
UNIVERSITY OF SALERNO
DEPARTMENT OF COMPUTER SCIENCE

PhD in COMPUTER SCIENCE
CYCLE XIII

PhD THESIS IN COMPUTER SCIENCE

# Coverage in Wireless Sensor Networks: Models and Algorithms

**Candidate**
Ciriaco D'Ambrosio

**Tutor**
Prof. Raffaele Cerulli

**Co-Tutor**
Dr. Francesco Carrabs

**Coordinator**
Prof. Giuseppe Persiano

2013/2014

*To my wonderful family*

# Acknowledgements

During my PhD I had the great pleasure to work with wonderful and professional people and I wish to express my gratitude to everybody for the great research experience that they have granted me. I would like to thank Professor Raffaele Cerulli, my wise tutor, for his help, his many precious suggestions and his constant encouragement during my PhD He was for me a real guide. I would like also to express my great gratitude to my exceptional co-tutor, Francesco Carrabs, for his collaboration, his patience and for teaching me to always pursue my goals. He was for me a precious point of reference. I am also very grateful to Andrea Raiconi for his guidance, his patience and his fundamental contributions to this research work. He demonstrated great dedication to help me in this research. I am also very thankful to Monica Gentili for her collaboration and her professional suggestions. Finally, I thank all my family and my dear friend Arcangelo Castiglione for their support and constant presence, and last but not least, I want to thank Sara with whom I shared every moment of my PhD.

# Abstract

Due to technological advances which enabled their deployment in relevant and diverse scenarios, Wireless Sensor Networks (WSNs) have been object of intense study in the last few years. Possible application contexts include environmental monitoring, traffic control, patient monitoring in healthcare and intrusion detection, among others (see, for example, [4], [16], [37]). The general structure of a WSN is composed of several hardware devices (*sensors*) deployed over a given region of interest. Each sensor can collect information or measure physical quantities for a subregion of the space around it (its *sensing area*), and more in particular for specific points of interest (*target points* or simply *targets*) within this area. The targets located in the sensing area of a given sensor s are *covered* by s. Individual sensors are usually powered by batteries which make it possible to keep them functional for a limited time interval, with obvious constraints related to cost and weight factors. Using a network of such devices in a dynamic and coordinated fashion makes it possible to overcome the limitations in terms of range extension and battery duration which characterize each individual sensor, enabling elaborate monitoring of large regions of interest. Extending the amount of time over which such monitoring activity can be carried out represents a very relevant issue. This problem, generally known as Maximum Lifetime Problem (MLP), has been widely approached in the literature by proposing methods to determine: i) several subsets of sensors each one able to provide coverage for the target points and ii) the activation time of these subsets so that the battery constraints are satisfied. It should be noted that while sensors could be considered

as belonging to different states during their usage in the intended application (such as receiving, transmitting, or idle) in this context two essential states can be identified. That is, each sensor may currently be active (i.e. used in the current cover, and consuming its battery) or not. Activating a cover refers therefore to switching all its sensors to the active state, while switching off all the other ones. This research thesis shows a detailed overview about the wireless sensor networks, about their applications but mainly about typical coverage issues in this field. In particular, this work focuses on the issue of maximizing the amount of time over which a set of points of interest (target points), located in a given area, can be monitored by means of such wireless sensor networks. More in detail, in this research work we addressed the maximum lifetime problem on wireless sensor networks considering the classical problem in which all targets have to be covered (classical MLP) and a problem variant in which a portion of them can be neglected at all times ($\alpha$-MLP) in order to increase the overall network lifetime. We propose an Column Generation approach embedding an efficient genetic algorithm aimed at producing new covers. The obtained algorithm is shown to be very effective and efficient outperforming the previous algorithms proposed in the literature for the same problems. In this research work we also introduce two variants of MLP problem with heterogeneous sensors. Indeed, wireless sensor networks can be composed of several different types of sensor devices, which are able to monitor different aspects of the region of interest including temperature, light, chemical contaminants, among others. Given such sensor heterogeneity, different sensor types can be organized to work in a coordinated fashion in many relevant application contexts. Therefore in this work, we faced the problem of maximizing the amount of time during which such a network can remain operational while assuring globally a minimum coverage for all the different sensor types. We considered also some global regularity conditions in order to assure that each type of sensor provides an adequate coverage to each target. For both these problem variants we

developed another hybrid approach, which is again based on a column generation algorithm whose subproblem is either solved heuristically by means of an appropriate genetic algorithm or optimally by means of ILP formulation. In our computational tests the proposed genetic algorithm is shown to be able to meaningfully speed up the global procedure, enabling the resolution of large-scale instances within reasonable computational times. To the best of our knowledge, these two problem variants has not been previously studied in the literature.

# Contents

# List of Figures

# Nomenclature

**List of Abbreviations**

$\alpha$-$MLP$ Maximum Network Lifetime Problem with partial coverage constraint

$CG$ Column Generation

$CG + GA$ Exact Approach for $MLMFP$

$CGonly$ Pure Exact Approach for $MLMFP$

$Exact$ Deschinkel Exact Approach for $MLP$

$GA$ Genetic Algorithm

$GCG$ Genetic Column Generation, exact approach for $\alpha$-$MLP$

$GR$ Gentili and Raiconi Approach for $\alpha$-$MLP$

$Heur$ Deschinkel Heuristic Approach for $MLP$

$ILP$ Integer Linear Programming

$MEMS$ Micro-Electro-Mechanical Systems

$Mixed$ Deschinkel Mixed Approach for $MLP$

$MLMFP$ Maximum Lifetime with Multiple Families Problem

$MLMFP - R$ Regular Maximum Lifetime with Multiple Families Problem

$MLP$ Maximum Network Lifetime Problem

## LIST OF FIGURES

$P$ Master Problem

$QoS$ Quality of Service

$RP$ Restricted Master Problem

$SP$ Separation Problem

$WSN$ Wireless Sensor Network

# Chapter 1

# Introduction

## 1.1 Wireless Sensor Network: Motivation

Wireless sensors networks (WSNs) were presented as one of the most promising technology that would change the world [88]. In the last ten years a lot of research have been conducted in this field and nowadays there is a growing interest in this technology. One could mention, for instance, the popularity of the recent technology concept known as *"The Internet of Things"* [7] [81], that is based on networks of smart objects which are globally connected to the Internet and include modular sensors as well as other technologies to collect and process information on the environment around them. There is therefore a current increasing economic interest for all smarter technologies and in particular for Wireless Sensor Networks, which justifies expectations of growing investments, as mentioned in some recent market research reports to industries, such as *"Semiconductor Wireless Sensor Networks Markets at $ 2.7 billion in 2013 are forecast to reach $ 12 billion Worldwide by 2020"* [87]. In today's world where people are constantly surrounded by smartphones, smart applications and smart objects, the sensor networks are constantly more and more widely used. Their main objective is to reach a better knowledge of the events occurring in the environment around people and in which people live. Sensor networks are an essential tool to better study the impact that natural and man-made phenomena may have on the environment including, the effects on climate, pollution, safety and many other

aspects. Such a technology is also a useful tool for scientists interested in studying some events and physics phenomena which still remain difficult to fully comprehend and predict. One could think of natural disasters such as earthquake, flood and so on. In these scenarios such networks are very useful, since they are able to constantly monitor the environment in unattended manner. In such scenarios the sensors may be deployed and controlled by flying drones, enabling secure remote control of a scenario without direct risks for the humans. Thanks to this scientific and economic interest, today Wireless Sensor Networks can be considered a mature technology even if there are some technology constraints, mainly related to the energy resources, that encouraged the research world to study and find new solutions. In these last years, the battery technology evolution has been less impressive than in others, such as micro electronic systems and even if there are new system that allow sensors to obtain a certain quantity of energy from the environment [104] [109], it remains a critical resource and should be used carefully. The constrained energy resource of a sensor is therefore one of the main issues in order to prolong as much as possible the sensor lifetime. Indeed, if a sensor depletes its energy resources, it generally becomes useless because it is difficult to supply additional energy to such sensors. These limits severely affect the availability of the network services. Prolonging the network operational time is a basic requirement in the design of a wireless network in terms of architecture, hardware and algorithms for the management of the sensors. For this reason, this line of research has been object of intense studies, which led to the design of heuristic and approximate algorithms, among others, based on distributed or centralized approaches. The time for which a network is able to guarantee the monitoring activities is typically called Network Lifetime. Besides energy limitations, the networks are also subject to others environment constraints, such as coverage or connectivity constraints. The mostly used approaches for energy efficiency can be divided into two main families, known as power-aware scheduling and duty scheduling of the sensors activities. A power-aware configuration algorithm aims to identify network configurations able to minimize the energy consumption associated with the network operations/tasks. They can be generally adopted in structured network scenarios with low-density sensors as discussed [1] [11] [94]. The second approach principally aims to assign a working status to a subset of

sensors able to satisfy the network coverage requirements for a proper amount of time. Only a subset of sensors is active at any given time, avoiding to waste the energy of unnecessary nodes. This technique is widely used for high density network scenarios, for instance in adverse scenarios with hostile environments in which a precise placement of the sensors is not possible, nor is possible to supply additional energy resources [17] [30] [90]. Even if the approaches of the first type such as routing schemes and power aware nodes configurations try to address the problem of extending the network lifetime as well, they suffer of other problems such as the well known "Hot-spot Problem" as reported in [1]. The approaches of the second type are generally preferable because they provide comprehensive solutions that are not affected by this type of issue. Generally, in unstructured networks with high density of sensors, there is a large number of possible subsets of sensors able to satisfy the coverage constraints, named *feasible covers* or simply *covers*. Therefore the maximum lifetime can be found by searching these covers and activating them, one at a time, for proper amounts of time. This approach is known as "duty scheduling", and plans to design the activities of the sensors in the network to ensure a monitoring for as long as possible [22] [30] [29] [54] [84] [112]. Starting from these basic motivations, in this thesis, we study the current literature about the Wireless Sensor Networks to design and propose exact efficient approaches for the Maximum Network Lifetime Problem under different coverage constraints. Motivated by heterogeneity of modern networks such as the *Internet of Things*, we investigated two variants of the basic problem in the case of heterogeneous sensor networks.

## 1.2 Contributions of this thesis

This thesis consists of an in-depth study on wireless sensor network on their applications and on typical coverage issues in this field. Particular focus is dedicated to the problem of maximizing the amount of time over which a set of points of interest (target points) located in a given geographic region can be monitored by means of a wireless sensor network. The problem is well known in the literature as the Maximum Network Lifetime Problem (MLP). We focused mainly on an algorithmic aspect of the problem rather than on a technology aspect. It is exam-

ined the column generation technique and how to apply it to the mathematical formulation MLP and three problem variants. It is showed how to embed in the column generation a genetic meta-heuristic aimed at solving the separation problem, that is shown to be very efficient for all the considered problem variants. The problems studied in this work include the $\alpha$-MLP problem, a variant in which a subset of sensors can be left uncovered. However, since $\alpha$-MLP is a generalization of MLP, our algorithms can be used to solve the classical problem as well. As will be shown in the discussion of our computational tests our algorithm is proven to be highly efficient with respect to the previous algorithms available in the literature. The other two studied MLP variants are related to heterogeneous sensors. Today wireless sensor networks are generally composed of several different types of sensor devices, which are able to monitor different aspects of the region of interest (including sound, vibrations, chemical contaminants, among others) and may be deployed together in a heterogeneous network. In this work, we address also the problem of maximizing the amount of time during which such a network can remain operational while assuring globally a minimum coverage for all the different sensor types. The second problem variant in this context also considers some global regularity conditions, in order to guarantee a fair coverage for each sensor type to each target. In our computational tests the proposed resolution approach is shown to be very effective, enabling the resolution of large-scale instances within reasonable computational times.

## 1.3 Organization of Dissertation

This work is organized as follows:

- Chapter 2 introduces some general concepts on wireless sensor networks and their functionalities. This chapter also presents the main sensor coverage models, some coverage issues and the related design choices. Then are shown some aspects of coverage optimization and the main literature results related to this field are discussed.

- Chapter 3 introduces a mathematical formulation of the MLP. Then it shows how to apply the column generation technique to solve the problem. It is

shown how to hybridize such an exact approach with heuristics. Finally, the chapter presents an overview on genetic algorithms.

- Chapter 4 presents the hybrid exact approach for the MLP and $\alpha$-MLP problems. It describes all the building blocks of specialized genetic metaheuristic that we designed to solve the separation problem of the Column Generation approach.

- Chapter 5 illutrates our research work on two novel variants of the MLP problem defined on heterogeneous sensor networks. Starting from the description of the two problems, we present the related mathematical formulations, the hybrid exact approaches we developed to solve both of them and the results of our computational tests.

- Chapter 6 includes our conclusions on these works and a summary of the obtained results. It shows also some future research directions about our considered field of research.

## 1.4 Reading this document

This manuscript is structured in such a way that the six chapters can be read independently. This first chapter briefly introduces the reader to this work and its structure. The readers interested in the research content may directly refer to Chapter 4 and Chapter 5. Chapter 4 refers to homogeneous wireless sensor network with global or partial coverage requirements, and, it corresponds to a work submitted to Journal of Network and Computer Applications. Chapter 5 refers to heterogeneous wireless sensor networks, and it corresponds to a work published on Computers and Operations Research. Chapter 2 and Chapter 3 show the essential background for the research content, therefore the readers interested to the theory and the techniques adopted may start by reading Chapter 2 for an general introduction to the Wireless Sensor Networks and Chapter 3 for a detailed description of the column generation and its application to the problem as well as an introduction on genetic algorithms.

# Chapter 2

# Coverage Optimization in Wireless Sensor Networks

## 2.1 Introduction

Wireless Sensor Networks nowadays are one of the most advanced systems able to collect and process information from the environment. Wireless Sensor Networks perform monitoring through the installation of a significant number of sensors (or sensor nodes) that detect, store and communicate local information that will be eventually used to make global decisions on the environment. Unlike traditional computer systems that process data and information produced by men, WSNs deal with information coming from the environment in which they are installed [44] [21]. This growing "symbiosis" between the world and this innovative technology has attracted and stimulated the interest of many researchers. The technological improvements, in recent years, in the field of micro-electro-mechanical systems (MEMS), digital electronics and in the field of wireless communications, among others, allowed the wireless sensor networks to achieve, today, the state of very mature technology. The miniaturization of computing and sensing devices encouraged the development of a type of network with a very wide range of applications. Early research in this field was dictated by military applications such as acoustic surveillance and target detection. Today there are systems for 360-degree monitoring, systems for the monitoring and protection of civil infras-

tructures such as bridges, tunnels, meeting areas, power grids, water networks and pipelines. In particular sensor networks have already been used for pollution control, flooding, for the control of health, and in agriculture for the control of the use of fertilizers, pesticides as well as for the control of natural water usage in order to improve the crop's health and production. Another interesting field of application is the traffic control, some applications include the installation of sensors along the main streets and on cars in order to control and improve traffic flow and avoid jams. These and many other applications from general monitoring, to healthcare, and even more generally to national security are described and surveyed in [3] [107] [4] [36] [16] [37] and many other works. Since when the first sensor network prototypes have been proposed, such as Smart Dust [66], the research on wireless sensor networks has raised many optimization problems due to both the natural development requirements of sensor networks and the needs of developing increasingly large and efficient networks; recall for instance the IrisNet (Internet-scale Resource-Intensive Sensor Network Services) project at Intel Research, an ideal framework for a worldwide heterogeneous sensor web [55]. An important class of problems is known as *Coverage Problems*. According to the application of the network the concept of *coverage* can be defined through different points of view. Generally given a set of targets of interest and a set of coverage constraints, the main goal of the coverage problems is to have each target of interest under monitoring with respect to the coverage constraints. The concept of *coverage* expresses how well a physical space is monitored by a sensor network [22] and its evaluation expresses the quality of service (QoS)[78] offered by the network. There are different formulations of the coverage problems related to different aspects of the network such as the type of covered object (area, target), the type of sensor placement (random or deterministic) and based on many other properties. For example, one of the most important sensor/network characteristics is the *energy consumption* that strongly influences the time interval for which the network is able to satisfy the application for which it was designed, i.e. the *network lifetime*. Indeed it is one of the most critical aspects of the WSN applications and one of the most studied properties. There are different application dependent definitions in the literature. One of the first definitions considers the lifetime as the time interval between when the network starts to operate and

the first node fails [95]. This definition, as others, doesn't take in consideration any particular characteristic of the coverage. Today there are various definitions based on the coverage, on the node availability, and others characteristics, all definition can be reviewed in [42]. In this thesis, the network lifetime is the time interval for which the network is able to meet the specific coverage requests of the application. It is the most general definition which is suited to coverage problems. If a network cannot guarantee the desired coverage request then it is considered as not being operational. It is well known that the energy efficiency is one of the issues that generally arise in all kinds of wireless sensor networks and that affects the network lifetime. The networks are composed of sensors of limited size and weight. These structural constraints severely affect the availability of services offered by the WSN. Since sensors generally have limited energy resources and limited communication features. If a sensor depletes its energy resources, it becomes useless since it is difficult to supply additional energy to it. For these reasons, prolonging the life time of a sensor network, as much as possible, by finding patterns of energy preservation is one of the main objectives to be met in order to achieve energy efficiency. This chapter aims to introduce general information about sensor networks and the basic technical details for the formulation of coverage problems with special emphasis on the design of energy efficient networks. In particular, we will focus on the problem of covering targets with known positions and energy constraints.

## 2.2 Wireless Sensor Networks Overview

The applications listed in the previous introduction (Sec 2.1) are only a subset of all possible uses of WSN. The underlying technology for sensor networks may vary for architectures and functionalities, but there are some common characteristics and features. The main common characteristic is the *integration* with the environment. A WSN is integrated in the phenomenon or environment for which it was developed. The sensors register the phenomenon in which they are immersed, they monitor the target points in surrounding space, they communicate with each others, transmitting detailed informations on the environment under monitoring. The analysis of the environment, through the capture of light,

temperature, sound, vibrations and so on, can reveal the nature of the physical space. The word *"sensing"* includes all the measurement activities and control of status changes of the phenomenon or of the environment under monitoring. Measurement, preliminary elaboration and reporting of the sensed information are in short the main objectives of the small devices called sensors. In this work we will use the term *sensor* and *sensor node* interchangeably. However to be precise, a *sensor node* is the sensing unit which includes with tools such as battery, antenna, possible actuators among others, while sensor refers only to the actual hardware device which is able to perceive the status changes of the phenomenon. Another well known term used to refer to a sensor is *transducer*, i.e. a system able to transform physical quantities into electrical signals. For more details on the structure of a sensor node, see Figure 2.1. As shown in the figure, its main four components are a sensing unit, a processing unit, a transceiver unit and a power unit. Obviously, the sensing unit can vary depending on what is to be monitored. In addition the sensor may be equipped, depending on the application, with additional components such as tracking systems, eventual generators of additional energy [109] and also actuators. The analog signals, perceived from the environment by the "sensor" subcomponent of the sensing unit, are converted to digital, by the "ADC" unit. Once converted to digital, the signals are passed to the "Processing Unit". The processing unit, typically coupled to a "Storage Unit", manages the procedures and scheduling activities required to efficiently collaborate with the other nodes in the network. The "Power Unit" is among the most important ones and as previously said is sometimes supported by specific hardware ("Power Generator" in the Figure 2.1) to obtain energy from the environment, see for example [109]. Some sensor nodes use location mechanisms through which the sensor identifies its position in space and/or actuators that allow to complete more complex sensing tasks. The sensors, as reported in Figure 2.1, can be equipped with a transceiver for the network connection through RF or optical drives. All these components are subject to tight dimensional constraints. Indeed, in some cases a sensor node is contained in a box [64] of a cubic centimeter [83]. They are also subject to many other general constraints such as low power consumption, operational capacities in high density condition, low production costs, absolute autonomous control and adaptation to the environment.

Given such constraints it is straightforward to understand that the lifetime of a WSN strongly depends on the battery duration of the sensors that compose it. Generally sensors can be either *static* or *mobile*, however in this work we focus on static nodes. A static (or fixed) node does not change its position, and represents the most common case. Conversely a mobile sensor may be equipped with mobility systems [33] or be positioned on moving devices such as robots or vehicles [34] [73].

### 2.2.1 Sensor Coverage Models

Sensors nodes may have many different characteristics from both a physical and a theoretical point of view. Taking into account the various aspects described in the introduction it is straightforward to note that different sensing models may be adopted. The sensing models can be based on the environment, on the design choices and on the application requirements and they express a measure of the sensing ability and its quality, evaluating the relation among the environment, the sensors and the targets. As reported in [77] and remarked later in [111] and other works, the sensor nodes typically have two intrinsic common characteristics: (1) the quality of sensing diminishes as distance increases (2) the sensing ability improves as the sensing time increases. Typically sensing models can be expressed in function of the *Euclidean Distance* among target points and sensors. We generally consider the concept of coverage, as in the majority of the works cited in this thesis, in the two dimensional space. Given a point $z$ in the space under monitoring and the set of sensors $S = \{s_1, s_2, ..., s_n\}$, spread over the area of interest, the Euclidean Distance between a generic sensor $s$ and point $z$ can be expressed as follows:

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

where obviously $s$ and $z$ are individuated by their Cartesian coordinates $(s_x, s_y)$ for $s$ and $(z_x, z_y)$ for $z$. As reported in [102] we could also consider a $\varphi$ angle among the targets and the sensors with $0 \leq \varphi(s, z) < 2\pi$. Evaluating the

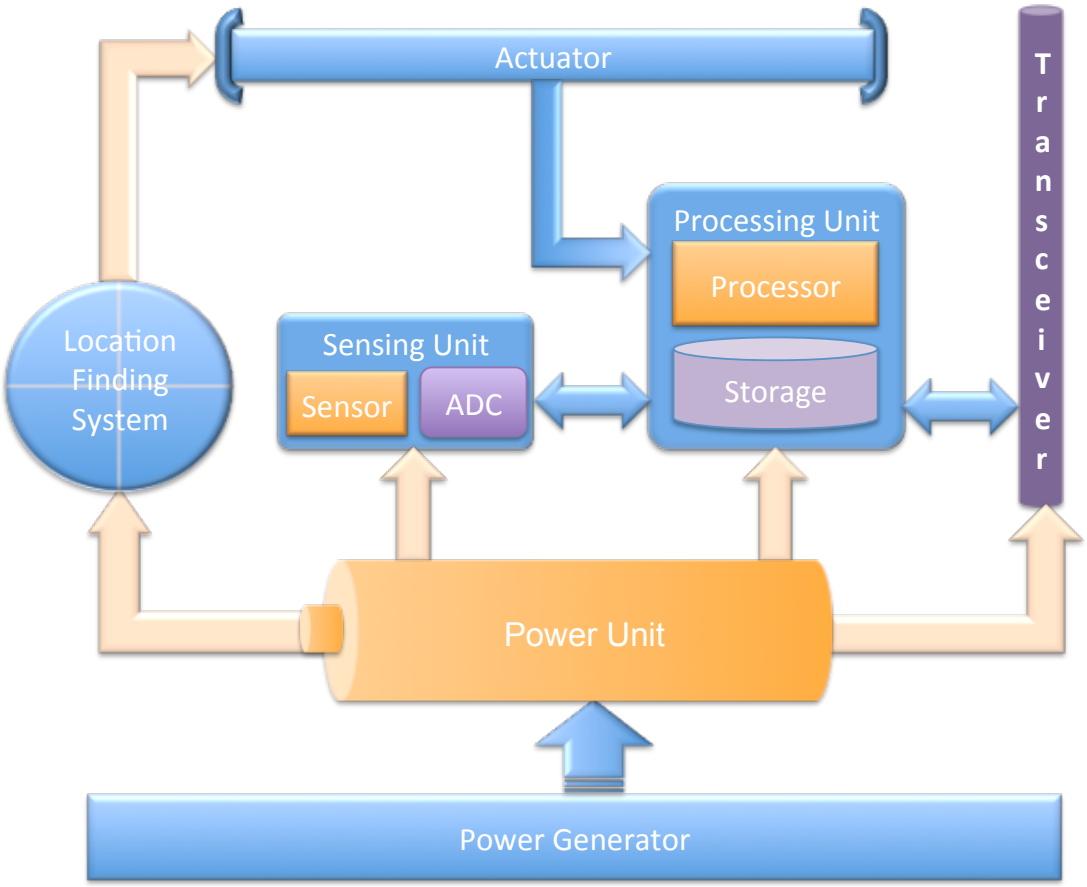



Figure 2.1: *Components of a Sensor Node*

Euclidean Distance of each space point with respect to the whole set of sensors, can be considered a function $f$ that for each space point expresses a coverage measure through this formula $f$:$(d(S,z),\varphi(S,z)) \rightarrow \Re^{+}$ where S is the set of all the sensors in the network. Also in [102] it is reported that by means of simple modifications this evaluation can be used in three-dimensional contexts. Generally the sensing models that consider a boolean evaluation of the coverage, i.e. 0 or 1, are called *Boolean* or *Binary Coverage Models*. Models that consider non-negative coverage measures are called *General Coverage Models*. Models that do not consider the angle as function input are called *Omnidirectional Coverage Models*, as opposed to *Directional Coverage Models*. For a more detailed and extensive review of the sensing models the reader can refer to [102]. In our works we considered the most studied and simplified sensing model, the binary disk coverage model that considers as coverage function the following formulation:

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

where we can recall that $d(s,z)$ is the Euclidean Distance between a generic sensor $s$ and $z$ a generic target point that we want to monitor. $R_{sense}$ is called *sensing range* or *sensing radius*, it is defined by the sensor technology and it defines the sensing area of the sensor centered on the device. It does not consider an angle as input therefore it is an omnidirectional coverage model. All the space point within the sensing area of a given sensor are defined as being covered by it. Generally for each space point we can consider the sum of the function values evaluated for all sensors that defines whether a space point is covered or not by the network, as follows:

$$f(S,z) = \sum_{i=1}^{n} f_i(d(s_i,z)) \tag{2.3}$$

If this $f(S,z) = K$ then the sensor can be defined as $K$-*Covered* (i.e. there exist $K$ sensors that cover $z$). There are also other models such as *Boolean Sector*

*Coverage Models*, *Attenuated Disk Coverage Models*, *Detection Coverage Models* and others [102] that model different coverage types.

### 2.2.2 Coverage Problems and Design Issues

Besides the previously mentioned elements there are other differentiations that generally complicate the network design as well as other aspects. Different coverage problems arise from the possible design choices. Coverage Problems constitute a research topic of crucial relevance for the design of communication protocols and algorithms for the efficient management of the sensors. Coverage problems may differ on the basis of the object of sensing (area or discrete points), deployment type (random or deterministic), sensor mobility capabilities, or network structure (simple or composite), among other factors. In the course of this chapter, we will give a general overview on the different coverage types and their related design issues, and will finally focus on the problem definition on which our research is based and that, as we will see, generalizes all formulations.

#### 2.2.2.1 Coverage Problem Types

Generally the coverage problems can belong to three main coverage families, that is: (*i*) *area coverage problems*, (*ii*) *point or target coverage problems* and (*iii*) *barrier coverage problems* [25]. In the former the main objective is to monitor a whole area of interest. In the second family the objective is to monitor only a set of discrete points which may be specific objectives within the environment. Figure 2.2-*a* shows an irregular area covered completely by the sensor network. Figure 2.2-*b* shows the same area and a set of targets which are covered by the network. In the second case, the targets coverage request can leave some zones of the area uncovered. However in the literature it is well know that an area coverage instance can be easily transformed into a target coverage one [98] and we can refer to both indifferently. The polynomial time transformation is based on the concept of "field", i.e. a subset of the area which is covered by the same set of sensors. We achieve the transformation by replacing each field with a target point. Figure 2.3 gives an idea of such simple transformation. Therefore, for simplicity we generally refer to target coverage problems.

