Lancaster University

Exploring Various Set Covering Problem Techniques to Tackle the Optimal Camera Placement Problem

Malek Almousa, BSc, MSc

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy

in the

Lancaster University Management School
Department of Management Science

June 2025

Abstract

Optimal Camera Placement (OCP) problem is a combinatorial optimization problem, where the aim is to find an optimal placement of cameras, given that a target area is completely covered. A single-objective OCP is commonly formulated in two possible ways. The first one is to maximize the area coverage, such that minimal camera cost is maintained. The second one is to minimize camera cost, such that complete area coverage is achieved. The idea of the second formulation of OCP can be similar to a popular $\mathcal{NP}$-hard combinatorial optimization problem, namely, the Set Covering Problem (SCP). Thus, OCP can be formulated as SCP. However, a special variant of SCP is used in this paper. This variant is called the unicost SCP (USCP), where the cost is assumed to be unified and, therefore, not a factor. The reason behind using USCP is that our problem instances do not consider cost as a factor, which makes USCP well-suited for this study. Thus, the objective of our OCP problem is to minimize the number of cameras rather than the cost.

Despite the connection between OCP and SCP, not many studies have used techniques from the former's literature and applied them to address the latter. This study exploits this connection by implementing various SCP techniques to address 69 OCP instances retrieved from the GECCO 2021 competition on "the optimal camera placement problem and the unicost set covering problem". First, our OCP problems are solved using classic exact methods. The initial results seem to be promising, but can be improved. Next, SCP problem reduction techniques are introduced and implemented to address the complexity of our problem. A comparison between the results before and after reduction is given. After that, the OCP is formulated as a bi-objective problem, where the two objectives are to, simultaneously, minimize the number of cameras and maximize the area coverage. An effective multi-objective technique is used to obtain the efficient solutions of this problem.

Keywords: Combinatorial Optimization, Optimal Camera Placement, Set Covering Problem, Reduction Techniques, Bi-Objective, Multi-Criteria Decision Making

Acknowledgements

I would like to express my sincere gratitude to both my supervisors, Professor Matthias Ehrgott and Dr. Ahmed Kheiri, for their invaluable guidance, patience, trust in my abilities, and constant support throughout the past four years of my PhD journey. Their exceptional supervision, valuable feedback, and motivation have been crucial in shaping my academic growth as well as the progress of this thesis. Without their help since the first day of my PhD, I would not have made it until this point. I am eternally grateful for everything they have done for me.

I would also like to express my genuine appreciation to Lancaster University, and more specifically, the Department of Management Science, for providing me with the perfect academic environment and giving me access to invaluable resources, which were pivotal to my academic growth. My heartfelt thanks to the faculty, staff, and colleagues for making this journey special.

I am also grateful to my examiners, Professor Daniel Gartner and Dr. Thu Dang, for their time, thoughtful feedback, and constructive suggestions, which helped improve the quality of this thesis.

Finally, I would like to extend my heartfelt thanks to my family and friends for their continuous support. Their belief in me and, generally, their constant presence in the last four years have given me the strength and motivation to keep going. I am forever grateful for them.

Declaration

I declare that this thesis represents my own original work, which was done in the past four years as a PhD student at Lancaster University, under the supervision of Professor Matthias Ehrgott and Dr. Ahmed Kheiri. To help achieve this work, datasets were retrieved from the GECCO 2021 competition on "the optimal camera placement problem and the unicost set covering problem", which are available online. In addition, I declare that this thesis has not been submitted as part of an application for any other degree at Lancaster University or any other institution.

I also declare the following:

This is a thesis by publication, where chapters 3, 4, and 5 represent different papers. Thus, each one of those chapters has been written as an independent piece of work and is supposed to be viewed that way.

Chapter 3 of this thesis has been published as: Malek Almousa, Matthias Ehrgott, and Ahmed Kheiri. Exploring the optimal camera placement problem and its relationship with the set covering problem. In *International Conference on Business Analytics in Practice*, pages 295–306. Springer, 2024.

Chapter 4 and Chapter 5 of this thesis represent two papers that have not been published yet. We plan to submit them to respected journals as soon as possible.

Malek Almousa

June 2025

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, installing cameras for various purposes, including security, tracking, or general monitoring purposes, has become a very common practice in public places, businesses, or even personal spaces. Some people install cameras at their homes to track their children or pets while they are at work, to obtain solid evidence in case of accidents, or generally just to feel safe. As for businesses, having cameras installed can be very valuable in tracking employees for performance purposes, ensuring the safety of everyone in the building, and preventing any potential theft or general assault. Regarding public places or cities in general, having cameras does not only reduce crime activities, but it also helps in monitoring large crowds, dealing with traffic, and improving such places by gathering data.

When people wish to install cameras in their homes, the process is normally not too complicated. To illustrate more, if a family wants to monitor their two-bedroom apartment, installing three cameras, one in each room, might be enough to monitor most of the place, depending on the structure of the apartment. However, for big venues, such as museums or even airports, the process becomes more challenging and costly. The reason behind this is that an efficient camera placement plan would be needed, where the minimum number of cameras is used to monitor most, if not all, the venue. This is where an optimal camera placement can be useful.

Optimal Camera Placement (OCP) is the process of finding a camera placement strategy that utilizes the cheapest camera plan possible to fully monitor a certain location (i.e., rooms, houses, theatres, cities, and so on). OCP is an optimization problem which can be formulated using two common objectives: maximizing area coverage or minimizing

camera cost. Using the first objective, the main goal of OCP would be to maximize area coverage such that the camera cost is minimal; on the other hand, when the second objective is used, the main goal of OCP would be to minimize camera cost such that complete area coverage is achieved.

The origins of OCP come from the field of computational geometry in the 70s of the last century, where Chvátal [1] studied the Art Gallery Problem (AGP). AGP is the process of finding the optimal guard placement in an art gallery. The main goal of AGP is to minimize the number of guards in an art gallery, such that the whole gallery is covered by the selected guards. From an OCP point of view, we employ cameras instead of guards, and the goal of an OCP problem would be to minimize the number of cameras, such that the whole target location is covered by the selected cameras. Ever since the introduction of this problem, AGP has become the source of inspiration for many OCP studies in various fields, such as robotics ([2]; [3]), motion capture systems ([4]; [5]; [6]) and surveillance ([7]; [8]; [9]). This thesis mainly focuses on OCP for surveillance purposes; nevertheless, this thesis discusses various applications of OCP to provide the reader with a better understanding of the idea of camera placement and the recent work that has been done on this topic. More on this is discussed in chapters 3, 4, 5.

1.1 Research Motivation and Contribution

As mentioned earlier, OCP can be modelled in two common ways, where the the first one maximizes the area coverage, given that the cost of the camera must be minimal; while the other one minimizes the camera cost, given that complete coverage of the target area must be achieved. When adopting the second model, OCP can be formulated as the Set Covering Problem (SCP). SCP is a widely studied combinatorial optimization problem, and, in the strong sense, is classified as an $\mathcal{NP}$-hard problem [10]. Given that a binary matrix containing m number of rows and n number of columns, where a certain cost c_j is assigned to each column j, SCP aims to find a subset of columns that minimizes the total cost, such that each row i is covered by at least one column j. From an OCP point of view, columns can be cameras and rows can be the certain locations that must covered by a subset of the given cameras. In other words, the goal of an OCP problem formulated as a set covering problem would be to find a subset of cameras (columns) that minimizes the total cost, such that each location (row) must be covered by at least one camera. Since more work has been done on SCP than on OCP, the similarity between the two problems can be exploited by finding techniques from the former's literature

and implement them to address the latter. In addition, exploring various SCP methods could inspire inventing new methods to tackle the OCP problem. More on SCP and its techniques will be discussed in Chapter 2.

Despite the fact that OCP can be formulated as SCP, not many OCP studies have exploited this connection [9]. Thus, this thesis explores the SCP literature to find various techniques that can be used to address the OCP problem. However, this is not as simple as it sounds. Our problem instances, which will be discussed in detail later, are quite large, which means that it can be time consuming to solve them using the standard optimization techniques. To elaborate on that, it can take weeks or even months just to solve one problem instance, especially the larger instances. Considering that we have to deal with 69 instances, solving all of them could take years. Therefore, we propose a reduction algorithm, which was inspired from the SCP literature, to address the size issue and the $\mathcal{NP}$-hard nature of this problem. We are interested in testing the impact of our proposed reduction algorithm on our problem instances and then compare that to other studies from the literature.

