS as a special case. The next section describes a greedy algorithm for the α_c-MLPTS. It will be embedded into a novel and generalized greedy approach known as Carousel Greedy [63] which allow us to improve the quality of the solutions of

the greedy procedure. Furthermore, in order to perform a complete analysis of α_c-MLPTS, the proposed approaches are compared with a modified version of the genetic algorithm presented in [106]. Finally, in order to avoid complicating the notation, the apex α_c is not used when it is clear from the context.

5.3 A greedy algorithm for solving the α_c-MLPTS

The proposed approaches are based on the *residual battery lifetime*, as defined in [106]. Given a collection of α_c-covers $\Gamma = \{C_1, \ldots, C_l\}$, the boolean variable x_j^i states that a sensor s_i belongs to a cover C_j. Specifically, $x_j^i = 1$ if the sensor s_i belongs to the cover C_j, and $x_j^i = 0$ otherwise. The residual battery lifetime of a sensor s_i is denoted by $rlt_i(\Gamma)$, and it is defined as follow:

$$rlt_i(\Gamma) = lt_i - \sum_{j=1}^{l} x_j^i \tau_s \quad i = 1, \ldots, m. \tag{5.1}$$

Specifically, (5.1) refers to the residual battery life of a sensor s_i after it has been used in the collection Γ. Clearly, if the collection Γ is empty, $rlt_i(\Gamma) = 1$, $i = 1, \ldots, m$. Let us denote with $rlt(\Gamma)$ the whole vector of the residual battery lifetimes, in which the i-th entry represents the residual battery lifetime of s_i.

The greedy algorithm takes in input a WSN {S, T}, a time slot value τ_s, a value $\alpha_c \in (0, 1]$ and, finally, the vector rlt, containing the residual lifetime of each sensor. Starting from an empty collection Γ, the algorithm builds a collection of feasible covers $\Gamma = \{C_1, \ldots, C_l\}$ where the sensors in each cover are selected with a greedy criterion. A detailed description of the greedy algorithm follows.

In the line 1, the subroutine *GetFeasibleCoversNumberUB* calculates the upper bound $\bar{l}$ on the number of feasible covers, as described in [106] for the case $\alpha_c = 1$. When $\alpha_c < 1$, the *GetFeasibleCoversNumberUB* returns as upper bound the value $\sum_{i \in S} \lfloor \frac{lt_i}{\tau_s} \rfloor$. Line 2 initializes Γ to the empty set while line 3 initializes the flag value

input : $\{S,T\}$, τ_s, α_c, rlt
output: A collection Γ of feasible covers

1 $\bar{l} \leftarrow$ `GetFeasibleCoversNumberUB` (S, T, τ_s, α_c);
2 $\Gamma \leftarrow \emptyset$;
3 *coverFound* $\leftarrow$ **true**;
4 **while** *coverFound* **and** $|\Gamma| < \bar{l}$ **do**
5 $\quad$ *coverFound* $\leftarrow$ **false**;
6 $\quad$ $C \leftarrow \emptyset$;
7 $\quad$ $S' \leftarrow \{s_i \in S \mid rlt_i(\Gamma) \geq \tau_s\}$;
8 $\quad$ **while** $S' \neq \emptyset$ **and not** `isFeasibleCover` *(C,* α_c*)* **do**
9 $\quad\quad$ $s \leftarrow \underset{s_i \in S'}{\operatorname{argmax}}(\mid T(s_i) \setminus \bigcup_{k \in C} T(k) \mid \cdot rlt_i(\Gamma))$;
10 $\quad\quad$ $C \leftarrow C \cup \{s\}$;
11 $\quad\quad$ $S' \leftarrow S' \setminus \{s\}$;
12 $\quad$ **end**
13 $\quad$ **if** `isFeasibleCover` *(C,* α_c*)* **then**
14 $\quad\quad$ $\Gamma \leftarrow \Gamma \cup C$;
15 $\quad\quad$ `updateResidualLifeTimes` (C, rlt);
16 $\quad\quad$ *coverFound* $\leftarrow$ **true**;
17 $\quad$ **end**
18 **end**
19 **return** *(*Γ*,* $\bar{l}$*)*;

Algorithm 3: Greedy algorithm for α_c-MLPTS

coverFound to *true* in order to manage the iterations of the loop 4-18. Each iteration of the external loop 4 - 18, builds a feasible cover C, until it is possible. This loop iterates until a feasible cover is found, and the size of the current collection Γ is less than $\bar{l}$. So the algorithm stops if it is not able to find a feasible cover or when a solution, containing $\bar{l}$ covers, has been found. In the line 6 the algorithm initializes the cover under construction C, which will contain the sensors selected in the current iteration of the external while loop. Since each cover is activated each time for an amount of time equal to τ_s, in the line 7 the algorithm selects and stores into S' the sensors that can potentially be chosen to belong to C, that is sensors with a residual lifetime greater or

equal to τ_s. The greedy selection is implemented by the while loop at the lines 8 - 12. In each iteration of this loop, the algorithm computes, for each sensor $s_i \in S'$, a value $\mid T(s_i) \setminus \bigcup_{k \in C} T(k) \mid \cdot rlt_i(\Gamma)$, that is the product between the amount of uncovered targets that s_i can cover if added to C and the residual lifetime of s_i. The greedy algorithm selects then the sensor for which such product is maximum. The aim of this choice is twofold because we want to select sensors that cover many uncovered targets and we want to preserve the sensors with a limited residual lifetime. Indeed, the residual battery lifetime value of a sensor over a collection Γ, allow us to discriminate sensors that cover the same number of targets but have a different residual battery lifetime. In such situations, a sensor with a greater lifetime is selected, in order to preserve the lifetime of a sensor with a lower residual battery lifetime value. Lines 10 and 11 add the selected sensor s to the cover under construction C and removes the same sensor from the set S', respectively. The internal while loop of the lines 8 - 12 ends when S' becomes empty or a feasible cover is found. The feasibility of a cover is checked by $isFeasibleCover(C, \alpha_c)$. Such subroutine checks if the sensors in the set C cover at least $\alpha_c \cdot k$ targets. In this case, lines 14 and 16 add the built cover C to the collection Γ, update the residual lifetimes with $updateResidualLifetime$ subroutine and assert the *coverFound* flag. Otherwise, if the algorithm exits the while loop, because S' has become empty, it means that all sensors with a residual lifetime greater than or equal to τ_s added to C are not able to cover the required number of targets. It means that there are no new feasible covers compared to those already included in Γ. In this case, since the *coverFound* flag remains false, the algorithm ends and returns the collection found and the value $\bar{l}$ to the caller.

Regarding the computational complexity, it is possible to observe that the number of iterations of the external loop 4 - 18 is limited by $\bar{l}$. The internal loop 8 - 12, is performed $O(m)$ times because in each iteration an element is extracted from S', whose cardinality can be at most m. The calculation of the union and the extraction of the maximum and the subsequent rearrangement of the data structures in line 9 have a complexity of $O(m + \log m)$. Therefore, the complexity of the aforementioned Greedy algorithm is $O(\bar{l} \cdot m \cdot (m + \log m))$. The pre-

sented Greedy algorithm is used both as the main procedure for solving α_c-MLPTS instances and as a subroutine of the Carousel approach presented in the next section.

5.4 A Carousel Greedy algorithm for α_c-MLPTS

Carousel Greedy is a generalized paradigm that can be used to improve the performances of a standard greedy algorithm in terms of solution quality. Carousel Greedy has been presented for the first time in [63]. Here, the authors showed that the Carousel Greedy can be successfully applied on a wide set of classical optimization problems such as the minimum vertex cover problem, the maximum indipendent-set problem, the minimum weight vertex cover and the minimum label spanning tree problem. Furthermore they showed that Carousel Greedy can be easily implemented through a procedure which is almost as simple and fast as a classical greedy algorithm. Carousel Greedy has been used in several other works [15][61][90] that show the effectiveness of the proposed method, in particular in most cases the authors showed that Carousel Greedy is able to obtain results comparable to metaheuristics, while being much faster than them. In more detail, in [61] the authors presented an improvement of a column generation algorithm proposed for the classical MLP with conflict constraints [62], by facing the pricing subproblem with a Carousel Greedy algorithm instead of a genetic algorithm. In [61], it can be observed that the Carousel exhibits even better performances of the previous genetic algorithm used in [62].

The underlying intuition of the Carousel Greedy is that during the execution of a greedy algorithm, the early decisions taken to construct a solution are likely to be less informed and valid that the later ones. During the first steps of the construction of the solution, indeed, the greedy choices could be even arbitrary or not very effective, since many choices may lead to similar solutions. So, the overall quality of the solution can be compromised by inadequate early choices.

Given this intuition, the Carousel Greedy (CG) approach operates

in the following main steps, to extend a generic greedy algorithm:

1. An initial solution is built, using the greedy algorithm.

2. A partial solution is obtained from the initial one. The partial solution is built by discarding a given percentage of the last choices made to build the initial solution.

3. The partial solution is then modified for a given number of iterations. In each iteration the oldest choice is replaced with a new one.