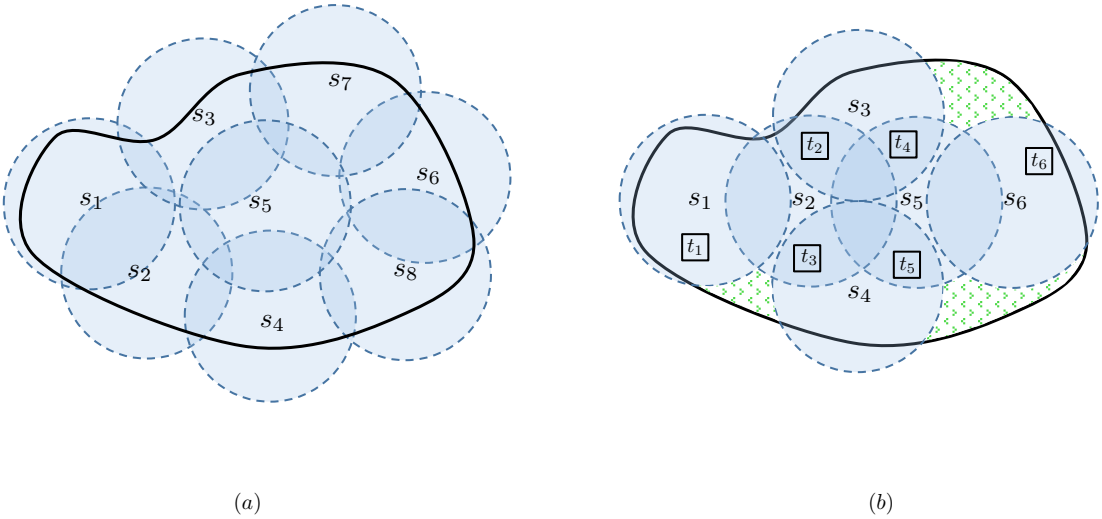
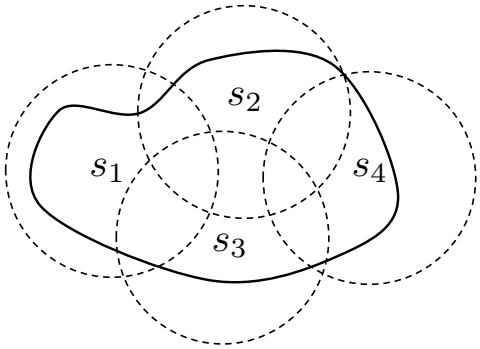



Figure 2.2: *Coverage Types: (a) area coverage - (b) points coverage*

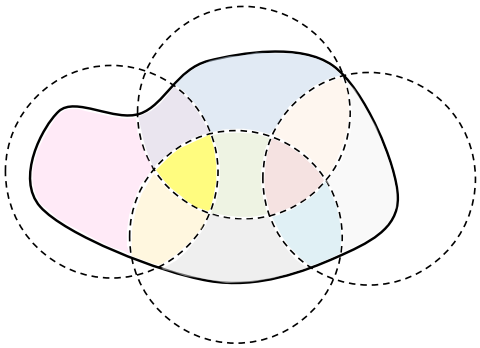
In barrier coverage problems the main objective is to build intrusion barriers for the detection of moving objects that can cross over or enter the area of interest. Let us consider Figure 2.4. This figure reports an illustration of barrier coverage. In Figure 2.4-a we can see an area that we want to monitor, a barrier coverage made of sensors along the boundaries of the area and an intrusion movement from the upper side of the area towards the lower side. In this case we can observe a real barrier coverage because the area can not be crossed from a one side to another without intersecting the sensing disk of the sensors. In Figure 2.4-b we can see a movement that can cross the region of interest without intersecting the sensing disk of the sensors. Therefore in this second case the network cannot provide a barrier coverage. It is straightforward to note that the barrier coverage doesn't require to coverage the whole area of interest. More details on intrusion barriers and penetration paths can be found in [102] [68] [69] [76].

#### 2.2.2.2 Design Issues

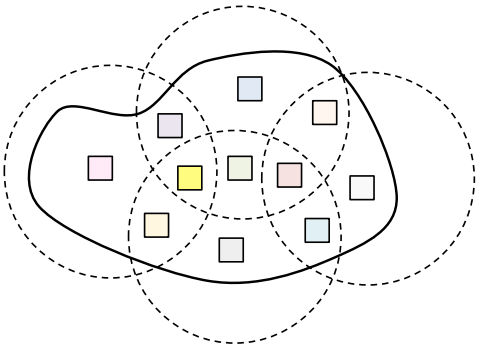
Given the sensing models and the three main families of coverage problems, we can now analyze the various design issues or design choices as reported in [102] [21], that may characterize a coverage problem:




*(a) Area Coverage Instance*



*(b) Fields Identification*



*(c) Target Coverage Instance*

Figure 2.3: *Transformation*

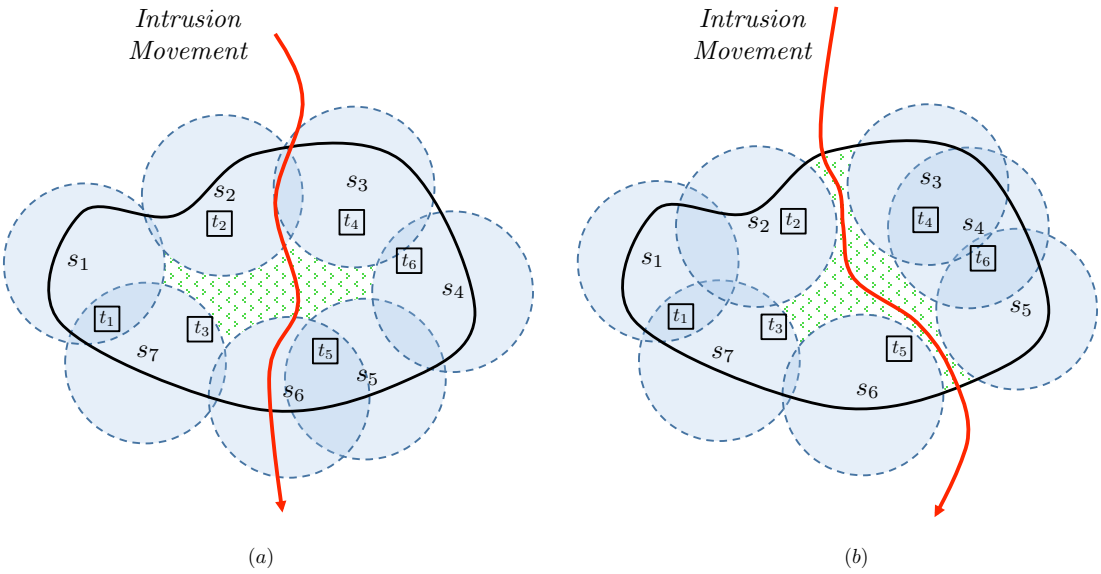



Figure 2.4: *Barrier Coverage Example*

- *Coverage Type*: as previously mentioned and described in Section 2.2.2.1 we can have area coverage, target coverage and barrier coverage problems.

- *Deployment Type*: there exist two deployment types that strongly affect the network topology: *deterministic* (or structured) and *random* (or unstructured) [107]. Deterministic installations are considered in the case of small and easily accessible area and the positioning is designed ad-hoc for the surrounding environment. Sensor placement can be designed to use as few sensors as possible in order to reduce both the management and maintenance costs. In the case of large and/or difficult to access area, random positioning could be preferable or mandatory. In the unstructured case, the network is composed of a large number of sensors which are positioned randomly and after the installation, the network is left unattended to perform its monitoring activities. In this type of network is it more difficult to address the issues of a typical communication network, such as how to manage the connectivity or possible failures. The choice of the deployment type depends on the environment.

- *Coverage Ratio*: this aspect refers to how many points in the area or which

percentage of the area the sensor network needs to cover in order to satisfy the coverage requirements. Generally we refer to *complete coverage* when the network covers the whole area or the whole set of targets. We refer to *partial coverage* when the network needs to cover only a subset of points to satisfy the coverage requirements. We can say that if the network covers 70 targets out of 100, its coverage ratio is 70%.

- *Coverage Degree*: each target can be monitored by one or more than one sensor at each time. When each node of the sensor network is covered by at least $k$ sensors the networks has a *k-coverage* degree. Typically this aspect is relevant when the network needs to guarantee a certain level of coverage robustness since a sensor network that satisfies a such requirement, can tolerate up to $k-1$ damages or faults for each sensor node.

- *Sensors type*: technology offers different sensors and there are cases in which monitoring is performed by sensors with different characteristics. The choice is often application dependent, some application requires that all sensor nodes in the network are equal or with the same characteristics, while others require different sensor nodes types (see for example the problems faced in Chapter 5 of this work).

- *Sensors Mobility*: as previously mentioned sensors may be static or mobile.

- *Network Type*: simple or mixed. The type is simple when a network is composed only of sensors that send information to a single collection node called *sink*, either in a centralized manner or a distributed one. The type is mixed or layered, if the network is composed of simple sensors (also with different technical and sensing characteristics) and a subset of more powerful nodes that act as collection nodes. In Figure 2.5 it is showed a dual layer sensor network in which a set of simple sensors (grey circles) can sense and process information about the environment, and a second layer of more powerful nodes (dotted circles) collect the information in order to efficiently manage the sensing tasks.

- *Collection Nodes Mobility*: some works assume that the collection nodes are static, other works assume that the sinks are mobile. Recent technologies

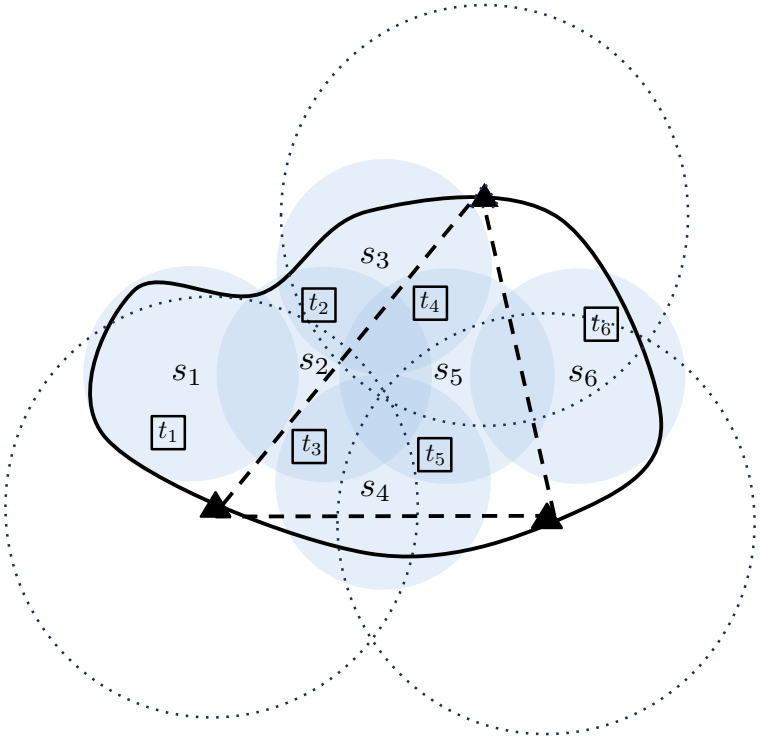



Figure 2.5: *Layered Wireless Sensor Network*

allow for instance the integration of robots which play the role of collection nodes in sensor networks [13] [70] [99] [39].

- *Events Type*: a network can monitors either atomic or composite events. As reported in [21] a fire event can be better monitored if there are more combined information as temperature, smoke, humidity. In this case an heterogeneous network has to satisfy specific coverage requirements for each atomic event [27].

- *Coverage Breach*: a target point or an area is defined *breached* when it is not covered by a sensor node. Some applications require a target or area monitoring that minimize the time for which the targets/areas remain uncovered. Sometimes the applications can require to minimize the total number of breached target points.

- *Activity Scheduling*: refers to the ability of the network to change the state of a sensor node. Generally it refers to the capacity of the sensor network to switch in a sleep energy-saving state the redundant active sensors. The main objective of this activity is typically to save energy in order to prolong the time for which the network can operate. This activity is governed by algorithms that belong to two main categories, namely *centralized* or *distributed*. Many centralized and distributed algorithms have been developed in this research field. Generally the distributed algorithms allow each node to decide about its state basing the process on the distributed information in the network. This process generally reduces the communication energy but intensifies the processing energy consumption. The centralized algorithms, on the other hand, leave each node to only send its sensed data to a central collector node that also makes global decisions about the working states of all sensors. This type of algorithms highly reduce the processing energy consumption. More details about the Activity Scheduling can be found in the next Sections.

- *Network Connectivity*: even if this requirement is typically related to the network layer it is also considered in the design of specialized algorithms to define certain cross layer operations. Two nodes can be defined as *connected*

if they are able to send and receive data directly between them. In other cases two nodes can be defined as *connected* if there are some others nodes between them that act as relays. Generally a network can be defined as *connected* if each couple of sensor nodes is *connected* and the data of each node can be sent to the collector node as well as to other nodes. In such a case sensors have a communication range in addition to the sensing range. Generally the communication range is larger than the sensing range.

All these design choices have some unifying requirements for the efficient functionality of the wireless sensor network. As already discussed a crucial one is the energy efficiency of the network in order to improve the "network lifetime".

## 2.3 Network Lifetime and Coverage Optimization

Network lifetime was originally defined as the amount of time until the first operational failure of a sensor occurs [42]. Definitions such as the previous one are incomplete because, especially in the case of unstructured networks with an high density of sensors, the network can be operational even if one of the sensors has depleted all its energy or has been damaged. These observations suggest different definitions, mainly linked to the coverage as in our research works (see Chapter 4 and 5), or related to the availability of nodes in the network or to the connectivity. In [42] the reader can find a full list of the main definitions. In this thesis, the network lifetime is the time interval for which the network is able to meet the specific coverage requests of the application. The absence of specific assumptions about the network, allows to adapt the definition to a wide range of different design choices.

### 2.3.1 Covers Scheduling on WSN

It is straightforward to understand that optimize the usage of constrained energy resource of a sensor is the main issue to be taken in consideration in order to prolong as much as possible the network lifetime. In the literature there are

different approach to address energy efficient coverage to extend the network lifetime. The first approach, on which are based our research works, is to define a schedule plan of the sensor activity that leaves some sensors in active state while the others are into a sleep state that does not consume energy. A second approach consists of designing an efficient coverage deployment plan, but is not always practical. A third approach is based on adjusting the sensing range in order to save energy. The first approach requires to identify *covers*, i.e. subsets of sensors, able to achieve the coverage requirements of the network [22] [23] [54] [59] [96]. This type of approach can be further divided in two main subcategories. A first one based on disjoint set of sensors as in [98] [24] and a second one based on non-disjoint set of sensor as in [22]. In the case of disjoint subsets, the covers do not share sensors, that is, the subsets have empty intersections. The second proposed approach allow the covers to share sensors among them. It has been proved that this approach can achieve greater lifetimes than the first one. In the case of disjoint subsets each cover is activated for all battery duration one at time while all sensors that do not belong to it are either in sleep mode or have been previously used. The authors in [98] proposed an heuristic algorithm to find as many covers as possible in order to extend the network lifetime. The duty scheduling was extensively investigated by researchers. Today there are many approaches that follow this idea while considering different characteristic of the network [31] [60] [65] [93] [54] [27]. It is important to note that the disjoint covers approach doesn't aim to maximize directly the lifetime, but rather tries to find the maximum number of possible covers. In the second case, in addition to the coverage constraints, covers can be activated even for very small amounts of time. The non-disjoint covers idea was investigated only in recent years and it is receiving more and more attention. Modeling lifetime problems on this idea has led to more realistic formulations. This approach aims at directly maximizing the lifetime by finding the optimal covers schedule and the related activation times, while satisfying all battery and network coverage constraints. The algorithms that follow this approach typically face hard optimization problems as in the case of the problems that we face in Chapters 4 and 5.

### 2.3.2 Common Scenario and Problem Definition

The scenario addressed in our work takes into account unstructured networks with random deployment and a high density of sensors. The sensors, with limited energy resources, are typically scattered in the region of interest and perform a monitoring activity on target points disposed within the area. The information gathered by the sensors is distributed among them, and we assume that such information will be collected and delivered to a central node at the end of the monitoring phase. The central node is also assumed to coordinate the activity of the sensors. Generally the sensor operational states are identified *TRANSMIT*, *RECEIVE*, *IDLE* or *SLEEP*. Taking into account the previous works in this field [22] [26] it is well known that the power usage of most units, such as the seismic sensor WINS Rockwell, is similar for the transmit state, the receive state and for the idle state while the sleep state requires a much lower, not negligible amount of energy. Therefore as in other works [54] [31], we assume for simplicity that two main operating states exist, called *ACTIVE* and *SLEEP*, which identify, generally, the cases in which a sensor is consuming battery for or not. Under these assumptions, we can observe that an accurate use of covers can improve considerably the network lifetime. Consider the example in Figure 2.6. In the example we have three targets $T=\{t_1, t_2, t_3\}$ and four sensors $S=\{s_1, s_2, s_3, s_4\}$. The sensor $s_1$ covers the targets $t_1$ and $t_2$. The sensor $s_2$ covers the targets $t_1$ and $t_3$. The sensor $s_3$ covers $t_2$ and $t_3$ and the sensor $s_4$ covers all targets. For all sensors we consider an energy resource normalized to 1 unit of time, i.e. each sensor can be in the active state for 1 unit of time before depleting all its energy. We recall that we want to cover all targets. If we active all sensors at the same time the overall network lifetime is equal to 1 since there are not other sensors available for monitoring. If we consider subsets of sensor, e.g. covers $C_1=\{s_1, s_2\}$, $C_2=\{s_1, s_3\}$, $C_3=\{s_2, s_3\}$, $C_4=\{s_4\}$, as in Figure 2.6-b-c-d-e, we can improve the network lifetime. Each one of these covers meets the coverage request, i.e. each subset of sensors covers all targets. Therefore we can design a strategy that activates first cover $C_1=\{s_1, s_2\}$ for 0.5 unit of time, then cover $C_2=\{s_1, s_3\}$ again for 0.5 units then $C_3=\{s_2, s_3\}$ for other 0.5 units and finally $C_4=\{s_4\}$ for 1 unit. Therefore, we can monitor the set of targets for 2.5 unit of time a value which is

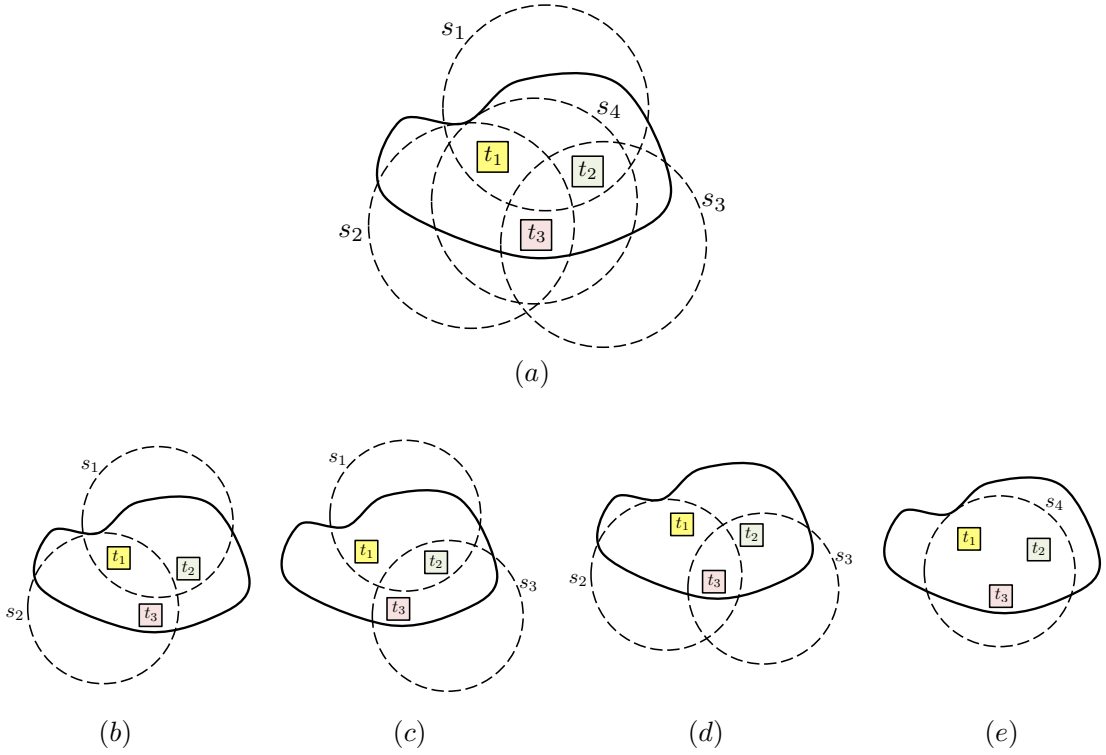



Figure 2.6: *Covers Examples*

2.5 times higher than the one obtained using the first, trivial strategy.

As shown, this approach can lead to considerable extensions of the network lifetime. This is particularly true on dense networks where targets are redundantly covered by sensors whose ranges present many overlaps. In this instances, indeed, a large number of feasible covers can exist and can be used to identify the optimal solution. This problem has been widely studied in recent years (refer to the literature overviews in chapters 4 and 5) and is usually known as Maximum Wireless Sensor Network Lifetime Problem (MLP). It has been shown to be Np-Complete by reduction from 3-SAT problem in [22]. There are also some related variants that consider different design choices such as the ones that we face in our research work described in Chapters 4 and 5.

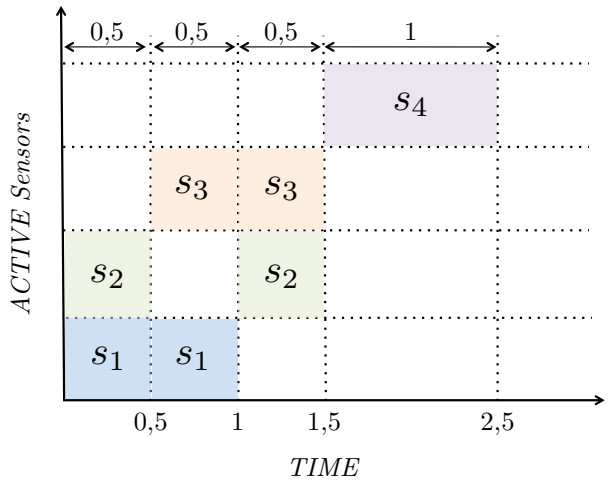



Figure 2.7: *Sensor scheduling in the optimal solution for the instance in figure 2.6*

# Chapter 3

# Maximum Lifetime Problem: Modeling and Algorithms

## 3.1 Modeling the problem

As extensively detailed in previous Chapter 2, Wireless Sensor Networks are composed of a huge number of sensors scattered over a geographic area that we want to monitor. Each sensor device has limited sensing and limited computational resources. Each of them senses the surrounding space around it defined by its sensing range. All the sensors perform a complex sensing task about the environment around them. A sensor can gather information about all its surrounding area or only about specific targets inside its sensing area. From now on we will refer only to target coverage problems for the motivations given in Section 2.2.2.1. Given the energy battery constraints, one of the main issue is to prolong as much as possible the network lifetime, i.e. the amount of time for which the network is able to guarantee the coverage constraints about the subject under monitoring. As reported in Section 2.3 the network lifetime can be extended by individuating covers and activating them, one at time, for a suitable amount time. In this chapter we formally define the Maximum Lifetime Problem (MLP) on Wireless Sensor Network and we describe the basic concepts of the column generation technique and the genetic algorithms that we will use to address this problem. Let $N = (S, T)$ be a wireless sensor network, where $S = \{s_1, \ldots, s_m\}$ is the set of

sensors and $T = \{t_1, \dots t_n\}$ is the set of targets. Each sensor has a given sensing range defined by its technical characteristics and each sensor is powered by a battery that can keep it in an active state for a limited amount of time. Here, for simplicity, we assume that each sensor has the same characteristics, i.e. same sensing range and same battery lifetime normalized to 1. We consider a omnidirectional binary sensing disk model (see Section 2.2.1). Therefore for each target $t_k \in T$ and sensor $s_i \in S$, we define a binary parameter $\delta_{ki}$ equal to 1 if $t_k$ is located inside the sensing area of $s_i$ (target $t_k$ is *covered* by the sensor $s_i$), 0 otherwise. Let $C$ be a subset of sensors ($C \subseteq S$). We formally define $C$ to be a *feasible cover*, for the network $N$, if all targets of the network are covered by at least one sensor, i.e. $\sum_{i \in C} \delta_{ki} x_i \geq 1 \ \forall k = 1, ..., n$. The Maximum Lifetime Problem consists in finding a collection of pairs $(C_j, w_j)$, where $C_j \subseteq S$ is a feasible cover and $w_j \geq 0$ is the amount of time for which the sensors belonging to $C_j$ are kept in an active state (i.e. activation time), such that the sum of the all activation times is maximized and each sensor is used globally for an amount of time that does not exceed its normalized energy resource. In the figure 3.1 we can see a very simple sensor network composed of 4 sensors and 3 targets. Possible examples of covers that can be activated to monitor all targets are: $C_1 = \{s_1, s_2\}$, $C_2 = \{s_1, s_3\}$, $C_3 = \{s_2, s_3\}$, $C_4 = \{s_4\}$. Therefore, assuming to be able to compute the whole set of feasible covers $C_1, \dots, C_\ell$ in advance, MLP could then be represented using the Linear Programming formulation [**P**], where $w_j$ are the variables associated to the columns of the matrix, $\forall j$, with $j = 1, ..., l$. The variables indicate the activation time of each cover. As seen in Section 2.3.2 we consider two particular relevant operating states, called *ACTIVE* and *SLEEP*, which identify the cases where a sensor is consuming battery for sensing or not. Then the binary parameter $a_{ij}$ indicates if a sensor $s_i$ is active in a covers $C_j$, i.e. $a_{ij} = 1$ if $s_i \in C_j$, 0 otherwise. We can note that each column $\underline{a}_j$ is a representation of a cover, therefore from now on we use the terms "cover" and "column" indifferently:

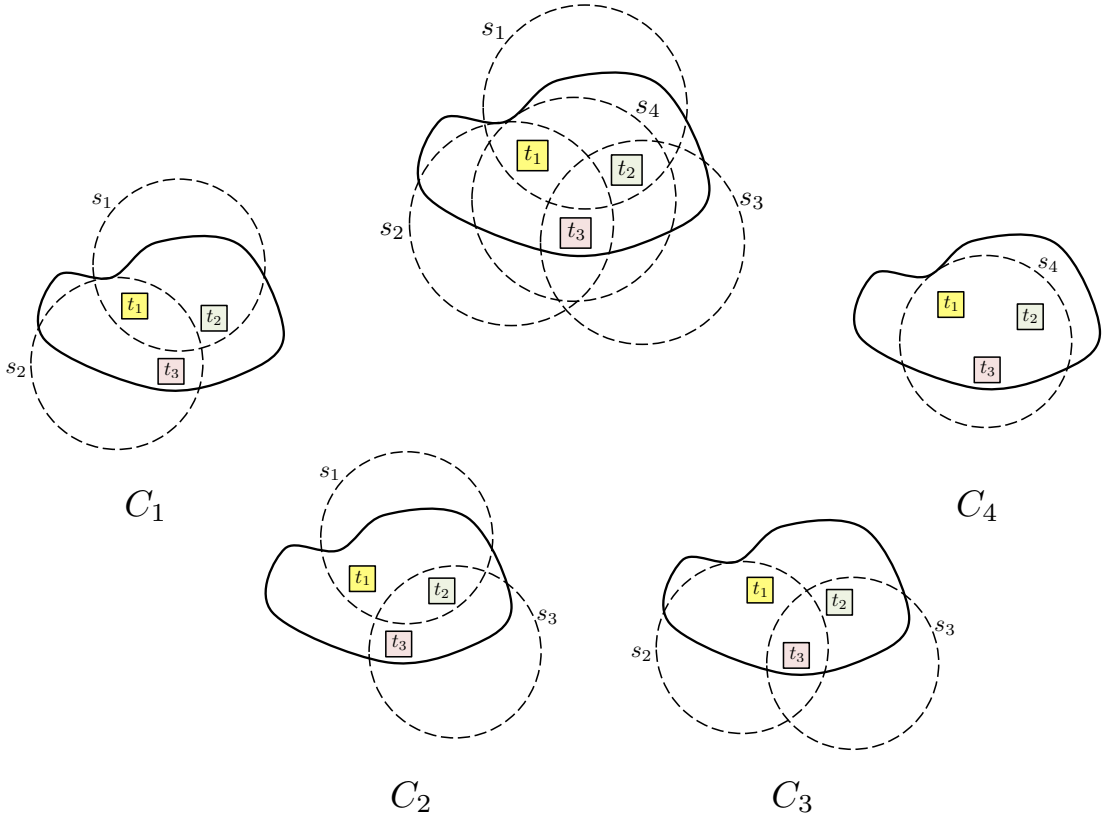



Figure 3.1: *Examples of covers*

$$[\mathbf{P}] \max \sum_{j=1}^{\ell} w_j \tag{3.1}$$

s.t.

$$\sum_{j=1}^{\ell} a_{ij} w_j \leq 1 \qquad \forall i = 1, ..., m \tag{3.2}$$

$$w_j \geq 0 \qquad \forall j = 1, ..., \ell \tag{3.3}$$

The objective function (3.1) maximizes the network lifetime i.e. the sum of the activation times $w_j$, while constraints (3.2) require that each sensor cannot be activated for more than its battery resource. In real world scenarios, the number of sensors is often very large with respect to the number of targets. Consequently, the number of covers (columns of the model) is too large (i.e. potentially exponential) for a direct application of the simplex method, both in terms of time (needed to check the optimality condition on the exponential non-basic variables) and space (needed to build the constraints matrix). A proper approach to the resolution of this problem is the well known *Column Generation* (**CG**) algorithm, a technique alternative to the Simplex algorithm, that starts by solving the model [P] with a small subset of columns and then introduces additional columns until the optimal solution of [P] is found. The column generation algorithm differs from the Simplex algorithm in *how* it performs the optimality test of the current basic solution and in how the new variable to eventually enter the basis is chosen. Indeed, in the case of the Column Generations these steps are performed by modeling and solving an auxiliary optimization problem.