In this study, we address 69 problem instances retrieved from the GECCO 2021 competition on the optimal camera placement problem and the unicost set covering problem [11]. 32 out of the 69 OCP problem instances are made up instances which are called *academic instances*. These academic instances consist of 3-dimensional grids that sample different sizes of rectangular-shaped rooms, where cameras can be installed on the ceilings to monitor these rooms. The remaining 37 problem instances are called *real-world instances*. As the name implies, these instances are modelled after real areas, such as urban cities. In real-world instances, different placements are used to install camera, such as the walls of buildings. Each problem instance, whether it is academic or real-world, includes two important terms that will be used throughout this thesis. The first term is called *candidates*. Each candidate represents a possible camera placement, which includes its angle and tilt. The other important term is called *samples*. Given a 3-dimensional space, samples represent 3-dimensional points within that space. From an OCP point of view, we can say that candidates can cover different samples; likewise, we can say that samples can be covered by different candidates. Thus, the goal of the OCP problem would be to find a subset of candidates that minimizes the total cost, such that each sample must be covered by at least one candidate.

The following are the main research questions of this thesis:

- Can we formulate our specific OCP problem using SCP?
- Can we solve the OCP problem using classic exact methods?
- Does reducing the size of our problem instances help in obtaining the solutions faster?
- Will the reduction algorithm itself be time consuming? If so, is there a way to fix this issue?
- Can we formulate our OCP problem as a bi-objective OCP and use SCP techniques to tackle it? If so, does the reduction algorithm have an impact on the bi-objective OCP problem?

We also list the main contributions of this thesis:

- Exploit the connection between SCP and OCP.
- Improve an already existing SCP reduction algorithm to address the size issue of our problem instances, then compare the impact of the original reduction algorithm with our new reduction algorithm.
- Compare our reduction results with other OCP studies that have used reduction techniques to address the same OCP problem instances we adopted in this thesis.
- Create an extensive OCP study, that focuses on the bi-objective version of the OCP problem.
- Propose an effective ϵ-constraint approach to tackle the bi-objective OCP problem.

1.2 Thesis Structure

Overall, this thesis consists of six chapters. An introduction to the thesis has already been given in Chapter 1. In Chapter 2, an extensive introduction optimization is given. This chapter contains different aspects of optimization that are relevant to this thesis. This includes discussions on optimization background, optimization concepts, types of optimization techniques, time complexity, and the different ways to solve optimization

problems. It also introduces the set covering problem, which will be used throughout this thesis.

The main goal of this thesis is to exploit the connection between OCP and SCP. Therefore, the next three chapters study that connection by using techniques from the SCP literature and implement them to tackle our specific OCP problem instances. In Chapter 3, the connection between the two problems is introduced. This includes discussions on both problems' literature, which helps in finding possible ways to address our problem instances. In this chapter, OCP is formulated as a USCP and then the problem instances are solved using a Linear Programming solver. For each problem instance, a certain time-limit is set. The results prove that OCP can be solved using the SCP formulation. Although, these initial results are promising, the time limit was not enough to produce all the results. The reasons behind this issue are discussed, but the main concern is the size of our problem instances, which prevents us from obtaining all the results in a reasonable time period. For future work, suggestions for addressing this complication are then discussed.

In Chapter 4, the connection between OCP and SCP is studied in more depth. The main aim of this chapter is to improve the results of Chapter 3 by addressing the size of the OCP problem instances. To achieve that, classic problem reduction techniques from the SCP literature are discussed and implemented to address the size issue. The initial results showed great promise as the sizes of these problem instances were dramatically reduced. However, it was still time consuming to reduce our OCP problems. As a result, we modified the SCP reduction techniques to make this process faster. The new results were exactly the same in terms of the size of the problems; however, the time needed to reach that outcome was significantly diminished. All the reduction and the optimization results are provided. In addition, a comparison between the results from Chapter 3 and Chapter 4 is given, which proves the effectiveness of the reduction techniques implemented in this chapter.

Chapter 5 takes a different path in terms of the optimization technique used. In this chapter, a multi-objective optimization technique, namely the ϵ-constraint method, is used to address the bi-objective OCP problem. In this chapter, OCP is formulated as a bi-objective problem, where the two objectives are to minimize the number of cameras and maximize the area coverage, simultaneously. All the relevant multi-objective terms are introduced and explained later in this thesis. With the help of the classic ϵ-constraint method, we managed to obtain the efficient solutions of a few problem instances. However, the size of the problem instances was also an issue for the bi-objective

OCP problem. Therefore, we implemented the reduction techniques used in Chapter 4 to address this issue. The new results show a dramatic improvement in terms of time consumed to obtain all the efficient solutions for each problem instance.

Finally, Chapter 6 concludes this thesis by providing a comprehensive summary of all the previous chapters. It also discusses the limitations of this thesis as well as explores the different possibilities for future work.

Chapter 2

Optimization Background

In this chapter, we start by introducing the field of Operations Research before jumping into discussing the topic of optimization, which includes its basic form, special types, techniques, time complexity of optimization problems, and finally an introduction to the main optimization problem of this thesis, namely, the Set Covering Problem.

2.1 Introduction to Operations Research

Operations Research (OR) is a quantitative field that employs quantitative tools to help decision makers make better decisions. Some of the popular OR concepts include statistical analysis, mathematical modelling, and optimization [12]. The purpose of statistical analysis generally is that it uses data for interpretations and identifying patterns; whereas, the purpose of mathematical modelling is to convert real-life systems into mathematical equations to understand said systems. As for optimization, which is the main focus of this section and thesis, it is the process of finding the best possible outcome for a problem that is defined mathematically. There are many other OR techniques that are not discussed in this thesis, but are widely popular, such as simulation, system dynamics, problem structuring methods, stochastic programming, artificial intelligence, machine learning, and many others [13]. The origins of OR come from World War II, where some OR techniques were invented to enhance military-related operations, such as radar detection, logistics, and resource allocation [12]. Because of the success of some of these OR techniques during the war, the OR field became widely popular in other domains, such as industry, business, and government. Nowadays, OR is still considered

as one of the most important fields, and is used in other domains, such as artificial intelligence, financial planning, healthcare operations, general scheduling, and supply chain management.

Generally, an OR process that is used for problem solving consists of the following five steps [12]:

1. Defining the problem.
2. Constructing the model.
3. Solving the problem.
4. Validating the model.
5. Implementing the solution.

In the first step, decision makers clearly define a problem, which would include the objectives of the problem, the constraints of the problems, and the decision variables. For example, a transportation company wants to minimize their overall travel time, while considering factors such as traffic, alternative routes, cost of petrol, number of vehicles available, and so on. The second step of an OR process is to construct a mathematical model that portrays the defined problem. After the problem has been defined and the model has been constructed, the model can be solved using various OR techniques. This is the third step. In the next step (Step 4), decision makers validate the model; meaning, they test the accuracy and how representative this solution is to the real problem. Finally, if all the previous steps are successfully achieved, the solution can be implemented to address the real problem. This process can be viewed as a straightforward step by step sequence; however, this is not the case. In each step, errors can happen, and this creates feedback loops between steps. For instance, in step 4, if the model cannot be validated, the process loops back to step 2 or even step 1, depending on the specific circumstances. Similarly, if various challenges appear in the implementation phase (step 5), then returning to the original model (step 2) might be necessary.

There are numerous OR techniques that have been implemented throughout the years; one of which is mathematical programming. Mathematical programming is a popular concept that falls under the umbrella of optimization. There are three main types of mathematical programming. The first one is called Linear Programming (LP), which is used to solve problems that consist of linear objectives and constrains. The second

one is called Nonlinear Programming (NLP), which deals with non-linear objectives and/or constraints. The third one is called Integer Programming (IP), which deals with discreet decision variables. Other OR techniques include simulation which is used to model stochastic systems, decision analysis which is usually adopted to address uncertain decisions using decision trees, and queuing theory which tries to analyse and optimize waiting lines.

OR has several benefits as well as some limitations. Starting with the benefits, OR improves the decision making process and outcome for complex problems, thrives in maximizing efficiency and minimizing waste, known to be adaptable with change, and, lastly, effective when it comes to minimizing costs of various operations in different domains. Moving to the limitations, using OR can sometimes be too challenging, as sampling real-world operations can be quite complicated. Similarly, applying OR solutions to real-world systems can be a complex process. Another limitation to OR is that it can be computationally expensive when it comes to the time and resources needed to solve OR problems. Finally, OR depends on real data, which most of the time is expensive, hard, or even impossible to obtain. For more details regarding OR in general, interested readers are advised to refer to [12].

2.2 Introduction to Optimization

Optimization is the process of solving a decision making problem by finding the best outcome out of a set of feasible outcomes, while taking into account a set of problem constraints. Generally, optimization problems focus on reaching certain objectives. These objectives are usually used to either maximize something (e.g., profit) or minimize something else (e.g., cost). Therefore, we can introduce the first component of an optimization problem, which is called the objective function. An example of an objective function for a business could be to maximize the profit. After deciding whether to maximize or minimize an objective function, decision variables can be defined. Decision variables are the second component of an optimization problem. They represent the parameters where their values are used in the objective function to find the optimal solution. An example of a decision variable for a business owner is the number of products that need to be sold to maximize the profit. The third component in optimization problems is known as the constraints. Constrains are basically the restrictions of the optimization problem. For example, the maximum number of products that can be manufactured per day is

considered as a constraint for a business owner aiming to maximize the profit. The final component is called the feasible region, from which the best outcome is selected.