4. In the last step, the partial solution is completed to produce a feasible solution by using the greedy heuristic.

Algorithm 4 presents the steps of the proposed Carousel Greedy (CG) approach. CG uses Greedy presented in section 5.3 and implements the previous four steps as follows:

1. Lines 1 and 2 initialize the collection of covers Γ to the empty set and each entry of the residual lifetime vector $rlt(\Gamma)$ to 1, respectively. Line 3 builds a full initial solution by using Greedy which returns, also, the upper bound on the number of feasible covers according to the method proposed in [106], that it is stored in $\bar{l}$. In the next line, the total number of the covers contained in the solution Γ is stored in l. Line 5 checks if l is equal to $\bar{l}$, in that case the algorithm has found the optimum and ends by returning Γ.

2. The for loop of lines 8 - 11 builds a partial solution by dropping the latest $\lfloor \beta \cdot l \rfloor$ covers from the collection Γ, where $0 < \beta < 1$ is a input parameter of the Carousel Greedy paradigm. The procedure *increaseResidualLifeTimes* increases the residual lifetime of the sensors in each dropped cover by a value τ_s.

3. This step is iterated $\alpha \cdot l$ times, where $\alpha \geq 1$ is a Carousel Greedy input parameter. In each iteration of the for loop of the lines 12 - 17, the cover corresponding to the oldest choice performed by *Greedy*, is removed from Γ and replaced with a new cover C.

input : $\{S,T\}$, τ_s, α_c, α, β
output: A collection Γ of feasible covers

1 $\Gamma \leftarrow \emptyset$;
2 $rlt_i(\Gamma) \leftarrow 1 \quad i = 1,\ldots,m$;
3 $(\Gamma, \bar{l}) \leftarrow$ `Greedy` $(\{S,T\}, \tau_s, rlt, \Gamma)$;
4 $l \leftarrow |\Gamma|$;
5 **if** $l = \bar{l}$ **then**
6 | **return** Γ;
7 **end**
8 **for** $i \leftarrow l$ **to** $l - \lfloor \beta \cdot l \rfloor$ **do**
9 | $\Gamma \leftarrow \Gamma \setminus \{C_i\}$;
10 | `increaseResidualLifeTimes` (C_i, rlt, τ_s);
11 **end**
12 **for** $i \leftarrow 1$ **to** $\alpha \cdot l$ **do**
13 | $\Gamma \leftarrow \Gamma \setminus \{C_i\}$;
14 | `increaseResidualLifeTimes` (C_i, rlt, τ_s);
15 | $C \leftarrow$ `GreedySingleIteration` $(\{S,T\}, \tau_s, rlt, \Gamma)$;
16 | $\Gamma \leftarrow \Gamma \cup \{C\}$;
17 **end**
18 $\Gamma \leftarrow$ `Greedy` $(\{S,T\}, \tau_s, rlt, \Gamma)$;
19 **return** Γ;

Algorithm 4: Carousel Greedy algorithm for α_c-MLPTS

Such a new cover C is built by calling *GreedySingleIteration*, that executes a single iteration of *Greedy*, taking as input the current solution Γ and the actual residual lifetimes.

4. In the line 18, the partial solution Γ is completed by invoking the *Greedy*.

In [63] the authors shown that given a generic greedy algorithm that has complexity $O(I)$, the complexity of a Carousel Greedy approach is equal to $(1+\alpha)\cdot O(I)$. So, considering the Greedy algorithm presented in the previous section, the CG complexity is equal to $(1+\alpha)O(\bar{l}\cdot m\cdot (m+\log m))$. As shown in the next Section, since the α will not be

too large, the CG will have a running time not much greater than our Greedy.

5.5 Experimental Evaluation

The objective of our computational tests is to evaluate the performances of the proposed algorithms in terms of solution quality and execution time. So, this section presents a comparison among Greedy presented in Section 5.3, Carousel Greedy (CG) presented in Section 5.4, and a genetic algorithm (α_c-GA) which is a slightly modified version of the genetic algorithm (GA) for MLPTS presented in [106], that is able to solve α_c-MLPTS. α_c-GA, essentially, implements the same evolutionary algorithm described in [106] but here it deals with α_c-covers instead of simple covers as for MLPTS. It is straightforward to consider that when $\alpha = 1$, α_c-GA can be used to solve the classical MLPTS. In more detail, in [106] the authors design a chromosome where each gene represents a cover for the MLPTS, that is, a subset of sensors able to cover all the targets. Here, in our α_c-GA, has been used the same chromosome structure of GA with the only difference that each gene represents an α_c-cover. Hence, in order to adapt GA to face α_c-MLPTS it has been sufficient to modify the feasibility test of the chromosome to check that each gene represents a feasible α_c-cover. The feasibility check now verifies that the active sensors of each cover monitor at least $\alpha \cdot |T|$ targets and that each sensor is not used for an amount of time that exceeds its battery capacity. Further details about the GA can be found in [106] while for a detailed description about genetic algorithms and recent evolutions see [107][108][109][110][111].

The algorithms were implemented in C++ and the tests were performed on a iMac platform configured with a 2.5 GHz Intel Core i5 processor and 8GB of RAM. Has been performed a preliminary tuning phase of the CG and of α_c-GA, in order to setup the parameters of each algorithm for the final test phase. Regarding the CG algorithm has been chosen the following values, $\alpha = 5$ and $\beta = 0.1$. While regarding the α-GA the size of the population has been set equal to 50. The maximum number of iterations has been set equal to 10 while the

crossover and mutation probabilities were set to 0.3 and 0.05 respectively.

5.5.1 Test instances

An extensive performance analysis was performed by running the algorithms on randomly built instances. For each one of $|S|x|T| = 10$ different combinations of sensors and target values, has been generated 10 different instances, with $|S| \in \{500, 750, 1000, 1250, 1500\}$ and $|T| \in \{15, 30\}$. In more detail, for each instance, were first generated at random the coordinates of the targets within a square area of 500x500. Then were generated the coordinates of the sensors. All the coordinates have been generated such that each sensor covers at least a target and each target is covered by at least one sensor, by assuming that each sensor has a sensing range equal to 100. During our test phase were considered two different *time slot* values with $\tau_s \in \{0.1, 0.3\}$ and three different coverage levels with $\alpha_c \in \{1, 0.9, 0.75\}$ for a total number of 60 scenarios and 600 performed tests for each algorithm.

5.5.2 Test results

Each table is divided in two parts. The first part reports the results obtained with $\tau_s = 0.1$ and a second part that reports the results when $\tau_s = 0.3$. Each line, in all the tables, corresponds to a scenario and reports the average values obtained on 10 instances. Tables 5.1 and 5.2 report the comparison of the three proposed approaches when $\alpha_c = 1$. In more detail, Table 5.1 shows the computational data about the three algorithms. Table 5.2, instead, carries out a comparison by considering the average percentage gaps among the solutions and the computational times of the algorithms. In Table 5.1, the first two columns report the characteristics of each scenario, that is the number of sensors ($|S|$), the number of targets ($|T|$). The third column (UB) reports the upper bound value on the lifetime of the scenario proposed in [106]. The remaining columns report the lifetime (LT) and the computational time in seconds ($Time$). As in Table 5.1, the first two columns of Table 5.2 report the characteristics of each scenario, the

Table 5.1: Solution and running time values, case $\alpha_c = 1$

$\alpha_c = 1$				α_c-GA		Greedy		CG	
	$\|\mathbf{S}\|$	$\|\mathbf{T}\|$	UB	LT	Time	LT	Time	LT	Time
$\tau_s = 0.1$	500	15	40.10	39.33	21.72	39.05	0.84	40.10	2.84
	500	30	29.80	29.43	16.79	29.05	1.25	29.78	4.70
	750	15	59.40	58.44	66.83	57.73	1.80	59.39	6.89
	750	30	43.50	42.71	49.21	42.14	2.64	43.48	8.92
	1000	15	85.10	83.64	151.43	82.20	3.67	85.10	12.06
	1000	30	58.10	56.99	123.49	56.11	4.71	58.07	22.70
	1250	15	114.60	112.87	416.52	110.01	6.16	114.60	18.34
	1250	30	74.90	73.92	190.86	72.79	7.96	74.84	28.74
	1500	15	139.80	138.39	648.47	136.02	9.04	139.80	30.69
	1500	30	93.30	91.80	514.11	90.77	12.30	93.30	44.74
$\tau_s = 0.3$	500	15	36.09	35.65	6.74	34.78	0.25	36.06	0.64
	500	30	26.82	26.49	5.26	25.98	0.38	26.79	1.39
	750	15	53.46	52.92	20.54	51.31	0.52	53.46	1.59
	750	30	39.15	38.67	14.97	37.73	0.81	39.06	2.15
	1000	15	76.59	75.81	45.95	73.52	1.09	76.53	3.01
	1000	30	52.29	51.76	38.24	50.92	1.38	52.29	5.13
	1250	15	103.14	102.00	126.47	99.31	1.80	103.11	4.62
	1250	30	67.41	66.39	58.16	64.97	2.37	67.26	8.33
	1500	15	125.82	123.29	196.74	121.66	2.68	125.82	5.69
	1500	30	83.97	82.36	156.03	81.86	3.52	83.82	12.73

meaning of the headings is the same. The following three columns (α_c-GA, *Greedy*, *CG*), under the multi-column GAP % UB, report the average percentage gap of the lifetimes when compared with the average UB value computed for the scenario. The average gap is computed as $100 \cdot (UB - LT)/UB$, where LT is the lifetime obtained by the algorithm indicated in the header of the column. The following multi-column (α_c-GA vs *Greedy*), report the average percentage gap (*GAP%*) between the solutions and the average percentage gap (*Time%*) between the computational times of the α_c-GA and *Greedy*. The average percentage gap (*GAP%*) between the solutions of α_c-GA and *Greedy* is equal to $100 \cdot (LT_{\alpha_c-GA} - LT_{Greedy})/LT_{\alpha_c-GA}$. In the same way, the next two multi-columns (*CG* vs *Greedy*), (*CG* vs α_c-