## 3.2 Column Generation

As reported in [75] and in [80], the first idea of *Column Generation* was presented in a work of Ford & Fulkerson (1958) [46]. This work suggests for the first time to deal with the variables of a problem in implicit manner. They consider a multi-commodity maximum flow problem and their idea was to begin by solving optimally a master LP formulation that just contains few columns (extreme flows in the work) for each commodity. Then they used an optimal dual solution to price out the columns not yet examined by means of the solution of a shortest path problem, for each commodity. Later, Dantzig & Wolfe (1960) [35] inspired by the work of Ford & Fulkerson [46], generalized the idea to obtain an algorithm for solving linear programming problems individuating a set of master constraints and a set of separation problem constraints, known as Dantzig-Wolfe Decomposition, as also reported in [80]. Later in the two seminal works, Gilmore and Gomory (1961-1963)[56] [57] implemented the technique to solve a problem that involve integer variables, the *Cutting Stock Problem*. However, the first oc-

currence of the column generation naming appeared in 1969 with a paper titled "A column generation algorithm for a ship scheduling problem" [6].

### 3.2.1 Restricted Master Problem Formulation

Given the mathematical model [P], let $R \subseteq \{1, ..., \ell\}$ be a subset of the indexes of all possible columns. Through the R columns, we build the [RP] model, named *Restricted Master Problem*, as follows:

$$[\mathbf{RP}] \max \sum_{j \in R} w_j \tag{3.4}$$

s.t.

$$\sum_{j \in R} a_{ij} w_j \leq 1 \qquad \forall i = 1, ..., m \tag{3.5}$$

$$w_j \geq 0 \qquad \forall j \in R \tag{3.6}$$

The coefficients $a_{ij}$, with $i \in \{1, .., m\}$ and $j \in R$, form the matrix $A_R$, i.e. the matrix obtained by considering only the columns of the original matrix $A$ with index belonging to the subset $R$. Therefore, the problem [RP] consists of just the variables of [P] associated to the column of $A_R$. Assuming that the R set is not too large, this restricted formulation can be directly solved by means of the Simplex algorithm. Let $\overline{w}_R$ be the optimal solution of [**RP**] that from now on we denote as the *incumbent solution*. Let $B \subseteq \{1, ..., \ell\}$ be the set of the basic columns, $N \subseteq \{1, ..., \ell\}$ the set of the non basic columns, $N' \subseteq R - B$ the set of non basic columns related to the restricted formulation and $\underline{b}$ the *righ-hand-side* column vector. It is easy to see that $(\overline{w}_R, \underline{0}_{N-N'})$ is a feasible solution for the original formulation [**P**]. In this situation there are two conditions that may occur:

- The optimal solution for [**RP**] is also optimal for [**P**]. Therefore the master incumbent solution is globally optimal and the column generation procedure stops and returns the incumbent solution.

- The optimal solution for [**RP**] is not optimal for [**P**]. It means that there exists a non-basic variable $w_j$ with $j \in N$, with $z_j - c_j < 0$, that can improve the value of the objective function. If the existence of this variable $w_j$ is proven, we have to construct the related column $a_j$ that has to be introduced in the matrix $A_R$. Once inserted the variable and the column in the restricted formulation, the column generation algorithm repeats the whole process.

A proper application of the method requires to face the following issues:

1. How do we find a variable $w_j$ having a negative reduced cost among the exponential variables in $N$?
2. Even if we find the variable $w_j$, how do we build the column $a_j$ of the coefficient matrix $A_R$?
3. How do we identify the initial set of columns $R$?
4. How do we update the set $R$ once the new column to be included is found?
5. How is degeneracy dealt with?

Now we will shortly address each of the above mentioned issues. For the issues 1 and 2, the column generation tries to identify the variable $w_j$ and the column $a_j$, through the resolution of a new optimization problem known as *Separation Problem* (or *SubProblem*) (**SP**). The separation problem is constructed in an ad-hoc manner depending on the master problem and often it corresponds to well known optimization problems for which efficient algorithms have already been proposed. Since the resolution of the RP is easy, the key point to obtain an efficient column generation algorithm is the ability to efficiently solve the subproblem [**SP**].

### 3.2.2 Modeling the Separation Problem

Now we see how to model the separation problem for the classical Maximum Lifetime Problem. Let $\underline{w}_R^*$ be the optimal solution of the restricted problem

[**RP**]. The matrix composed only of the columns of the variables belonging to the base will be indicated with the form $A_B$. Instead, the matrix corresponding to the restricted problem will be indicated with the form $A_R$. The columns of the matrix corresponding to non-basic variables of the restricted problem are indicated in the form $A_{R-B}$. We denote by $A_N$ all the columns not yet generated joined to the columns in $A_{R-B}$. Given the initial set of columns $R$, some of them will correspond to basic variables, while the remaining $N'$ will correspond to non-basic variables. Then we can write:

$$\underline{w}_R^* = \begin{bmatrix} \underline{w}_B \\ \underline{w}_{R-B} \end{bmatrix} = \begin{bmatrix} A_B^{-1}\underline{b} \\ \underline{0}_{R-B} \end{bmatrix} \tag{3.7}$$

$$\Downarrow$$

$$\underline{w} = \begin{bmatrix} \underline{w}_B \\ \underline{w}_{R-B} \\ \underline{w}_{N-R-B} \end{bmatrix} = \begin{bmatrix} A_B^{-1}\underline{b} \\ \underline{0}_{R-B} \\ \underline{0}_{N-R-B} \end{bmatrix} = \begin{bmatrix} A_B^{-1}\underline{b} \\ \underline{0}_N \end{bmatrix} \tag{3.8}$$

Taking into account the optimality conditions expressed through the reduced cost values we can state that a current solution is globally optimal if and only if the reduced cost values corresponding to all the non-basic $A_N$ columns are positive, i.e. $z_j - c_j \geq 0 \; \forall j \in N$. Considering the size of the problem it is clear that it is not possible to evaluate these coefficients in an enumerated manner. The idea of the column generation is based on the search for the smallest reduced cost value evaluating the following objective function that will be the objective of our separation problem:

$$\min_{j \in N} \quad z_j - c_j \tag{3.9}$$

If the value of the minimum reduced cost, corresponding to the objective function (3.9), evaluated on the incumbent solution, is non negative then the optimality conditions for problem [P] are met and therefore there are not non-basic variables that can improve the incumbent solution. Otherwise there exists a column that, once added to the restricted problem RP, may improve it. In this

case, the column corresponding to the optimal solution of the separation problem will be added to the matrix $A_R$. We observe that:

$$\min_{j \in N} \quad z_j - c_j \tag{3.10}$$

$$\Downarrow$$

$$\min_{j \in N} \quad \underline{c}_B^T A_B^{-1} \underline{a}_j - c_j \tag{3.11}$$

$$\Downarrow$$

$$\min_{j \in N} \quad \underline{\pi}^T \underline{a}_j - c_j \tag{3.12}$$

$$\Downarrow$$

$$\min_{j \in N} \quad \sum_{i=1}^{m} a_{ij} \pi_i - c_j \tag{3.13}$$

$$\Downarrow$$

$$\min_{j \in N} \quad \sum_{i=1}^{m} a_{ij} \pi_i - 1 \tag{3.14}$$

By applying the above shown substitutions to objective function (3.10) we can build the formulation of the objective function of the separation problem which is specific for the the Maximum Lifetime Problem. As we know from the theory of linear programming, $z_j = \underline{c}_B^T A_B^{-1} \underline{a}_j$ then substituting in (3.10) we get the formula (3.11). We observe that $\underline{c}_B^T A_B^{-1}$ corresponds to the simplex multipliers that appear in the evaluation of the reduced cost values, in other words these values are the dual prices and correspond to the vector $\underline{\pi}^T$. Substituting in (3.11), we get (3.12). By explicating (3.12), we get (3.13), where $\underline{a}_j$ is the $j$-th column of the coefficient matrix. Finally we note that the coefficients cost $c_j$ of [P] are all equal to 1, and then we obtain (3.14). It should be noted that in (3.14), each $a_{ij}$ value corresponds to the $i$-th entry of the new entering column $a_j$ that we want to find, and therefore are the variables of our separation problem. Then, we rewrite the new objective function as follows:

$$\min \quad \sum_{i=1}^{m} x_i \pi_i - 1 \tag{3.15}$$

The variable $x_i$ indicates whether the $i-th$ sensor is turned on or off in the cover which is represented by the new column built by the separation problem. The vector $\underline{x}$ must be a feasible cover of [**P**], i.e. its active sensors have to cover all the targets. To this end, we build the following subproblem [SP] in which the constraints impose to satisfy this covering condition.

$$\textbf{[SP]} \qquad \min \sum_{i=1}^{m} \pi_i x_i \tag{3.16}$$

s.t.

$$\sum_{i=1}^{m} \delta_{ki} x_i \geq 1 \qquad \forall k = 1, ..., n \tag{3.17}$$

$$x_i \in \{0, 1\} \qquad \forall i = 1, ..., m \tag{3.18}$$

The objective function (3.16) minimizes the sum of the dual prices of the sensors chosen to be part of the newly produced cover. If the optimal solution of this formulation is greater than or equal to 1, then the incumbent solution is guaranteed to be optimal for the original problem, otherwise the new cover is added to the master problem. The constraints (3.17) define a feasible coverage, that is, every target must be covered by at least one sensor in the current cover. It is straightforward to note that the [**SP**] formulation corresponds to a specialization of the Set Covering problem, a well known Np-Hard problem.

### 3.2.3 Working Scheme

Now we focus our attention on the working scheme of the column generation algorithm. Given the linear programming formulation [**P**], the column generation technique starts by considering the restricted master problem, and solving it to optimality. The optimal solution of the restricted master problem is feasible for the original problem [P], however there is no guarantee regarding global optimality since most of its columns have been discarded. The column generation then considers the specific separation problem which either produces an *attractive cover* to add to the restricted master problem for a new column generation iteration, or certifies that the incumbent solution is optimal. We recall that an *attractive cover* is a feasible cover corresponding to a non-basic variable with negative reduced cost, which could therefore improve the incumbent solution. The column generation procedure iterates until the above presented optimality condition is met. Therefore, this exact approach allows to implicitly discard most of the variables that will be non-basic in the optimal solution. The subsequent algorithm and the diagram 3.2 summarize the working scheme of the column generation approach:

**INPUT**: Maximum Lifetime Problem Instance

1. **Define the initial set of columns R**:
   The choice of the columns has to guarantee the feasibility for the main problem P.

2. **Costruct the RP formulation with the column belonging to R**:
   $\max\{\underline{c}_R^T \underline{w}_R : A_R \underline{w}_R \leq 1, \underline{w}_R \geq 0\}$.

3. **Solve the RP formulation**:
   Let $\underline{w}_R^*$ be the optimal solution of the RP problem.

4. **Construct and solve the separation problem**:
   If the optimal solution of the separation problem is $\geq 1$ then the solution $(\underline{w}_R^*, \underline{0}_{N-N'})$ is the optimal solution also for [**P**] and the algorithm stops. Otherwise the column computed by the separation problem SP is added to the RP formulation and then we return to the step 3.

We can now answer to the question 3 of the section 3.2.1 that is: 3) how do we identify the initial set of columns $R$? The initial choice of the set $R$ of columns has to guarantee the feasibility for the restricted master problem. The general method is to use an heuristic approach to compute any feasible solution of size $R$ for the current problem that we have to solve. Given such heuristic solution, the set $R$ will be composed by columns of the matrix $A$ related to the heuristic solution. A simple example of heuristic solution could be the follow: *for each target choose a sensor that covers it and activate it.* We don't care about how many targets a selected sensor covers, as long as we respect the target coverage constraints. This is a simple way to construct a cover. The process may be iterated to produce the desired R set. Another important question is linked to the update of the set $R$ of columns, i.e. 4) how do we update the set $R$? In literature there exist variants of the column generation approach based on the method used to update the set $R$ at each iteration:

1. the set $R$ can be composed of the basic columns deriving from the last [RP] solution and of the new column that enters the basis. At each iteration the column generated by the separation problem can be added to the set $R$ and the exiting one is deleted.

2. iteration by iteration a new column is added to the set $R$ without deleting previous ones.

3. columns of R which did not enter the basis for a large number of iterations without re-entering are deleted from it.

Indeed, there are cases in which even if the separation problem individuates a new attractive entering variable the objective function doesn't change its value. This is a situation that slows down the convergence of the algorithm. Generally in case of degeneracy we can apply anti-cycling rules and perturbation rules on the restricted problem. More details about the applicability of the approach can be found in [12] [41] [56] [57] [75] [80].

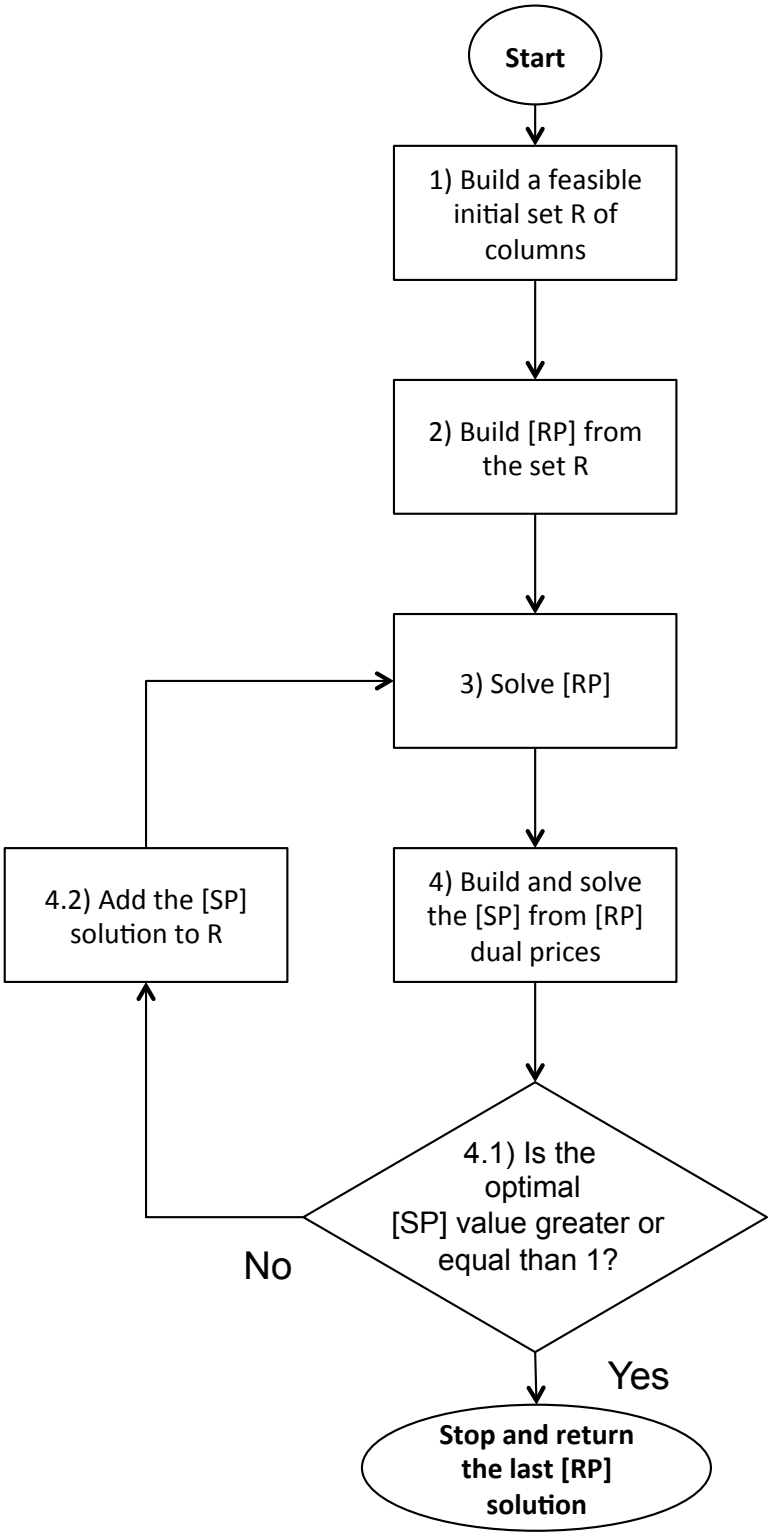



Figure 3.2: *CG Diagram*

### 3.2.4 How to solve the separation problem

In the previous section we saw the general working principles of the Column Generation approach and their application to the MLP problem.

As shown in section 3.2.3, the separation subproblem for the MLP is the well known weighted set covering problem that is NP-Hard [53][67]. Typically to solve hard optimization problems, it is helpful to use heuristic approaches that by improving the computational time required to solve the separation problem may help to speed-up the whole resolution framework. The column generation uses the optimal solution of the separation problem to test the optimality of the problem [P], however in order to improve the current objective function of the master problem we do not need necessarily the column with the best reduced cost. All the columns with objective function value lower than 1 can be used to improve the objective function of the master problem. Therefore we thought to apply an heuristic approach that can compute more than one column at a time. There exist various examples of column generation algorithms which use heuristics for the resolution of the separation problem, more detail can be found in [15] [101] [14]. Obviously, the heuristic approach does not guarantee to find the optimal solution, therefore every time the heuristic fails in finding attractive covers we are forced to invoke the exact resolution of the separation problem. This implies that the [SP] formulation must be solved to optimality at least once. In the next chapters we will show the effectiveness of genetic algorithms constructed in an ad-hoc manner to speedup the generation of several attractive covers at once with respect to the classical column generation approach. Indeed, we chose a population-based metaheuristic since it allows to consider and improve several good solutions, potentially reducing the number of required column generation iterations and thus also the computational effort. remaining part of this chapter we introduce genetic algorithms and their behavior in greater detail.

## 3.3 Genetic Algorithms

Genetic Algorithms represent a well-known and widely used meta-heuristic technique for optimization problems. Similarly to other evolutionary techniques, a

genetic algorithm (GA) emulates biological evolution and natural selection. The first evolutionary ideas, inspired by Charles Darwin's theory of evolution, appeared in computer science and in the field of optimization around 1960. As reported in [45] and [97], in 1960 Rechenberg, [86] presented the first approach which included an evolutionary simulation in the field of computer science. He was studying optimization problems in fluid-mechanics when he began to study general function optimization algorithms and evolutionary strategies. With his works he pioneered the filed of evolutionary computing. In the following years, these ideas were studied and extended by several researchers. As reported in [45], at that time, the concept of genetic algorithm appeared independently three times in a decade. Indeed, it was presented by Fraser in [48], [49] (1957), in [50] (1960), in [52] (1962), in[47] (1966), and in [51] (1968). In the same years it was also independently studied by Bremermann in [18] (1962), and Bremermann et al in [19] (1966). Other contributions were presented by John Holland's students at Michigan University (Bagley in [9] (1967), Rosenberg in [89](1967)), and Holland in [62] (1969) and in [63] (1973). However, historically the birth of genetic algorithms was assigned to Holland that in its publication *"Adaption in Natural Artificial Systems"* [61] (1957), as reported in [97], developed the first genetic algorithm inspired to the Darwin's theory for an optimization problem. Holland's ideas were intensively studied and now genetic algorithms represent a mature theory in the resolution of optimization problems and search. As reported in [97], genetic algorithms are today an example of mathematical technology transfer because simulating the evolutionary concept can solve many problems in different fields. We can remember genetic algorithm applications for very hard optimization problems about scheduling, timetabling and others [97]. As in nature, a GA considers the evolution process based on *chromosomes*, elements that represent the structure of an individual for the real world and a solution (e.g. a feasible cover in our case) for the optimization problem. The natural selection is the process through which the GA guarantees that new solutions are typically, step by step, better adapted to the environment. The environment is encoded by the *fitness function* that is used to rank each solution. The evolutionary step is achieved through two mechanisms, named *crossover* and *mutation*. The crossover operator combines, in a probabilistic manner, the genetic material of typically two

or more selected individuals (*parent* solutions). The mutation operator, instead, randomly modifies the value of one or more genes of a child chromosome derived from the crossover phase in order to increase diversity. The overall process is repeated until a stop condition is reached. Such a condition can be a maximum number of generations, a specified amount of time, a lack of improvements in the fitness function of the best individual, or other conditions related to the specific optimization problem. The resolution of an optimization problem aims to search the best solution in the search space, i.e. the space of feasible solutions. Genetic algorithms take advantage of the above mentioned evolutionary concepts in order to find the best solution, trying to avoid or escape from local optima. There are two key aspects on which the genetic algorithms are based, that is randomness and the current population (i.e. set of feasible solutions). The genetic algorithms are stochastic algorithms, meaning that randomness is used within the selection, reproduction and mutation procedures to better simulate the evolution process. They do not work on a single solution, indeed they work on a big set of solutions at the same time, which allow the genetic procedures to consider a significant amount of diversity at each iteration. In literature there are also other approaches inspired by Darwin's theory of evolution and by genetic algorithms, such as evolutionary strategies and genetic programming. Classifying this large set of different approaches, which is still in expansion, can become difficult; for this reason, they are generally referred to as Evolutionary Algorithms. Despite their good features, genetic algorithms also present some drawbacks. Indeed, they represent a general framework that works, typically, without knowing specific notions about the problem, and this generality leaves them the possibility to be applicable to a great set of problems. Obviously, specialized algorithms designed more specifically for a given problem can outperform the performance of a more general purpose genetic algorithm in terms of required computational effort and solution accuracy. However, given their features genetic algorithm are often useful to better investigate a big and complicated search space, or to hybridize other approaches, either exact or not. As reported in [97] John Holland, in its work [61], intended to investigate both the evolutionary process of natural species with their adaptation to the environment and to design artificial systems with similar characteristics. His main intuition has been that a set of a solutions for a given

problem can contain the representation of the optimal solution. The initial population of solutions can evolve toward better individuals (i.e. solutions) through the combination, permutation and mutation of part of their information and the one of other individuals, allowing new solutions to inherit the best features of previous ones.

### 3.3.1 Generational Genetic Algorithms and Steady State Algorithms

A simple genetic algorithm is composed of a set of steps that follow the paradigm of the evolution of the species. Today, as reported in [97], we can well understand that biological concepts such as the structure of DNA and RNA are very similar to the mechanisms of information storage in the computer, e.g. linear data structures or vectors or strings. This, in many cases, suggests us the natural application of the evolutionary approaches to optimization problems. The starting step, for the application of a genetic algorithm, is the representation of the individuals, i.e. the design of a structure that represents efficiently the information which represent a solution for our problem. This structure is known as *chromosome*. All the solutions that follow that structure can be part of the search space of the our problem. Obviously it is impractical to enumerate and build all possible solutions. Indeed, as in nature, genetic algorithms work with an *initial population* of individuals (solutions) of limited size, represented through the chromosome structure. Therefore the initial search space is reduced to this set of solutions. Given the current population and the details of the optimization problem that we would like to solve, a genetic algorithm uses specific operators which are able to combine two or more solutions to obtain new ones. The *fitness function*, as previously reported, encodes the environment and is used to rank solutions and give a measure of their quality. Typically the fitness function corresponds to the objective function of the optimization problem. As in nature the process of *natural selection* favors the best individuals, and genetic algorithms implement a *selection process* on the current population in order to select, according to the fitness value, only the best ones. However, during the evolutionary steps, elements with inferior fitness values might also be combined with others as a

mean to increase diversity and explore new regiorns of the search space. This diversification is further addressed by the mutation operator, which randomly alters part of the information belonging to newly produced chromosomes. The general structure of a genetic algorithm can be summarized in the following steps and schematically in diagram 3.3.

**General Genetic Algorithm structure**:

1. Build a random *Initial Population*: defined the chromosome structure and the details of the problem, we 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 problem, we evaluate this function on each solution belonging to the initial population. Therefore the algorithm can rank the quality of each solution.

3. Store the best individual and its fitness value.

4. Apply a *Selection* on current Population: at this point given the information coming from the previous steps and the characteristics of the solutions, we have to decide how to naturally select the mating pool i.e. what are the individuals from which to evolve the current population.

5. Create a *New Generation* that replaces the old one by means of these steps

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

   (b) Apply *Crossover*: here the algorithm combines the genetic material of the selected parents to generate one or more children. Typically the selection and the crossover operator are applied on good the individuals belonging to the mating pool since the hope is that by combining them even better solutions might be obtained. There are several crossover techniques such as Single point, Two Point, Multipoint, Uniform etc.

(c) Apply *Mutation*: with a predefined probability the algorithm applies on the current children a mutation operator. This operator alters some information in the new solution in order to allow the recovery and the conservation of genetic material that could be lost. At the same time it allows to vary the solution to improve the search and allows to avoid getting stuck in local optima.

(d) *Acceptance*: each new produced child, will be inserted in the new population that will replace the older population. In the described scheme, known as generational genetic algorithm, the new children will be used only in the new generation. In the case of *steady-state genetic algorithm* instead each new child 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.

6. If any *Stop condition* the algorithm terminates, otherwise a new iteration is carried out (Step 2).

The scheme 3.3 represents the operational flow of a generational genetic algorithm, in its standard definition. However, there are some variations, such as the already introduced steady state algorithm a technique that as we will see we applied to the research problems studied in chapters 4 and 5. This type of genetic algorithm has for many types of problems better performances, which is due to earlier insertion of the elements in the population. In practice, each new child is immediately available in the mating pool, making immediately a step towards better solutions, in the early stages of the evolutionary process. For a complete and detailed description of the genetic algorithms and their characteristics the reader can refer to [97] [58] [38] [105].

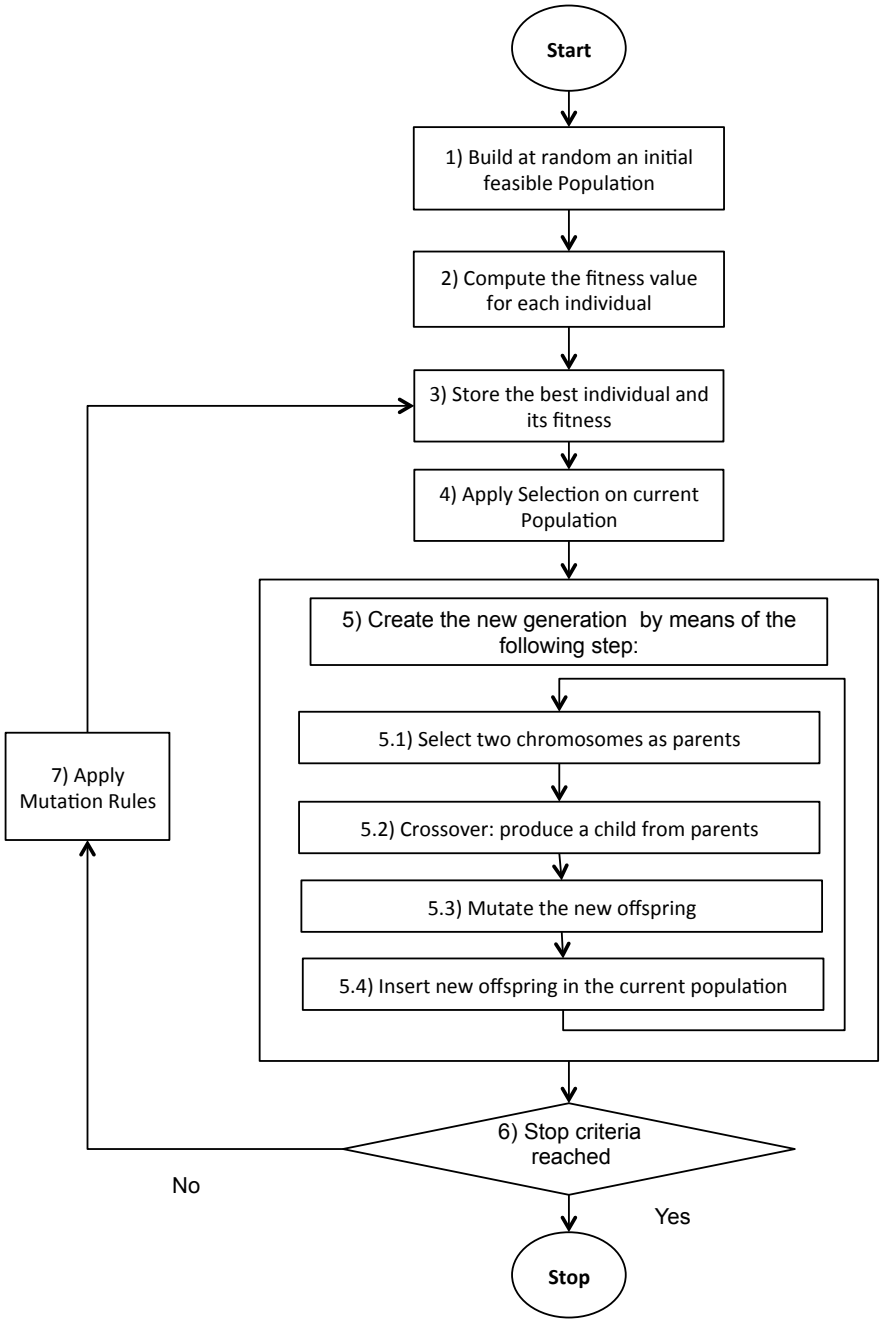



Figure 3.3: *GA Diagram*

# Chapter 4

# A Hybrid Exact Approach for Maximizing Lifetime in Sensor Networks with Complete and Partial Coverage Constraints

## 4.1 Introduction

In this chapter we address the well known Maximum Lifetime Problem (MLP) in wireless sensor network with full and partial coverage constraints. Here we describe, first, the problem and the essential literature on the problem to better introduce, subsequently, the mathematical formulation of the problem and the building blocks of the designed resolution approach. Wireless Sensor Networks (WSNs) are generally composed of a huge amount of small power-constrained sensing devices (*sensors*) scattered over the region of interest that we want to monitor. Each sensor is generally capable of monitoring the space around itself defined by its *sensing range*, this area is usually called *sensing area*. Each individual device has well known limits in terms of sensing capabilities and energy resources (see Section 2.2 for more details about sensors and wireless sensor networks). A coordinated set of sensors allows to perform monitoring activities in possibly large areas, in fields as diverse as environmental control, healthcare

and military applications, and others (see, for example, Chapter 2, [4], [82], [85]). Given the limited energy resources of the batteries that generally keep sensing devices in a active state for a limited amount of time, an issue which has drawn the attention of researchers in the last years is the optimization of energy consumption in order to improve the global network lifetime. Specifically, the problem of judiciously use a set of sensors to monitor specific points of interests (known as *targets*), placed inside the environment under monitoring, as long as possible has been widely studied. The MLP has been mainly addressed with strategies aimed at finding *covers*, i.e. several, potentially overlapping subsets of active sensors which can, one at a time, assure the coverage of all target points, as well as an activation time for each of them, such that the sum of the activation times of the covers in which each sensor appears is not greater than the amount of time that its battery can guarantee. The main idea is to activate the covers one at time, that is to turn on all the sensors which belong to the current cover, while keeping all other sensors turned off. In [22] the authors showed that MLP can be improved with respect to previous methods in which each sensor could only belong to a single cover, i.e. sensors were divided into disjoint sets. They also showed that the problem is NP-Complete and they designed an approximation algorithm to solve it. A Column Generation algorithm aimed at solving the MLP was proposed in [40]. In this work the authors propose a hybrid approach where the *Separation Problem* of the Column Generation technique is either solved by means of an heuristic algorithm or optimally by means of a specific ILP formulations. More details about this approach are given in Section 4.3. For more details on Column Generation and how to generally hybridize it the reader may refer to Chapter 3. For a survey on hybrid algorithms, including the embedding of heuristics and meta-heuristics into Column Generation frameworks, the reader may refer to [15]. Many variants of MLP have been proposed as well, in order to fit the original problem to different scenarios that need of different sensing models (see Section 2.2.1). Some of the suggested variants take into account cover connectivity ([5], [110], [84], [30], [28]) or reliability issues ([32]), or consider sensors with adjustable sensing ranges ([23], [31], [91]). For many of these variants, efficient algorithms based on Column Generation have been proposed ([5], [84], [31], [91], [92], [27], [30], [32], [28]). One of the most interesting variant of the

problem is the Maximum Network $\alpha$-Lifetime Problem ($\alpha$-MLP), which was addressed in [54]. In such a variant, a predefined quantity of the overall number of the targets is allowed to be not monitored in each cover. As can be easily deduced and will be better showed in Section 4.2, $\alpha$-MLP generalizes MLP and therefore each algorithm aimed at solving this problem can also be applied to address the original one. In [54] the authors presented both a heuristic algorithm and an exact one, showing that huge improvements in terms of global network lifetime can usually already be achieved by neglecting a small quantity of targets in each cover. Furthermore, the authors showed also that most of the advantage is generally maintained even if some additional regularity conditions are taken into account in order to assure a minimum global coverage level to each target.