Optimization is widely used in various fields, including but not limited to logistics, transportation, manufacturing, finance, artificial intelligence, and machine learning. Optimization is important because it helps businesses to enhance their decision making process, resource allocation, competitiveness by maximizing profit or minimizing cost, and adaptation to change in the nature of their specific business. For more details regarding optimization background, reader are advised to refer to [12].

A general optimization problem can be formulated as follows [14]:

$$
\begin{aligned}
&\text{minimize} \quad z(x) \\
&\text{subject to:} \\
&\quad g_i(x) \leq 0, \quad i = 1, 2, \ldots, m \\
&\quad h_j(x) = 0, \quad j = 1, 2, \ldots, p \\
&\quad x \in \mathbb{R}^n.
\end{aligned}
$$

Where $z(x)$ is the objective function that needs to be minimized. The first constraints, $g_i(x) \leq 0$, are called inequality constraints, while the second constraints, $h_j(x) = 0$, are called equality constraints. Finally, $x \in \mathbb{R}^n$ refers to the decision variable x, which belongs to the n-dimensional space. The goal of this optimization problem would be to minimize $z(x)$ by finding the relevant value of x, while considering the restrictions that $g_i(x) \leq 0$ where $i = 1, 2, \ldots, m$ and $h_j(x) = 0$ where $j = 1, 2, \ldots, p$.

Commonly, optimization problems are solved using exact methods. Exact methods ensure that the solution obtained will be the exact optimal solution (i.e., best outcome). Examples of exact methods include the Simplex method for LP problems, and Branch-and-Bound and Branch-and-Cut for IP problems. These methods are popular in OR, and specifically in optimization, since they provide the optimal solution. However, for some large problems, such as $\mathcal{NP}$-hard problems, finding the optimal solution can be computationally expensive in terms of time and resources. As a result, some experts adopt heuristics. Heuristics are algorithms designed to solve specific problems and aim to find good enough solutions (not necessarily optimal) in a reasonable time period. Despite the fact that heuristics do not guarantee optimal solutions, they are still widely popular as they can dramatically reduce the time needed to solve large-scale problems. Popular examples of heuristic techniques are the Greedy Algorithm and Local Search

Algorithms. Another technique used to find near-optimal solutions is called metaheuristic. The main difference between heuristics and metaheuristics is that the former is usually designed for specific problems, whereas the latter is used for general purposes (i.e., problem-independent). Popular examples of metaheuristics are Genetic Algorithms and Simulated Annealing.

2.3 Multi-Objective Optimization

Thus far, we discussed optimization problems assuming that they contain a single objective, where the solution would represent the best outcome. That being said, optimization problems can also be defined using multiple conflicting objectives. When a multi-objective optimization problem contains only two objectives, we can also call it as a 'bi-objective' optimization problem.

Generally, Multi-objective Optimization (MO) produces a set of *alternatives*, instead of one single solution. This is also known as the *feasible set*. These feasible solutions (or alternatives) consist of different combinations of the objective function values. Then, *efficient solutions* are selected from these feasible solutions. In MO, a solution is considered as efficient if there are no other alternatives that offer a"better" value. The next step would be to represent the solution in a graph which is created by plotting all the efficient solutions. This is called the *efficient frontier*, where any point that lies on the frontier is considered as an efficient solution. After the efficient set, which contains all the efficient solutions, has been identified and the efficient frontier has been produced, a solution can be selected by the decision makers based on their personal preference. For more details regarding MO, readers can refer to [15].

A general example of an MO problem is illustrated below:

$$\begin{aligned} &\text{minimize} \quad (f_1(x), \ldots, f_k(x)) \\ &\text{subject to:} \\ &x \in \mathcal{X}. \end{aligned}$$

Where $(f_1(x), \ldots, f_p(x))$ represents a vector of k objective functions that must be minimized. As for the constraint, $x \in \mathcal{X}$ represents the feasible decision space. The goal is to obtain all the efficient solutions, where each efficient solution $x \in \mathcal{X}$.

When the decision makers' preference is to obtain a set of efficient alternatives rather than one optimal solutions, MO is more valuable. MO is often a more realistic approach to adopt, as a single optimal solution can be too expensive or even infeasible to implement. Moreover, MO gives decision makers several options to choose from. Thus, if there is an issue with one solution, decision makers have other efficient solutions to choose from, where the solution could be worse in quality but comes at a lower cost.

There are several MO techniques used to tackle MO problems, one of which is called the weighted sum method. This method combines all objectives into one objective by assigning weights to each objective. Then, the new single objective that consists of all objectives can be optimized. The weighted sum method is a popular MO technique because it is usually not very difficult to implement. It is also useful for convex problems as it can obtain all the efficient solutions. However, weighted sum method can fail to obtain all the efficient solutions in non-convex problems. Another challenge in adopting this technique is to assign proper values for the weights. More details regarding the weighted sum method can be found in [15].

ϵ-constraint method is another popular MO technique. In this method, one objective is optimized, while all the other objectives are transformed into constraints, where each constraint (objective) is given a specific bound of ϵ. This method can work better than the weighted sum method when it comes to non-convex problems. However, it can be time consuming as the problem is solved multiple times based on the values of ϵ. More details regarding the ϵ-constraint method can be found in [15].

2.4 Combinatorial Optimization

Generally, optimization problems can be classified as continuous or discrete. As the name implies, *continuous optimization problems* can only take decision variables that represent real numbers, or generally any values within a specific range. On the other hand, decision variables in *discrete optimization problems* can represent only discrete values, such as binary values or, generally, any integers. *Combinatorial optimization problems* represent a special type of discrete problems.

Combinatorial optimization problems deal with problems that have a discrete nature. Given a finite set of feasible solutions, the goal, as always, is to find the optimal solution. The Travelling Salesman Problem (TSP), is a popular example of combinatorial problems, where the goal is to find the shortest path that passes through all the possible

locations and returns back to the original starting point. Another example of combinatorial problems is the Knapsack Problem, where the goal is to choose the items that offer the best value possible, subject to a weight capacity limit. SCP is another classic combinatorial optimization problem. As pointed out earlier, SCP is a major part of this thesis, which is why it is essential to discuses combinatorial problems in general. More information regarding SCP is given later in this chapter and generally throughout the thesis.

Solving combinatorial optimization problems can be a big challenge due to their nature. In combinatorial problems, the number of possible solutions can grow exponentially based on the size of the given problem. For instance, if we assume that a specific TSP has n cities that must be passed through, the number of possible routes to go through all those cities is $(n-1)!/2$. Thus, as the size of the problem grows, solving it using traditional optimization techniques can become computationally more expensive or even impossible in some cases. For more details regarding combinatorial optimization problems, readers are advised to refer to [16].

2.5 Time Complexity

As discussed earlier, combinatorial optimization problems can be computationally complex to be solved, especially when these problems grow in size. More specifically, time complexity represents the computational time needed by an algorithm based on the size of its input. In optimization, specifically combinatorial optimization, it is vital to understand the concept of time complexity as it allows us to decide if solving a problem can be done efficiently. This section focuses on the time complexity of optimization problems. For more details regarding time complexity, readers are advised to refer to [16].

When an algorithm's computational time grows at a rate similar to a polynomial function with an n size input, we say that this algorithm runs in polynomial time. More formally, we can say that an algorithm's runtime $T(n)$ is bounded by the big O, $O(n^k)$, where k is a polynomial constant. This can be expressed as:

$$T(n) = O(n^k)$$

Generally, optimization problems can be classified into four main classes in terms of their time complexity:

1. Class P
2. Class NP
3. Class NP-Hard
4. Class NP-Complete

A problem is classified as P when it is possible to solve it in polynomial time using deterministic algorithms. P problems tend to be efficient to solve. This is because as the problem size grows, the time to solve these problems remains reasonable. Class NP problems represent nondeterministic polynomial-time problems. Solving NP problems may not always be efficient, but once the solution is found, verifying said solution can be achieved in polynomial time. As for NP-hard problems, they are considered to be at least as hard as the most difficult NP problem. If all the problem instances of an NP-hard problem are solved optimally, it means that all NP problems can be solved too. This would mean that $NP = P$. However, not all NP-hard problems also belong to the NP class. Because NP-hard problems normally are not very easy to solve, especially for large-scale problems, techniques such as approximation algorithms, heuristics, and metaheuristics are occasionally adopted. Despite the fact that these techniques do not guarantee an optimal solution, they can solve such complex problems in a reasonable time. In this thesis, we implemented a different approach to deal with the nature of our combinatorial problem, OCP formulated as SCP, which is also classified as an NP-hard problem. More on that will be discussed in the next few chapters. The last time complexity class to discuss is NP-complete. NP-complete is a special case of the class NP-hard, that also belongs to the class NP. The main difference between NP-hard and NP-complete problems is that the former is at least as hard as the most difficult NP problem, but it does not have to be in NP; whereas the latter exists in both NP and NP-hard. This means that it may not be possible to verify the solution of NP-hard problems in polynomial time, whereas it is possible to verify the solution of NP-complete problems in polynomial time.