Table 5.2: Comparison among α_c-GA, Greedy and CG, case $\alpha_c = 1$

$\alpha_c = 1$			GAP % UB			α_c-GA vs Greedy		CG vs Greedy		CG vs α_c-GA	
	|S|	|T|	α_c-GA	Greedy	CG	GAP %	Time %	GAP %	Time%	GAP %	Time%
$\tau_s = 0.1$	500	15	1.92	2.62	0.00	0.72	3.84	2.70	29.39	1.96	13.08
	500	30	1.23	2.53	0.07	1.33	7.47	2.53	26.69	1.18	28.00
	750	15	1.62	2.82	0.02	1.23	2.70	2.88	26.17	1.63	10.31
	750	30	1.82	3.12	0.05	1.34	5.37	3.18	29.63	1.81	18.13
	1000	15	1.71	3.41	0.00	1.76	2.42	3.53	30.40	1.74	7.96
	1000	30	1.92	3.42	0.05	1.55	3.81	3.48	20.74	1.90	18.38
	1250	15	1.51	4.01	0.00	2.60	1.48	4.18	33.59	1.53	4.40
	1250	30	1.31	2.81	0.08	1.54	4.17	2.81	27.70	1.25	15.06
	1500	15	1.01	2.71	0.00	1.75	1.39	2.78	29.47	1.02	4.73
	1500	30	1.61	2.71	0.00	1.13	2.39	2.79	27.48	1.64	8.70
$\tau_s = 0.3$	500	15	1.23	3.63	0.08	2.49	3.75	3.68	39.43	1.16	9.51
	500	30	1.24	3.14	0.11	1.96	7.14	3.12	27.06	1.14	26.39
	750	15	1.02	4.02	0.00	3.13	2.55	4.19	33.01	1.03	7.73
	750	30	1.23	3.63	0.23	2.49	5.41	3.52	37.57	1.01	14.39
	1000	15	1.01	4.01	0.08	3.13	2.38	4.10	36.36	0.94	6.55
	1000	30	1.02	2.62	0.00	1.64	3.61	2.69	26.88	1.03	13.43
	1250	15	1.11	3.71	0.03	2.70	1.42	3.82	38.88	1.09	3.66
	1250	30	1.51	3.61	0.22	2.18	4.08	3.52	28.46	1.31	14.32
	1500	15	2.01	3.31	0.00	1.34	1.36	3.42	47.05	2.05	2.89
	1500	30	1.91	2.51	0.18	0.62	2.26	2.39	27.69	1.77	8.16

GA) report a comparison among the respective algorithms, and the average percentage gap is calculated analogously. The percentage gap between the computational times is always calculated as the percentage of time taken by the faster algorithm between the two considered, with respect the slowest one. More in detail, in the multi-column α_c-GA vs *Greedy*, *Time%* is equal to $100 \cdot Time_{Greedy}/Time_{\alpha_c - GA}$, in the next multi-column named *CG* vs *Greedy*, *Time%* is equal to $100 \cdot Time_{Greedy}/Time_{CG}$, while in the last multi-column named *CG* vs α_c-GA, *Time%* is equal to $100 \cdot Time_{CG}/Time_{\alpha_c - GA}$.

The first thing that it is possible to observe, from Table 5.1, is that the number of targets impacts differently on the computational time of the algorithms. The α_c-GA exhibits a lower computational time when the number of targets increases. This behaviour, probably, is a consequence of the nature of the genetic meta-heuristic and intuitively, it spends less time in optimization since it has a lower number of solution improvements. While given, the choices of the greedy-based algorithms, it is possible to see that the more are the targets to cover

Table 5.3: Solution and running time values, case $\alpha_c = 0.75$

$\alpha_c = 0.75$			α_c-GA		Greedy		CG	
	$\|\mathbf{S}\|$	$\|\mathbf{T}\|$	LT	Time	LT	Time	LT	Time
$\tau_s = 0.1$	500	15	85.85	68.65	85.18	1.63	91.19	8.85
	500	30	74.07	82.22	73.19	2.74	77.89	15.34
	750	15	137.03	234.88	135.45	3.87	145.75	21.42
	750	30	111.35	272.51	110.08	6.15	117.44	34.59
	1000	15	164.00	576.31	162.45	6.43	177.49	35.32
	1000	30	150.36	665.91	148.47	10.79	158.10	60.76
	1250	15	217.66	1097.51	216.81	10.74	232.61	58.21
	1250	30	181.96	1304.99	180.85	17.30	195.73	97.61
	1500	15	266.77	2381.87	263.52	15.03	278.06	83.19
	1500	30	216.46	2963.60	214.99	24.67	230.88	139.24
$\tau_s = 0.3$	500	15	76.26	20.66	75.57	0.48	80.79	2.68
	500	30	64.32	24.92	63.90	0.85	69.12	4.66
	750	15	119.67	70.86	118.95	1.15	129.60	6.45
	750	30	98.85	82.21	97.86	1.95	104.01	10.46
	1000	15	147.84	173.97	147.75	1.99	158.31	10.87
	1000	30	131.82	200.60	130.89	3.37	140.61	18.43
	1250	15	196.62	330.05	194.01	3.11	207.06	17.17
	1250	30	163.35	390.72	162.72	5.58	174.36	29.12
	1500	15	231.78	755.29	229.50	4.54	247.17	24.71
	1500	30	194.52	866.99	192.60	7.93	205.32	41.33

the bigger is the computational time. This behaviour seems to be quite normal since the more are the targets to cover the more are the choices to be made and hence the computational time increases too. As expected, the computational time of the CG algorithm is always bigger than Greedy algorithm. The same trend can be observed with both $\tau_s = 0.1$ and $\tau_s = 0.3$. τ_s influences the computational times of all the algorithms in the same way and can be noticed that the lower it is the bigger are the values reported in the *Time* columns. Furthermore, but is easy to comprehend, for each scenario the lifetime computed when $\tau_s = 0.3$ is always lower then the lifetime when $\tau_s = 0.1$. Continuing to stress the analysis on the computational time, can be seen that, in the best case the α_c-GA requires at least 5.26 seconds and in the worst case around 650 seconds. While, in the worst case, the Greedy requires at most 12.30 seconds and the CG at most 44.74 seconds. Looking at Table 5.2, can be performed a more detailed analysis on

Table 5.4: Comparison among α_c-GA, Greedy and CG, case $\alpha_c = 0.75$

$\alpha_c = 0,75$			α_c-GA vs Greedy		CG vs Greedy		CG vs α_c-GA	
	$\|\mathbf{S}\|$	$\|\mathbf{T}\|$	GAP %	Time %	GAP %	Time%	GAP %	Time %
$\tau_s = 0.1$	500	15	0.78	2.37	7.05	18.37	6.22	12.90
	500	30	1.21	3.33	6.43	17.84	5.16	18.66
	750	15	1.16	1.65	7.60	18.07	6.36	9.12
	750	30	1.16	2.26	6.69	17.79	5.47	12.69
	1000	15	0.95	1.12	9.26	18.22	8.23	6.13
	1000	30	1.27	1.62	6.49	17.77	5.15	9.12
	1250	15	0.39	0.98	7.29	18.45	6.87	5.30
	1250	30	0.61	1.33	8.23	17.72	7.57	7.48
	1500	15	1.23	0.63	5.52	18.06	4.23	3.49
	1500	30	0.68	0.83	7.39	17.72	6.66	4.70
$\tau_s = 0.3$	500	15	0.91	2.34	6.91	18.05	5.94	12.99
	500	30	0.66	3.41	8.17	18.24	7.46	18.72
	750	15	0.61	1.63	8.95	17.89	8.30	9.11
	750	30	1.01	2.37	6.28	18.63	5.22	12.73
	1000	15	0.06	1.14	7.15	18.26	7.08	6.25
	1000	30	0.71	1.68	7.43	18.31	6.67	9.19
	1250	15	1.35	0.94	6.73	18.09	5.31	5.20
	1250	30	0.39	1.43	7.15	19.18	6.74	7.45
	1500	15	0.99	0.60	7.70	18.38	6.64	3.27
	1500	30	1.00	0.91	6.60	19.19	5.55	4.77

the algorithms. Looking at multicolumn *GAP % UB*, can be seen that the percentage GAP of the α_c-GA is at least 1.01% and at most 2.01%. Greedy exhibits worst results since the GAP value ranges from 2.51% up to 4.01%. The best approach, in terms of quality of solutions, when α_c is equal to 1, is CG since it is able to find the optimal solution for 8 out of 20 scenarios while, for 16 out 20 scenarios the percentage gap is at most 0.1%. CG exhibits a worst case solution with a GAP equal to 0.23%. Furthermore, looking at multi-columns *α_c-GA vs Greedy* it is possible to see that Greedy requires at most 7.47% of the computational time required by the α_c-GA, indeed Greedy is faster than α_c-GA, however α_c-GA finds solution with a greater lifetime up to 3.13% if compared to Greedy. While looking at *CG vs α_c-GA*, can be seen that CG requires at most the 28% of the computational time of α_c-GA however it finds better solutions than α_c-GA, up to 2.05%. So, even if the computational time required by CG is bigger than Greedy, it is at least lower than the 72% of the α_c-GA, furthermore CG finds