In this Chapter we propose an exact approach for the $\alpha$-MLP problem, named GCG. While the general structure of the method is again based on the Column Generation, the main contribution of this research work consists in the proposal of a specific designed genetic meta-heuristic which is applied to solve the related *Separation Problem*. For an introduction on genetic algorithms and how they work the reader may refer to Chapter 3, while more details on evolutionary approaches can be found in [97],[38], [58]. As will be shown by the computational tests (see Section 4.5) our algorithm is highly efficient in terms of computational time with respect to both the algorithms presented in [40] for MLP and in [54] for $\alpha$-MLP.

The rest of this chapter is organized as follows. Section 4.2 formally introduces the problems and a mathematical formulation to describe them. Section 4.3 briefly resumes the approaches presented in [40] and [54] to solve MLP and $\alpha$-MLP. Section 4.4 describes the design of the genetic algorithm, while Section 4.5 describes the results of our computational experiments.

## 4.2 Problems Definition and Mathematical Formulation

Let $N = (T, S)$ be a wireless sensor network, where $T = \{t_1, \dots t_n\}$ is the set of the targets and $S = \{s_1, \dots, s_m\}$ is the set of sensors. As previously introduced, each sensor is assumed to have a given *sensing range* and a battery that can

keep it in an active state for a limited amount of time. In this research work we assume each sensor to be identical. All sensors have the same sensing range and the same battery characteristics. The battery durations are normalized to 1. In Figure 4.1(a) a sensor network is shown with a set of six sensor $S = \{s_1, \ldots, s_6\}$ and set of six targets $T = \{t_1, \ldots, t_6\}$. The sensing ranges are represented by circles.

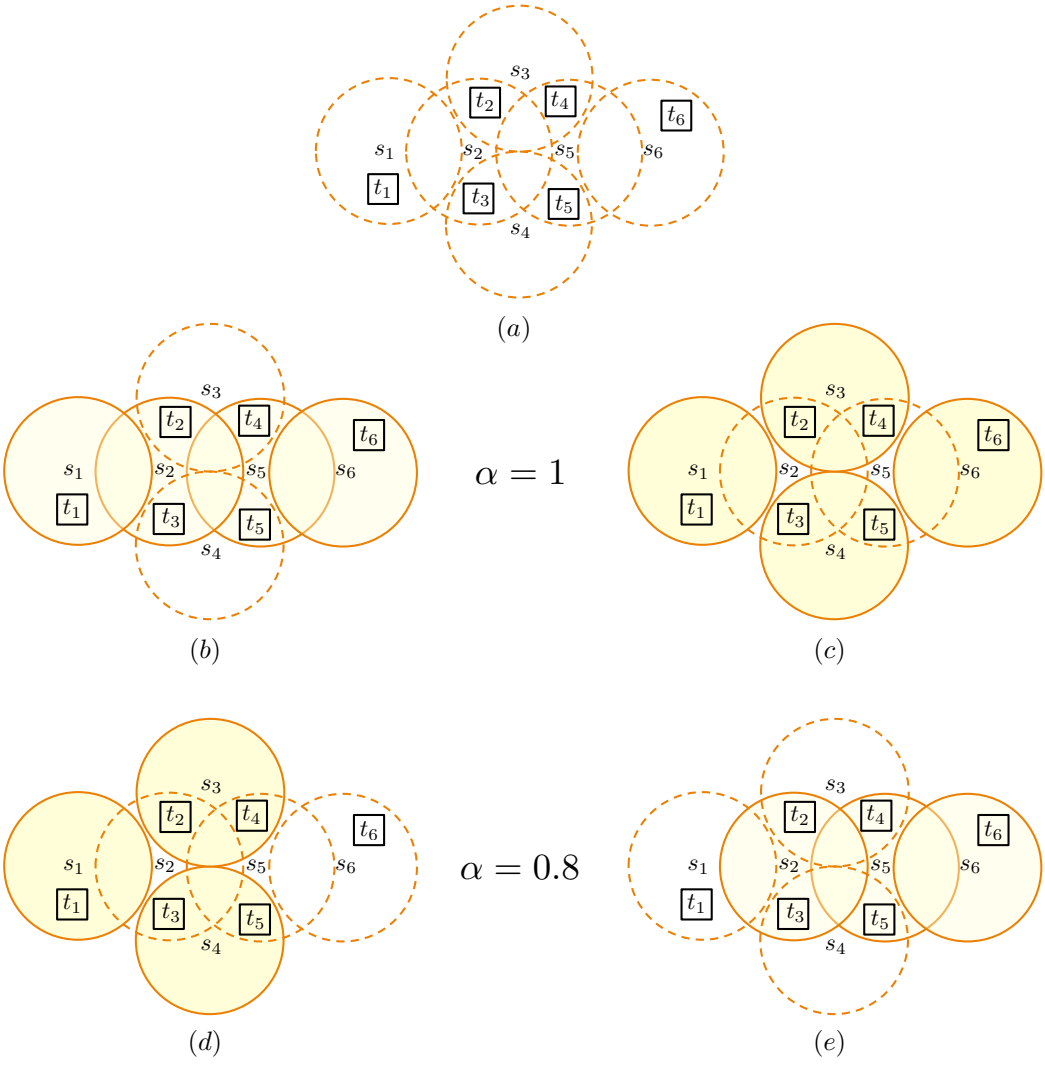



Figure 4.1: *Example Network*

For each target $t_k \in T$ and sensor $s_i \in S$, let $\delta_{ki}$ be a binary parameter equal

to 1 if $t_k$ is positioned within the sensing range of $s_i$ (i.e. $t_k$ is *covered* by the sensor $s_i$), 0 otherwise. For a subset of sensors $S' \subseteq S$ and $t_k \in T$, let $\Delta_{kS'}$ be another binary parameter equal to 1 if $\delta_{ki} = 1$ for a given $s_i \in S'$, 0 otherwise.

Given a value $\alpha \in (0, 1]$, we define $C \subseteq S$ to be a *feasible cover* for the network if its active sensors cover at least $T_\alpha = \alpha \times n$ targets, i.e. $\sum_{t_k \in T} \Delta_{kC} \geq T_\alpha$. Furthermore, we define a cover to be *non-redundant* if it does not contain another cover as a proper subset.

The Maximum Network $\alpha$-Lifetime Problem ($\alpha$-MLP) consists then in finding a collection of pairs $(C_j, w_j)$ where each $C_j \subseteq S$ is a feasible cover and each $w_j \geq 0$ is an activation time, such that the sum of the activation times is maximized and each sensor is used for an amount of time that does not exceed its normalized battery duration. It is straightforward to note that an optimal solution can be found only considering non-redundant covers.

It is also interesting to observe that, on the same wireless sensor network instance, the maximum lifetime for the $\alpha$-MLP is always greater than or equal to the maximum lifetime for the MLP. For example, let us consider again the network in Figure 4.1(a). It is straightforward to observe that the only two feasible covers for MLP would be $\{s_1, s_2, s_5, s_6\}$ (Figure 4.1(b)) and $\{s_1, s_3, s_4, s_6\}$ (Figure 4.1(c)). In this case, it is possible to achieve a network lifetime equal to 1 time unit by activating them for any couple of time $w_1, w_2 \geq 0$ such that $w_1 + w_2 = 1$. However, after this operation, no more feasible covers can be obtained by the remaining sensors with a non-empty battery (we can note that the batteries of sensors $s_1$ and $s_6$ are exhausted), and then the final solution is equal to 1. Let us consider now on the same network an $\alpha$-MLP problem with $\alpha = 0.8$, that is 1 out of 6 targets can be uncovered. In this case there are four feasible not redundant covers $\{s_1, s_3, s_4\}$ (Figure 4.1(d)), $\{s_2, s_5, s_6\}$ (Figure 4.1(e)), $\{s_1, s_2, s_5\}$ and $\{s_3, s_4, s_6\}$ and we can easily achieve a lifetime equal to 2 time unit by activating in sequence the covers $\{s_3, s_4, s_6\}$ and $\{s_1, s_2, s_5\}$, for 1 time unit.

Assuming that we are able to compute in advance the whole set of feasible covers $C_1, \ldots, C_\ell$, $\alpha$-MLP could then be modeled using the following Linear Programming formulation, where the binary parameter $a_{ij} = 1$ if $s_i \in C_j$, 0 otherwise:

$$[\mathbf{P}] \max \sum_{j=1}^{\ell} w_j \tag{4.1}$$

s.t.

$$\sum_{j=1}^{\ell} a_{ij} w_j \leq 1 \qquad \forall i = 1, ..., m \tag{4.2}$$

$$w_j \geq 0 \qquad \forall j = 1, ..., \ell \tag{4.3}$$

Objective function (4.1) maximizes the global lifetime, i.e. the sum of the activation times, while constraints (4.2) enforce the respect of the lifetime constraints for each sensor.

In the classical Maximum Network Lifetime Problem (MLP), each cover has to collect information on the whole set of targets in order to be feasible; therefore, MLP corresponds to the $\alpha$-MLP with $\alpha = 1$ (and hence $T_\alpha = n$). Under these assumptions, the problem definition and the **[P]** formulation presented above represent the classical problem as well.

The potentially exponential number of covers, in particular for large scale instances deriving from real-world scenarios, prevents us to directly apply the previous formulation. This can be especially true for lower values of $\alpha$; indeed, it is easily to observe that given $(\alpha_1, \alpha_2) \in (0, 1]^2$ with $\alpha_2 < \alpha_1$, each cover for $\alpha_1$-MLP is also feasible for $\alpha_2$-MLP. For this reason, it is necessary to apply more efficient approach to solve [**P**] such as Column Generation algorithms proposed by [40] for MLP and by [54] for $\alpha$-MLP. We use the same approach to solve the [**P**] formulation, but we focus our attention on the *Separation Problem*, since solving it efficiently is the key to obtain a fast column generation algorithm. To this end, we design a fast genetic meta-heuristic, defined GA from now on, whose main characteristic is the ability to return several useful feasible covers at once and, as we will see in Section 4.5, this ability will make the difference, in terms of computational time, with respect to the previous algorithms.

## 4.3 Column Generation Approaches for $\alpha$-MLP and MLP

Given the dual prices $\pi_i$ associated to each constraint of the Master Problem, all the feasible covers with a reduced cost lower than 1 are attractive covers for the Master Problem. We can therefore define as subproblem the following formulation [**SP**], where objective function (4.4) minimizes the sum of the dual prices of the sensors chosen to be part of the newly produced cover, while constraints (4.5)-(4.8) define a feasible cover:

$$[\mathbf{SP}] \qquad \min \sum_{i=1}^{m} \pi_i x_i \tag{4.4}$$

s.t.

$$\sum_{i=1}^{m} \delta_{ki} x_i \geq y_k \qquad \forall k = 1, ..., n \tag{4.5}$$

$$\sum_{k=1}^{n} y_k \geq T_\alpha \tag{4.6}$$

$$x_i \in \{0, 1\} \qquad \forall i = 1, ..., m \tag{4.7}$$

$$y_k \in \{0, 1\} \qquad \forall k = 1, ..., n \tag{4.8}$$

For each sensor $s_i$, the binary variable $x_i$ represents the choice on including it in the new cover, while, for each target $t_k$, the variable $y_k$ represents whether the target is monitored in the cover. Constraints (4.5) make sure that each $y_k$ can have value 1 only if at least one of the sensors that cover the target has been added, while constraints (4.6) impose that at least $T_\alpha$ targets are covered. The current incumbent solution is optimal if the value of the objective function (4.4) is greater or equal than 1, otherwise the new attractive cover is added to the master problem. When $\alpha = 1$, that is we are considering the MLP problem, constraints (4.6) reduce to $\sum_{i=1}^{m} \delta_{ki} x_i \geq 1 \ \forall k = 1, ..., n$, and constraints (4.5) as well as variables $y_k$ are not necessary.

### 4.3.1 Heuristic approaches to speed-up the Column Generation Approach

The main disadvantage of the column generation approach proposed above is that the [**SP**] is strongly NP-hard, being a specialization of the Set Covering problem. For this reason, it is advisable to limit as much as possible the number of times in which it is required to be solved. For instance, in [40], the author proposes three column generation based approaches. A first exact approach, named Exact, solves the subproblem to optimality as discussed, while the other two (named *Heur* and *Mixed*, respectively) make use of a constructive heuristic to attempt the generation of new attractive covers. This heuristic iteratively builds a single cover by first choosing a random uncovered target and then selecting the sensor with minimal dual price value that can cover it, until complete coverage has been reached. The *Heur* algorithm transforms the whole CG framework into a heuristic approach by substituting the subproblem formulation with the above described heuristic, ending as soon as it fails. The *Mixed* algorithm, instead, is again an exact approach that uses both the heuristic and the exact MILP formulation to solve the subproblem. More in detail, the subproblem is solved to optimality only when the heuristic fails to produce an attractive cover, in order to find such a cover or certify that it does not indeed exist. In [54], the authors propose instead a heuristic meant to independently produce a complete solution for $\alpha$-MLP (that is, a collection of covers and activation times). Each cover in this approach is again built iteratively, adopting some heuristic criteria to favor the coverage of targets which has been covered for fewer amounts of time so far in the partial solution. Each newly produced cover is assigned a predefined amount of time, and the algorithm ends when the residual energy in the sensors do not allow to produce a new feasible one. Finally, the set of produced covers is used as initial restricted set for the master problem.

In this work, we attempt to heuristically solve [**SP**] at each iteration, by using a genetic meta-heuristic instead of a simple constructive heuristic as the one proposed in [40]. As in the Mixed algorithm, the exact subproblem formulation is used when the genetic algorithm fails in order to guarantee that an exact solution is always found. We define this hybrid exact approach GCG. As we will see later,

thanks to this choice our column generation approach will overcome the above mentioned previous algorithms for MLP and $\alpha$-MLP. Here in Figure 4.2, recalling the introduction in Chapter 3, we can see a simple working scheme that briefly synthesizes our hybrid exact approach. In the first phase (Figure 4.2 (1)) we use our GA to initialize the Master Formulation considered by Column Generation. In the starting phase GA takes as input random dual prices to produce the first set of feasible columns. The master problem restricted to this columns is then solved and is computed and used a first set of dual prices to properly run GA (Figure 4.2 (4)). When the GA reaches a stop condition, we check if in the current population there are columns whose fitness value (corresponding to the objective function of the separation problem) is lower than 1. If is it the case, we add those columns to the RMP formulation and we go back to step 3, otherwise we solve the [SP] problem, we check if there is a new column that we can add to the RMP (in which case we go back to step 3) or if the current solution is the optimal one.

## 4.4 A Genetic Algorithm to Solve the Subproblem [SP]

In this section we describe our genetic algorithm designed to hybridize and to enhance the column generation approach. For a complete and detailed description about the genetic algorithms the reader can refer to [97],[38], [58]. We briefly recall that a genetic algorithm is a naturally stochastic technique that emulates the typical steps of the biological evolution based on the concept of *natural selection*, *crossover* and *mutation*. Each problem solution is expressed by an element, named *chromosome*, that represents the structure of an individual. Given a starting population $P$ of chromosomes, the genetic algorithm produces new chromosomes by means of the crossover operator that combines, in a probabilistic manner, the genetic information of typically two or more naturally selected chromosomes of the population. On the newly built chromosome, a mutation operator is applied in order to provide a perturbation of the solution without irreversible loss of genetic material. The natural selection together with the *fitness* function, used to rank each solution, guarantee that new chromosomes are typically better

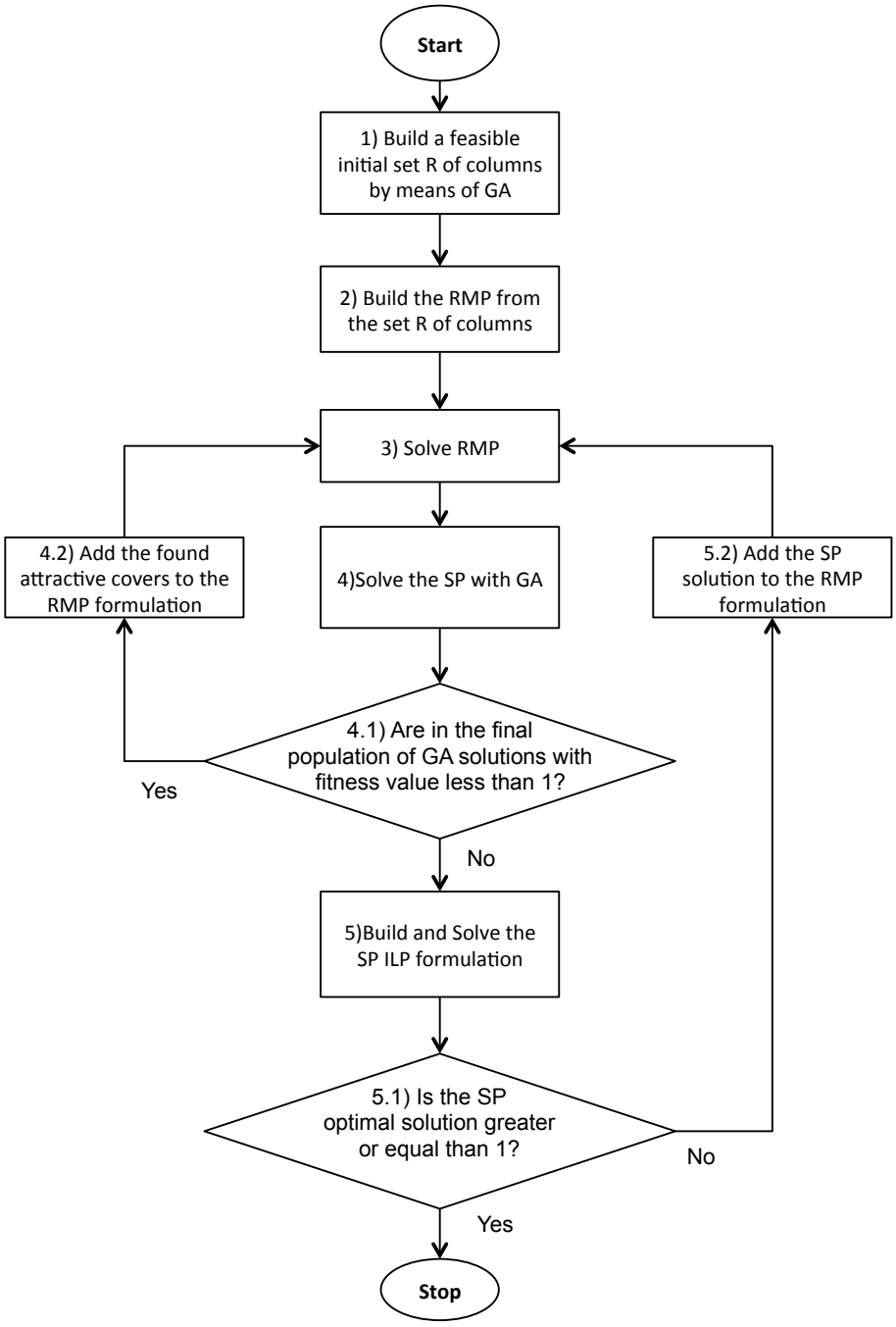



Figure 4.2: *Hybrid Scheme*

adapted to the environment. The genetic algorithms iterations are typically regulated by stop conditions as a maximum number of iterations, a specific amount of time, the lack of improvements in the fitness function of the best individual or by a specific subset of conditions related to the problem. As previously introduced, the aim of the GA algorithm is to quickly find attractive covers and return them to the master problem. As we will see in the section 4.5, GA is very effective since often it fails just once, i.e. when the optimal solution is found. Moreover, our genetic approach has the ability to produce several attractive covers at once, reducing dramatically the number of required iterations. As a consequence, our GCG algorithm converges noticeably faster than previous approaches. In the next subsections the details of our genetic algorithm are given.

### 4.4.1 Chromosome Representation and Fitness Function

GCG is based on the binary chromosome, shown in Figure 4.3, composed by $m = |S|$ positions, each one associated to a sensor of the network. In our genetic algorithm, each chromosome represents a feasible not-redundant cover and each position $i$ of the chromosome is equal to 1 if the sensor $s_i$ is *active* in the cover and 0 otherwise.

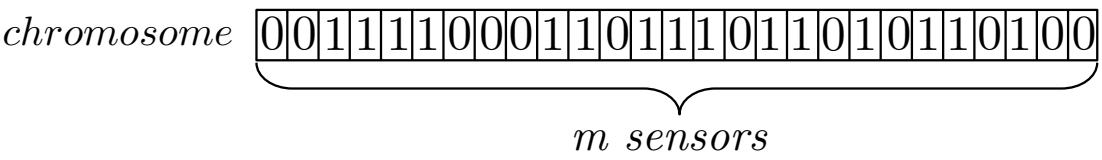



Figure 4.3: *The chromosome representation.*

The value of the $i$-th position ($i = 1, ..., m$) in the chromosome corresponds to the binary value of the $x_i$ variable in the **[SP]** formulation. Analogously to covers, a chromosome is defined to be redundant if it is possible to switch off at least one of its active sensors, and the related cover remains feasible. Since, as already mentioned, an optimal solution can always be found by only considering non-redundant feasible covers, during the GA execution we only allow non-redundant chromosomes to be part of the population.

The fitness function for a given chromosome is the dot product of the binary chromosome vector and the dual prices vector coming from the last iteration of the Master Problem (and therefore corresponds to objective function (4.4) for [**SP**]). At the end of the GA procedure, each chromosome, with a fitness lower then 1, will be included in the Master Problem as a new column.

### 4.4.2 Crossover

One of the main aspects that influence the effectiveness of a genetic algorithm is the crossover operator. This operator allows the creation of new chromosomes starting from previous ones. In particular, the crossover usually selects two members of the population (defined *parents*), and generates a new one starting from them (the *child*), which hopefully inherits their best features. During the evolution process of a genetic algorithm, special care should be taken in order to avoid the case in which several identical chromosomes exist in the population; indeed, in that case the crossover operator has failed to create offspring that is different from their parents. This situation penalizes the effectiveness of the algorithm and therefore the quality of the final solutions. In our crossover, the selection of the parents is carried out through a typical binary tournament (see Algorithm 1). To this end, the chromosomes of the population are initially sorted, in ascending order, according to their fitness values. Subsequently, two chromosomes ($C_{1rand}$ and $C_{2rand}$ in Algorithm 1) are selected randomly among all them and the one with best fitness is chosen as first parent ($p_1$ in Algorithm 1). As we can see in the pseudocode, the second parent $p_2$ is chosen in the same way, avoiding the first parent to be chosen as participant of the second tournament. Our crossover operator works exactly like the AND logical operator. The Figure 4.4 shows, on the left, the AND truth table and, on the right, two example chromosomes, $parent1$ and $parent2$, from whose the crossover operator builds the *child* one. This type of operator ensures the common heritage belonging to both parents to be transmitted to the child.

*crossover operator* $child_i = parent1_i \wedge parents2_i \ \forall i \in \{1, ..., m\}$

| *parent* 1 | *parent* 2 | *child* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

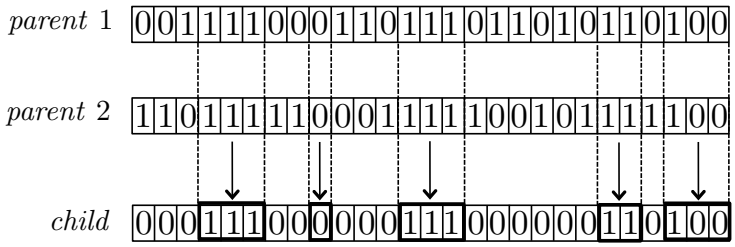



Figure 4.4: *The crossover operator.*

**Algorithm 1:** *tournament*

**Input**: $P$;
**Output**: couple of parents $(p_1, p_2)$;

```
1  p1 ← ∅;
2  p2 ← ∅;
3  while p1 = ∅ ∨ p2 = ∅ do
4      C1rand ← randomSelect(P − {p1});
5      C2rand ← randomSelect(P − {p1, C1rand});
6      if p1 = ∅ then
7          if fitness(C1rand) <= fitness(C2rand) then
8              p1 ← C1rand;
9          else
10             p1 ← C2rand;
11     else
12         if fitness(C1rand) <= fitness(C2rand) then
13             p2 ← C1rand;
14         else
15             p2 ← C2rand;
```

### 4.4.3 Mutation

Mutation is a genetic operator that alters one or more genes in a chromosome to introduce perturbation and therefore provides diversification in the new generated chromosomes. We recall that in our genetic algorithm no duplicated chromosomes are allowed in the population. This means that generating a duplicate is a waste of computational time since it would be rejected. Since we do not generate new chromosomes taking into account all the chromosomes of the population, we try to differentiate each child from at least both its parents, if possible. It is common for a new generated chromosome to coincide with one of its parents, in particular when they are very similar, that is the most part of their genes are identical. Indeed, in this situation, the child generated by our crossover operator will also be similar to the parents and the following operations carried out on it (see section 4.4.4) could make it identical to one of them. In order to face this problem, we use mutation to change the value of one of a random single gene in the child whose value is identical into its parents, if it exists, in order to differentiate it from both of them. This gene will be switched back only if strictly needed by the feasibility or redundancy operator described in the next section.

### 4.4.4 Fixing and Redundancy Operators

It is easy to see that the chromosome produced by the crossover and mutation operators could be unfeasible since it is not guaranteed that the $T_\alpha$ coverage is satisfied. For this reason, it is necessary to apply another operator, named *fixing* operator, that restores the feasibility of this chromosome. To this end, the fixing operator selects randomly one of the genes in the child with value equal to zero and switches its value to one (switching the sensor in the active state). This process is repeated until the threshold $T_\alpha$ is satisfied. The Algorithm 2 shows the pseudocode of this operator. The while loop (line1) is repeated until the threshold is reached. The procedure individuates the set of uncovered targets $\hat{T}$ (line 2) and, randomly selects one of these targets (line 3), let us say $t$. Then it randomly selects and activates one of the sensors, $s$, that cover $t$ (line 4). Finally, the last two lines update the chromosome child, thus switching to 1 the gene associated to $s$, and the set of covered targets, adding the new targets covered by $s$.