Figure 2.1 summarizes the connection between the four classes of optimization problems in terms of time complexity. The blue circle represents NP-hard problems, whereas the red circle represents NP problems. We can see that the two circles intersect, because

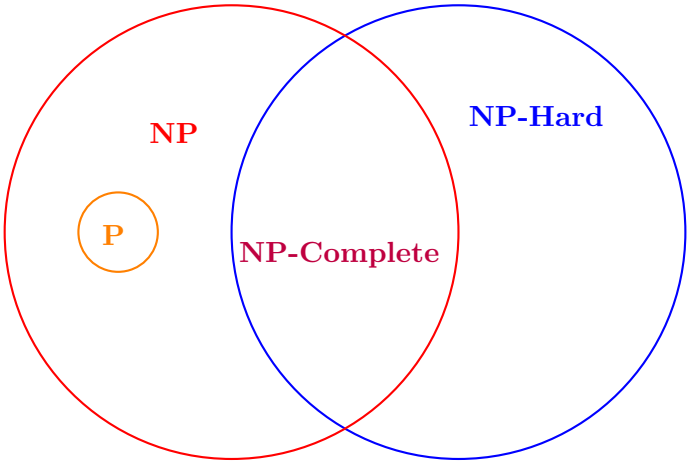

FIGURE 2.1: A visual illustration of the relationship between the four classes of optimization problems based on time complexity [16]

NP-hard problems may or may not be also NP. NP-complete problems are represented by the intersection of the blue and the red circles, because an NP-complete problem is both NP-hard and NP. Finally, class P is represented by the orange circle. It is placed inside the red circle, because P is part of NP.

2.6 Set Covering Problem

As emphasized in a few occasions earlier, we formulate our OCP problem as a set covering problem (SCP) and exploit the relationship between the two problems by exploring techniques from the SCP literature and apply them to solve the OCP problem. SCP is a classic combinatorial optimization problem, and is classified as an $\mathcal{NP}$-hard problem [10]. This means that solving SCP optimally may not be efficient in terms of time, especially when the size of the problem gets larger. Moreover, solving SCP using exact methods becomes more challenging when $n > 1000$, as the number of possible solutions grows exponentially [16]. Given the size of some of our problem instances, solving our OCP problems using only exact methods is expected to be time consuming. More on that is discussed in Chapter 4.

Formally, SCP can be defined as follows: firstly, a universal set U, where $U = \{1, 2, \ldots, n\}$, is given. Secondly, a group of subsets S, where $S = \{S_1, S_2, \ldots, S_m\}$, is also given. Each S_i from S is a subset of U, and a cost c_i is assigned to each S_i. Given that U is covered by the union of the selected subsets of S ($\bigcup_{S_j \in C} S_j = U$), the goal of SCP would be to find a subset with minimum cost ($C \subseteq S$): minimize $\sum_{S_i \in C} c_i$.

SCP is a widely studied combinatorial optimization problem and is involved in different applications, such as crew scheduling ([17]; [18]; [19]), facility location ([20]; [21];[22]; [23]), network design ([24]; [25]; [26]; [27]), and bioinformatics ([28]; [29]; [30]). Table 2.1 lists those studies with the year of publication of each study. It can be noted that SCP has had various applications throughout the years; however, OCP has not been explicitly studied in SCP until 2019 [9]. This shows that OCP application in SCP is still a new research area and there could be numerous possibilities on how to address it.

TABLE 2.1: Key studies for popular SCP application

Application	**Key References and Year of Publication**
Crew Scheduling	(1973 [17]; 1997 [18]; 1997 [19])
Facility Location	(1974 [20]; 1976 [21]; 2011 [22]; 2013 [23])
Network Design	(2005 [24]; 2010 [25]; 2015 [26]; 2022 [27])
Bioinformatics	(2014 [28]; 2014 [29]; 2020 [30])

In the past few decades, various techniques have been implemented to address SCP, including exact methods, heuristics, and metaheuristics. For finding optimal solutions, many SCP studies adopted exact methods, despite the fact that SCP is $\mathcal{NP}$-hard and and could be difficult to solve optimally. Some of the techniques implemented to help solve SCP using exact methods include branch-and-bound ([31]; [32]) and column generation ([33]; [34]). Heuristic and metaheuristic techniques have also been implemented to tackle SCP, but are used mainly to address the $\mathcal{NP}$-hard nature of the problem by finding near-optimal solution in a reasonable time period. Some of the heuristic techniques include greedy algorithms ([35]; [36]) and local search algorithms ([37]; [38]); while some of the metaheuristic techniques include genetic algorithms ([39]; [40]) and tabu search ([41]; [42]). An elaboration on some of the key methods from the SCP literature is provided in the next few chapters.

2.6.1 SCP Single-Objective Formulation

In this subsection, we formulate our single-objective OCP problem as an SCP. Given a 0-1 matrix A with m rows and n columns, where element a_{ij} of the matrix A is equal to 1 when row i can be covered by column j (a_{ij} is equal to 0 otherwise), and a specific cost c_j is assigned to each column j, then the SCP model is be expressed as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad \forall i \in \{1, \ldots, m\} \\ & x_j \in \{0, 1\}, \qquad j \in \{1, \ldots, n\} \end{aligned} \tag{2.1}$$

Where x_j is a decision variable and is equal to 1 when column j is chosen; otherwise, x_j is equal to 0. The goal here is to minimize the total column cost, subject to the constraint that ensures that each row i is covered by at least one column j. From an OCP point of view, rows are the samples and columns are the candidates that can cover those samples. The goal of this OCP problem would be to minimize the total candidate cost, subject to the constraint that ensures that each sample i is covered by at least one candidate j.

In this study, our specific instances do not consider camera cost as a factor. As a matter of fact, this specific OCP problem can be formulated as a special variant of SCP, called the Unicost SCP (USCP), where columns are not associated with any cost. Hence, the goal of our specific OCP problems would be to find a subset of candidates that minimizes the *number* of candidates used, such that each sample must be covered by at least one candidate. The only difference between SCP and USCP models is that we do not include the cost parameter c_j in the latter, as cost is assumed to be unified, which means it is not a factor in our problem. The formulation of USCP is expressed as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad \forall i \in \{1, \ldots, m\} \\ & x_j \in \{0, 1\}, \qquad j \in \{1, \ldots, n\} \end{aligned} \tag{2.2}$$

The goal of this USCP is to minimize the number of columns, subject to the constraint that ensures that each row i is covered by at least one column j. From an OCP point of view, the goal would be to minimize the total number of candidates, subject to the

constraint that ensures that each sample i is covered by at least one candidate j. The single-objective OCP problem is addressed in Chapter 3 and Chapter 4.

2.6.2 SCP Bi-Objective Formulation

$$
\begin{aligned}
&\text{minimize} && \sum_{j=1}^{n} x_j \\
&\text{maximize} && \sum_{i=1}^{m} y_i \\
&\text{subject to} && \sum_{j=1}^{n} a_{ij} x_j \geq y_i, \quad \forall i \in \{1, \ldots, m\} \\
& && x_j \in \{0, 1\}, \qquad j \in \{1, \ldots, n\} \\
& && y_i \in \{0, 1\}, \qquad i \in \{1, \ldots, m\}
\end{aligned}
\tag{2.3}
$$

After formulating our single objective OCP problem as USCP, now we can transform this problem into a bi-objective OCP problem. OCP can naturally be considered as a bi-objective problem with two conflicting objectives: minimize candidates and maximize samples coverage. Since OCP can be formulated as an SCP, bi-objective OCP can also be formulated as a bi-objective USCP. Given a 0-1 matrix A with m rows and n columns, and two binary decision variables named x_j and y_i, the bi-objective USCP is expressed in model 2.3 , where x_j is the columns decision variables and is equal to 1 if column j is selected; otherwise, x_j is equal to 0. As for y_i, it represents the rows decision variable. This means if row i is covered by at least one column j, y_i would be equal to 1; Otherwise, y_i would be equal to 0. The goal of this bi-objective USCP problem is to, simultaneously, minimize the number of columns and maximize the number of rows, such that if row i is selected, all the relevant columns are also selected. From an OCP point of view, the goal of this bi-objective OCP problem is to, simultaneously, minimize the number of candidates and maximize the number of samples, such that if sample i is selected, all the relevant candidates are also selected. The bi-objective OCP problem is addressed in Chapter 5.