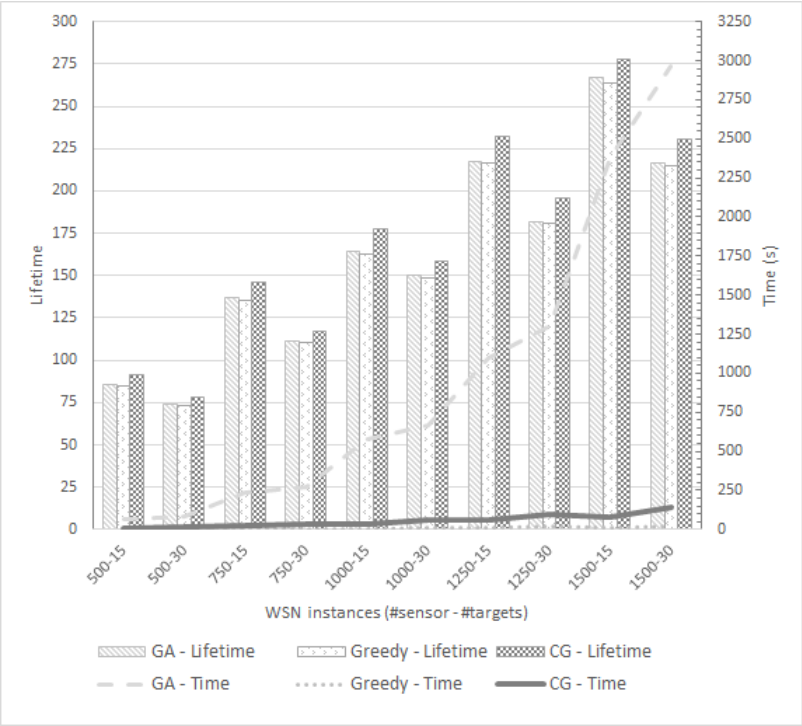


Figure 5.4: solution and running time values, case $\alpha_c = 0.75$, $\tau_s = 0.1$

always better solutions. Indeed, in the end, looking to *CG vs Greedy* multicolumn we discover that CG improves the Greedy solutions from 2.39% up to 4.19%, providing solutions of higher quality.

Table 5.3 and 5.4 report the comparison of the three proposed approaches when $\alpha_c = 0.75$. In more detail, Table 5.3 shows the computational data about the three algorithms the same values are also plotted in the figures 5.4 and 5.5. The x-axis reports the instance characteristics, while the Lifetime and Time values are reported on the primary and secondary y-axis, respectively. The structure of the Table 5.3 and the meaning of its headings are the same of Table 5.1. The first thing that can be observed is that, by simply requiring the coverage of 75% of the targets, the problem becomes more and more challenging. The number of solutions grows exponentially and the computational time required by the algorithms increases too. In this case the number of

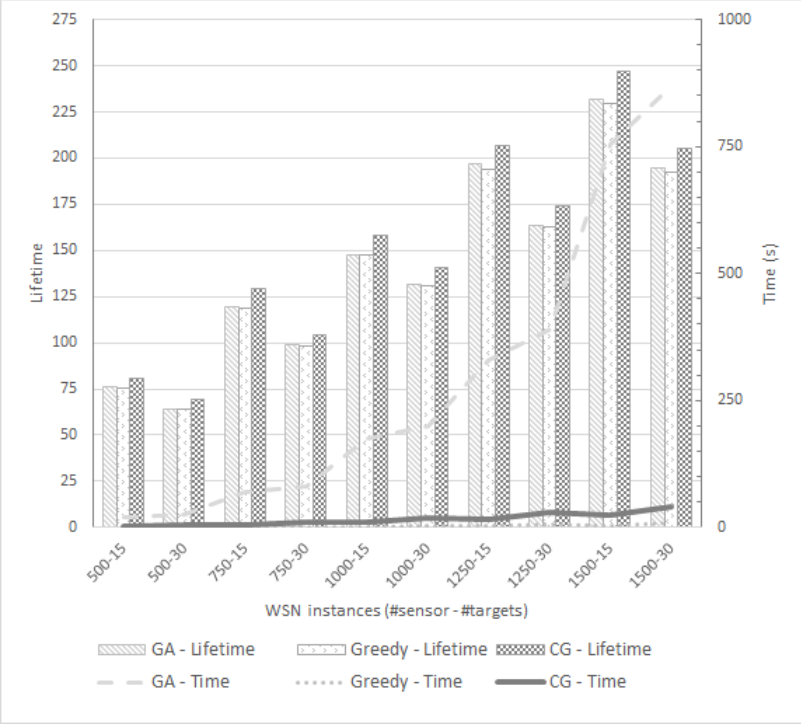


Figure 5.5: solution and running time values, case $\alpha_c = 0.75$, $\tau_s = 0.3$

the targets impacts all the algorithms in the same way, the bigger is the number of the targets the bigger is the computational times. The same trend can be observed on all the algorithms. The computational time of α_c-GA ranges grows up to 2963.60 seconds, that is almost 4.6 times worst than the worst case when $\alpha_c = 1$. The Greedy just doubled the computational time required in the worst case while CG is almost 3.2 times worst. However CG exhibits a computational time such that, for 15 out of 20 scenarios, is lower than 41.33 seconds that is a computational time lower than the worst case when $\alpha_c = 1$. In the case of partial coverage is not available a have a good upper bound that can be used to evaluate the quality of the solutions. So we compared the solutions found among the algorithms in Table 5.4. In such table the first two columns report the characteristics of each scenario in terms of number of sensors and targets. The following multi-columns (α_c-

Table 5.5: Solution and running time values, case $\alpha_c = 0.9$

$\alpha_c = 0,9$			α_c-GA		Greedy		CG	
	$\|\mathbf{S}\|$	$\|\mathbf{T}\|$	LT	Time	LT	Time	LT	Time
$\tau_s = 0.1$	500	15	58.83	46.38	58.32	1.36	63.15	7.08
	500	30	53.22	70.78	52.62	2.32	56.34	12.46
	750	15	97.14	157.42	96.10	3.26	103.71	16.91
	750	30	79.55	208.81	79.53	5.05	85.60	27.76
	1000	15	118.15	388.24	117.40	5.51	127.29	28.49
	1000	30	105.55	396.23	104.47	8.90	112.77	48.32
	1250	15	159.90	662.94	157.58	9.30	167.59	47.31
	1250	30	131.80	899.03	131.33	14.46	139.43	77.59
	1500	15	189.14	1456.82	185.87	12.53	199.02	66.03
	1500	30	156.46	1974.62	154.96	20.39	167.20	111.65
$\tau_s = 0.3$	500	15	52.86	14.08	52.35	0.38	56.58	2.02
	500	30	46.74	21.90	46.44	0.67	49.68	3.61
	750	15	86.94	47.64	85.53	0.91	92.85	4.92
	750	30	72.06	63.85	71.16	1.45	75.90	8.10
	1000	15	106.77	117.58	105.24	1.55	113.55	8.18
	1000	30	93.87	120.26	92.64	2.49	99.69	13.73
	1250	15	139.50	200.42	138.48	2.59	149.31	13.71
	1250	30	114.63	271.63	114.21	4.08	122.85	22.46
	1500	15	167.70	411.62	166.74	3.61	178.41	19.29
	1500	30	138.24	633.51	136.41	5.91	147.27	32.43

GA vs *Greedy*), (*CG* vs *Greedy*) and (*CG* vs α_c-GA) have the same meaning as in Table 5.2. It can be observed from α_c-GA vs *Greedy* multicolumn that the quality of the solution found by α_c-GA is always better than the Greedy, with an improvement that ranges from 0.39% up to 1.27%. However this improvement results to be lower than the case when $\alpha_c = 1$. While looking at *CG vs* α_c*-GA*, it can be seen that CG requires at most the 18.72% of α_c-GA however it finds better solutions than α_c-GA, up to 8.30%. It is possible to note that even if the computational time required by CG is bigger than Greedy, it is at least lower than the 81% of the α_c-GA, furthermore CG finds always better solutions. In the end, looking to *CG vs Greedy* multicolumn can be discovered that CG improves the Greedy solutions from 5.52% up to 9.26%.