**Algorithm 2:** Fixing Operator

**Input**: unfeasible $Child$ chromosome;
**Output**: feasible $Child$ chromosome;

```
1 while |T_covered| < T_α do
2     T̂ ← T \ T_covered;
3     t ← randomSelect(T̂);
4     s ← randomSelect(S, t);
5     update(Child, s);
6     update(T_covered);
7 return Child;
```

The application of the $fixing$ operator can produce a redundant chromosome. Since we don't allow redundant chromosome in the population, we remove the redundancy by applying another operator, named *Redundancy*. The operator tries to switch off active sensors without compromising the feasibility. Checking each sensor, step by step, the procedure generates a list of redundant sensors, then it switches off a randomly chosen element of the list. The chromosome is updated, the list is rebuilt and the procedure iterates until the list is equal to the empty set.

### 4.4.5 Building the Initial Population

The initial population $P$ is composed of $Size_P$ different chromosomes. The procedure works iteratively applying on a starting vector, with all the positions set to zero, the $fixing$ and $redundancy$ operators. As soon as a feasible chromosome is obtained, it is added to the population, if it is not already present in it, otherwise it is rejected. The procedure iterate until a fixed desired number $Size_P$ of different chromosomes is obtained. Due to the instances given in input, there are situations in which it is hard or not possible to generate $Size_P$ different chromosomes. To face these situations, we set a threshold *maxinitDB* and when the number of rejected chromosomes reaches this threshold, the procedure stops and returns the current population whatever is the number of chromosomes in it updating the value os $Size_p$ accordingly.

**Algorithm 3:** *InitP*

**Input**: $(S, T), Size_P$;
**Output**: initial population P;

```
1  P ← ∅;
2  doubles ← 0;
3  while |P| ≤ Size_P do
4      C ← ∅;
5      C ← fixingOperator(C);
6      C ← redundancyOperator(C);
7      if C ∉ P then
8          Insert(C, P);
9      else
10         if doubles ≤ maxinitDB then
11             doubles++;
12         else
13             Size_p ← |P|;
```

### 4.4.6 GA Overall Structure

This section describes the main GA structure used in our CG approach. The pseudocode is listed in Algorithm 4. The input consists of a wireless sensor network $(S, T)$, where $S$ is the set of sensors and $T$ the set of targets, and a vector of dual prices $DP$ coming from the last iteration of the current Restricted Master Problem. The GA first generates a starting population $P$ of feasible solutions and identifies the initial best chromosome fitness value $BestFit$. During the evolution process the population is kept ordered based on the fitness, $BestFit$ is kept updated to store the value of the *incumbent* solution and it is used as a comparison parameter through the overall procedure. The population of individuals has a fixed size dimension, named $(Size_P)$, and it is initialized taking into account the coverage requirements as described in Section 4.4.5. The genetic algorithm iterates the typical evolutionary steps of *crossover* (Section 4.4.2) and *mutation* (Section 4.4.3) until one or both of the stop criteria is reached. At each iteration the GA applies two fundamental operators, *fixing* and *redundancy* (Section 4.4.4) on each new generated chromosome. The new child produced in this way is then

**Algorithm 4:** WSNGenetic

**Input**: $(S, T), DP$;
**Output**: a subset of chromosomes(i.e. columns) for the MasterProblem;

```
1  P ← InitP();
2  BestFit ← bestFitness(P, DP);
3  criteria ← setCriterion(MaxIT, MaxDB);
4  while check(criteria) do
5      (p1, p2) ← tournament(P);
6      C ← Crossover(p1, p2);
7      C ← Mutation(C);
8      C ← fixingOperator(C);
9      C ← redundancyOperator(C);
10     if C ∉ Pop then
11         Insert(C, P);
12         if fitness(C) ≥ BestFit then
13             update(criteria);
14         else
15             BestFit ← fitness(C);
16     else
17         update(criteria);
18 Chromos ← chromosomes with fitness ≤ 1;
19 return Chromos;
```

inserted in current population $P$ only if it does not belong to it and it replaces a randomly selected chromosome among the $|P/2|$ individuals with worst fitness value. Then the population is sorted according to the fitness values. The first stop criterion is based on the *MaxIT* parameter, that is the maximum number of iterations without improvements with respect to the *BestFit* value, it is a parameter that is updated every time a new child has a fitness bigger or equal than the *BestFit*. The second stop condition is the maximum number of consecutive duplicate chromosomes, *MaxDB*, generated by the evolutionary steps, in details this parameter is updated when a new child is already present in the current population and it is reset when a new one child is not already resent in the current population. Once a stopping condition is reached, the chromosomes in the current population $P$ with a fitness value less than 1 are introduced in the

Master Problem as new columns. The GA algorithm was also used in our tests to provide the initial set of columns which is required by the first step of the master problem. In this case, however, the vector of dual prices which is used to evaluate the chromosomes is not available. For this reason, in this first iteration a random positive value is used as dual price for each sensor. The whole set of $Size_P$ individuals is returned to the master problem in this case.

## 4.5 Computational Results

The purpose of the computational experience presented in this section is to study the performance of our GCG algorithm with respect to column generations approaches proposed in literature by [54] ($GR$) for the $\alpha$-coverage problem and by [40] for the case when $\alpha = 1$. We tested our algorithm on the same set of instances used in [54] and [40], which were provided by the authors. Our algorithm was coded in C++ on a (SUSE) Linux platform running on a Intel Core2 Duo 2.4GHz desktop computer with 4GB RAM (single thread mode). We have performed our column generation algorithm using the Concert library of IBM ILOG CPLEX 12.5. We first ran a set of tuning tests to define the values of the parameters used by the our GA algorithm. The $Size_p$ population size was set to be equal to 50. The population initialization threshold $maxinitDB$ was chosen equal to 100. Finally, the two stopping criteria $MaxDB$ and $MaxIT$ were set to 100 and 2000, respectively.

Let us start our comparison from the benchmark instances proposed in the literature by Deschinkel [40]. In Table 4.1 the results of GCG, and of the genetic algorithm in it embedded, are reported. Each line in the table represents a *scenario* composed by 10 instances with the same characteristics but different topologies. Therefore, the results reported in each line are the average values on these 10 instances. For a detailed description of the characteristics of these scenarios see [40]. The first two columns ($Sensors$ and $Targets$) report the number of sensors and targets into the scenarios. The columns $Lifetime$ and $Time$ report the solution values and the CPU times, in seconds. The last three columns $Inv$, $Col$ and $Flr$ report how many times the genetic algorithm is invoked by the restricted master problem after the initialization phase, the average number of

| Sensors | Targets | Lifetime | Time | Inv | Col | Flr |
|---|---|---|---|---|---|---|
| 50 | 30 | 3.80 | 0.21 | 1.0 | 0.0 | 1.0 |
| | 60 | 3.00 | 0.31 | 1.0 | 0.0 | 1.0 |
| | 90 | 2.80 | 0.40 | 1.0 | 0.0 | 1.0 |
| | 120 | 2.70 | 0.51 | 1.0 | 0.0 | 1.0 |
| 100 | 30 | 8.70 | 0.44 | 1.6 | 10.1 | 1.0 |
| | 60 | 7.20 | 0.65 | 1.4 | 6.1 | 1.0 |
| | 90 | 6.90 | 1.11 | 1.6 | 8.5 | 1.0 |
| | 120 | 6.70 | 1.57 | 1.5 | 7.4 | 1.0 |
| 150 | 30 | 14.70 | 0.80 | 2.6 | 20.4 | 1.0 |
| | 60 | 12.30 | 1.41 | 2.4 | 18.8 | 1.0 |
| | 90 | 11.80 | 2.40 | 2.3 | 19.6 | 1.0 |
| | 120 | 11.30 | 3.38 | 2.3 | 19.9 | 1.0 |
| 200 | 30 | 19.60 | 1.24 | 2.9 | 24.4 | 1.0 |
| | 60 | 17.30 | 2.39 | 2.6 | 23.2 | 1.0 |
| | 90 | 16.60 | 4.10 | 3.0 | 24.5 | 1.0 |
| | 120 | 15.50 | 5.14 | 2.7 | 24.4 | 1.0 |
| **Avg** | | | | **1.93** | **12.96** | **1.0** |

Table 4.1: Results obtained by the GCG algorithm on the benchmark instances proposed in [40].

columns (i.e. attractive covers) returned by the genetic algorithm at each invocation (again excluding the starting one), and how many times the genetic algorithm returns zero columns (i.e. the number of failures), respectively. Finally, the last line of the table reports the average values of the last three columns. From the values of the $Time$ column, it is evident that GCG is very fast because it finds the optimal solution in few seconds on all the scenarios. However, we postpone to the Table 4.2 the performance comparison among GCG and the other algorithms because, with the results of Table 4.1, we want to highlight the impact of the genetic algorithm inside our column generation approach. To this end, we analyze the values reported in the last three columns of the Table 4.1. The values of the column $Inv$ show that very few invocations of the genetic algorithm are needed

to provide to the master problem all the columns needed to find the optimal solution. In particular, on the scenarios with 50 sensors, it is invoked just once and this means that the starting columns, provided by genetic algorithm during the initialization phase, already contain the columns of the optimal solution. Indeed, a single invocation after the initialization means that the GA failed and the exact subproblem certified that an optimal solution was indeed reached, otherwise GA would have been invoked again in the following iteration. Moreover, on average, we have less then 1.6 invocations on the instances with 100 sensors and less than 3 invocations on the instances with 150 and 200 sensors, with an overall average equal to 1.93. The number of invocations is small since the genetic algorithm returns a significant number of attractive covers, at each invocation, which is on average equal to 12.96 columns, with a peak of 24.5, which brings the columns needed to reach an optimal solution to be quickly added to the master problem. In particular, on the largest instances with 200 sensors the average number of returned columns is above 24, that is, almost 50% of the chromosomes in the final population are attractive covers for the restricted master problem. However, the more interesting results are these reported into the column $Flr$ which measure the effectiveness of the genetic algorithm. Every time the genetic algorithm does not find any attractive cover for the restricted master problem, we are forced to solve the subproblem exactly in order to certify the optimality of the incumbent solution or to find a new attractive cover that the genetic algorithm has missed. This means that the genetic algorithm have to fail at least once, that is when the optimal solution is found. Remarkably, on all the scenarios reported in [40] the number of failures of our genetic algorithm is always equal to 1. This means that we solve the subproblem ILP formulation just once for each instance, when it is used to certify the optimality of the current incumbent solution. According to the results of the Table 4.1, we expect the effectiveness of the genetic algorithm to be able to speed up the convergence of our column generation approach, making it competitive with respect to the other algorithms proposed in literature. In order to verify this, the computational times of our GCG and of $Exact$, $Heur$ and $Mixed$ algorithms, proposed by [40], are reported in the Table 4.2. The first three columns show the characteristics of the scenarios already mentioned in Table 4.1 (columns *sensors*, *targets*, *Lifetime*). The following four columns

| Sensors | Targets | Lifetime | Exact | Heur | Mixed | GCG | GAP | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Time | Time | Time | Time | vs Exact | vs Heur | vs Mixed |
| 50 | 30 | 3.80 | 0.25 | 0.30 | 0.12 | 0.21 | | | |
| | 60 | 3.00 | 1.03 | 0.53 | 0.52 | 0.31 | | | |
| | 90 | 2.80 | 2.95 | 0.82 | 1.55 | 0.40 | 86.42% | | 74.15% |
| | 120 | 2.70 | 8.40 | 1.20 | 4.03 | 0.51 | 93.87% | | 87.22% |
| 100 | 30 | 8.70 | 3.29 | 2.97 | 1.03 | 0.44 | 86.75% | 85.32% | |
| | 60 | 7.20 | 26.53 | 4.25 | 8.41 | 0.65 | 97.55% | 84.71% | 92.28% |
| | 90 | 6.90 | 243.95 | 6.82 | 74.19 | 1.11 | 99.55% | 83.77% | 98.51% |
| | 120 | 6.70 | 749.46 | 9.70 | 220.64 | 1.57 | 99.79% | 83.79% | 99.29% |
| 150 | 30 | 14.70 | 17.17 | 14.51 | 4.94 | 0.80 | 95.37% | 94.52% | 83.89% |
| | 60 | 12.30 | 315.66 | 22.21 | 48.96 | 1.41 | 99.55% | 93.65% | 97.12% |
| | 90 | 11.80 | 2365.65 | 30.61 | 525.21 | 2.40 | 99.90% | 92.17% | 99.54% |
| | 120 | 11.30 | 9249.81 | 48.15 | 1987.04 | 3.38 | 99.96% | 92.98% | 99.83% |
| 200 | 30 | 19.60 | 38.80 | 34.85 | 9.50 | 1.24 | 96.80% | 96.44% | 86.93% |
| | 60 | 17.30 | 750.40 | 56.34 | 126.39 | 2.39 | 99.68% | 95.75% | 98.11% |
| | 90 | 16.60 | 8229.53 | 132.46 | 1297.82 | 4.10 | 99.95% | 96.91% | 99.68% |
| | 120 | 15.50 | 28942.49 | 105.87 | 4393.04 | 5.14 | 99.98% | 95.15% | 99.88% |
| **AVG** | | | **3184.09** | **29.47** | **543.96** | **1.63** | **96.79%** | **91.26%** | **93.57%** |

Table 4.2: Comparative of $GCG$, $Exact$, $Heur$ and $Mixed$ algorithms on the Deschinkel's benchmark instances.

report the CPU time, in seconds, required by the four algorithms. The last three columns report the percentage gap, among GCG and the other three algorithms, computed as $100 \times (Alg - GCG)/Alg$ where $Alg \in \{Exact, Heur, Mixed\}$, and $GCG$, $Alg$ refer to the computational time of the related procedure. Finally, the last line of the table reports the average values of the last seven columns. Note that when the CPU Time gap between two algorithms is lower than 1 second, we do not report the percentage gap, in the last three columns, because we consider this gap negligible and we do not want that this value to affect the significance of the average gaps contained in the last line of the table. As previously mentioned, the results of the Time column for GCG show that it is able to find the optimal solution in less than 6 seconds on average whatever are the characteristics of the considered scenario. Therefore, the increment in terms of requested CPU time, as the size of scenarios grows, is bounded to few seconds. The situation appears to be completely different for the other three algorithms, that appear to be much slower, and whose computational times are significantly affected by the scenarios characteristics. More in details, from the average values of the last line it is evident that GCG is faster than Exact by three orders of magnitude with a gap that is always greater than 86%. It is impressive to observe the performance

difference on the last scenario of the table (200 sensors and 120 targets) where the Exact algorithm spends more than 8 hours to find the optimal solution while GCG requires only 5 seconds. The Mixed algorithm results to be faster than the Exact algorithm but when compared to the GCG algorithm it is slower by two order of magnitude. Moreover, the performance gap between these two algorithm is always greater than 83%. Finally, it is remarkable to note that GCG results to be 20 times faster than the heuristic approach *Heur*, with a percentage gap which is always greater than 74%.

It has to be highlighted that this comparison cannot be completely accurate since the algorithms proposed in [40] were run on a different hardware and the mathematical models were solved using GLPK. However, since the running time gap can be quantified in orders of magnitude, we believe that the comparison still provides solid evidence about the effectiveness of our approach.

The results of GCG, on one hand, confirm our expectations on the effectiveness and efficiency of our GA algorithm and, on the other hand, prove that a column generation approach, paired with a fast and effective method to generate new columns, results to be a very suitable approach for lifetime problems on sensor networks.

We now present the results of GCG when used to solve the Group 2 set of benchmark instances proposed in [54] for the $\alpha$-coverage problem. This is the hardest set of instances considered in that paper, and therefore we considered the results on it to be more revelevant and interesting. Nevertheless, we also tested our approach on the Group 1 dataset, and the related tables are contained in the Appendix A. As will be shown, GCG performs well on all these instances as well.

The Group 2 instances contain 100 targets, while the number of sensors is not fixed *a priori*, but is rather computed assuring that each target is covered by at least 3 sensors. The instances are further divided in two subgroups, named *Scattering* and *Design* respectively. In the *Scattering* group sensors are added randomly until the desired coverage level is reached, while in the *Design* group, sensors are added only when needed to reach such coverage. For a detailed description of the characteristics of these instances see [54].

In Table 4.3 the results of GCG on the *Scattering* and *Design* are reported. The first two columns specify the type of instance and the number of targets

| Inst. Subgroup | Targets | $T_\alpha$ | LifeTime | Time | Inv | Col | Flr |
|---|---|---|---|---|---|---|---|
| **Design** | 100 | 50 | 8.32 | 0.41 | 4.30 | 19.20 | 1.03 |
| | | 75 | 5.42 | 0.77 | 9.90 | 13.63 | 2.00 |
| | | 85 | 4.50 | 0.90 | 11.50 | 12.27 | 2.87 |
| | | 93 | 3.65 | 0.52 | 6.97 | 14.03 | 1.57 |
| | | 95 | 3.34 | 0.39 | 4.80 | 11.87 | 1.20 |
| | | 97 | 3.04 | 0.26 | 2.13 | 7.43 | 1.00 |
| | | 99 | 3.00 | 0.27 | 2.03 | 11.90 | 1.00 |
| | | 100 | 3.00 | 0.29 | 2.37 | 13.63 | 1.00 |
| **Scattering** | 100 | 50 | 20.50 | 1.19 | 6.20 | 23.30 | 1.03 |
| | | 75 | 13.36 | 9.14 | 39.77 | 10.03 | 8.07 |
| | | 85 | 10.57 | 9.12 | 52.57 | 8.13 | 10.47 |
| | | 93 | 7.73 | 2.35 | 16.10 | 17.77 | 2.10 |
| | | 95 | 6.64 | 1.22 | 7.63 | 20.27 | 1.40 |
| | | 97 | 5.37 | 0.74 | 3.37 | 17.00 | 1.03 |
| | | 99 | 3.83 | 0.56 | 1.67 | 7.17 | 1.00 |
| | | 100 | 3.00 | 0.48 | 1.00 | 0.00 | 1.00 |
| **Avg** | | | | | **10.77** | **12.98** | **2.36** |

Table 4.3: Results obtained by the GCG algorithm on the Group 2 benchmark instances proposed in [54].

present in it. The column $T_\alpha$ specifies the number of target that must be covered while the columns $Lifetime$ and $Time$ reports the solution value and the CPU time, respectively. Finally, the last 3 columns show the results of the genetic algorithm already mentioned in Table 4.1. Each line in the table represents a scenario composed by 30 instances with the same characteristics but a different topologies. Therefore, the results reported in each line are the average values on these 30 instances.

The results of Table 4.3 show that for these instances the number of GA invocations is on average 10.77, the number of columns returned is approximately 12.98 and the number of average failures is 2.36. More in detail, on the *Design* scenarios we register a peak of GA invocations equal to 11.50 for the case $T_\alpha = 85$, which also corresponds to the peak of failures, equal to 2.87. The average number of columns returned for each iteration is greater than 10 in all cases except one, in the case $T_\alpha = 97$. The *Scattering* instances result to be harder to solve, with a peak of GA invocations and failures corresponding to 52.57 and 10.47,

| Inst. Subgroup | $T_\alpha$ | GR | | GCG | | GAP |
|---|---|---|---|---|---|---|
| | | LifeTime | Time | LifeTime | Time | |
| **Design** | 50 | 8.32 | 3.20 | 8.32 | 0.41 | 87.28% |
| | 75 | 5.42 | 13.94 | 5.42 | 0.77 | 94.46% |
| | 85 | 4.50 | 11.46 | 4.50 | 0.90 | 92.15% |
| | 93 | 3.65 | 7.03 | 3.65 | 0.52 | 92.56% |
| | 95 | 3.34 | 2.68 | 3.34 | 0.39 | 85.41% |
| | 97 | 3.04 | 1.43 | 3.04 | 0.26 | 81.61% |
| | 99 | 3.00 | 0.59 | 3.00 | 0.27 | |
| | 100 | 3.00 | 0.21 | 3.00 | 0.29 | |
| **Scattering** | 50 | 20.50 | 11.13 | 20.50 | 1.19 | 89.30% |
| | 75 | 13.36 | 216.98 | 13.36 | 9.14 | 95.79% |
| | 85 | 10.56** | 302.91 | 10.57 | 9.12 | 96.99% |
| | 93 | 7.38* | 36.18 | 7.73 | 2.35 | 93.50% |
| | 95 | 6.64 | 8.02 | 6.64 | 1.22 | 84.78% |
| | 97 | 5.37 | 2.01 | 5.37 | 0.74 | 63.15% |
| | 99 | 3.83 | 0.56 | 3.83 | 0.56 | |
| | 100 | 3.00 | 0.05 | 3.00 | 0.48 | |
| **AVG** | | | **38.65** | | **1.79** | **88.08%** |

Table 4.4: Computational results of GCG and GR algorithms for the $\alpha$-coverage WSN problem.

respectively (again in the case $T_\alpha = 85$). This can be explained considering the additional number of sensors, and therefore the higher amount of feasible covers which exists in such instances.

It can be noticed that also on this dataset GA only fails once for the highest values of $T_\alpha$, and therefore the problem approaches the classical MLP. In particular, this happens for each instance with $T_\alpha \geq 97$ for the *Design* dataset and with $T_\alpha \geq 99$ for the *Scattering* one.

Despite the results appear to be less favorable than the ones presented in 4.1, the values in the Time column show that GCG is still very fast. Indeed, the algorithm finds the optimal solution in less than 1 second on average in all scenarios for the *Design* instances, and always in less than 10 seconds on average for the *Scattering* ones.

In Table 4.4 a performance comparison between GCG and the GR algorithm is

performed. As mentioned above, we do not evaluate gaps when both procedures report a computational time which is below 1 second. On the *Design* scenarios, the GR algorithm finds all solution within the considered 1 hour time limit. However, it is clear that GCG is generally much faster, with a percentage gap greater than 81% on the first 6 scenarios and a CPU Time always lower than a second. More interesting are the results on the *Scattering* scenarios, where some of its instances are not solved within 1 hour time limit by the GR algorithm. More in detail, it reaches the time limit on 2 instances of the scenario with $T_\alpha = 85$ and 3 instances of the scenario with $T_\alpha = 93$. The solution values of these scenarios are marked into the table with the symbols "*" and "**" to highlight that these values are averages evaluated only on the subset of instances which were solved to completion.

The values of column GAP show that GCG is at least 63% faster than GR with a peak equal to 97% and an average equal to 88%. The values reported in the last line show that GCG is faster than GR by an order of magnitude with a CPU time lower than 2 seconds with respect to the 38 seconds required by GR algorithm. These results certify that GCG is the fastest algorithm and that it is also more effective, since it can solve within 10 seconds at most on average all the considered scenarios.

# Chapter 5

# The Maximum Lifetime Problem of Sensor Networks with Multiple Families

## 5.1 Introduction

Prompted by the heterogeneity of modern networks and by the wide range of different sensor devices existing nowadays, in this chapter we address the well known Maximum Lifetime Problem (MLP) in Wireless Heterogeneous Sensor Networks. Here we describe the problem and the essential literature to better introduce, subsequently, the mathematical formulation of the problem and the designed resolution approach. Due to technological advances, introduced in Chapter 1 and 2, which enabled their deployment in relevant and diverse scenarios, WSNs have been object of intense study in the last few years. Possible application contexts include traffic control, environmental monitoring, intrusion detection and patient monitoring in healthcare among others (see, for example, Chapter 1 and 2, [4], [16], [37]). The typical structure of a WSN is composed of several hardware devices (*sensor nodes*) installed over a given area that we want to monitor. Each sensor can collect information or measure physical quantities about the space around it (its *sensing area* defined by its *sensing range*), and more in particular about specific points (*target points* or simply *targets*) within this area. The target

points inside the sensing area of a given sensor are defined as *covered* by it.

Each sensor is generally powered by a battery that can guarantee it functional for a limited amount of time, due mainly to well known cost and structural constraints. Using such sensing devices in a dynamic and coordinated fashion makes it possible to realize a sensor network able to overcome the hardware constraints in terms of range extension and battery duration which characterize each sensor node, enabling complex sensing activities on large areas of interest. Prolonging the amount of time (i.e. lifetime), over which such control activities can be performed, has therefore emerged as an issue of great relevance. Generally known as Maximum Lifetime Problem (MLP), it has been widely addressed in the literature by proposing approaches to compute several, possibly overlapping subsets of sensors which are independently able to guarantee the coverage request for the target points (*covers*), and by activating them one at a time for proper amounts of time such that energy constraints are not violated. It should be noted, as reported in Chapter 2, that while sensors could be considered as belonging to different states during their usage in the intended application (such as receiving, transmitting, or idle) (see Section 2.3.2) in this context two essential states can be identified. That is, each sensor may currently be active (i.e. used in the current cover, and consuming its battery) or not. Activating a cover refers therefore to switching all its sensors to the active state, while switching off all the other ones.

A considerable amount of research has been proposed in the literature to approach MLP and its variants. As reported in Chapter 2, the problem was shown to be NP-Complete in [22]. Earlier works such as [10] and [22] proposed approximation and heuristic algorithms to solve it. Among the variants of the problem there are cases where a certain quantity of target points may be left uncovered by each cover ([54], [90], [103]), cases where connectivity issues are taken into account in order to route the sensed and processed information to a central processing facility ([5], [84], [110]), or in which the sensing ranges can be adjusted in order to provide optimal trade-offs among energy consumption and coverage ([23], [31], [91]). Furthermore, while the sensing radius of each sensor node is generally only limited by a certain threshold distance (i.e. they provide coverage on 360 degrees around them), some authors also investigated the case in which the sensing activity is limited to an adjustable restricted angle

([2], [20], [92]), as in the case of ultrasonic sensors or video cameras. Recently, among the proposed resolution methods for MLP variants, Column Generation approaches have proved to be efficient methods to solve reasonably large instances to optimality ([5], [31], [32], [54], [84], [91], [92]).

Most of the above cited works take into account homogeneous networks, that is, networks whose sensing devices are perfectly equal and therefore have the same sensing and processing capabilities. This assumption is reasonable for scenarios where a large number of sensor nodes based on the same hardware is installed. Nevertheless, in the context of WSNs and coverage problems defined on them, heterogeneity has been studied as well, in terms of different metrics. In [43], [72], [79], [100], [8], sensors belonging to a limited subset are provided with bigger batteries, and in other cases have longer transmission ranges and better processing capabilities, often in relation to heterogeneous schemes (recall Section 2.2.2.1) where such sensors serve as supernodes. Other works consider heterogeneity in a non-hierarchical context, allowing individually different sensors or sensing capabilities. For example, sensors with possibly variable energy resources are discussed in [74] and [92], while heterogeneous sensing ranges were addressed in [71] and [108].

Less attentions have been dedicated to the case of networks composed of distinct categories of sensors, where each one fulfills a different sensing task. This network type, as also reported in the Chapter 1 and 2, is becoming more and more interesting thanks to the newer network conceptualizations. Indeed, it could be necessary to monitor different characteristics of the same area of interest. For example, during the monitoring of a certain geographical area for environmental control purposes, different types of sensors could be installed to check the temperatures, the pollution levels, vibrations, as well as for intrusion detection and other peculiar properties of the area under monitoring. This kind of heterogeneity was discussed in [106], where the authors propose a hardware and software testbed for wireless sensor network applications, including sensors with auxiliary systems able to gather additional energy from the ambient.

In this work, we study WSNs where sensors belong to different types, from now on defined as *families*, and propose two variants of MLP defined on such networks. We call such variants the Maximum Lifetime with Multiple Families

Problem (MLMFP) and the Regular Maximum Lifetime with Multiple Families Problem (MLMFP-R).

Indeed, if each target needs to be covered by every family where the WNS is activated, then finding a solution would merely reduce to solving MLP separately for each family, with an objective function value equal to the minimum among such maximum lifetimes. In fact, such separate sets of covers could be activated in parallel, and the monitoring activity would continue until one of the families had no covers available. However, such a hard requirement could be too restrictive for many real-world cases. It could be reasonable for a portion of the targets to be left uncovered by each family in each cover, as long as some minimum family-dependent threshold is met, and coverage of all the targets is provided by at least one of the families at all times.

Consider, for instance, a fire detection scenario which makes use of different types of sensors to monitor heat, humidity and smoke levels. While perfect knowledge using all types of sensors for all target points would be ideal, detections with a high level of accuracy may still be possible if each target is covered by only one or two types of sensors, and the information gathered by sensors monitoring a subset of targets located in the same portion of the area suggest accordingly that a fire event is indeed happening. Some sensor types may be more relevant for the detection of the phenomenon of interest (for example, heat or smoke); therefore, appropriate tradeoffs between network lifetime and detection accuracy may be obtained by choosing a percentage of the targets that should be covered by such families at all times, representing the above mentioned threshold.

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

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

The rest of the work is organized as follows. In Section 5.2 we formally introduce the MLMFP and the MLMFP-R. In Section 5.3 an exact Column Generation approach is presented. In Section 5.4 we present our genetic algorithm and describe its integration within the CG framework. Section 5.5.3 presents the results

of our intensive computational experiments.