Chapter 3

Exploring the Optimal Camera Placement Problem and its Relationship with the Set Covering Problem

Optimal Camera Placement (OCP) is the process of finding a subset of cameras that either maximizes the coverage, such that the cost of cameras is reduced, or minimizes the total cost of cameras, such that coverage constraints are satisfied. By adopting the latter formulation, the OCP problem can be formulated as a Set Covering Problem (SCP), as the concepts of the two problems are inherently similar. Until recently, the literature has not explicitly discussed this similarity. Hence, this paper examines the OCP problem by leveraging the formulation established in prior research. Our focus lies in the practical application, as we implement the model on all instances to derive meaningful insights. Furthermore, we explore techniques from the SCP literature that can be applied to address the OCP problem in future studies. In this study, we address 69 problem instances, utilising a benchmark set generated by other researchers. These instances were employed as part of the GECCO 2021 competition on the optimal camera placement problem and the unicost set covering problem. We provide detailed results, and we conclude with recommendations for future research.

3.1 Introduction

In recent years, establishing an optimal camera network for surveillance purposes has been the subject of interest in several studies. This increased attention is prompted by the worldwide spread of surveillance systems, which are being employed to address various issues, including analysing crowd movements, monitoring transportation systems, or simply observing certain places for general purposes [9]. The idea of camera placement was first discussed in computational geometry in the 1970s by Chvátal [1]. The author's widely known Art Gallery Problem (AGP) has inspired numerous camera placement studies since its introduction. AGP is an approach for placing guards in an art gallery, where the goal is to minimize the number of guards, ensuring that every point in the art gallery is covered by at least one guard. Transforming this idea to camera placement, guards become cameras, and the general goal is to select the smallest subset of cameras that achieves full coverage.

Generally and more formally, OCP is the process of finding a subset of cameras that either maximizes the coverage, such that the cost (or number) of cameras is reduced; or minimizes the total cost (or number) of cameras, such that coverage constraints are satisfied. When working with the minimization objective, the problem can be viewed as a set covering problem. However, the similarity between the two problems has not been explicitly discussed in the OCP literature until recently [9]. Therefore, the aim of our study is to explore the connection between the two problems and discuss techniques from the SCP literature that can be applied to the OCP problem.

The rest of the paper is structured as follows: Section 3.2 provides a brief summary of the OCP literature, discussing different techniques that have been used to deal with the problem. This is followed by a description of SCP in Section 3.3. Section 3.4 provides a detailed problem description and formulations of our OCP problem. Then, a summary of the results of the problem is given in Section 3.5. Finally, Section 3.6 concludes our study and gives a hint of what can be done in future work.

3.2 OCP Literature

Inspired by the AGP, the use of optimal camera networks for surveillance has increased in the past few decades in order to fully monitor different areas, including public places,

warehouses, buildings, and so on. The general goal is to maximize coverage or to minimize the cost (or number) of cameras, given a set of constraints [8]. When it comes to solving OCP, researchers have employed various tools. Some used exact methods to deal with the problem and find optimal solutions, while others used heuristic methods to find near-optimal solutions within a reasonable time. To elaborate on the latter, since the OCP is an $\mathcal{NP}$-hard problem [8, 9], finding optimal solutions can be time-consuming for sufficiently large instances. Moreover, if a client's priority is time rather than solution quality, then it might be sensible to use heuristic methods instead of exact methods to find a satisfactory solution in a reasonable amount of time. Table 3.1 provides a summary of some of the key studies that will be discussed in this thesis. This includes, the objectives used, constraints added, and methods applied to solve every unique OCP problem.

Table 3.1: A summary of some of the key OCP studies

Study	**Objective(s)**	**Constraint(s)**	**Methods used**
Brévilliers et al. (2018) [43]	Minimize number of cameras	Full coverage	Set-based Differential Evolution
Kritter et al. (2019) [44]	Minimize number of cameras	Full coverage	Row-Weighting Local Search
Jun et al. (2018) [45]	Minimize cost of cameras	Minimum coverage level	Greedy, Genetic Algorithms and ULA
Yang (2018) [46]	Minimize cost of cameras	Full coverage	Branch-and-bound
Yang (2018) [46]	Maximize coverage	Budget constraint	Dynamic programming
Puligandla and Lončarić (2022) [47]	Maximize coverage	Predefined number of cameras	Clustering-based optimization

A study by [45] employed three heuristic algorithms to tackle an OCP. The authors focused on finding the best algorithm capable of solving OCP problems for the surveillance of bridges. Their model aimed to minimize the total cost of cameras, while ensuring that a minimum coverage level is satisfied. Moreover, this study included different types of cameras to cover specific target points in a three-dimensional space, resulting, for example, in an increase in the number of camera locations, which can lead to the expansion of the size of the problem instances. Because of that, the authors started by examining their OCP instance using two popular heuristic methods, greedy and genetic algorithms, as well as a novel heuristic method that is called the Uniqueness Score with Local Search Algorithm (ULA). For the first method, the greedy algorithm allocates cameras starting from the cheapest one and going up until it reaches the highest possible number of cameras or achieves the minimum coverage level. At this point, the algorithm halts and provides the proposed camera network. For the second method, the genetic algorithms begins by randomly generating a population of chromosomes, where each chromosome represents a camera placement. This process continues until the minimum coverage level is reached by each chromosome. Subsequently, a subset of chromosomes that satisfy the minimum coverage is selected. Mutation and crossover operators are then applied to this subset, creating new generations and ensuring that the solutions do not become

stuck in a local optimum. As demonstrated in [45], the last method, proven to be more effective than the other two approaches, begins with the first solution of ULA, inspired by the uniqueness score. It emphasises specific areas that are left uncovered by other cameras. Subsequently, a local search is used to enhance the solution by changing the selected cameras, aiming to find a new solution with a lower cost.

Another study in [46] also examined three different approaches for three OCP cases, one of which utilized an exact method, while the other two employed heuristic methods. In the first case, the objective was to maximize the coverage within a limited budget. The authors formulated the optimization problem as a set covering problem and then applied dynamic programming to address it. For the second case, the goal was to minimize the total cost while ensuring full coverage constraints. Similar to the first case, the optimization problem was formulated as an SCP, and branch-and-bound algorithms were employed. This involved relaxing the binary constraints, solving the new optimization problem using primal and dual simplex methods, and re-introducing the binary constraints through branching. This process is repeated recursively until a feasible (binary) solution is obtained. Lastly, the third case integrated the previous two cases to form a multi-objective problem, aiming to both maximize coverage and minimize cost. Instead of looking for one optimal solution, the goal was to find a set of optimal trade-offs (Pareto optimal solutions). In this context, 'Pareto optimal' refers to a set of selected candidate cameras, where no feasible candidate cameras could improve coverage without worsening the cost simultaneously. To address this problem, the authors suggested using a heuristic method, specifically the multi-objective genetic algorithm NSGA-II [48].

Another method that was used to address an OCP problem is Differential Evolution (DE). In [49], this heuristic method was utilized to improve the performance of greedy algorithms in order to achieve full coverage. Their DE starts by using an array containing a number of cameras, where each array represents an individual within the population. The algorithm then uses the vector differential of two individuals from the previous generation to create a new individual. Consequently, the algorithm continues to generate improved individuals compared to those in the previous generations.

The work in [43] used a variant of DE, called set-based DE, inspired by the study in [50], to address their OCP problem. In contrast to the study in [49], the authors focused on minimizing the number of cameras (i.e., cost reduction) while ensuring complete coverage. The difference between set-based DE and the original DE lies in the fact that the former is primarily utilized to solve permutation-based problems, whereas the latter can be applied to solve general problems (including set-based problems).

Other methods employed in recent literature include dynamic algorithms [51], simulated annealing [5], greedy algorithms [52] and hill climbing [53]. While some OCP studies have formulated their problem as an SCP, the study in [44] argues that almost none of these studies have fully exploited this similarity or employed techniques from the SCP literature to address the OCP problem. As a result, many techniques employed in the SCP literature have not yet been utilized to address the OCP problem. This suggests potential opportunities for contributions that can make a difference in the OCP field.

3.3 Set Covering Problem (SCP)

The Set Covering Problem (SCP) is a popular combinatorial optimization problem classified as $\mathcal{NP}$-hard [10]. Throughout the years, numerous studies have explored the SCP to tackle a diverse range of real-world applications. These include solving transit crew scheduling problems [54], optimising transit crew scheduling design with multiple objectives [26], finding optimal quantity and location of gas detectors [55], assigning fire stations with ladder trucks [56], and scheduling wireless sensor networks [57], among others. For more information regarding the SCP and its applications, readers are advised to refer to [58].

3.3.1 Introduction to SCP

A brief definition of SCP would be: given a zero-one matrix, the goal is to obtain a subset of columns that minimizes the total cost associated with the selected columns, ensuring that all the rows of the matrix are covered by these columns [59]. To elaborate on that, consider matrix A with three rows and three columns:

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

In this matrix, the value 1 indicates that a given row is covered by a given column, and 0 otherwise. For example, element a_{11} shows that row 1 is covered by column 1. While element a_{33} shows that row 3 is not covered by column 3. If each column is associated with a specific cost, the objective of SCP would be to find a subset of columns that minimizes the total cost while ensuring that all the rows of the matrix are covered by

this subset. For example, if the cost is the same for all columns, an optimal solution could be selecting both columns 1 and 3, as this combination covers all given rows.