Table 5.5 and 5.6 report the comparison of the three proposed approaches when $\alpha_c = 0.9$. Table 5.5 shows the computational data about

Table 5.6: Comparison among GA, Greedy and Carousel, $\alpha_c = 0.9$

$\alpha_c = 0,9$			α_c-GA vs Greedy		CG vs Greedy		Carousel vs α_c-GA	
	$\|\mathbf{S}\|$	$\|\mathbf{T}\|$	GAP %	Time %	GAP %	Time%	GAP %	Time %
$\tau_s = 0.1$	500	15	0.88	2.93	8.28	19.16	7.34	15.27
	500	30	1.14	3.28	7.06	18.60	5.86	17.61
	750	15	1.08	2.07	7.91	19.27	6.76	10.74
	750	30	0.03	2.42	7.64	18.20	7.61	13.30
	1000	15	0.64	1.42	8.42	19.35	7.74	7.34
	1000	30	1.03	2.25	7.94	18.42	6.84	12.19
	1250	15	1.47	1.40	6.35	19.66	4.81	7.14
	1250	30	0.36	1.61	6.17	18.63	5.79	8.63
	1500	15	1.76	0.86	7.08	18.97	5.22	4.53
	1500	30	0.97	1.03	7.90	18.26	6.86	5.65
$\tau_s = 0.3$	500	15	0.97	2.68	8.08	18.74	7.04	14.31
	500	30	0.65	3.06	6.98	18.57	6.29	16.48
	750	15	1.65	1.92	8.56	18.59	6.80	10.32
	750	30	1.26	2.27	6.66	17.89	5.33	12.68
	1000	15	1.45	1.32	7.90	18.98	6.35	6.96
	1000	30	1.33	2.07	7.61	18.16	6.20	11.42
	1250	15	0.74	1.29	7.82	18.92	7.03	6.84
	1250	30	0.37	1.50	7.57	18.17	7.17	8.27
	1500	15	0.58	0.88	7.00	18.72	6.39	4.69
	1500	30	1.34	0.93	7.96	18.22	6.53	5.12

the three algorithms, the same values are also plotted in the figures 5.6 and 5.7. The *x*-axis reports the instance characteristics, while the Lifetime and Time values are reported on the primary and secondary *y*-axis, respectively. The meaning of the Table 5.5 headings are the same of Table 5.3. The structure and the meaning of the heading of Table 5.6 are instead the same of the Table 5.4. The behaviour of the algorithms follows the same trend expressed in the case $\alpha_c = 0.75$. Indeed, looking to α_c-GA vs *Greedy* multicolumn it is possible to observe that the quality of the solution found by α_c-GA is always better than the Greedy, while looking to the multi-columns α_c*-GA vs CG* and *CG vs Greedy* it can be seen that even if the computational time required by CG is bigger than Greedy, CG is more fast and finds better solutions with respect to the α_c-GA.

Table 5.7 allows to highlight the network lifetime improvement obtained by the CG when α_c is equal to 0.75 or 0.9. The first two columns have the same meaning as the previous tables. The third

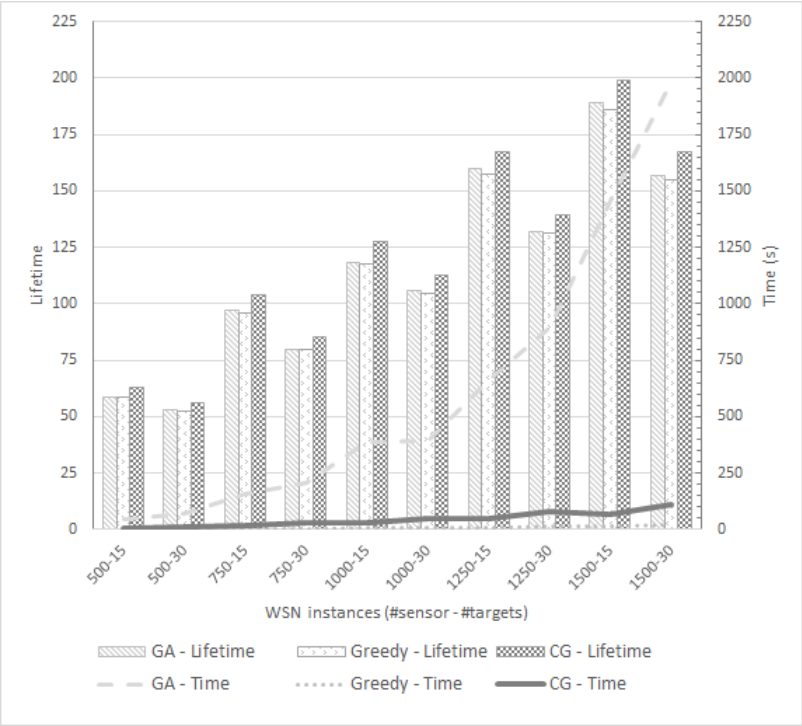


Figure 5.6: solution and running time values, case $\alpha_c = 0.9$, $\tau_s = 0.1$

column, LT shows the average value of network lifetime obtained by the CG in the case $\alpha_c = 1$, while the fourth and sixth columns report the average value of network lifetime obtained by the CG algorithm in the cases $\alpha_c = 0.9$ and $\alpha_c = 0.75$, respectively. The fifth column, $Impr\%$, reports the percentage improvement of the LT obtained when $\alpha_c = 0.9$, compared to the case of total coverage, calculated as $100 \cdot (LT_{\alpha_c=0.9} - LT_{\alpha_c=1})/LT_{\alpha_c=1}$. The seventh column, similarly, reports the percentage improvement of the network lifetime obtained with $\alpha_c = 0.75$ compared to the case of total coverage. It is possible to observe that when $\alpha_c = 0.9$ and $\tau_s = 0.1$, the improvement of the lifetime ranges from a minimum of 42.36% to a maximum of 96.87%, with an average value of 71.61%. Similarly, when $\tau_s = 0.3$ the lifetime improvement goes from 41.80% up to 94.32% and on average it is equal to 69.43%. It is possible to deduce that the neglecting the 10%

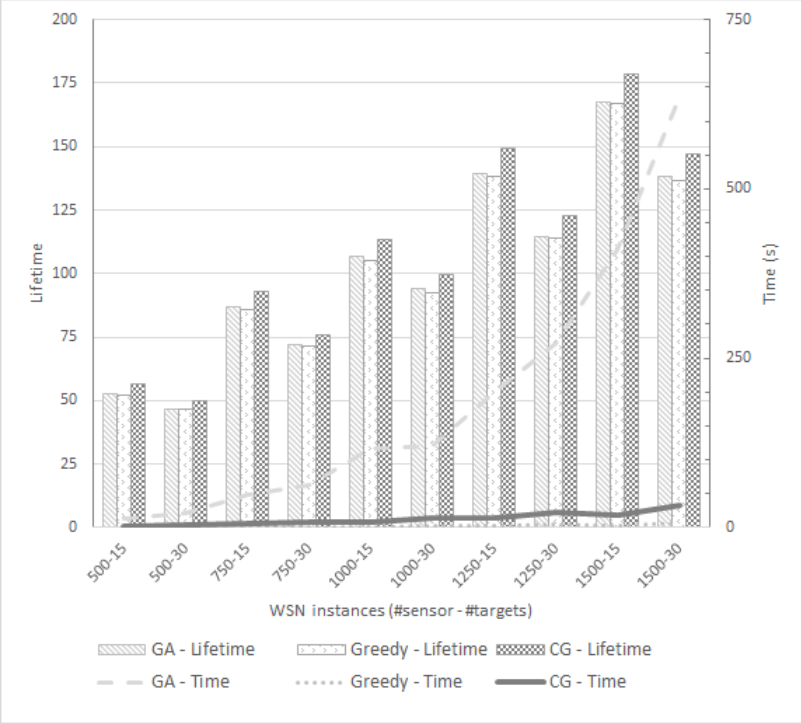


Figure 5.7: solution and running time values, case $\alpha_c = 0.9$, $\tau_s = 0.3$

of the targets, leads an improvement of the network lifetime of about the 70% for both $\tau_s = 0.1$ and $\tau_s = 0.3$. When $\alpha_c = 0.75$ and $\tau_s = 0.1$, the lifetime improvement increases from the 98.90% up to 172.26% with an average value of 139.62%. While, when $\tau_s = 0.3$, the lifetime improvement rages between the 96.45% and 168.90%, whit a mean value of 136.80%. Furthermore, it easy to observe from this table that, τ_s influences the lifetime computed by CG and it is possible to see that the lower τ_s is the bigger is the lifetime.

To deepen the analysis of the lifetime improvement in the case of partial coverage, consider the Table 5.8 that highlights the network lifetime improvement obtained by the α_c-GA, in the cases $\alpha_c = 0.75$ and $\alpha_c = 0.9$. The structure of the Table 5.8 and the meaning of its headings are the same of Table 5.7. When $\alpha_c = 0.9$ and $\tau_s = 0.1$, the lifetime improvements ranges from a minimum of 36.67% to a maxi-

Table 5.7: CG: comparison of lifetime values in function of α_c

			$\alpha_c = 1$	$\alpha_c = 0.9$		$\alpha_c = 0.75$	
	$\|\mathbf{S}\|$	$\|\mathbf{T}\|$	LT	LT	Impr %	LT	Impr %
$\tau_s = 0.1$	500	15	40.1	63.15	57.48	91.19	127.41
	500	30	29.78	56.34	89.19	77.89	161.55
	750	15	59.39	103.71	74.63	145.75	145.41
	750	30	43.48	85.6	96.87	117.44	170.10
	1000	15	85.1	127.29	49.58	177.49	108.57
	1000	30	58.07	112.77	94.20	158.1	172.26
	1250	15	114.6	167.59	46.24	232.61	102.98
	1250	30	74.84	139.43	86.30	195.73	161.53
	1500	15	139.8	199.02	42.36	278.06	98.90
	1500	30	93.3	167.2	79.21	230.88	147.46
$\tau_s = 0.3$	500	15	36.06	56.58	56.91	80.79	124.04
	500	30	26.79	49.68	85.44	69.12	158.01
	750	15	53.46	92.85	73.68	129.6	142.42
	750	30	39.06	75.9	94.32	104.01	166.28
	1000	15	76.53	113.55	48.37	158.31	106.86
	1000	30	52.29	99.69	90.65	140.61	168.90
	1250	15	103.11	149.31	44.81	207.06	100.81
	1250	30	67.26	122.85	82.65	174.36	159.23
	1500	15	125.82	178.41	41.80	247.17	96.45
	1500	30	83.82	147.27	75.70	205.32	144.95

mum of 86.27%, with an average value of 63.65%. When $\tau_s = 0.3$ the lifetime improvement goes from 36.02% up to 86.35% and on average it is equal to 61.09%. So, neglecting the 10% of the targets, the α_c-GA allows to obtain an improvement of the network lifetime on average equal to 62,37% for both $\tau_s = 0.1$ and $\tau_s = 0.3$. In the case $\alpha_c = 0.75$ and $\tau_s = 0.1$, the lifetime improvement increases from the 92.76% up to 163.85% with an average value of 129.27%. While, when $\tau_s = 0.3$, the minimum lifetime improvement is 87.99% and the maximum lifetime improvement is 155.62%, whit a mean value of 125.12%. Similarly to the CG, τ_s influences the lifetime computed by α_c-GA and it can be seen that the lower τ_s is the bigger is the lifetime.