## 5.2 Notation and Problems Definition

Consider a wireless network $(S, T, F)$, where $S = \{s_1, \ldots, s_m\}$ is the set of the sensors, $T = \{t_1, \ldots, t_n\}$ is the set of the targets, and $F = \{f_1, \ldots, f_z\}$ is the set of the sensor families. As previously introduced, each sensor is assigned to a family and is able to monitor a subset of targets defined by its sensing range. For each $t_k \in T$ and $s_i \in S$, let $\gamma_{ki}$ be a binary parameter equal to 1 if $t_k$ is covered by $s_i$, 0 otherwise. Furthermore, let $\{S_1, \ldots, S_z\}$ be a partition of $S$, such that $s_i \in S_a$ if the family of sensor $s_i$ is $f_a$, $\forall a \in \{1, \ldots, z\}$. A cover $C_j \subseteq S$ is defined in the classical MLP problem as a subset of sensors such that each target of $T$ is covered by at least one sensor in $C_j$, that is, $\sum_{s_i \in C_j} \gamma_{ki} \geq 1$, $\forall t_k \in T$. For a cover to be feasible, we consider an additional condition which imposes a minimal coverage threshold to be satisfied by each family. More in detail, given the *coverage requirement* $0 \leq \tau_a \leq n$ associated with $f_a$, $C_j$ is feasible if and only if the sensors in $C_j \cap S_a$ cover at least $\tau_a$ different targets. The MLMFP problem consists then of finding a set of feasible covers $C_1, \ldots C_u$ and of assigning a positive activation time $w_1, \ldots, w_u$ with each of them, such that the overall network lifetime is maximized and the battery duration constraint for each sensor are not violated. Let us assume that we can compute in advance the complete set of feasible covers $\mathcal{C} = \{C_1, \ldots, C_\ell\}$. For each $s_i \in S$ and $C_j \in \mathcal{C}$, let $\phi_{ij}$ be a binary parameter equal to 1 if $s_i$ belongs to $C_j$ and 0 otherwise. Let us assume that each battery duration is normalized to 1 time unit. Then, MLMFP can be described by the following Linear Programming formulation:

$$[\mathbf{P}] \max \sum_{C_j \in \mathcal{C}} w_j \tag{5.1}$$

s.t.

$$\sum_{C_j \in \mathcal{C}} \phi_{ij} w_j \leq 1 \qquad \forall s_i \in S \tag{5.2}$$

$$w_j \geq 0 \qquad \forall C_j \in \mathcal{C} \tag{5.3}$$

The objective function (5.1) maximizes the total network lifetime. Constraints (5.2) ensure that, for each sensor, the sum of the activation times of the covers, where it is contained, does not exceed its normalized battery duration. Let us consider a solution of MLMFP composed of a set of feasible covers and the related activation times. Additionally, for each sensor $t_k$ and each family $f_a$, let $w_{ka}$ be the amount of time $t_k$ is covered by sensors belonging to $S_a$ in the solution. We define the solution to be *regular* if $w_{min} = \min\{w_{ka} | t_k \in T, f_a \in F\}$ is maximized. The regular version of the problem (i.e., MLMFP-R) consists of finding a regular solution which maximizes the network lifetime. The motivation for that is to ensure the time $w_{ka}$, when target $t_k$ is covered by sensors of family $S_a$, is balanced among all the targets and all the families. Let us consider the full set of feasible covers $\mathcal{C} = \{C_1, \ldots, C_\ell\}$. For each $t_k \in T$, $f_a \in F$ and $C_j \in \mathcal{C}$, let $\psi_{kaj}$ be a binary parameter equal to 1 if a sensor in $S_a$ belongs to $C_j$ and covers $t_k$, 0 otherwise. The problem is then defined as follows:

$$\textbf{[P2]} \max(W + \varepsilon) w_{min} + \sum_{C_j \in \mathcal{C}} w_j \tag{5.4}$$

s.t.

(5.2), (5.3)

$$(\sum_{C_j \in \mathcal{C}} \psi_{kaj} w_j) - w_{min} \geq 0 \qquad \forall t_k \in T, \forall f_a \in F \tag{5.5}$$

$$w_{min} \geq 0 \tag{5.6}$$

Constraints (5.5) ensure, for each $t_k \in T$ and $f_a \in F$, the quantity $w_{min}$ to be not greater than $w_{ka}$ (that is, the sum of the activation times $w_j$ for each $C_j \in \mathcal{C}$ such that $\psi_{kaj} = 1$). In the objective function (5.4) the $W$ parameter represents an upper bound on the maximum lifetime $\sum_{C_j \in \mathcal{C}} w_j$ and $\varepsilon$ is a small positive coefficient, such that the weighting ensures that a regular solution is sought as primary objective. It should be noted that while MLMFP and MLMFP-R have different objective functions and the latter introduces additional constraints, each individual cover which may be part of a solution has to satisfy the same conditions

in order to be feasible, and therefore the set $\mathcal{C}$ is the same for both problem variants. The provided formulations cannot be used to solve real world instances of MLMFP or MLMFP-R, since the cardinality of the set of feasible covers $\mathcal{C}$ is potentially exponential. For this reason, we developed Column Generation algorithms to solve both the problems, as described in Section 5.3. Section 5.2.1 to follow discusses how to adapt model formulations [P] and [P2] when hardware differences among the sensors are taken into account. Section 5.2.2 discusses the issue of redundant covers in the feasible region of the two problems.

### 5.2.1 Modeling Hardware Differences

The above presented models represent the problems as long as different sensor families can be assumed to have the same battery durations. Indeed, since they may be based on widely different hardware, this may not be the case. However, the model can be easily adapted to take this factor into account.

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

Furthermore, sensors may individually have an initial charge level which is different from the maximum for their family (for example, if the sensor was previously employed for different activities). For a given sensor $s$, let $0 < charge_s \leq 1$ be its initial charge percentage. Again, let $s \in S_a$ with $\Delta_a = 2$, and let $charge_s = 0.5$. Then, sensor $s$ can be used for $charge_s/\Delta_a = 0.25$ units of time. For both problems, constraints (5.2) can then be modeled in the following more general form:

$$\sum_{C_j \in \mathcal{C}} \phi_{ij} w_j \leq charge_{s_i}/\Delta_a \qquad \forall f_a \in F, s_i \in S_a \tag{5.7}$$

### 5.2.2 MLMFP, MLMFP-R and cover redundancy

Given a feasible cover $C_1$, we define it to be *redundant* if it contains another feasible cover $C_2$ as a proper subset. It is straightforward to observe that if an optimal solution for MLMFP contains $C_1$, then an alternative one where $C_2$ replaces $C_1$ can be found. Therefore, when looking for optimal solutions for MLMFP, in the methods described in Sections 5.3 and 5.4 we focus on individuating non-redundant covers, in order to reduce the search space and speed-up the convergence of our Column Generation algorithm. Conversely, it can be shown that an optimal solution for MLMFP-R may involve redundant coverage. To illustrate this, consider a simple network with $T = \{t_1, t_2\}$, $S = \{s_1, s_2, s_3\}$, $F = \{f_1, f_2\}$, $S_1 = \{s_1, s_2\}$, $S_2 = \{s_3\}$, $\tau_1 = \tau_2 = \Delta_1 = \Delta_2 = charge_{s_1} = charge_{s_2} = charge_{s_3} = 1$. Furthermore, let $s_1$ and $s_2$ cover $t_1$ and $t_2$, respectively, while $s_3$ covers both of them. This network is shown in Figure 5.1C, where sensors belonging to $S_1$ and $S_2$ are represented by dotted and dashed lines, respectively. The only two feasible non-redundant covers in this network are $C_1 = \{s_1, s_3\}$ and $C_2 = \{s_2, s_3\}$, shown in Figures 5.1A-5.1B. Indeed, due to $\tau_2$ being nonzero, $s_3$ needs to be in each feasible cover, and since it already covers both targets either $s_1$ or $s_2$ can be used to also satisfy the $\tau_1$ requirement. Using this set of covers, the maximum achievable $w_{min}$ value is 0.5, obtained when both $C_1$ and $C_2$ are activated for such amount of time. This is easy to verify, since the sum of the activation times of the two covers cannot be higher than the lifetime of $s_3$ which is 1, and any other feasible activation time choice (e.g., 0.6 for $C_1$ and 0.4 for $C_2$) would bring a reduction to the amount of time for which either $t_1$ or $t_2$ are covered by sensors belonging to family $f_1$. Conversely, by activating the redundant cover $C_3 = \{s_1, s_2, s_3\}$ for a full time unit, both $w_{min}$ and the network lifetime are equal to 1.

## 5.3 Column Generation Approach

Delayed Column Generation (CG) is a widely used linear programming approach for LP problems with a large number of variables. The approach initially considers the original LP formulation (in our case formulations [**P**] and [**P2**]), called the

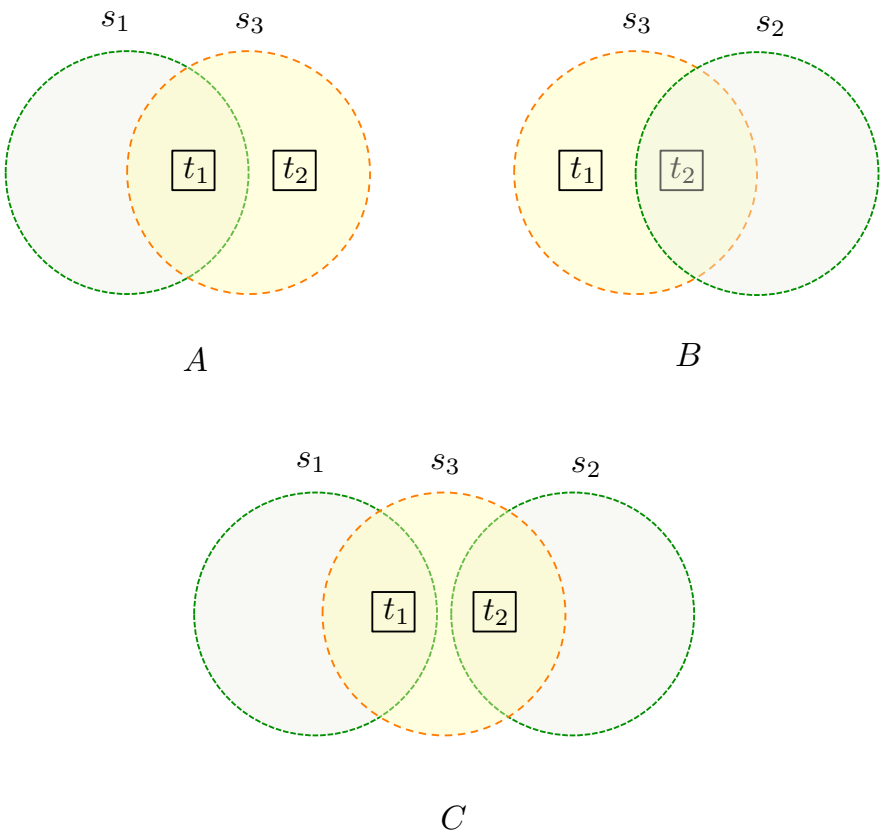



Figure 5.1: *Sample network. A-B: Feasible non-redundant covers* $C_1, C_2$. *C: Complete network and feasible redundant cover* $C_3$.

*Master Problem*, restricted to a subset of variables, and optimally solves it. CG then considers a specific optimization problem (called the *Separation Problem*) which either identifies a new attractive variable to be entered in the problem or certifies the optimality of the last solution found. If a new variable is identified, it is included in the Master Problem and the procedure iterates until optimality test is satisfied. The solution of the Separation Problem therefore avoids the enumerative assessment of all the (potentially exponential) variables that will be nonbasic in the final solution. Consider the MLMFP problem first and let us call **[SP]** its Separation Problem. Given the last iteration of the master problem, let $\pi_i$ be the dual prices associated with its constraints, that is, with each sensor. The current solution is optimal if and only if the reduced costs associated with

all nonbasic variables are non negative, i.e. $\sum_{i:s_i \in C_j} \pi_i - c_j \geq 0$ for each nonbasic $C_j$. In our case $c_j$ are the coefficients of the objective function (5.1) and are all equal to 1; therefore, the optimality condition reduces to $\sum_{i:\phi_{ij}=1} \pi_i \geq 1$ for each nonbasic $C_j$. The **[SP]** objective function minimizes the sum of the dual prices of the sensors selected to be part of the new cover, and the optimality test is satisfied if the optimum value of **[SP]** is greater than or equal to 1. Constraints in **[SP]** define the construction of a feasible cover.

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

$$[\mathbf{SP}] \min \sum_{s_i \in S} \pi_i x_i \tag{5.8}$$

s.t.

$$\sum_{s_i \in S_a} \gamma_{ki} x_i \geq y_{ka} \qquad \forall f_a \in F, t_k \in T \tag{5.9}$$

$$y_{ka} \geq \gamma_{ki} x_i \qquad \forall f_a \in F, s_i \in S_a, t_k \in T \tag{5.10}$$

$$\sum_{t_k \in T} y_{ka} \geq \tau_a \qquad \forall f_a \in F \tag{5.11}$$

$$\sum_{f_a \in F} y_{ka} \geq 1 \qquad \forall t_k \in T \tag{5.12}$$

$$x_i \in \{0, 1\} \qquad \forall s_i \in S \tag{5.13}$$

$$y_{ka} \in \{0, 1\} \qquad \forall f_a \in F, t_k \in T \tag{5.14}$$

The objective function (5.8) makes sure that the reduced cost of the newly generated column is minimized. Constraints (5.9)-(5.10) bind the two sets of variables, by letting $y_{ka}$ be equal to 1 if and only if at least one sensor $s_i$ that belongs to $f_a$ and covers $t_k$ is selected.

Constraints (5.11) ensure that the coverage requirement for each family is respected. Finally, Constraints (5.12) impose that all the targets are covered by

at least one family (and therefore by at least one sensor).

Note that new redundant columns may be introduced when dual prices are equal to zero. This may occur in particular in the first iterations of the CG procedure. To avoid this, we modify objective function (5.8) by adding a small positive coefficient $\varepsilon$ to each dual price, as follows:

$$\min \sum_{s_i \in S} (\pi_i + \varepsilon) x_i \tag{5.15}$$

By assigning a positive weight to each $x_i$ variable, the new objective function (5.15) ensures that each sensor added to the new column is needed. Note that the value of the original objective function (5.8) still has to be evaluated after each [**SP**] iteration in order to determine whether the optimality test is satisfied.

We now define the [**SP2**] subproblem for the MLMFP-R problem. Let $\pi_i$ and $q_{ka}$ be the dual prices related to Constraints (5.2) (or their generalized form (5.7)) and (5.5) for the last iteration of the [**P2**] master problem; the current solution is optimal if $\sum_{i:\phi ij=1} \pi_i + \sum_{k,a:\psi_{kaj}=1} q_{ka} \geq 1$ for each nonbasic $C_j$. Therefore, [**SP2**] can be expressed as follows:

$$[\mathbf{SP2}] \min \sum_{s_i \in S} \pi_i x_i + \sum_{t_k \in T} \sum_{f_a \in F} q_{ka} y_{ka} \tag{5.16}$$

s.t.

(5.9)-(5.14)

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

$$\sum_{s_i \in S_a} x_i \leq T \qquad \forall f_a \in F \tag{5.17}$$

The main drawback of the CG approach presented above is that the subproblems are NP-Hard combinatorial optimization problems, being specializations of the set covering problem. For this reason, in the next section we introduce a

genetic algorithm able to quickly compute good feasible solutions for the subproblems. We embedded this genetic algorithm in our CG approach to improve its performance.

## 5.4 Genetic Algorithm

As discussed in Section 5.3, the subproblems are NP-Hard and therefore it is preferable to solve them heuristically, especially for instances of considerable size. We addressed this problem by developing a genetic algorithm (GA) able to return new *attractive covers*, i.e. covers with an objective value lower than 1. The procedure generates feasible solutions for both the problems and evaluates the associated objective function value according to (5.8) for MFMLP and (5.16) for MFMLP-R. Furthermore, the GA for MFMLP always removes redundancy, while redundant covers may be generated by the GA for MLMFP-R, due to the motivations provided in Section 5.2.2.

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

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

The GA is a well-known and widely used meta-heuristic technique for optimization problems. Similarly to other evolutionary techniques, the GA emulates biological evolution and natural selection. As in nature, GA considers the evolution process based on *chromosomes*, elements that represent the structure of an individual for the real world and a solution (e.g. a feasible cover) for the optimization problem. The natural selection is the process through which the GA guarantees that new solutions are typically, step by step, better adapted to the environment. The environment is encoded by the *fitness function* that is used to rank each solution. The evolutionary step is achieved through two mech-

anisms, named *crossover* and *mutation*. The crossover operator combines, in a probabilistic manner, the genetic material of typically two or more selected individuals (*parent* solutions). The mutation operator, instead, randomly modifies the value of one or more genes of a child chromosome derived from the crossover phase in order to increase diversity. The overall process is repeated until a stop condition is reached. Such a condition can be a maximum number of generations, a specified amount of time, a lack of improvements in the fitness function of the best individual, or other conditions related to the specific optimization problem. For a complete and detailed description of the genetic algorithms and their characteristics the reader can refer to [38].

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

### 5.4.1 Chromosome representation and fitness function

The chromosome representation is based on the binary encoding represented in Figure 5.2. It stores the set of sensors activated in a given candidate cover, as well as the related covered targets for each sensor family.

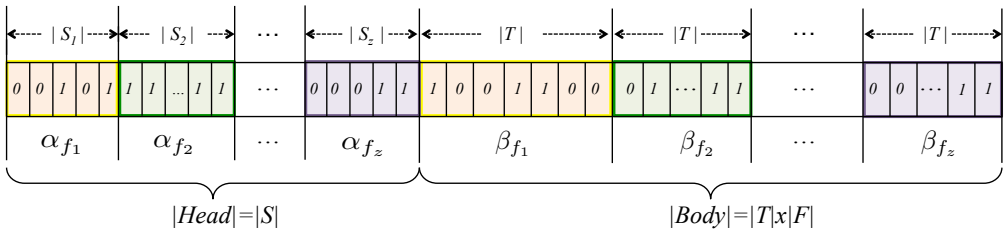



Figure 5.2: *Chromosome Structure*

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

The *Head* component is a binary vector of length $|S|$. Each position is related to a sensor, and is equal to 1 if it belongs to the cover, 0 otherwise. Moreover, the sensors are sorted by family so that the first $|S_1|$ positions (defined section $\alpha_{f_1}$) of the Head contain the binary values related to the sensors of family $f_1$, the subsequent $|S_2|$ positions ($\alpha_{f_2}$) refer to the sensors of family $f_2$, and so on for all families. For instance, the $\alpha_{f_1}$ segment in Figure 5.2 consists of 5 positions

(meaning that $|S_1| = 5$ in the network), and two of them (the third and fifth) are currently activated in the chromosome.

The *Body* component represents which targets are covered by each family. This component is partitioned in $|F|$ segments of size $|T|$, sorted by family. More in detail, the $i$-th position in $\alpha$-th body segment, defined $\beta_{f_a}$, is equal to 1 if there is at least one sensor of $f_a$ that covers target $t_i$ and that is currently activated in $\alpha_{f_a}$, 0 otherwise. For instance, in Figure 5.2 the segment $\beta_{f_1}$ consists of the first $|T| = 7$ positions of the Body component, and it shows that the three sensors activated in $\alpha_{f_1}$ cover the targets $t_1$, $t_4$ and $t_5$.

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

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

$$\sum_{t_k \in T} \beta^C_{ka} \geq \tau_a \qquad \forall f_a \in F \tag{5.18}$$

$$\sum_{f_a \in F} \beta^C_{ka} \geq 1 \qquad \forall t_k \in T \tag{5.19}$$

Condition (5.18) ensures that each family meets its coverage requirement, while condition (5.19) states that each target must be covered at least once. For instance, in the example in Figure 5.2, condition (5.18) is respected for family $f_1$ if $\tau_1 \leq 3$.

Our GA considers in each iteration a population consisting of only feasible chromosomes. In the case of MFMLP, chromosomes will always be non-redundant as well. The chromosomes are evaluated according to the objective function of **[SP]** for MFMLP and of **[SP2]** for MFMLP-R. That is, in the case of MFMLP, given the vector of dual prices provided by the last Master Problem iteration and sorted by family, the fitness function of a given chromosome is equal to the dot product of its *Head* component and the dual prices vector. For each cover that is found to be attractive at the end of the GA procedure, the *Head* component

**Algorithm 5:** Genetic Algorithm for MLMFP or MLMFP-R

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

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

corresponds to the new column to be included in the restricted columns set of master problem [**P**]. In the case of MFMLP-R, both the *Head* and the *Body* components are used to evaluate the fitness function of a given chromosome, and both components represent the column to be added to [**P2**].

### 5.4.2 GA overall structure

In this section we describe the general structure of the GA, whose pseudocode is given in Algorithm 5.

The procedure takes as input the wireless sensor network $(S, T, F)$ and the vector $DP$ of the dual prices provided by the Master Problem. The first step is the generation of an initial population $Pop$ and the identification of the chromosome

with the best fitness ($BestFit$). This chromosome is the *incumbent* solution and it will be used for comparisons during the evolution process. The population consists of a predefined number ($Size_{Pop}$) of feasible covers and it is initialized by the procedure described in Section 5.4.7.

The while loop (line 4) iterates until either $MaxIT$ consecutive iterations, without improvements in the incumbent solution fitness $BestFit$, are carried out or $MaxDB$ consecutive duplicates chromosome are generated. A chromosome is a *duplicate* if it is already present in the population. Forbidding the presence of duplicates in the population makes it possible to avoid looping over a solution space that has almost been exhausted.

Each iteration includes a tournament for the selection of two parent chromosomes (see Section 5.4.3), a crossover function (Section 5.4.4) and a mutation function (Section 5.4.5). Furthermore, two operators, called *fixing* and *redundancy*, are applied. The first one is used to check and eventually restore feasibility for the newly generated chromosome. The redundancy operator always removes eventual redundancy for the MFMLP while, for the MFMLP-R, it may return a redundant cover if it is considered useful to improve the objective function. The two operators are described in Section 5.4.6.

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

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

### 5.4.3 Tournament selection

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

chosen parent.

### 5.4.4 Crossover

The crossover function represents the process of coupling between two selected parents. Recall from the chromosome description in Section 5.4.1 that each family-related segment $\alpha_{f_a}$ in the *Head* component is strongly linked to a specific segment $\beta_{f_a}$ in the *Body* section, since the former represents the selected sensors for a given family, and the latter the related covered targets. By definition, a feasible chromosome ensures that each couple $(\alpha_{f_a}, \beta_{f_a})$ satisfies the related constraint (5.18). Therefore, in our genetic algorithm we consider such couples to be genes of the chromosome, which will be used as building blocks for the child chromosome during the crossover. A graphical representation of the gene structure is given in Figure 5.3.

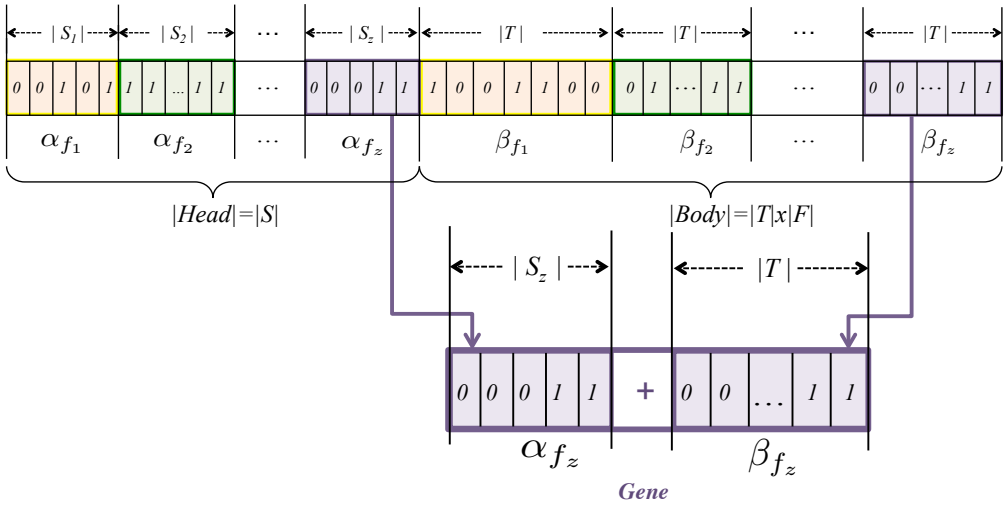



Figure 5.3: *Gene structure*

Given the chromosome structure and the gene definition, the crossover function randomly selects each gene one at a time between the two input parents.

In particular, let $C_1$ and $C_2$ be the two parents, and $Child$ the child chromosome to be built. For each family $f_a \in F$, the gene $(\alpha_{f_a}^{Child}, \beta_{f_a}^{Child})$ will be equal to $(\alpha_{f_a}^{C_1}, \beta_{f_a}^{C_1})$ with probability 0.5, otherwise it will be equal to $(\alpha_{f_a}^{C_2}, \beta_{f_a}^{C_2})$. The crossover is illustrated in Figure 5.4. It is straightforward to observe that this construction ensures that the coverage requirements (5.18) are satisfied for each family, since by definition both parents are feasible, and therefore each of their genes satisfies the requirement as well.

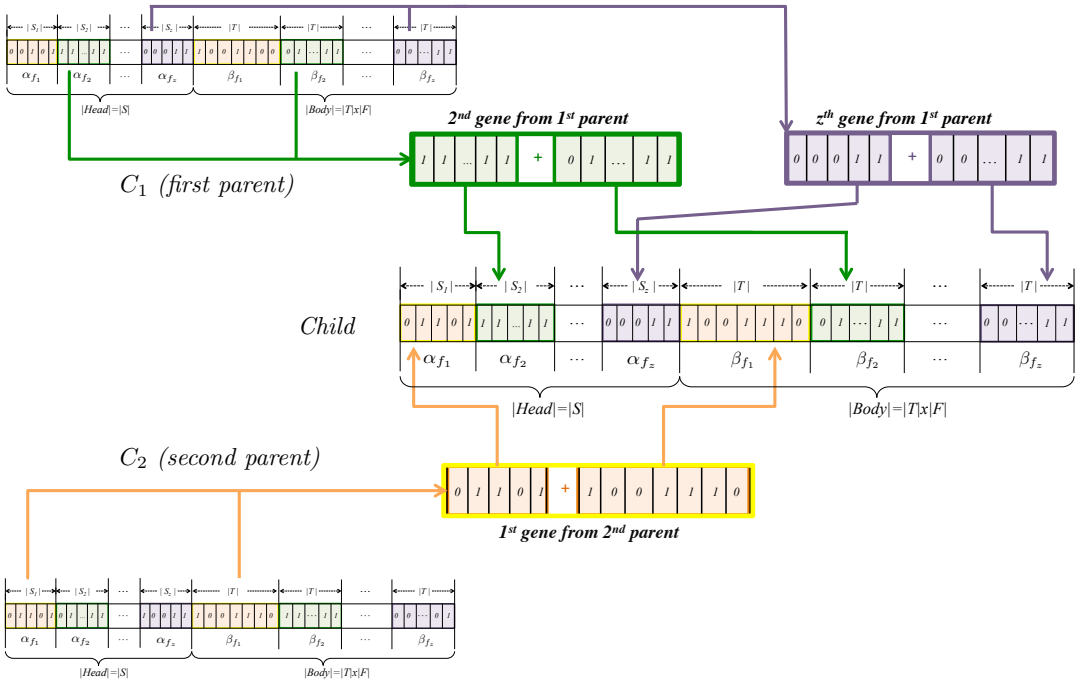



Figure 5.4: *Crossover*

### 5.4.5 Mutation

The mutation randomly alters the child chromosome produced by the crossover function in order to create diversity during the exploration of the solution space. The mutation operates in two steps. In the first step, it randomly selects a single family $f_a$, $1 \leq a \leq z$. Subsequently, it randomly selects a sensor $s_i \in S_a$, and switches its position in the $\alpha_{f_a}$ component either from 1 to 0, or from 0 to 1. The

mutation also involves a change in the $\beta_{f_a}$ segment if deactivating or activating the selected sensor leads to a different target coverage for family $f_a$.

### 5.4.6 Fixing and redundancy operators

As noted in Section 5.4.4, at the end of the crossover phase, the coverage requirement is respected for all families (that is, condition (5.18)). However, due to the perturbation brought by the mutation, this may no longer be the case for one of them. Moreover, there is no guarantee that condition (5.19) is respected; that is, there could be targets that are not covered by any sensor. Therefore, after crossover and mutation, the fixing operator (whose pseudocode is given in Algorithm 6) is applied on each generated chromosome to ensure feasibility. The operator works in two phases, one for each of the two conditions.

If the first condition is not met for some family $f_a$ (Algorithm 6, lines 1-6), let $\hat{S}_a \subseteq S_a$ be the set of sensors of family $f_a$ that are currently not activated in $Child$. Futhermore, let $\hat{T}_a$ be the targets which are currently not covered by the family in the chromosome. The procedure iteratively selects sensors in $\hat{S}_a$ which cover some elements of $\hat{T}_a$, until $\tau_a$ targets are covered by the family. The gene $(\alpha_{f_a}^{Child}, \beta_{f_a}^{Child})$ is updated accordingly at each step.

Regarding the second feasibility condition (Algorithm 6, lines 7-12), the algorithm puts all the globally uncovered targets in $\hat{T}$, if they exist. Then, the procedure iteratively selects a target $t \in \hat{T}$ and a sensor $s \in S$ that can cover $t$. The gene of $Child$ related to the family of $s$ is updated to include the new sensor, and $t$ is removed from $\hat{T}$, along with any previously uncovered target which is covered by $s$. The procedure iterates until $\hat{T}$ is empty.