Formally, given a finite universal set $U = \{1, 2, \ldots, n\}$ and a family of subsets $S = \{S_1, S_2, \ldots, S_m\}$ where each S_i is a subset of U, and each subset S_i has an associated cost c_i, the set covering problem is to find a minimum-cost subset $C \subseteq S$ such that the union of the selected subsets covers the entire universal set, i.e., $\bigcup_{S_j \in C} S_j = U$. The goal is to minimize $\sum_{S_i \in C} c_i$, the total cost of the selected subsets.

3.3.2 The Connection between SCP and OCP

From an optimization perspective, the study in [9] asserts that the Optimal Camera Placement (OCP) can be reformulated as a set covering problem after certain pre-processing steps. In this context, the pre-processing phase involves the transformation of the OCP problem into a visibility matrix. This matrix matches each location in the surveillance area with every possible configuration (position and orientation) of the given cameras, resulting in a 0-1 matrix similar to matrix A. To elaborate, columns in this matrix represent the cameras, and rows represent the locations that need to be covered by the cameras. The objective of the transformed OCP problem is to find a subset of cameras that minimizes the cost, ensuring that all the specified locations are covered by this subset. In essence, this conversion enables the application of SCP methodologies to address the OCP challenge effectively.

In the study conducted in [9], an in-depth illustration is provided regarding the possibility of using methods from the SCP literature and applying them to OCP problems. The study introduces different heuristic and metaheuristic methods that were employed to address SCP, and consequently, these methods could be adapted for solving OCP problems. For instance, greedy algorithm is one of the heuristic methods utilized in the SCP literature to address the problem's $\mathcal{NP}$-hard nature. As will be pointed out later, the greedy algorithm was also used in the OCP literature in recent years. Despite being tailored specifically for OCP problems, the main conceptual framework remains the same for both domains. Another approach discussed in this study involves the work of [38], who utilized the Row-Weighting Local Search (RWLS) algorithm for a special type of SCP known as the Unicost SCP. According to [9], this approach has not been explored in the OCP literature, and questions arise regarding its potential efficiency in solving OCP problems. In a subsequent study [44], different approaches from both OCP and SCP literature were employed on real OCP cases. Notably, they utilized the RWLS algorithm

and demonstrated its efficacy in solving OCP problems. This illustrates the possibility of leveraging techniques from the SCP literature to effectively address challenges in the OCP domain.

3.4 Problem Description and Formulation

In this study, we address 69 three-dimensional problem instances, comprising 32 academic problem instances and 37 real-world instances. These instances were utilized as part of the GECCO 2021 competition on the optimal camera placement problem and the unicost set covering problem [11]. The academic problem instances represent different sizes of rectangularly modelled rooms, where ceilings are used for camera placement. On the other hand, the real-world instances represent diverse sizes of actual urban spaces, with the walls of multiple buildings serving as potential camera locations.

For each problem instance, a set of camera configurations (locations and orientations coordinates), referred to as *candidates*, is provided. Additionally, a grid containing a set of points in three-dimensional space, denoted as *samples*, must be covered by a subset of candidates. For each sample, there exists multiple candidates capable of overseeing it.

Consider the example with two samples (a and b) and four candidates (1, 2, 3, and 4). Suppose sample a can be covered by candidates 1, 2, and 4, while sample b can be covered by candidates 2 and 3. We can create a visibility matrix A for this example, where the first row represents sample a, the second row represents sample b, and columns represent candidates 1, 2, 3, and 4. Matrix A is presented below.

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

The objective of the OCP problem is to cover all the given samples by identifying a subset of candidates that achieves this goal at the minimum cost. In the presented OCP example, assuming equal costs for each candidate, the optimal solution would be to select candidate 2. This choice is optimal because candidate 2 is the only one overseeing both samples, enabling the coverage of both samples with a single candidate. Other solutions, such as candidates 1 and 3 or candidates 3 and 4, would require paying for

two candidates to cover the two samples. However, the optimal solution, in this case, is achieved by selecting just one candidate (i.e., candidate 2).

As previously mentioned, the OCP problem shares similarities with the set covering problem. Consequently, we model the OCP problem as an SCP. The formulation involves a binary matrix A ($a_{ij} = 1$ if row i can be covered by column j, and 0 otherwise) with m rows and n columns, where each column j is assigned a specific cost c_j. The mathematical model is expressed as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad \forall i \in \{1, \ldots, m\} \\ & x_j \in \{0, 1\}, \qquad j \in \{1, \ldots, n\} \end{aligned} \tag{3.1}$$

Here, $x_j = 1$ if column j is selected, and $x_j = 0$ otherwise. The objective function minimizes the total cost, and the first constraints ensure that each row i is covered by at least one selected column j.

Now, for our specific OCP problem, all cameras have the same cost [11]. This simplification transform our SCP model into a Unicost SCP (USCP), where the cost parameter c_j is omitted:

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad \forall i \in \{1, \ldots, m\} \\ & x_j \in \{0, 1\}, \qquad j \in \{1, \ldots, n\} \end{aligned} \tag{3.2}$$

In this formulation, $x_j = 1$ if candidate j is included in the solution, and $x_j = 0$ otherwise. The objective remains to minimize the total number of selected candidates, ensuring that each sample is covered by at least one candidate. This transition to a USCP model simplifies the objective by focusing on minimizing the number of candidates, each having an equal cost, while maintaining the essential constraints for effective camera placement in the OCP context.

TABLE 3.2: Results for academic problem instances. m is the number of samples, n is the number of candidates, UB is the upper bound, LB is the lower bound, Gap is the optimality gap, and Time is the time taken in seconds

Instance	m	n	UB	LB	Gap	Time
AC_01	605	2904	7.00	7.00	0.00%	9.33
AC_02	2205	10584	4.00	4.00	0.00%	140.30
AC_03	4805	23064	3.00	3.00	0.00%	1411.43
AC_04	8405	40344	5.00	5.00	0.00%	7542.09
AC_05	13005	62424	-	-	-	-
AC_06	18605	89304	-	-	-	-
AC_07	32805	157464	-	-	-	-
AC_08	51005	244824	-	-	-	-
AC_09	73205	351384	-	-	-	-
AC_10	605	2904	20.00	17.37	10.00%	10800.00
AC_11	2205	10584	72.00	52.00	26.39%	10800.00
AC_12	4805	23064	168.00	109.81	34.52%	10800.00
AC_13	8405	40344	344.00	187.18	45.35%	10800.00
AC_14	13005	62424	723.00	0.00	100.00%	10800.00
AC_15	18605	89304	-	-	-	-
AC_16	32805	157464	-	-	-	-
AC_17	51005	244824	-	-	-	-
AC_18	73205	351384	-	-	-	-
AC_19	99405	477144	-	-	-	-
AC_20	129605	622104	-	-	-	-
AC_21	163805	786264	-	-	-	-
AC_22	202005	969624	-	-	-	-
AC_23	244205	1172184	-	-	-	-
AC_24	290405	1393944	-	-	-	-
AC_25	340605	1634904	-	-	-	-
AC_26	394805	1895064	-	-	-	-
AC_27	453005	2174424	-	-	-	-
AC_28	515205	2472984	-	-	-	-
AC_29	581405	2790744	-	-	-	-
AC_30	651605	3127704	-	-	-	-
AC_31	725805	3483864	-	-	-	-
AC_32	804005	3859224	-	-	-	-

3.5 Computational Results

For this study, we utilized the PuLP[1] package in Python to build the model, which was then solved using Gurobi, a commercial Linear Programming solver. Experiments were conducted on an Intel(R) Core(TM) i5-8500T CPU @ 2.10GHz 2.11GHz with 8.00GB RAM. A time limit of 3 hours was imposed on each problem instance, encompassing both model building and problem solving. The results for both the academic and real-world problem instances are presented in Table 3.2 and Table 3.3, respectively. For instances

[1]https://pypi.org/project/PuLP/

TABLE 3.3: Results for real-world problem instances. m is the number of samples, n is the number of candidates, UB is the upper bound, LB is the lower bound, Gap is the optimality gap, and Time is the time taken in seconds