Table 5.8: α_c-GA: comparison of lifetime values in function of α_c

			$\alpha_c = 1$	$\alpha_c = 0.9$		$\alpha_c = 0.75$	
	$\|\mathbf{S}\|$	$\|\mathbf{T}\|$	LT	LT	Impr %	LT	Impr %
$\tau_s = 0.1$	500	15	39.33	58.83	49.59	85.85	118.29
	500	30	29.43	53.22	80.82	74.07	151.66
	750	15	58.44	97.14	66.22	137.03	134.48
	750	30	42.71	79.55	86.27	111.35	160.73
	1000	15	83.64	118.15	41.25	164	96.07
	1000	30	56.99	105.55	85.22	150.36	163.85
	1250	15	112.87	159.9	41.67	217.66	92.84
	1250	30	73.92	131.8	78.31	181.96	146.17
	1500	15	138.39	189.14	36.67	266.77	92.76
	1500	30	91.80	156.46	70.44	216.46	135.80
$\tau_s = 0.3$	500	15	35.65	52.86	48.29	76.26	113.93
	500	30	26.49	46.74	76.46	64.32	142.83
	750	15	52.92	86.94	64.30	119.67	126.15
	750	30	38.67	72.06	86.35	98.85	155.62
	1000	15	75.81	106.77	40.83	147.84	95.00
	1000	30	51.76	93.87	81.37	131.82	154.69
	1250	15	102.00	139.5	36.77	196.62	92.77
	1250	30	66.39	114.63	72.66	163.35	146.05
	1500	15	123.29	167.7	36.02	231.78	87.99
	1500	30	82.36	138.24	67.84	194.52	136.17

5.6 Conclusion

In this chapter we investigated and generalized the maximum lifetime problem with additional time slots constraints (MLPTS) introduced in [106]. We considered the possibility to neglect a small percentage of targets in each subset of active sensors. We defined such new problem as α_c-MLPTS, where α_c represents the percentage of targets that we have to monitor constantly. For this new scenario we proposed two new greedy based approaches, that is a classical greedy algorithm and an enhanced greedy algorithm based on a novel greedy paradigm known as Carousel Greedy. In order to deeply investigate the problem we also considered the implementation of a modified version of the genetic algorithm already proposed for MLPTS in the literature. The

comparison of the three approaches is carried out through an extensive computational test phase. The experimental evaluation shows that the Carousel Greedy results to be the best choice in terms of quality of the solutions and of computational times. Carousel Greedy always finds better solutions than both Greedy and Genetic Algorithm with a computational time that is slightly higher than the computational time required by Greedy but much lower than the computational time required by the Genetic Algorithm. Furthermore, as already proved for the classical MLP, the partial coverage of the targets, when possible, results to be a valid strategy to considerably improve the network lifetime.

Chapter 6

Conclusions

In this research thesis we presented an overview of wireless sensor networks, their applications and typical coverage problems. One of the most important issues in this field is the maximization of the amount of time during which the WSN is able to perform its monitoring task. This problem, generally known as Maximum Lifetime Problem (MLP), is a well known and challenging optimization problem which has been addressed successfully with several approaches in the last years. While a lot of results can be found in literature on the MLP and variants considering classical technical issues (as connectivity and multi-role issues among others), few research effort has been devoted to investigate specific operational requirements of the sensors. In this thesis we focus on such scenario in which, in order to perform the monitoring activity, each sensor must be active for a predefined period of time defined as *operating time slot*. This context characterizes the *periodic sensing applications* in which a WSN monitors the phenomenon under observation according to a sensitivity cycle which is repeated periodically.

We formally defined this problem as Maximum Lifetime Problem with Time Slots (MLPTS). For this new scenario we derived an upper bound on the maximum achievable lifetime and proposed a genetic algorithm for finding a near-optimal node activity schedule. The performance evaluation results obtained on numerous benchmark instances, showed the effectiveness of the proposed approach. Further,

we generalized MLPTS by taking into account the possibility to neglect the coverage of a small percentage of the whole set of targets since, in some applications, the status of the phenomenon under observation, can be estimated or inferred by monitoring even only a subset of all targets. We defined such new problem as α_c-MLPTS, where α_c defines the percentage of targets that the WSN has to monitor in each time slot. For this new scenario we proposed three approaches: a classical greedy algorithm, a modified version of the genetic algorithm already proposed for MLPTS and a Carousel Greedy algorithm. The comparison of the three approaches is carried out through extensive computational experiments. The computational results showed that the Carousel Greedy represents the best trade-off between solutions quality and computational times, and confirm that the network lifetime, also in the case of sensors with operational time constraints, can be considerably improved by omitting the coverage of a modestly percentage of the targets.

To the best of our knowledge, the two problems MLPTS and α_c-MLPTS, variants of the MLP problem, have not been previously studied in the literature. Therefore we believe that this study represents an important contribution in this research area.

With respect to future research there are many directions that can be followed. In particular, several well known design issues deriving from well known variants of the classical MLP problem such as connectivity, routing and robustness (i.e. fault tolerance), which may arise in specific periodic sensing applications will be faced. The use of multiple sensor families, each with a different time slot, will be addressed.

Bibliography

[1] Y. et al., "Internet of things: Wireless sensor networks," International Electrotechnical Commission (IEC), Tech. Rep., 2019.

[2] P. P. Ray, "A survey on internet of things architectures," *Journal of King Saud University-Computer and Information Sciences*, vol. 30, no. 3, pp. 291–319, 2018.

[3] D. E. Kouicem, A. Bouabdallah, and H. Lakhlef, "Internet of things security: A top-down survey," *Computer Networks*, vol. 141, pp. 199–221, 2018.

[4] ReportsnReports, "Global wireless sensor networks (wsn) market size, status and forecast 2020-2026," *https://www.reportsnreports.com/reports/3153645-global-wireless-sensor-networks-wsn-market-size-status-and-forecast-2020-2026.html*, 2020.

[5] MordorIntelligence, "Wireless sensors network market - growth, trends, and forecast (2020 - 2025)," *https://www.mordorintelligence.com/industry-reports/wireless-sensor-networks-market*, 2020.

[6] B. Etikasari, S. Kautsar, H. Riskiawan, D. Setyohadi *et al.*, "Wireless sensor network development in unmanned aerial vehicle (uav) for water quality monitoring system," in *IOP Conference Series: Earth and Environmental Science*, vol. 411, no. 1. IOP Publishing, 2020, p. 012061.

[7] I. Deaconu and A. Voinescu, "Mobile gateway for wireless sensor networks utilizing drones," in *RoEduNet Conference 13th Edition: Net-*

working in Education and Research Joint Event RENAM 8th Conference, 2014. IEEE, 2014, pp. 1–5.

[8] F. Deng, X. Yue, X. Fan, S. Guan, Y. Xu, and J. Chen, "Multisource energy harvesting system for a wireless sensor network node in the field environment," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 918–927, 2018.

[9] H. Sharma, A. Haque, and Z. A. Jaffery, "Maximization of wireless sensor network lifetime using solar energy harvesting for smart agriculture monitoring," *Ad Hoc Networks*, vol. 94, p. 101966, 2019.

[10] A. Abdulla, H. Nishiyama, and N. Kato, "Extending the lifetime of wireless sensor networks: A hybrid routing algorithm," *Computer Communications*, vol. 35, no. 9, pp. 1056–1063, 2012. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84860481685&partnerID=40&md5=67b4869a432f836647d030b76d39e38c

[11] A. Santos, C. Duhamel, L. Belisário, and L. Guedes, "Strategies for designing energy-efficient clusters-based wsn topologies," *Journal of Heuristics*, vol. 18, no. 4, pp. 657–675, 2012. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84865206102&partnerID=40&md5=a48d9d87aab963d68790a586ac87b45c

[12] T. Bokareva, W. Hu, S. Kanhere, B. Ristic, T. Bessell, M. Rutten, and S. Jha, "Wireless sensor networks for battlefield surveillance," in *in Proc. of the Land Warfare Conference*, 2006.

[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*, vol. 52, no. B, pp. 220–230, 2014.

[14] A. Rossi, A. Singh, and M. Sevaux, "Column generation algorithm for sensor coverage scheduling under bandwidth constraints," *Networks*, vol. 60, no. 3, pp. 141–154, 2012.