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

Also the redundancy$_2$ operator builds the $S_{red}$ list of redundant sensors. Then, the procedure checks whether removing a random sensor $s_{r1} \in S_{red}$ from $Child$

**Algorithm 6:** Fixing Operator

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

```
for a ← 1 to z do
    while Σ_{t_k∈T} β_ka^Child < τ_a do
        Ŝ_a ← inactiveSensors(Child, f_a);
        T̂_a ← uncoveredTargets(Child, f_a);
        s ← randomSelect(Ŝ_a, T̂_a);
        update(Child, s);
T̂ ← uncoveredTargets(Child);
while |T̂| > 0 do
    t ← randomSelect(T̂);
    s ← randomSelect(S, t);
    update(T̂);
    update(Child, s);
return Child;
```

would lead to a worse fitness function. If that is the case, a second element $s_{r2} \in S_{red} \setminus \{s_{r1}\}$ is randomly selected and checked for removal. Iteratively, the elements in $S_{red}$ are visited according to a random order; as soon as one can be removed is found $Child$ is updated and $S_{red}$ is recomputed. The procedure ends when either $S_{red}$ is empty or all its elements have been visited. The redundancy$_2$ operator results to be more computationally intensive than redundancy$_1$.

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

### 5.4.7 Building the initial population

The procedure for initializing GA builds the initial population $Pop$, composed of $Size_{Pop}$ random feasible individuals. The population is built iteratively. For each individual, the procedure applies the fixing and redundancy operators, as discussed in Section 5.4.6, with the only difference that they start from an empty chromosome. If a chromosome is equal to a previously generated one, it is dis-

carded and generated again. If the procedure fails to build a new chromosome for $MaxInitDB$ consecutive iterations, it is interrupted, and $Size_{Pop}$ is set to the current value of $|Pop|$.

## 5.5 Computational Results

This section presents the test scenarios and the results obtained by performing our extensive computational phase. The algorithms were coded in C++ and the tests were performed on a computer with an Intel Xeon 2 GHz processor and 8GB of RAM, equipped with the IBM ILOG CPLEX 12.5.1 solver and the Concert Technology Library for the mathematical formulations. Section 5.5.1 described our instances and the test scenarios being considered. The values used for the GA parameters as well as a description of the CG initialization are given in Section 5.5.2. Finally Section 5.5.3 presents our computational results organized in tables, along with several comments on them.

### 5.5.1 Description of instances and test scenarios

The instances were generated by randomly placing targets and sensors on an area of size $500 \times 500$. We assumed the sensing range of each sensor to be equal to 150. We considered instances containing a number of target points $|T|$= 30, 60, 90 or 120, and whose sensors are divided in $|F| = 2$, 4 or 6 sensor families.

For each value of $|F|$, we considered 6 different values for the overall number of sensors $|S|$, corresponding to the cases in which each family has on average 50, 100, 150, 200, 300 or 400 sensors, leading to the values reported in Table 5.1. However, to better model the heterogeneity which may characterize real-world scenarios, sensors were not evenly distributed among the different families, but rather randomly assigned to them, leading to families with different numbers of sensors. However, each family is always guaranteed to cover each target with at least one sensor in order to ensure feasibility for each possible coverage request value, as well as strictly positive $w_{min}$ optimal solution value, for the whole set of instances.

For each combination of the above mentioned parameters, we generated 5

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

Table 5.1: Settings of the $|S|$ parameter

different instances. The total number of test instances is therefore equal to 360.

Furthermore, for each instance, two different scenarios were considered, related to the possible values of the coverage request parameters. In the *uniform* coverage request scenario, each family $f_a$ is required to provide coverage for $\tau_a = \lfloor |T|/|F| \rfloor$ target in each feasible cover. In the *variable* coverage request scenario, we assigned either 2 or 3 different coverage request values to the families, with the lowest one being set to 0. In particular, when $|F| = 2$, one of the families has a coverage request equal to $\lfloor 3/4|T| \rfloor$, while the coverage request is equal to 0 for the other family. For $|F| = 4$, the coverage request is set to $\lfloor 3/8|T| \rfloor$ for a family, $\lfloor 3/16|T| \rfloor$ for 2 of them and 0 for the remaing one. Finally, for instances with 6 families, the three coverage request values are $\lfloor 3/12|T| \rfloor$, $\lfloor 3/24|T| \rfloor$ and 0, and are assigned to 2 families each. Furthermore, for both scenarios we considered the case in which the consumption ratio of the family with index $i \in \{1, \ldots |F|\}$ is equal to $(1.0) + (0.1)(i-1)$. All sensors are always assumed to have fully charged batteries at the beginning of the monitoring phase (that is, $charge_{s_i} = 1 \; \forall s_i \in S$). The coverage request and the consumption ratio values being considered for the two scenarios are summarized in Table 5.2.

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

As discussed in Section 5.5.3, we also ran some tests for a "pure" CG approach which does not embed the GA after the CG initialization, and therefore relies on the **[SP]** formulation to generate new covers. We performed this comparison on a subset of the generated instances for the MFMLP problem, as explained in

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

Table 5.2: Coverage request and consumption ratio values

Section 5.5.3.

### 5.5.2 Parameter setting and CG initialization

Parameter values were chosen after a preliminary tuning phase. The population size $Size_{Pop}$ was chosen to be equal to $50 + \lceil \sqrt{|S|} \rceil$. The two termination criteria, namely the maximum number of iterations without improvements $MaxIT$ and the maximum number of consecutive duplicates $MaxDB$ were chosen to be equal to 1500 and 100, respectively. During the initialization phase, the limit on the number of consecutive duplicates $MaxInitDB$ was set to 100 as well. Finally, the value 0.9 was chosen for the fitness threshold value $fitThreshold$ when solving MFMLP, and 0.5 when solving MFMLP-R.

As introduced in Section 5.3, in order to initialize the CG algorithm, a subset of feasible covers has to be provided for the first iteration of the master problem. We generated these covers using a first run of the GA. As a heuristic criterion, during this GA execution each sensor is given an equal, strictly positive weight, meaning that when fitness function is evaluated covers with fewer sensors are favored. Furthermore, for this iteration the $fitThreshold$ value is unbounded, meaning that the whole set of $Size_{Pop}$ covers is returned and added to the master

problem.

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

### 5.5.3 Test and Results

We now analyze the impact of embedding the proposed GA within the CG scheme for the easier [**P**] formulation to solve. We first compare our proposed algorithm (referred to as CG+GA) with a pure CG approach (referred to as CGonly) which only uses the genetic algorithm for its initialization as reported in Section 5.5.2, and which generates each subsequent attractive cover by solving the [**SP**] formulation. The two approaches are compared based on the subset of 60 instances corresponding to the lowest values of the $|S|$ parameter, that is $|S| = 100$ for $|F| = 2$, $|S| = 200$ for $|F| = 4$ and $|S| = 300$ for $|F| = 6$. For each of those instances, computational tests were performed for both the coverage request scenarios. The comparison is given in Table 5.3 for the basic (i.e. non regular) version of the problem, similar conclusions can be derived for the regular version of the problem whose results are reported in Table B.1 in the Appendix B. As shown in the tables, the performances of the pure approach tend to degrade quickly as the size of the instances grows, therefore, it cannot be expected to find solutions in reasonable time on the largest ones.

Each entry reported in the table shows average values and standard deviations for the 5 tests corresponding to the related choices of $|T|$, $|F|$, $|S|$ and coverage scenario type. Columns *avg.* and *std. dev.* are average and standard deviation values computed among the five different instances generated for each scenario, respectively. The *solution* column contains the average solution value computed among the five different instances of the scenario. Heading *CGonly* stands for the pure CG described above, while *CG+GA* is related to the algorithm that takes full advantage of the genetic algorithm. For both procedures, column *SP it.* reports the number of times the subproblem [**SP**] was solved to optimality by means of the solver CPLEX, and *time* the requested computational time in

seconds. For CG+GA, *GA it.* contains the number of times in which the GA is invoked and solved. Finally, column *speed-up* measures the speed-up factor between the computational time of CGonly and that of CG+GA.

Looking at the table, it can be noticed that the benefit provided by the GA is remarkable even for the for the basic problem on these smaller instances, and the performances of the two procedures diverge as the number of sensors increases.

*The CG+GA approach consistently outperforms the pure CG approach, and the computational times difference between the two procedures increases with the number of sensors.* The minimum average speed-up (column speed-up in Table 5.3) is equal to 7.35 for $|S| = 100$, 31.75 for $|S| = 200$, and 46.43 for $|S| = 300$. Overall, the CG+GA is up to 112.85 times faster than CGonly and required a maximum computational time of 6.83 seconds on average, on a set of 5 instances which required on average 683.68 seconds when solved by CGonly (which is the maximum value for this procedure as well). CG+GA shows a consistent and robust behavior on all the instances with a coefficient of variation (i.e., the ratio between standard deviation and average value) for the speed-up always less than 50% for the uniform coverage request scenarios and less than 42% for the variable requests scenarios (except for two instances for $|S| = 100$ for both the cases).

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

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

*The performances of CG+GA scale well when bigger instances are considered*

Table 5.3: **Comparison of our approach (CG+GA) and a pure column generation approach (CGonly) when solving MFMLP.**

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

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

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

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

*and, overall, all tests could be executed within reasonable computational times for both problems versions.* In particular, the computational time increases with the size of the instance, as expected, for both the problems and for both the two different coverage requests scenarios.

This trend is evident from Figure 5.5 and column *time* of Table 5.4, where the computational times of our algorithm when solving the basic version of the problem and the regular version of the problem, for the uniform coverage request scenario and the variable coverage request scenario, with $|F| = 4$, are shown. The same figures and tables for $|F| = 2$ (Figure B.1 and Table B.2) and $|F| = 6$ (Figure B.2 and Table B.3) are given in Appendix B. The average computational time when solving the basic version of the problem with uniform coverage request is equal to 5.75 seconds (varying between 0.35 and 29.83) for $|F| = 2$, 27.95 seconds (varying between 0.92 and 120.88) $|F| = 4$, and 155.06 seconds (varying between 2.02 and 1040.16) for $|F| = 6$. The average computational time when solving the basic version of the problem with variable coverage request is equal to 8.10 seconds (varying between 0.36 and 29.65) for $|F| = 2$, 23.36 seconds (varying between 0.95 and 99.79) for $|F| = 4$, and 94.12 seconds (varying between 1.49 and 510.03) for $|F| = 6$. The average computational time when solving the regular version of the problem with uniform coverage request is equal to 32.65 seconds (varying between 0.51 and 344.57) for $|F| = 2$, 321.01 seconds (varying between 1.62 and 1590.86) for $|F| = 4$, and 1100.89 seconds (varying between 4.01 and 5219.41) for for $|F| = 6$, while it is equal to 37.46 seconds (varying between 0.42 and 142.02) for $|F| = 2$ , 262.93 seconds (varying between 1.95 and 1381.18 ) $|F| = 4$, and 836.97 seconds (varying between 2.18 and 3563.90) for $|F| = 6$, on scenarios with variable coverage requirement.

*The requested average time to solve the regular version of the problem is generally higher than the average time required to solve the basic version of the problem.* Indeed, the GA requires more iterations when solving the MLMFP-R as can be observed in our results (column *GA it.* in Table 5.4 for $|F| = 4$, and Table B.2 for $|F| = 2$ and Table B.3 for $|F| = 6$ in the Appendix B) where the number of GA iterations is higher when solving the regular version of the problem and the total number of returned columns is much greater. Solving the regular version of

Table 5.4: **Results of CG+GA for $|F| = 4$ scenarios when solving the basic version of the problem (MLMFP) and the regular version (MLMFP-R).**

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

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

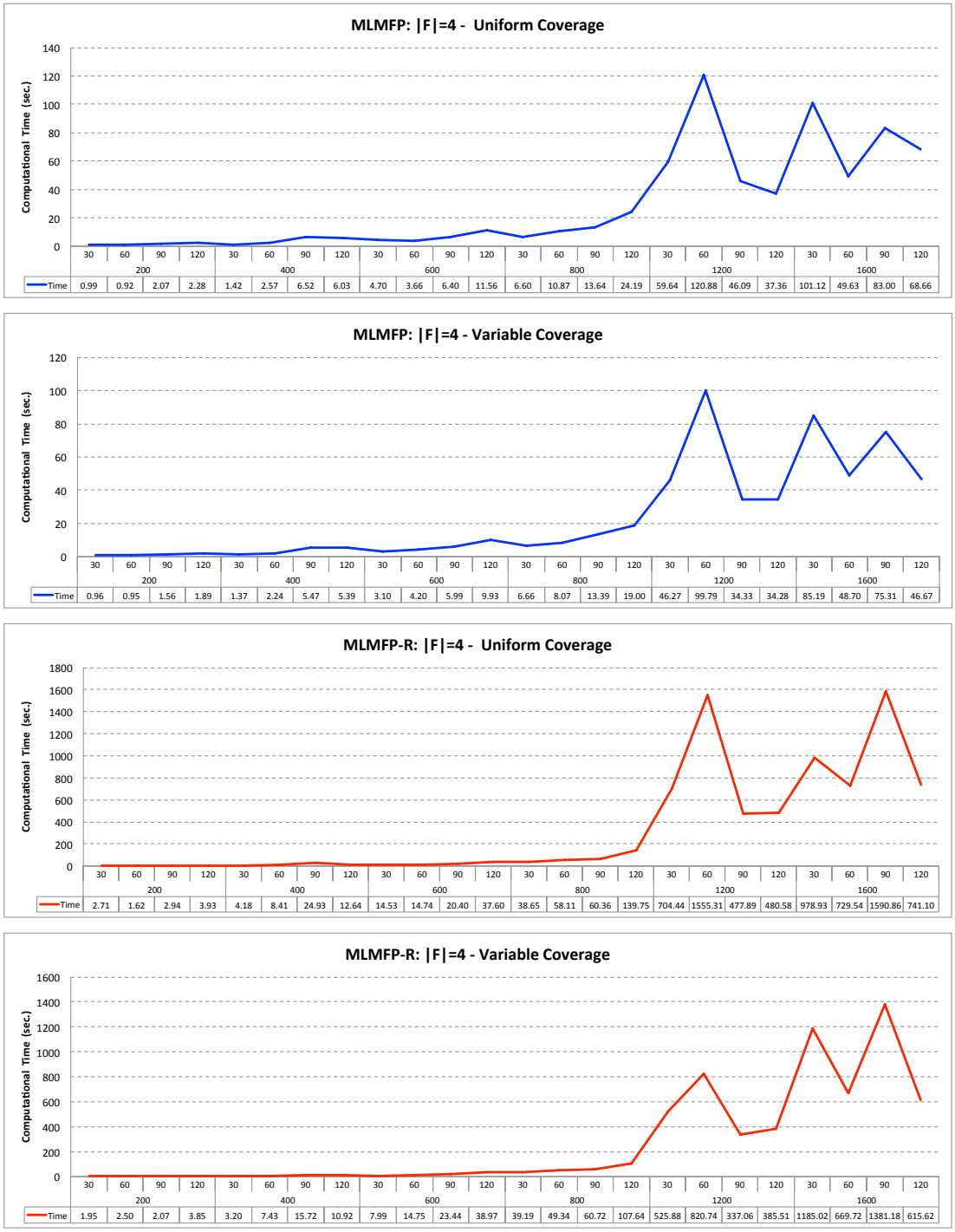



Figure 5.5: *Computational time of CG+GA when solving the basic version of the problem (on the top half) and the regular version of the problem (on the bottom half), for the uniform coverage request scenario and the variable coverage request scenario, with* $|F| = 4$.

the problem, with uniform coverage requests, requires on average 124.86% more GA iterations with respect to the solution of the basic version of the problem for $|F| = 2$, 141.23% more iterations for $|F| = 4$, and 98.09% more iterations for $|F| = 6$. Solving the regular version of the problem, with variable coverage requests, requires on average 104.16% more iterations with respect to the solution of the basic version of the problem for $|F| = 2$, 154.50% more iterations for $|F| = 4$, and 117.80% more iterations for $|F| = 6$. We believe that this could be due to the different objective function of the regular problem which forces the GA to explore more deeply the solution space to find the right combination of covers to satisfy the regularity condition. The number of subproblem iterations (column *SP it.* in the tables) keeps being low for both problem variants, witnessing the effectiveness of the GA algorithm. More in particular, for MLMFP and uniform coverage requests, the subproblem is solved on average 1.43, 1.02 and 1.42 times for $|F = 2|$, $|F = 4|$ and $|F = 6|$, respectively, while for variable coverage requests it is solved on average 2.39, 1.07 and 1.1 times. For MLMFP-R, the correspondent numbers of subproblem invocations are 6.78, 1.21 and 2.84 for uniform coverage requests and 6.38, 1.25 and 1.72 for variable coverage requests.

When comparing the quality of the solutions returned by the two problems we can observe that *the maximum lifetime is the same on all the instances both for the original version of the problem and for the regular version.* This is a counterintuitive result since, by enforcing the individuation of a regular solution, one would expect a deterioration in terms of lifetime for MLMFP-R with respect to MLMFP. Furthermore, as will be discussed later, MLMFP-R is indeed able to improve significantly the value of $w_{min}$ in the returned solution, in particular for the variable scenario. Therefore, we believe that this result is due to the existence of several alternative solutions corresponding to the same optimal lifetime value in the feasible region on each instance. Hence, a regular solution could always be found for the considered set of instances without compromising the maximum lifetime value which can be obtained when regularity is not enforced.

*The maximum lifetime increases, as expected, with the size of the instance for both the problem variants,* as we can observe in Figure 5.6 for $|F| = 4$ (Figure

B.3, B.4 in Appendix B for $|F| = 2$ and $|F| = 6$, respectively). This was an easily expected result since a larger number of sensors allow more covers to exist.

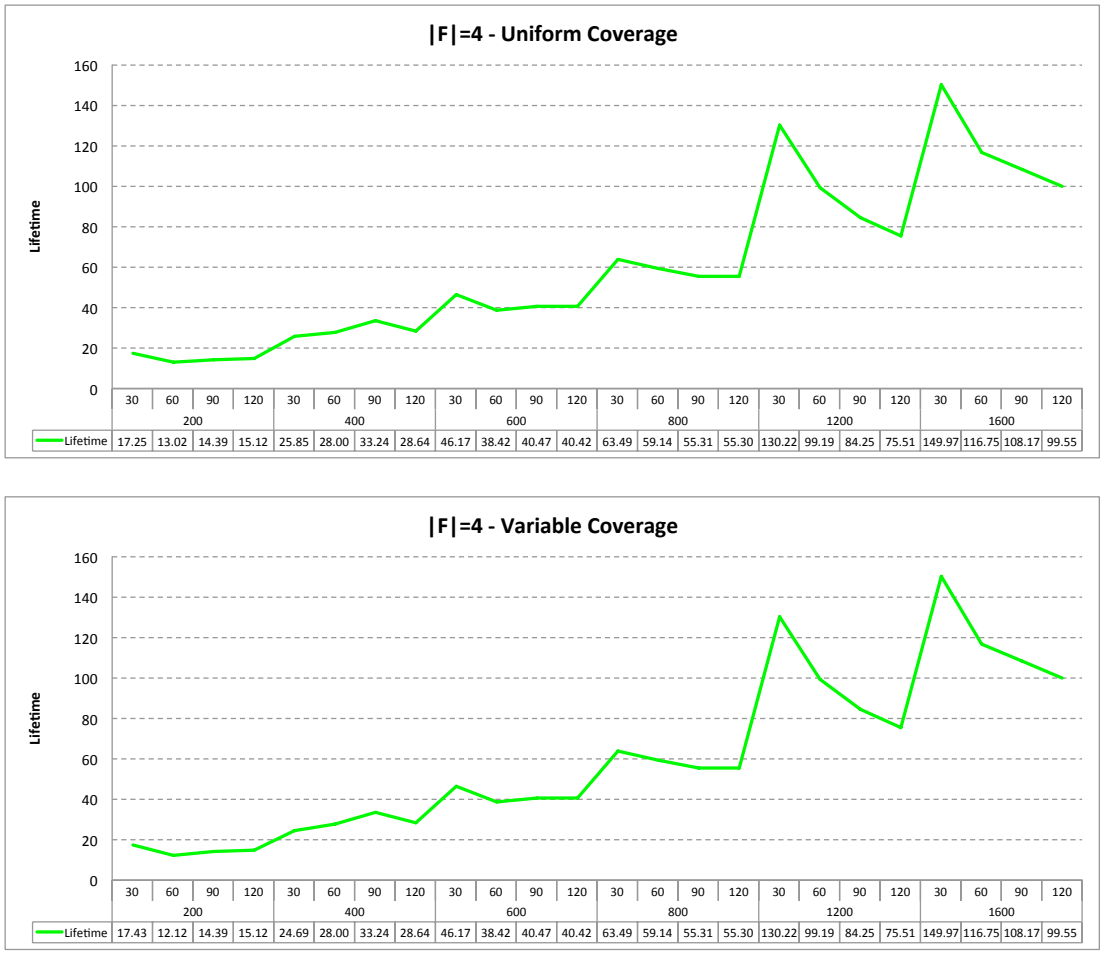



Figure 5.6: *Lifetime values when solving the two problems for the uniform coverage request scenario (on the top) and the variable coverage request scenario (on the bottom), with* $|F| = 4$.

*For a given version of the problem, the network lifetime is usually the same for the two coverage requirement scenarios, except for instances with fewer number of sensors.* For example, identical solutions values were found in 21 out of 24 cases for $|F| = 4$ except for two cases with $|S| = 200$, and one case with $|S| = 400$. Overall, in these three cases the difference between average solutions is always less than 7.43%. On these datasets, one could expect a fewer number

of feasible covers to exist. Therefore, if some of the covers are feasible for a given type of coverage requirement and not for the other, on bigger instances a larger set of alternative covers may be available and help to converge to the same optimal lifetime. This result is significant, since it suggests that when a particular robustness level is needed for a given application in terms of coverage request for a subset of particularly relevant sensor families, it can be expected to be obtained with reasonable trade-offs in terms of solution quality, especially if many sensors are available in the network.

We can also compare the quality of the optimal solutions of the two problems with respect to the level of regularity by comparing the value of the variable $w_{min}$, which, we recall, is the minimum amount of time, among all the families, for which a target is covered. Refer to Table 5.5 for $|F| = 4$, and Tables B.4 and B.5 in the Appendix B for $|F| = 2$ and $|F| = 6$, respectively.

*Solving the regular version of the problem improves the value of $w_{min}$.* This is an expected result, since, when solving the regular version of the problem, we look for solutions such that $w_{min}$ is maximized, while there are no requirements for $w_{min}$ in the basic version of the problem. Hence, alternative optimal solutions with the same lifetime but lower value of $w_{min}$ can be generally selected when solving the basic variant of the problem. In particular, the average percentage difference between the optimum $w_{min}$ obtained when solving MLMFP-R, and the value of $w_{min}$, obtained when solving MLMFP, with uniform coverage requests is equal to 11.89% for $|F| = 2$, 11.66% for $|F| = 4$, and 4.60% for $|F| = 6$. The average percentage difference when solving the problems with variable coverage requests is equal to 86.03% for $|F| = 2$, 28.11% for $|F| = 4$, and 52.43% for $|F| = 6$.

*The value of $w_{min}$, when solving the regular version of the problem are the same for both the uniform and the variable request scenarios.* This is due to the fact that the $w_{min}$ value depends on target-family combinations of the most unfortunate coverage situations for each instance. In order to investigate this aspect, we checked for each solution provided by MLMFP-R, which target-family

Table 5.5: **Values of $w_{min}$ for $|F| = 4$ scenarios.**

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

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

combinations corresponded to the $w_{min}$ coverage level. We found that, for any given input instance at least one of such unfortunate combinations was always found to be common to the uniform and variable scenarios.

*The value of $w_{min}$, when solving the basic version of the problem, is higher*

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

# Chapter 6

# General Conclusions

This research thesis presents an overview about the wireless sensor networks, their applications and typical coverage issues. Our research focus has been dedicated to one of the most important issues in this field, which is related to maximizing the amount of time over which a set of points of interest located in a given area can be monitored by means of such wireless sensor networks. This problem is well known in the literature as the Maximum Network Lifetime Problem. The exact column generation technique has been investigated and it has been showed how to apply the technique to this problem, such an exact approach has then been adapted to solve three variants of the classical problem. This research work also presents the basic idea about how to embed heuristics in such type of exact approach. This idea has been used to embed in the above mentioned column generation algorithms an ad-hoc designed genetic algorithm for each of the studied problem variants, which proved to be very efficient. More in detail, in this work we addressed the maximum lifetime problem on wireless sensor networks considering both the classical variants in which all sensors have to be covered and the one in which a portion of them can be neglected at all times ($\alpha$-MLP) in order to increase the overall network lifetime. Our proposed algorithm is shown to be highly efficient and to outperform significantly the algorithms available in literature for both these cases. The other two problem variants considered in this research work have been proposed for the first time by us and are related to wireless sensors networks with heterogeneous sensors. Nowadays wireless sensor networks can be composed of several different types of sensor devices, which

are able to monitor different aspects of the region of interest (including sound, vibrations, chemical contaminants, among others). These different sensors can be organized to work in a coordinated fashion in many relevant application contexts. Therefore we faced the problem of maximizing the amount of time during which such a network can remain operational while assuring globally a minimum coverage for all the different sensor types. In one of the two problem variants considered in this context we considered also some global regularity conditions, in order to guarantee an adeguate coverage for each sensor type to each target. In our computational tests the proposed specific genetic algorithm has been shown to be able to meaningful speed up the global column generation procedure, enabling the resolution of large-scale instances within reasonable computational times. Indeed we were able to solve several large scale instances to optimality in less than one thousand seconds. 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. First of all the general column generation framework might be improved by trying to adopt more specific initialization strategies to obtain fast good feasible covers early in the procedure. Moreover, the use of multiple sensor families will be further addressed, due to its great relevance for real world scenarios. In particular, several well known design issues deriving from well known variants of the classical problem such as connectivity, routing and robustness (i.e. fault tolerance), which may arise in specific applications will be faced in this heterogeneous context.

# Appendix A

Tables A.1 and A.2 contain the results related to the Group 1 instances proposed in [54] solved using the GCG and GR algorithms (see chapter 4). Each instance in this group contain 15 targets, while the number of sensors for the different scenarios is specified by the *Sensor* heading in the tables. Each line in the tables contain averages over 5 different instances with the same characteristics. For a description of the table headings, refer to the description of Tables 4.3 and 4.4 in the paper.

| Sensors | $T_\alpha$ | Lifetime | Time | Inv | Col | Flr |
|---|---|---|---|---|---|---|
| 25 | 8 | 13.60 | 0.29 | 3.00 | 11.80 | 1.00 |
| | 11 | 10.40 | 0.26 | 3.40 | 16.40 | 1.00 |
| | 13 | 6.60 | 0.19 | 2.80 | 17.00 | 1.00 |
| | 15 | 3.60 | 0.19 | 2.00 | 19.60 | 1.00 |
| 50 | 8 | 27.23 | 0.59 | 4.60 | 19.40 | 1.00 |
| | 11 | 19.40 | 0.40 | 4.60 | 18.80 | 1.00 |
| | 13 | 13.93 | 0.33 | 3.60 | 21.00 | 1.00 |
| | 15 | 9.40 | 0.26 | 2.40 | 15.20 | 1.00 |
| 100 | 8 | 54.90 | 1.27 | 9.20 | 21.00 | 1.00 |
| | 11 | 41.49 | 1.25 | 11.00 | 23.20 | 1.20 |
| | 13 | 30.40 | 0.87 | 7.00 | 26.60 | 1.00 |
| | 15 | 15.40 | 0.52 | 3.00 | 21.80 | 1.00 |
| 150 | 8 | 87.60 | 2.39 | 11.80 | 22.00 | 1.00 |
| | 11 | 66.98 | 2.40 | 15.40 | 22.80 | 1.40 |
| | 13 | 51.72 | 1.97 | 12.20 | 27.60 | 1.00 |
| | 15 | 25.00 | 0.89 | 4.00 | 24.60 | 1.00 |
| **AVG** | | | | **6.25** | **20.55** | **1.04** |

Table A.1: Results obtained by the GCG algorithm on the Group 1 benchmark instances proposed in [54].

| **Sensors** | $T_\alpha$ | **GR** | | **GCG** | | **GAP** |
|---|---|---|---|---|---|---|
| | | LifeTime | Time | LifeTime | Time | |
| 25 | 8 | 13.60 | 0.26 | 13.60 | 0.29 | |
| | 11 | 10.40 | 0.44 | 10.40 | 0.26 | |
| | 13 | 6.60 | 0.11 | 6.60 | 0.19 | |
| | 15 | 3.60 | 0.01 | 3.60 | 0.19 | |
| 50 | 8 | 27.23 | 1.11 | 27.23 | 0.59 | |
| | 11 | 19.40 | 0.68 | 19.40 | 0.40 | |
| | 13 | 13.93 | 0.39 | 13.93 | 0.33 | |
| | 15 | 9.40 | 0.01 | 9.40 | 0.26 | |
| 100 | 8 | 54.90 | 5.95 | 54.90 | 1.27 | 78.72% |
| | 11 | 41.49 | 8.03 | 41.49 | 1.25 | 84.39% |
| | 13 | 30.40 | 2.74 | 30.40 | 0.87 | 68.42% |
| | 15 | 15.40 | 0.02 | 15.40 | 0.52 | |
| 150 | 8 | 87.60 | 15.24 | 87.60 | 2.39 | 84.31% |
| | 11 | 66.98 | 13.90 | 66.98 | 2.40 | 82.74% |
| | 13 | 51.72 | 9.79 | 51.72 | 1.97 | 79.86% |
| | 15 | 25.00 | 0.02 | 25.00 | 0.89 | |
| **AVG** | | | **3.67** | | **0.88** | **79.74%** |

Table A.2: Computational results of GCG and GR algorithms on the Group 1 benchmark instances proposed in [54].