Instance	m	n	UB	LB	Gap	Time
RW_01	153368	32430	-	-	-	-
RW_02	285698	56132	-	-	-	-
RW_03	161099	32040	-	-	-	-
RW_04	304655	59137	-	-	-	-
RW_05	206900	34568	-	-	-	-
RW_06	380420	65691	-	-	-	-
RW_07	214889	42046	-	-	-	-
RW_08	382651	77986	-	-	-	-
RW_09	206816	39003	-	-	-	-
RW_10	368114	71323	-	-	-	-
RW_11	82437	15632	316.00	309.88	1.90%	10800.00
RW_12	136555	28109	-	-	-	-
RW_13	293138	61741	-	-	-	-
RW_14	81062	14916	337.00	334.77	0.59%	10800.00
RW_15	141309	27008	-	-	-	-
RW_16	105829	21063	-	-	-	-
RW_17	180453	35635	-	-	-	-
RW_18	79947	14423	338.00	336.65	0.30%	10800.00
RW_19	141114	26483	-	-	-	-
RW_20	332300	50284	-	-	-	-
RW_21	654068	90050	-	-	-	-
RW_22	83835	17203	399.00	392.93	1.50%	10800.00
RW_23	142326	31038	-	-	-	-
RW_24	201967	33880	-	-	-	-
RW_25	375680	59851	-	-	-	-
RW_26	105566	18043	-	-	-	-
RW_27	181090	32669	-	-	-	-
RW_28	136755	27838	-	-	-	-
RW_29	273964	49267	-	-	-	-
RW_30	263518	49354	-	-	-	-
RW_31	472660	87248	-	-	-	-
RW_32	124289	30189	-	-	-	-
RW_33	229231	55000	-	-	-	-
RW_34	134479	27329	-	-	-	-
RW_35	238546	47590	-	-	-	-
RW_36	135043	28162	-	-	-	-
RW_37	238492	50702	-	-	-	-

where obtaining the optimal solution was unattainable, we provide the upper and lower bounds along with the calculated optimality gap using the formula:

$$\text{Gap} = \frac{|UB - LB|}{|UB|}$$

where UB is the upper bound, and LB is the lower bound. In cases where the optimal solution was found, both the upper and lower bounds will have the same value. Instances where no results were produced due to insufficient memory or time limitations are denoted by a dash '-' in the respective table entries.

As seen in both tables, there are many cells filled with a dash '-', mostly because the three-hour time limit was reached before completing the model building stage, rendering it impossible to solve the problem. This issue arises mainly due to the substantial size of these problem instances, requiring hours or even days solely for model building. Another observation is that, even when the model is built for some instances, an optimal solution is not always achieved. In such cases, having the upper and lower bounds is particularly valuable, offering useful information about how close we are to reaching the optimal solution. For some instances, like RW_18, it appears that we are quiet close to the optimal solution. For these, extending the runtime may yield the optimal solutions. However, there are instances, such as AC_13, where the gap is notably large, indicating the need for an extended runtime to solve them.

If we take a look at the first problem instance in Table 3.2, AC_01 contains 605 samples and 2904 candidates. The optimal solution for this problem instance involved using 7 candidates to cover all 605 samples, and the computation took 9.33 seconds. If we compare this with another academic instance, such as AC_02, we can observe that it takes less time to solve AC_01 than AC_02. This is because the size of the latter is bigger than that of the former. In other words, AC_02 contains more samples as well as candidates than AC_01. This is similar when comparing AC_01 with other instances, such as AC_03, AC_04, AC_05. The bigger the size of the instance, the more time is needed to solve the given instance. However, if we take a look at AC_10, it contains the exact same amount of candidates and samples as AC_01, but the former could not obtain any results within the time-limit. The reason behind this is that even when the number of samples and candidates is similar, the structure of the problem might be different. In the case of AC_10, it consists of a more complicated structure than AC_01, which could not be solved during the given time-limit, unlike in the case of AC_01. Figure 3.1 visualizes this problem instance. The figure represents a rectangular room, where the yellow points on top of the "room" represent all candidates and the blue points represent all the 3-d samples that must be covered by a subset of those candidates. The solution of AC_01 is also depicted in Figure 3.1, where 7 yellow points on the graph represent the optimal locations of the candidates that cover all 605 samples.

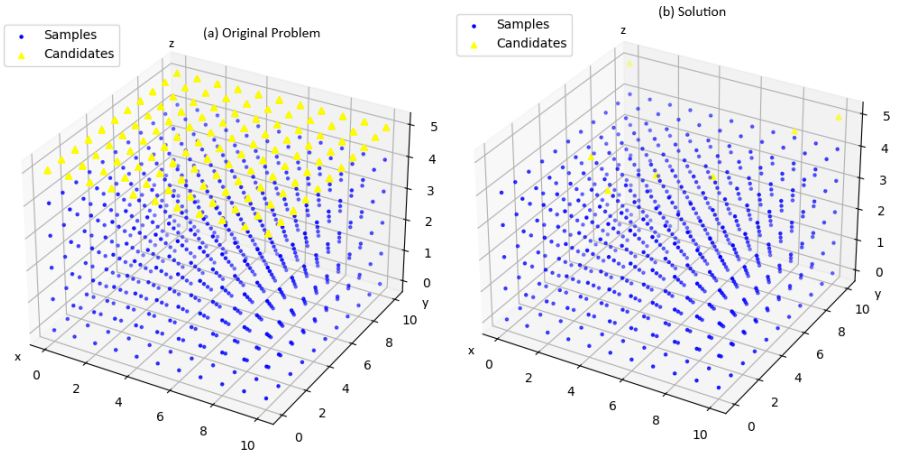

FIGURE 3.1: Visualisation of the AC 01 problem instance (a) and its optimal solution (b), where yellow points represent candidates and blue points represent samples to be covered

Similar to AC_01, we visualise another problem instance in Figure 3.2, namely the problem RW_22. This real-world instance contains 17,203 candidates and 83,835 samples. A feasible solution was obtained with an upper bound of 399, a lower bound of 392.93, and a gap of 1.5%. The visual representation of this problem's solution can be observed in the same figure.

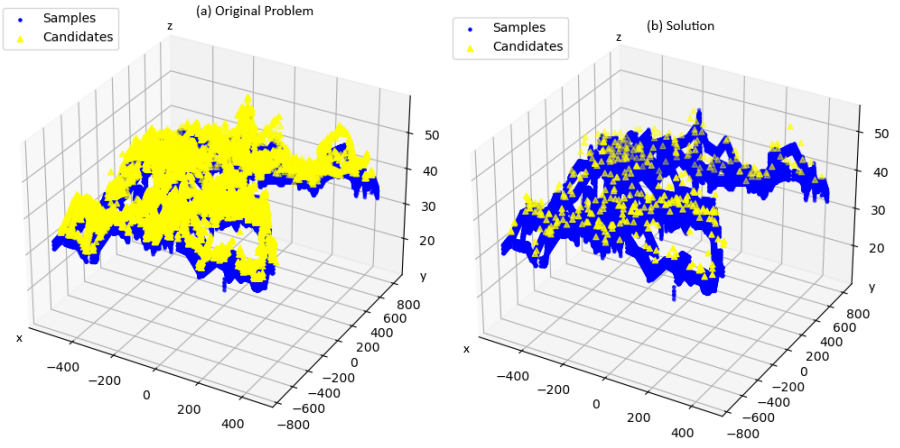

FIGURE 3.2: Visualisation of the RW_22 problem instance (a) and its feasible solution (b), where yellow points represent candidates and blue points represent samples to be covered

Addressing the time issue is paramount for achieving improved results. One approach involves tackling the size of the problem; for instance, reducing it by eliminating unnecessary sets from the visibility matrix. Additionally, exploring different optimization

methods could be beneficial, such as employing multi-objective optimization. In this approach, the main objectives would involve maximizing area coverage while minimizing the number of cameras. Another viable approach is the utilisation of heuristic techniques. By studying various heuristic methods employed in the SCP literature and applying them to our OCP problem instances, we can potentially obtain near-optimal results within a reasonable amount of time.

3.6 Conclusions

OCP problem involves determining the optimal locations and orientations for a set of cameras, with the objective either being to maximize the coverage of a given surveillance area or to minimize the total cost or number of selected cameras. Although this problem has been formulated in a manner resembling the SCP, this similarity has only recently been explicitly discussed in the literature.

This chapter studied the OCP problem by exploring its literature and understanding its relationship with SCP. Given that OCP is an $\mathcal{NP}$-hard problem, and the majority of problem instances are large, relying solely on exact methods did not provide solutions for all instances. Therefore, addressing the $\mathcal{NP}$-hard nature of the problem becomes crucial for future work. Potential strategies include reducing the size of problem instances, and/or resorting to heuristic techniques.

Chapter 4

Solving the Optimal Camera Placement Problem with the Help of Effective Problem Reduction Techniques

This paper explores different reduction techniques to address the size of 69 Optimal Camera Placement (OCP) problem instances, which were obtained from the GECCO 2021 competition on the optimal camera placement problem and the unicost set covering problem. Given a number of candidate cameras is provided to monitor a specific area for surveillance purposes, OCP aims to maximize area coverage, such that camera cost is reduced, or minimize camera cost, such that coverage requirements are met. OCP can be formulated as a Set Covering Problem (SCP) when using the latter formulation. The reason behind this is that the basic idea of the two problems would be quite similar, when minimizing the camera cost in OCP. Thus, techniques from the SCP literature can be used to address the OCP problem. Despite this similarity, SCP techniques have not been utilized to address the OCP problem until recently. Therefore, this paper exploits the relationship between SCP and OCP by studying and improving upon some SCP reduction techniques and applying them to address OCP problem instances. The reduction results as well as the impact of the reduction algorithms on the OCP solutions are provided.