[15] F. Carrabs, R. Cerrulli, C. D'Ambrosio, and A. Raiconi, *Maximizing Lifetime for a Zone Monitoring Problem Through Reduction to Target Coverage*. Springer International Publishing, 2018, pp. 111–119.

[16] F. Carrabs, R. Cerulli, C. D'Ambrosio, and A. Raiconi, "Exact and heuristic approaches for the maximum lifetime problem in sensor networks with coverage and connectivity constraints," *RAIRO - Operations Research*, vol. 51, no. 3, pp. 607–625, 2017.

[17] 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*, vol. 3, 2005, pp. 1976–1984.

[18] F. Castaño, A. Rossi, M. Sevaux, and N. Velasco, "On the use of multiple sinks to extend the lifetime in connected wireless sensor networks," *Electronic Notes in Discrete Mathematics*, vol. 41, pp. 77–84, 2013. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84879706424&partnerID=40&md5=0ccae4b8dada22daae2da246e9304390

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

[20] D. Zorbas, D. Glynos, P. Kotzanikolaou, and C. Douligeris, "Solving coverage problems in wireless sensor networks using cover sets," *Ad Hoc Networks*, vol. 8, no. 4, pp. 400–415, 2010. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-75149197226&partnerID=40&md5=0fabfc89e2e02624d7fa671bd840d9e0

[21] M. Z. A. Bhuiyan, G. Wang, J. Wu, J. Cao, X. Liu, and T. Wang, "Dependable structural health monitoring using wireless sensor networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 4, pp. 363–376, 2017.

[22] A. B. Noel, A. Abdaoui, T. Elfouly, M. H. Ahmed, A. Badawy, and M. S. Shehata, "Structural health monitoring using wireless sensor networks: A comprehensive survey," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1403–1423, 2017.

[23] C. Tang, H. F. Rashvand, G. Y. Tian, P. Hu, A. I. Sunny, and H. Wang, *Structural Health Monitoring with WSNs*. John Wiley and Sons, Ltd, 2017, ch. 18, pp. 381–408. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119126492.ch18

[24] M. Navarro, T. W. Davis, Y. Liang, and X. Liang, "A study of long-term wsn deployment for environmental monitoring," in *2013 IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Sep. 2013, pp. 2093–2097.

[25] B. Bathiya, S. Srivastava, and B. Mishra, "Air pollution monitoring using wireless sensor network," in *2016 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*, Dec 2016, pp. 112–117.

[26] P. Arroyo, J. L. Herrero, J. I. Suárez, and J. Lozano, "Wireless sensor network combined with cloud computing for air quality monitoring," *Sensors*, vol. 19, no. 3, p. 691, 2019.

[27] "Lufdaten.info," http://www.luftdaten.info.

[28] T. Ojha, S. Misra, and N. S. Raghuwanshi, "Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges," *Computers and Electronics in Agriculture*, vol. 118, pp. 66 – 84, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169915002379

[29] A. P. Abidoye and I. C. Obagbuwa, "Models for integrating wireless sensor networks into the internet of things," *IET Wireless Sensor Systems*, vol. 7, no. 3, pp. 65–72, 2017.

[30] J. Elson and D. Estrin, "Wireless sensor networks," C. S. Raghavendra, K. M. Sivalingam, and T. Znati, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 2004, ch. Sensor Networks: A Bridge to the Physical World, pp. 3–20.

[31] M. Cardei, "Coverage problems in sensors networks," in *Handbook of Combinatorial Optimization (2nd edition)*, P. M. Pardalos, D. Z. Du, and R. Graham, Eds. New York: Springer, 2013, pp. 899–927.

[32] J. Hwang, C. Shin, and H. Yoe, "A wireless sensor network-based ubiquitous paprika growth management system," *Sensors*, vol. 10, no. 12, pp. 11 566–11 589, 2010.

[33] A. A. Pasi and U. Bhave, "Flood detection system using wireless sensor network," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 5, no. 2, 2015.

[34] J. Fernández-Lozano, M. Martín-Guzmán, J. Martín-Ávila, and A. García-Cerezo, "A wireless sensor network for urban traffic characterization and trend monitoring," *Sensors*, vol. 15, no. 10, pp. 26 143–26 169, 2015.

[35] S. J. Ramson and D. J. Moni, "Applications of wireless sensor networks - a survey," in *2017 international conference on innovations in electrical, electronics, instrumentation and media technology (ICEEIMT)*. IEEE, 2017, pp. 325–329.

[36] T. Ojha, S. Misra, and N. S. Raghuwanshi, "Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges," *Computers and Electronics in Agriculture*, vol. 118, pp. 66 – 84, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169915002379

[37] 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*, vol. 68, no. 1, pp. 1–48, 2014.

[38] M. Pejanovic Durisic, Z. Tafa, G. Dimic, and V. Milutinovic, "A survey of military applications of wireless sensor networks," in *Proceedings of the Mediterranean Conference on Embedded Computing*, 2012, pp. 196–199.

[39] I. Dietrich and F. Dressler, "On the lifetime of wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 5, no. 1, 2009. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-60449083692&partnerID=40&md5=1cd3d215145edc8b047962235c55706a

[40] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks,"

in *Proceedings of the 6th annual international conference on Mobile computing and networking*. ACM, 2000, pp. 56–67.

[41] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors," *Communications of the ACM*, vol. 43, no. 5, pp. 51–58, 2000.

[42] S. Megerian, F. Koushanfar, G. Qu, G. Veltri, and M. Potkonjak, "Exposure in wireless sensor networks: theory and practical solutions," *Wireless Networks*, vol. 8, no. 5, pp. 443–454, 2002.

[43] C. Zhu, C. Zheng, L. Shu, and G. Han, "A survey on coverage and connectivity issues in wireless sensor networks," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 619–632, 2012.

[44] B. Wang, "Coverage problems in sensor networks: A survey," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 32, 2011.

[45] M. Cardei and J. Wu, "Coverage in wireless sensor networks," *Handbook of Sensor Networks*, pp. 422–433, 2004.

[46] S. Slijepcevic and M. Potkonjak, "Power efficient organization of wireless sensor networks," vol. 2, 2001, pp. 472–476. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-0034865562&partnerID=40&md5=324c52c8abf6f225ab82abd4d5828450

[47] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[48] M. Cardei, J. Wu, and M. Lu, "Improving network lifetime using sensors with adjustable sensing ranges," *International Journal of Sensor Networks*, vol. 1, no. 1-2, pp. 41–49, 2006.

[49] M. Gentili and A. Raiconi, "$\alpha-$coverage to extend network lifetime on wireless sensor networks," *Optimization Letters*, vol. 7, no. 1, pp. 157–172, 2013.

[50] Y. Gu, B.-H. Zhao, Y.-S. Ji, and J. Li, "Theoretical treatment of target coverage in wireless sensor networks," *Journal of Computer Science and Technology*, vol. 26, no. 1, pp. 117–129, 2010. [Online]. Available: http:

//www.scopus.com/inward/record.url?eid=2-s2.0-79951504703&partnerID=40&md5=b6e2c0bb45d3d4d93e9c5c892b6976fa

[51] A. Singh, A. Rossi, and M. Sevaux, "Matheuristic approaches for q-coverage problem versions in wireless sensor networks," *Engineering Optimization*, vol. 45, no. 5, pp. 609–626, 2013. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84876401704&partnerID=40&md5=94e6c2758a3564f68e95f189ccc26b6d

[52] S. Slijepcevic and M. Potkonjak, "Power efficient organization of wireless sensor networks," in *IEEE International Conference on Communications*, vol. 2, 2001, pp. 472–476.

[53] 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*, vol. 4, 2011, pp. 209 – 214.

[54] 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*, vol. 220, no. 1, pp. 58–66, 2012.

[55] 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*, 2007, pp. 1118–1123.

[56] R. Cerulli, M. Gentili, and A. Raiconi, "Maximizing lifetime and handling reliability in wireless sensor networks," *Networks*, vol. 64, no. 4, pp. 321–338, 2014.

[57] 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," *Journal of Network and Computer Applications*, vol. 58, pp. 12–22, 2015.

[58] ——, "Extending lifetime through partial coverage and roles allocation in connectivity-constrained sensor networks," *IFAC-PapersOnline*, vol. 49, no. 12, pp. 973–978, 2016.

[59] ——, "Extending lifetime through partial coverage and roles allocation in connectivity-constrained sensor networks," *IFAC-PapersOnline*, vol. 49, no. 12, pp. 973–978, 2016.

[60] H. Zhang and J. C. Hou, "Maintaining sensing coverage and connectivity in large sensor networks," *Ad Hoc & Sensor Wireless Networks*, vol. 1, no. 1-2, pp. 89–124, 2005.

[61] F. Carrabs, C. Cerrone, C. D'Ambrosio, and A. Raiconi, "Column generation embedding carousel greedy for the maximum network lifetime problem with interference constraints," in *Optimization and Decision Science: Methodologies and Applications. ODS 2017.*, ser. Springer Proceedings in Mathematics & Statistics, S. A. and S. C., Eds., vol. 217. Springer, Cham, 2017, pp. 151–159.

[62] F. Carrabs, R. Cerulli, C. D'Ambrosio, and A. Raiconi, "Prolonging lifetime in wireless sensor networks with interference constraints," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10232 LNCS, pp. 285–297, 2017.