# Appendix B

This appendix contains additional figures and captions for chapter 5.

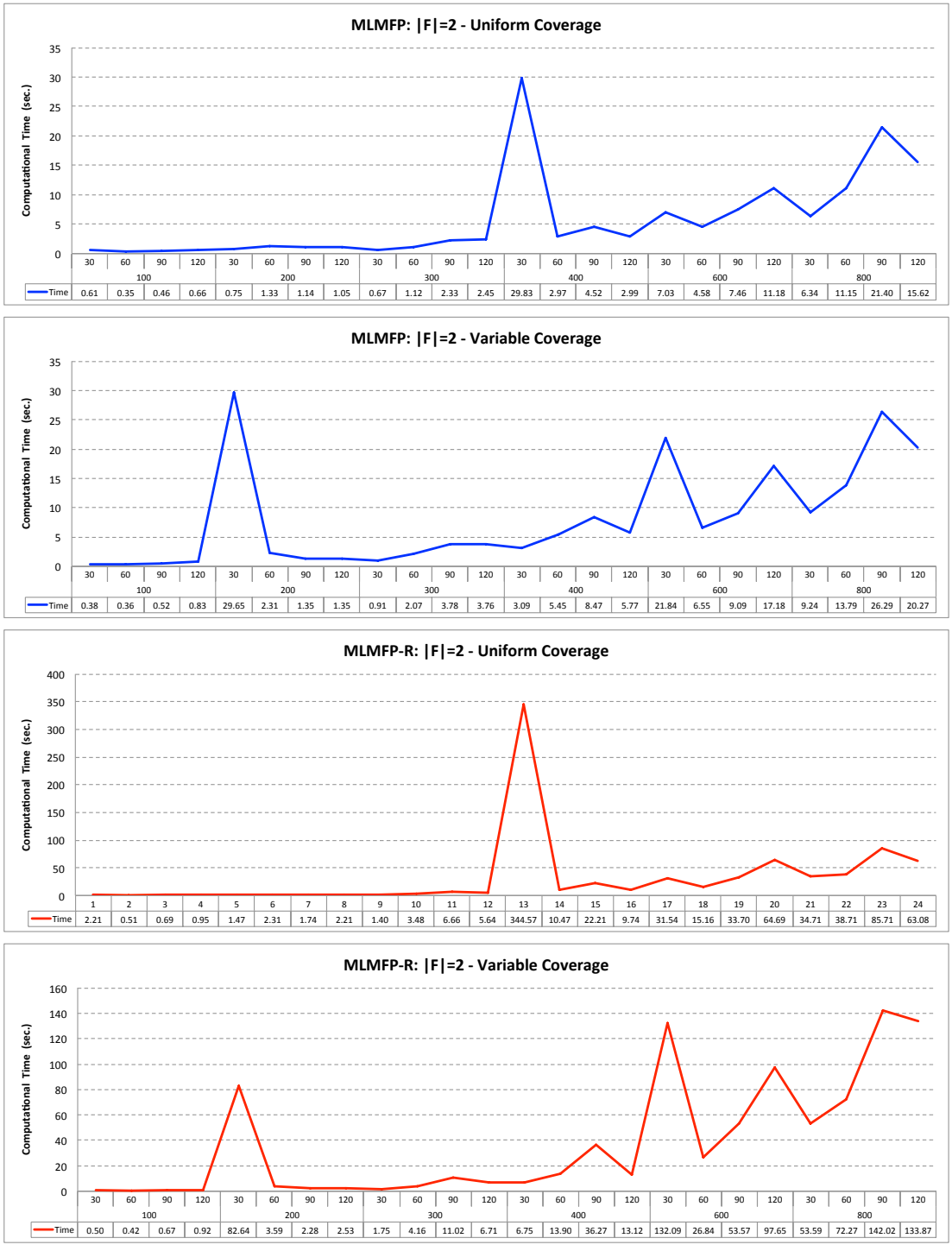



Figure B.1: *Computational time of CG+GA when solving the basic version of the problem (on the top half) and the regular version of the problem (on the bottom half), for the uniform coverage request scenario and the variable coverage request scenario, with* $|F| = 2$.

# . APPENDIX B

## Table B.1: **Comparison of our approach (CG+GA) and a pure column generation approach (CGonly) when solving MFMLP-R.**

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

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

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

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

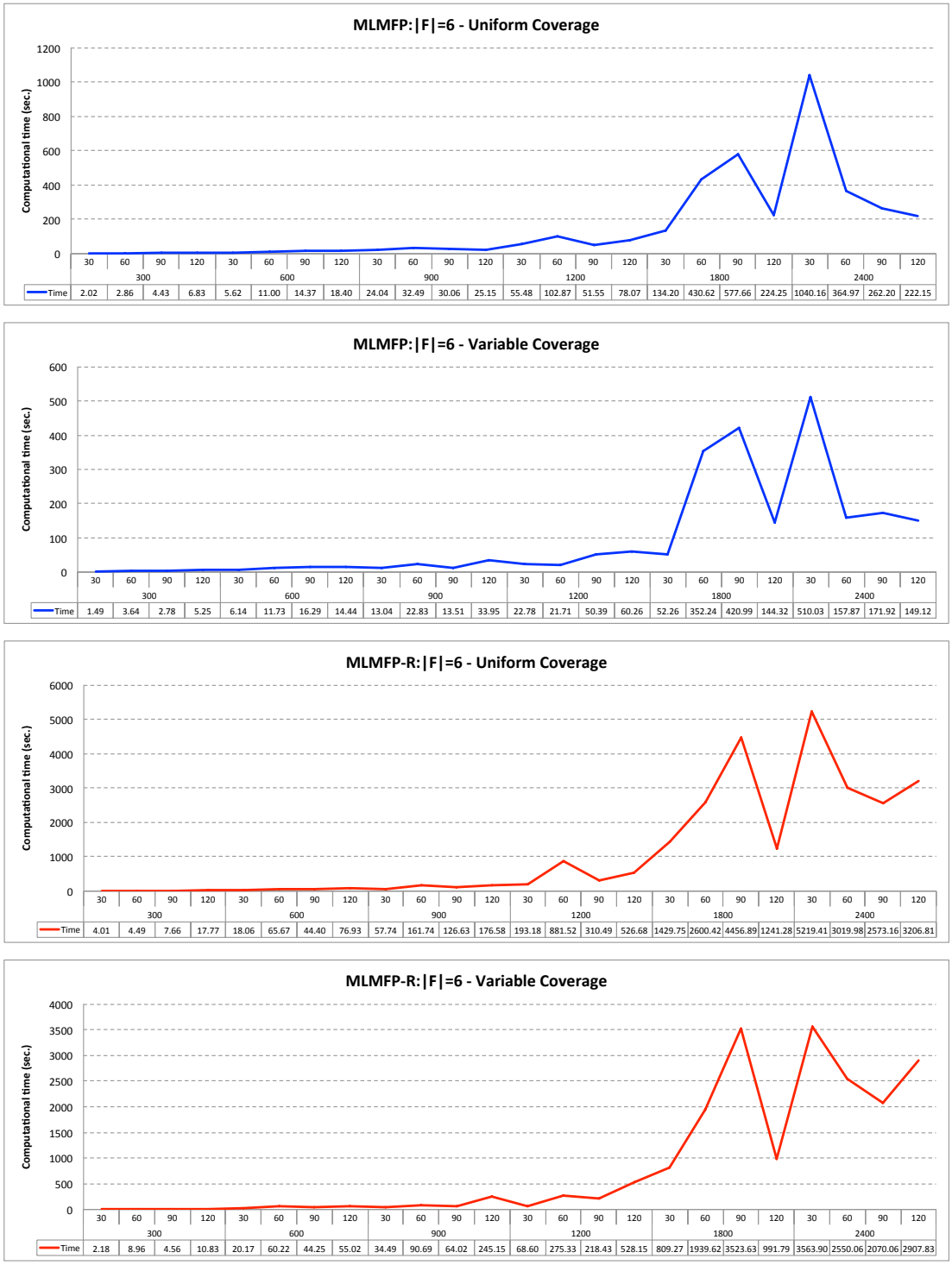



Figure B.2: *Computational time of CG+GA when solving the basic version of the problem (on the top half) and the regular version of the problem (on the bottom half), for the uniform coverage request scenario and the variable coverage request scenario, with* $|F| = 6$.

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

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

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

Table B.3: **Results of CG+GA for $|F| = 6$ scenarios when solving the basic version of the problem (MLMFP) and the regular version (MLMFP-R).**

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

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

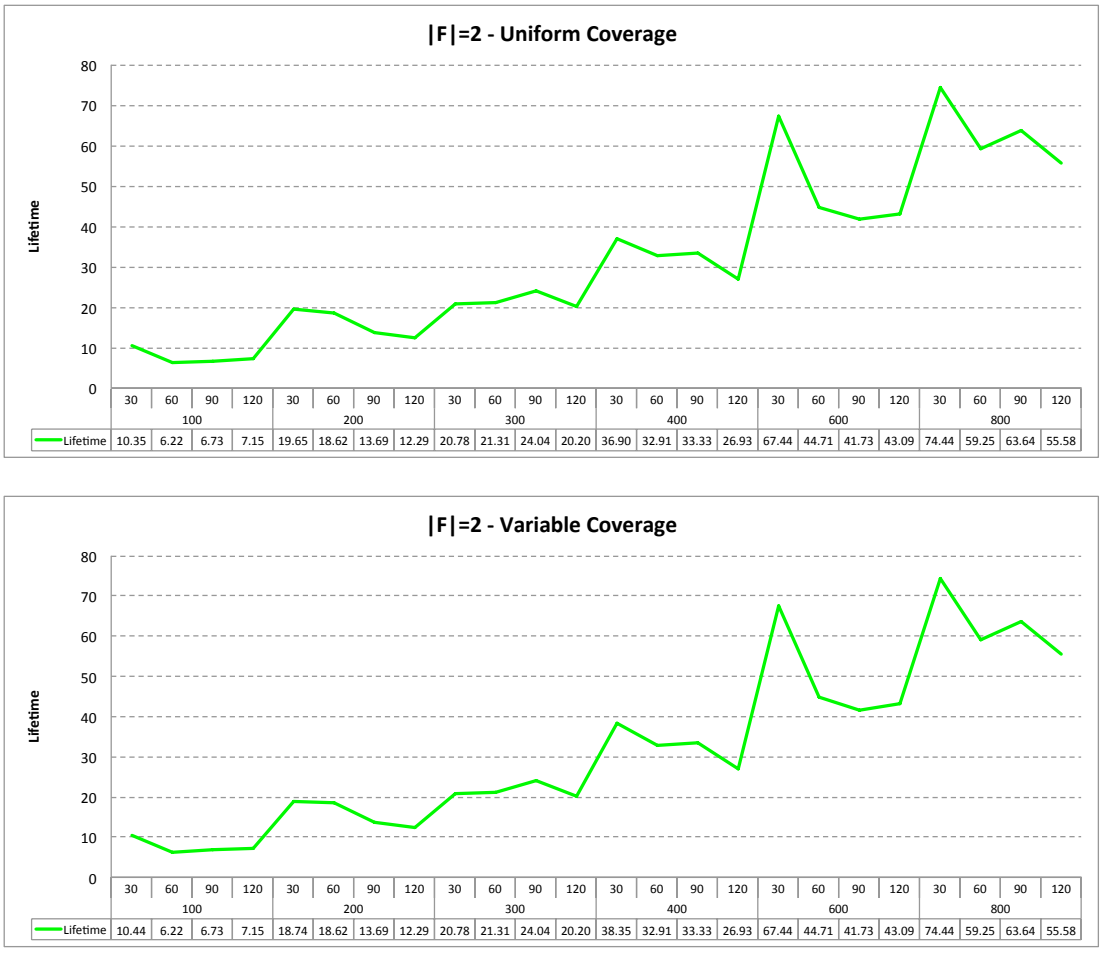



Figure B.3: *Lifetime values when solving the two problems for the uniform coverage request scenario (on the top) and the variable coverage request scenario (on the bottom), with* $|F| = 2$.

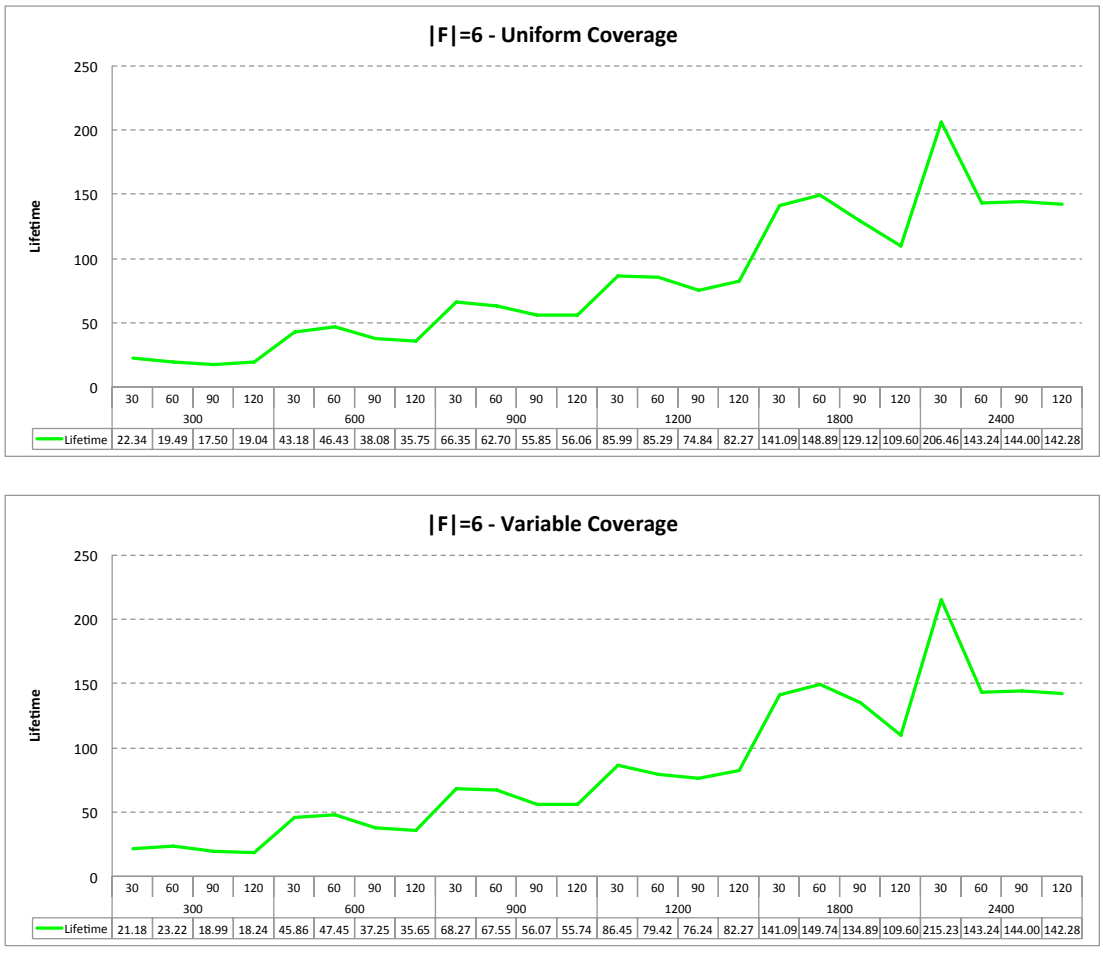



Figure B.4: *Lifetime values when solving the two problems for the uniform coverage request scenario (on the top) and the variable coverage request scenario (on the bottom), with* $|F| = 6$.

Table B.4: **Values of $w_{min}$ for $|F| = 2$ scenarios**.

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

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

Table B.5: **Values of $w_{min}$ for $|F| = 6$ scenarios**. Each entry is an average of five instances. Results are reported for both the problems and for both the coverage requirements. Column $w_{min}$ for MLMFP is evaluated by checking the minimum amount of time for which, among all the families, a target is covered in the optimum solution. Column $w_{min}$ for MLMFP-R is the optimum value of the related variable obtained when solving this problem variant. Column *% Gap* reports the percentage difference between the optimum $w_{min}$ obtained when solving MLMFP-R and the value of $w_{min}$ obtained when solving MLMFP.

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

# References


[1] A.E.A.A. Abdulla, H. Nishiyama, and N. Kato. Extending the lifetime of wireless sensor networks: A hybrid routing algorithm. *Computer Communications*, 35(9):1056–1063, 2012.

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

[3] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422, 2002.

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

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

[6] Leif H Appelgren. A column generation algorithm for a ship scheduling problem. *Transportation Science*, 3(1):53–68, 1969.

[7] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

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

[9] John Daniel Bagley. The behavior of adaptive systems which employ genetic and correlation algorithms. *Ann Arbor: The University of Michigan*, 1967.

[10] P. Berman, G. Calinescu, C. Shah, and A. Zelikovsky. Power efficient monitoring management in sensor networks. In *Proceedings of the Wireless Communications and Networking Conference*, volume 4, pages 2329 – 2334, 2004.

[11] P. Berman, G. Calinescu, C. Shah, and A. Zelikovsky. Efficient energy management in sensor networks. In *Ad Hoc and Sensor Networks. Nova Science Publishers*. Nova Science Publisher, 2005.

[12] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[13] Yanzhong Bi, Limin Sun, Jian Ma, Na Li, Imran Ali Khan, and Canfeng Chen. Hums: an autonomous moving strategy for mobile sinks in data-gathering sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2007, 2007.

[14] Maria J. Blesa, Christian Blum, and Stefan Voß, editors. *Hybrid Metaheuristics - 9th International Workshop, HM 2014, Hamburg, Germany, June 11-13, 2014. Proceedings*, volume 8457 of *Lecture Notes in Computer Science*. Springer, 2014.

[15] C. Blum, M. J. Blesa Aguilera, A. Roli, and M. Sampels, editors. *Hybrid Metaheuristics - An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*. Springer-Verlag, Berlin/Heidelberg, 2008.

[16] V. L. Boginski, C. W. Commander, P. M. Pardalos, and Y. Ye, editors. *Sensors: theory, algorithms, and applications*, volume 61 of *Springer Optimization and Its Applications*. Springer-Verlag, New York, 2011.

[17] 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.

[18] Hans J Bremermann. Optimization through evolution and recombination. *Self-organizing systems*, pages 93–106, 1962.

[19] Hans J Bremermann, M Rogson, and S Salaff. Global properties of evolution processes. *Natural automata and useful simulations*, pages 3–41, 1966.

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

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

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

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

[24] Mihaela Cardei and Ding-Zhu Du. Improving wireless sensor network lifetime through power aware organization. *Wireless Networks*, 11(3):333–340, 2005.

[25] Mihaela Cardei and Jie Wu. Coverage in wireless sensor networks. *Handbook of Sensor Networks*, pages 422–433, 2004.

[26] Mihaela Cardei and Jie Wu. Energy-efficient coverage problems in wireless ad-hoc sensor networks. *Computer communications*, 29(4):413–420, 2006.

[27] F. Carrabs, R. Cerulli, C. D'Ambrosio, M. Gentili, and A. Raiconi. The maximum lifetime problem for sensor networks with multiple sensor families. Submitted, 2014.

[28] F. Castaño, E. Bourreau, N. Velasco, A. Rossi, and M. Sevaux. Exact approaches for lifetime maximization in connectivity constrained wireless

multi-role sensor networks. *European Journal of Operational Research*, 241(1):28–38, 2015.

[29] 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*, 41:77–84, 2013.

[30] F. Castaño, A. Rossi, M. Sevaux, and N. Velasco. A column generation approach to extend lifetime in wireless sensor networks with coverage and connectivity constraints. *Computers & Operations Research*, 52(B):220–230, 2014.

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

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

[33] Sriram Chellappan, Xiaole Bai, Bin Ma, Dong Xuan, and Changqing Xu. Mobility limited flip-based sensor networks deployment. *Parallel and Distributed Systems, IEEE Transactions on*, 18(2):199–211, 2007.

[34] Karthik Dantu, Mohammad Rahimi, Hardik Shah, Sandeep Babel, Amit Dhariwal, and Gaurav S Sukhatme. Robomote: enabling mobility in sensor networks. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 55. IEEE Press, 2005.

[35] George B Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

[36] W. Dargie and C. Poellabauer, editors. *Fundamentals of Wireless Sensor Networks: Theory and Practice*. John Wiley and Sons, London, 2010.

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

[38] L. Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[39] Ioan Deaconu and Andrei Voinescu. Mobile gateway for wireless sensor networks utilizing drones. In *RoEduNet Conference 13th Edition: Networking in Education and Research Joint Event RENAM 8th Conference, 2014*, pages 1–5. IEEE, 2014.

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

[41] Jacques Desrosiers and Marco E Lübbecke. *A primer in column generation*. Springer, 2005.

[42] I. Dietrich and F. Dressler. On the lifetime of wireless sensor networks. *ACM Transactions on Sensor Networks*, 5(1), 2009.

[43] E. J. Duarte-Melo and M Liu. Analysis of energy consumption and lifetime of heterogeneous wireless sensor networks. In *Proceedings of the IEEE Global Telecommunications Conference*, volume 1, pages 21–25, 2002.

[44] Jeremy Elson and Deborah Estrin. Wireless sensor networks. chapter Sensor Networks: A Bridge to the Physical World, pages 3–20. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[45] David B Fogel. *Evolutionary computation: toward a new philosophy of machine intelligence*, volume 1. John Wiley & Sons, 2006.

[46] Lester Randolph Ford Jr and Delbert R Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958.

[47] A Fraser. Comments on mathematical challenges to the neo-darwinian concept of evolution. In *The Wistar Institute symposium monograph*, volume 5, pages 107–107, 1966.

[48] Alex S Fraser. Simulation of genetic systems by automatic digital computers i. introduction. *Australian Journal of Biological Sciences*, 10:484–491, 1957.

[49] Alex S Fraser. Simulation of genetic systems by automatic digital computers ii. effects of linkage on rates of advance under selection. *Australian Journal of Biological Sciences*, 10:492–499, 1957.

[50] Alex S Fraser. Simulation of genetic systems by automatic digital computers vi. epistasis. *Australian Journal of Biological Sciences*, 13(2):150–162, 1960.

[51] Alexander S Fraser. The evolution of purposive behavior, 1968.

[52] AS Fraser. Simulation of genetic systems. *Journal of Theoretical Biology*, 2(3):329–346, 1962.

[53] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman, 2002.

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

[55] P.B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. Irisnet: an architecture for a worldwide sensor web. *IEEE Pervasive Comput.*, 2:22–33, 2003.

[56] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

[57] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting stock problem-part ii. *Operations research*, 11(6):863–888, 1963.

[58] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[59] 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*, 26(1):117–129, 2010.

[60] Yu Gu, Yusheng Ji, Jie Li, and Baohua Zhao. Qos-aware target coverage in wireless sensor networks. *Wireless Communications and Mobile Computing*, 9(12):1645–1659, 2009.

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

[62] John H Holland. Adaptive plans optimal for payoff-only environments. Technical report, DTIC Document, 1969.

[63] John H Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88–105, 1973.

[64] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah 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*, pages 56–67. ACM, 2000.

[65] Jie JIA, Jian Chen, Gui-Ran Chang, and Ying-You WEN. Efficient cover set selection in wireless sensor networks. *Acta Automatica Sinica*, 34(9):1157–1162, 2008.

[66] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for "Smart Dust". In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, MobiCom '99, pages 271–278, New York, NY, USA, 1999. ACM.

[67] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[68] Santosh Kumar, Ten H Lai, and Anish Arora. Barrier coverage with wireless sensors. In *Proceedings of the 11th annual international conference on Mobile computing and networking*, pages 284–298. ACM, 2005.

[69] Santosh Kumar, Ten H Lai, and Anish Arora. Barrier coverage with wireless sensors. *Wireless Networks*, 13(6):817–834, 2007.

[70] Anthony LaMarca, Waylon Brunette, David Koizumi, Matthew Lease, Stefan B Sigurdsson, Kevin Sikorski, Dieter Fox, and Gaetano Borriello. Making sensor networks practical with robots. In *Pervasive Computing*, pages 152–166. Springer, 2002.

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

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

[73] Uichin Lee, Eugenio Magistretti, Biao Zhou, Mario Gerla, Paolo Bellavista, and Antonio Corradi. Efficient data harvesting in mobile sensor platforms. In *Pervasive Computing and Communications Workshops, 2006. PerCom Workshops 2006. Fourth Annual IEEE International Conference on*, pages 5–pp. IEEE, 2006.

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

[75] M.E. Lbbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[76] Seapahn Megerian, Farinaz Koushanfar, Miodrag Potkonjak, and Mani B Srivastava. Worst and best-case coverage in sensor networks. *Mobile Computing, IEEE Transactions on*, 4(1):84–92, 2005.

[77] Seapahn Megerian, Farinaz Koushanfar, Gang Qu, Giacomino Veltri, and Miodrag Potkonjak. Exposure in wireless sensor networks: theory and practical solutions. *Wireless Networks*, 8(5):443–454, 2002.

[78] Seapahn Meguerdichian, Farinaz Koushanfar, Miodrag Potkonjak, and Mani B. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *in IEEE INFOCOM*, pages 1380–1387, 2001.

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

[80] G. L. Nemhauser. Column generation for linear and integer programming. *Documenta Mathematica*, EXTRA VOLUME ISMP:65–73, 2012.

[81] The Array of Things. The array of things project. *"https://arrayofthings.github.io"*, 2014.

[82] 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*, pages 196–199, 2012.

[83] Gregory J Pottie and William J Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[84] A. Raiconi and M. Gentili. Exact and metaheuristic approaches to extend lifetime and maintain connectivity in wireless sensors networks. In J. Pahl, T. Reiners, and S. Voss, editors, *Network Optimization*, volume 6701 of *Lecture Notes in Computer Science*, pages 607–619. Springer, Berlin/Heidelberg, 2011.

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

[86] I. Rechenberg. Cybernetic solution path of an experimental problem. In *Royal Aircraft Establishment Translation No. 1122, B. F. Toms, Trans.* Ministry of Aviation, Royal Aircraft Establishment, 1965.

[87] ReportsnReports. Semiconductor wireless sensor internet of things (iot): Market shares, strategies, and forecasts, worldwide, 2014 to

2020. *"www.reportsnreports.com/reports/271010-wireless-sensor-networks-market-shares-strategies-and-forecasts-worldwide-2013-to-2019.html"*, 2014.

[88] Mit Technology Review. 10 emerging technologies that will change the world. *"www2.technologyreview.com/featured-story/401775/10-emerging-technologies-that-will-change-the/2/"*, 2003.

[89] Richard S Rosenberg. Stimulation of genetic populations with biochemical properties. *Ann Arbor: The University of Michigan*, 1967.

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

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

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

[93] André Rossi, Alok Singh, and Marc Sevaux. Lifetime maximization in wireless directional sensor network. *European Journal of Operational Research*, 231(1):229–241, 2013.

[94] A.C. Santos, C. Duhamel, L.S. Belisrio, and L.M. Guedes. Strategies for designing energy-efficient clusters-based wsn topologies. *Journal of Heuristics*, 18(4):657–675, 2012.

[95] Yi Shi. *Algorithms and Optimization for Wireless Networks*. PhD thesis, Virginia Polytechnic Institute and State University, 2007.

[96] A. Singh, A. Rossi, and M. Sevaux. Matheuristic approaches for q-coverage problem versions in wireless sensor networks. *Engineering Optimization*, 45(5):609–626, 2013.

[97] S.N. Sivanandam and S.N. Deepa. *Introduction to genetic algorithms*. Springer, 2008.

[98] S. Slijepcevic and M. Potkonjak. Power efficient organization of wireless sensor networks. volume 2, pages 472–476, 2001.

[99] Arun A Somasundara, Aman Kansal, David D Jea, Deborah Estrin, and Mani B Srivastava. Controllably mobile infrastructure for low energy embedded networks. *Mobile Computing, IEEE Transactions on*, 5(8):958–973, 2006.

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

[101] EG Talbi. Hybrid metaheuristics, volume 434 of studies in computational intelligence, 2013.

[102] Bang Wang. Coverage problems in sensor networks: A survey. *ACM Computing Surveys (CSUR)*, 43(4):32, 2011.

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

[104] Roy Want, Keith I Farkas, and Chandrasekhar Narayanaswami. Guest editors' introduction: Energy harvesting and conservation. *Pervasive Computing, IEEE*, 4(1):14–17, 2005.

[105] Thomas Weise. Global optimization algorithms ? theory and application ?, 2008.

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

[107] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008.

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

[109] Chen Zhao, Sam Yisrael, Joshua R Smith, and Shwetak N Patel. Powering wireless sensor nodes with ambient temperature changes. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 383–387. ACM, 2014.

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

[111] Chuan Zhu, Chunlin Zheng, Lei Shu, and Guangjie Han. A survey on coverage and connectivity issues in wireless sensor networks. *Journal of Network and Computer Applications*, 35(2):619–632, 2012.

[112] D. Zorbas, D. Glynos, P. Kotzanikolaou, and C. Douligeris. Solving coverage problems in wireless sensor networks using cover sets. *Ad Hoc Networks*, 8(4):400–415, 2010.