[63] C. Cerrone, R. Cerulli, and B. Golden, "Carousel greedy: A generalized greedy algorithm with applications in optimization," *Computers and Operations Research*, vol. 85, pp. 97 – 112, 2017.

[64] K. Trojanowski, A. Mikitiuk, F. Guinand, and M. Wypych, "Heuristic optimization of a sensor network lifetime under coverage constraint," in *Computational Collective Intelligence*, N. T. Nguyen, G. A. Papadopoulos, B. Jkedrzejowicz, Piotrand Trawinski, and G. Vossen, Eds. Cham: Springer International Publishing, 2017, pp. 422–432.

[65] K. Trojanowski, A. Mikitiuk, and M. Kowalczyk, "Sensor network coverage problem: A hypergraph model approach," in *Computational Collective Intelligence*, N. T. Nguyen, G. A. Papadopoulos, P. Jedrzejowicz, B. Trawinski, and G. Vossen, Eds. Cham: Springer International Publishing, 2017, pp. 411–421.

[66] K. Trojanowski, A. Mikitiuk, and K. J. M. Napiorkowski, "Application of local search with perturbation inspired by cellular automata for

heuristic optimization of sensor network coverage problem," in *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, E. Deelman, and K. Karczewski, Eds. Cham: Springer International Publishing, 2018, pp. 425–435.

[67] A. Tretyakova, F. Seredynski, and F. Guinand, "Heuristic and meta-heuristic approaches for energy-efficient coverage-preserving protocols in wireless sensor networks." New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3132114.3132119

[68] K. Brezinski, M. Guevarra, and K. Ferens, "Population based equilibrium in hybrid sa/pso for combinatorial optimization: Hybrid sa/pso for combinatorial optimization," *International Journal of Software Science and Computational Intelligence*, vol. 12, pp. 74–86, 04 2020.

[69] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated annealing*. Dordrecht: Springer Netherlands, 1987, pp. 7–15. [Online]. Available: https://doi.org/10.1007/978-94-015-7744-1_2

[70] E. Fitzgerald, M. Pioro, and A. Tomaszewski, "Network lifetime maximization in wireless mesh networks for machine-to-machine communication," *Ad Hoc Networks*, vol. 95, p. 101987, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870519303415

[71] C. Francesco, C. D'Ambrosio, and A. Raiconi, "Optimization of sensor battery charging to maximize lifetime in a wireless sensors network," *Optimization Letters*, 2020.

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

[73] S. Jevtic, M. Kotowsky, R. P. Dick, P. Dinda, and C. Dowding, "Lucid dreaming: Reliable analog event detection for energy-constrained applications," 04 2007, pp. 350–359.

[74] X. Vilajosana, P. Tuset-Peiro, F. Vazquez-Gallego, J. Alonso-Zarate, and L. Alonso, "Standardized low-power wireless communication technologies for distributed sensing applications," *Sensors*, vol. 14,

no. 2, pp. 2663–2682, 2014. [Online]. Available: https://www.mdpi.com/1424-8220/14/2/2663

[75] K. Chintalapudi, T. Fu, J. Paek, N. Kothari, S. Rangwala, J. Caffrey, R. Govindan, E. Johnson, and S. Masri, "Monitoring civil structures with a wireless sensor network," *IEEE Internet Computing*, vol. 10, no. 2, pp. 26–34, March 2006.

[76] Jeongyeup Paek, K. Chintalapudi, R. Govindan, J. Caffrey, and S. Masri, "A wireless sensor network for structural health monitoring: performance and experience," in *The Second IEEE Workshop on Embedded Networked Sensors, 2005. EmNetS-II.*, May 2005, pp. 1–9.

[77] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *2007 6th International Symposium on Information Processing in Sensor Networks*, April 2007, pp. 254–263.

[78] M. M. Ahmadi and G. A. Jullien, "A wireless-implantable microsystem for continuous blood glucose monitoring," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 3, no. 3, pp. 169–180, June 2009.

[79] J. Holland, "Adaption in natural and artificial systems," *Ann Arbor MI: The University of Michigan Press*, 1975.

[80] S. Sivanandam and S. Deepa, *Introduction to genetic algorithms*. Springer, 2008. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84890007909&partnerID=40&md5=9428a3197d84e5b2265771201673a6a2

[81] L. N. d. Castro, *Fundamentals of Natural Computing (Chapman & Hall/Crc Computer and Information Sciences)*. Chapman & Hall/CRC, 2006.

[82] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[83] R. C. Prim, "Shortest connection networks and some generalizations," *The Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, 1957.

[84] R. L. Graham and P. Hell, "On the history of the minimum spanning tree problem," *Annals of the History of Computing*, vol. 7, no. 1, pp. 43–57, 1985.

[85] J. Nešetřil, "A few remarks on the history of mst-problem," *Archivum mathematicum*, vol. 33, no. 1, pp. 15–22, 1997.

[86] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[87] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, vol. 4, no. 3, pp. 233–235, 1979.

[88] E. L. Lawler, J. K. Lenstra, A. H. R. Kan, and D. B. Shmoys, "Sequencing and scheduling: Algorithms and complexity," *Handbooks in operations research and management science*, vol. 4, pp. 445–522, 1993.

[89] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[90] C. Cerrone, C. D'Ambrosio, and A. Raiconi, "Heuristics for the strong generalized minimum label spanning tree problem," *Networks*, vol. 74, no. 2, pp. 148–160, 2019.

[91] C. Duin and S. Voß, "The pilot method: A strategy for heuristic repetition with application to the Steiner problem in graphs," *Networks*, vol. 34, no. 3, pp. 181–191, 1999.

[92] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *Journal of global optimization*, vol. 6, no. 2, pp. 109–133, 1995.

[93] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 2033–2049, 2007.

[94] S.-C. Hu, Y.-C. Wang, C.-Y. Huang, and Y.-C. Tseng, "Measuring air quality in city areas by vehicular wireless sensor networks," *Journal of*

Systems and Software, vol. 84, no. 11, pp. 2005 – 2012, 2011, mobile Applications: Status and Trends.

[95] B. Bathiya, S. Srivastava, and B. Mishra, "Air pollution monitoring using wireless sensor network," in *2016 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*, Dec 2016, pp. 112–117.

[96] L. Davis, Ed., *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.

[97] A. Konak, D. Coit, and A. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering and System Safety*, vol. 91, no. 9, pp. 992–1007, 9 2006.

[98] T. Popoviciu, "Sur les équations algébriques ayant toutes leurs racines réelles," *Mathematica*, vol. 9, pp. 129–145, 1935.

[99] L. N. d. Castro, *Fundamentals of Natural Computing (Chapman & Hall/Crc Computer and Information Sciences)*. Chapman & Hall/CRC, 2006.

[100] S. M. Dehnavi, M. Ayati, and M. R. Zakerzadeh, "Three dimensional target tracking via underwater acoustic wireless sensor network," in *2017 Artificial Intelligence and Robotics (IRANOPEN)*, 2017, pp. 153–157.

[101] J. Luo, Z. Zhang, C. Liu, and H. Luo, "Reliable and cooperative target tracking based on wsn and wifi in indoor wireless networks," *IEEE Access*, vol. 6, pp. 24 846–24 855, 2018.

[102] N. Dey, A. S. Ashour, F. Shi, S. J. Fong, and R. S. Sherratt, "Developing residential wireless sensor networks for ecg healthcare monitoring," *IEEE Transactions on Consumer Electronics*, vol. 63, no. 4, pp. 442–449, 2017.

[103] A. Díaz-Ramírez, F. A. Bonino, and P. Mejía-Alvarez, *Human Detection and Tracking in Healthcare Applications Through the Use of a Network of Sensors*. Cham: Springer International Publishing, 2014, pp. 171–190. [Online]. Available: https://doi.org/10.1007/978-3-319-10807-0_8

[104] S. Tennina, M. Santos, A. Mesodiakaki, P. . Mekikis, E. Kartsakli, A. Antonopoulos, M. Di Renzo, A. Stavridis, F. Graziosi, L. Alonso, and C. Verikoukis, "Wsn4qol: Wsns for remote patient monitoring in e-health applications," in *2016 IEEE International Conference on Communications (ICC)*, 2016, pp. 1–6.

[105] M. R. Garey and D. S. Johnson, "Computers and intractability," *A Guide to the*, 1979.

[106] C. D'Ambrosio, A. Iossa, F. Laureana, and F. Palmieri, "A genetic approach for the maximum network lifetime problem with additional operating time slot constraints," *Soft Computing*, 2020.

[107] C. Cerrone, R. Cerulli, and M. Gaudioso, "Omega one multi ethnic genetic approach," *Optimization Letters*, vol. 10, no. 2, pp. 309–324, Feb 2016. [Online]. Available: https://doi.org/10.1007/s11590-015-0852-0

[108] D. B. Fogel, *Evolutionary computation: toward a new philosophy of machine intelligence*. John Wiley & Sons, 2006, vol. 1.

[109] D. E. Goldberg, *Genetic algorithms*. Pearson Education India, 2006.

[110] J. H. Holland *et al.*, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[111] V. Toğan and A. T. Daloğlu, "An improved genetic algorithm with initial population strategy and self-adaptive member grouping," *Computers & Structures*, vol. 86, no. 11-12, pp. 1204–1218, jun 2008.