4.1 Introduction

Nowadays, surveillance systems are being used for various reasons, including but not limited to transportation systems tracking, crowd movement analysis, or general monitoring purposes [9]. This has led to an increase in interest in creating an optimal camera network. Generally, OCP aims to maximize area coverage, such that camera cost is reduced, or minimize camera cost, such that coverage requirements are achieved. Using the latter formulation can transform OCP into the popular combinatorial optimization problem, namely, the Set Covering Problem or SCP. The basic idea of the two problems would be similar when using the minimization objective. Thus, OCP can be formulated as an SCP. Nonetheless, many OCP studies have not exploited this similarity by using techniques from the SCP literature to address different OCP problems. Hence, this study explores some SCP techniques that can be used to address our OCP problem instances. Moreover, the focus of this paper is using different reduction methods that can be found in the SCP literature and apply them to our OCP instances. These methods can be very useful in dealing with the $\mathcal{NP}$-hard nature of our problem instances. To expand on that, solving the OCP problem can take a significant period of time as the problem instances get bigger in size. The aim of this paper is to prove that reducing the sizes of these problem instances would reduce the time needed to solve our OCP problem instances.

The rest of the paper is organized as follows. Section 4.2 thoroughly discusses OCP then its relationship with SCP. Section 4.3, then, describes the given problem instances in depth. Next, Section 4.4 introduces a few reduction techniques from the SCP literature, and explains how these techniques would work on our OCP instances. After that, Section 4.5 provides an extensive discussion on the reduction as well as the optimization results, including the impact of the former on the latter. Lastly, Section 4.6 concludes this paper by summarizing the main points, before suggesting a few recommendations for future work.

4.2 Optimal Camera Placement (OCP)

This section is divided into two parts. Firstly, we provide a brief background on the OCP problem. Secondly, we explore the OCP literature, by discussing various techniques used to address the OCP problem.

4.2.1 OCP Background

OCP is an $\mathcal{NP}$-hard optimization problem [9] that originates from the field of computational geometry from the 70s of the 20^{th} century. The idea of OCP was first introduced as an Art Gallery Problem (AGP) [1]. AGP is a technique for finding an optimal placement for guards in an art gallery. Given that each location in the gallery must be covered by at least one guard, the objective of AGP would be to minimize the total number of guards. From an OCP point of view, guards can be replaced by cameras, and the objective would be to minimize the number (or cost) of cameras, such that every area in a given location is covered by at least one camera.

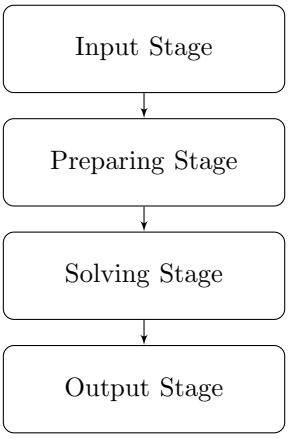

FIGURE 4.1: The main stages of camera placement [8]

As Figure 4.1 suggests, OCP goes through four different stages. The first stage is called the Input Stage, and it concerns identifying the inputs of the OCP problem. This includes the types of cameras available and the given surveillance area. In addition, some requirements can be set, such as the resolution quality of the cameras or the proportion of the surveillance area that must be monitored by said cameras. Other input information might be needed if some tasks are given for the camera network, such as object tracking. The second stage is called the preparing stage. It takes place when it is decided to use a discrete domain instead of continuous, where the parameters of the targeted area as well as the camera configurations (locations and orientations) are discretised, and a set of cameras is created. After generating a pool of candidate cameras, a mathematical model of the OCP problem can be formulated and solved using any suitable optimization tools. This is the third stage, and it is called the solving stage. The final stage is the output stage, which is basically the OCP solution that was obtained. This would be the optimal camera layout. For more details regarding the four main stages of OCP, interested readers can refer to [8].

4.2.2 OCP Literature

In the OCP literature, there are numerous studies that have used a variety of techniques to solve the OCP problem ([43]; [60]; [45]; [47]; [61]; [46]; [49]). For finding optimal solutions, exact techniques were implemented. However, to address the $\mathcal{NP}$-hard nature of the problem, some studies have adopted heuristic techniques to obtain near-optimal solutions in a shorter period of time. This subsection explores the different techniques, employed to address various OCP problems.

In a study by [46], three techniques were adopted to tackle three OCP problems. One problem had a budget limit, and the objective was to maximize area coverage. For this specific problem, OCP was, in fact, formulated as an SCP. To solve this problem, dynamic programming was used. Another problem was formulated to minimize camera cost, such that full coverage is achieved. This problem was also formulated as an SCP, but instead of dynamic programming, branch-and-bound algorithms were used. The third problem considered the previous two problems and formulated a bi-objective problem. This means that instead of one objective, the new problem included two objectives: maximizing area covering and minimizing camera cost. Since a bi-objective model was used, the aim would be to obtain a set of efficient solutions, rather than one solution. Each efficient solution in this set represents a combination of coverage and cost values where no other solution offers a better combination value. For obtaining these solutions, multi-objective genetic algorithm NSGA-II [48] was adopted.

In another study by [45], three techniques were also used to address one problem. These techniques are greedy algorithms, genetic algorithms, and a novel heuristic technique, named the Uniqueness score with Local search Algorithm (ULA). The objective of this problem was to minimize the camera cost, such that minimum area coverage level is achieved. For the first technique, a maximum camera limit and a minimum coverage level are set. Then, a greedy algorithm allocates cameras from the least to the most expensive and terminates once one of the two requirements is met. When that is done, a camera placement solution is provided. For the second technique, a genetic algorithm produces a population that contains a number of camera placement plans, where each plan is called a 'chromosome'. Each chromosome should reach a minimum coverage requirement. Once that is done, a subset of these chromosomes is selected, and mutation and crossover operations begin. These operations are used to create new 'generations' of chromosomes and to avoid getting stuck in a local optimum. Inspired by the uniqueness score, the last technique obtains one ULA solution. Then, the process of finding a new

solution that costs less begins. For achieving this, local search is used and the locations that are not covered by other cameras are emphasized.

Differential Evolution (DE) is another technique mentioned in the OCP literature [49]. This specific method was used to enhance the greedy algorithm performance. In this study, DE creates individuals, in the form of arrays, where each individual includes a number of cameras. New and improved individuals are, then, created using vector differential between two individuals. In another study, by [43], DE was also the main technique used to address an OCP problem. This study was inspired by [50] to create a special type of DE, named set-based DE, which can solve general problems, unlike the original DE which can mainly solve permutation-based problems.

There are many other techniques used to address the OCP problem. A clustering-based optimization method was used to tackle their specific OCP problem [47]. The authors utilized a clustering approach to implement branch-and-bound algorithms. The reason behind that is that traditional branch-and-bound can be time consuming when addressing large real-life problems and using a clustering method can significantly reduce the time needed to solve such problems. Last but not least, grey wolf algorithm is yet another optimization technique implemented to deal with an OCP problem [60]. This population based metaheuristic approach, inspired by the behaviour of wolves, was proven to be effective when compared with other algorithms, such as the Particle Swarm Optimization and the Moth Flame Optimization methods. There are still more techniques in the OCP literature that have been studied, including simulated annealing [5], hill climbing [53], greedy algorithms [52], dynamic algorithms [51], and more.

A study, by [9], emphasised on the fact that despite the clear similarity between the two problems, work from the SCP literature has not been applied to address the OCP problem (until recently). As a result, they discuss several techniques from the SCP literature that can be used to solve OCP problems. An example of these techniques is greedy algorithm, which we have already pointed out that it is being used in the OCP literature. Another technique that they discussed from the SCP literature is Row-Weighting Local Search (RWLS) [38], which was successfully utilized to address an OCP problem in a later study [44]. This proves that techniques from the SCP literature can be used to tackle the OCP problem. Thus, we exploit this similarity by implementing some reduction techniques, which were discussed in the SCP literature, and apply them to our OCP problem instances.

As for reduction techniques, there are some OCP papers that have used these methods in their work to improve their results. A paper by [61] proposed a neighbourhood search algorithm, named weighting-based variable neighborhood search (WVNS). This method was used to address the OCP problem instances taken from the GECCO 2020 Competition. This algorithm performs four different reduction methods before transforming the main OCP problem into smaller sub-problems. These sub-problems are, then, addressed using local search techniques. In another study, [62] have also utilised a reduction technique, known as the dominated column reduction method. In their study, this reduction algorithm was applied to a number of instances from the GECCO 2020 Competition, which we are using in this thesis. Their results are discussed later in this chapter, where we compare our reduction results with theirs. After reducing the problem instances, the authors solved them using CPLEX solver, greedy algorithm, and RWLS. After that, the authors proposed two hybrid DE methods to address the OCP instances.

4.3 Problem Description

The problem instances that we used in this study were obtained from the GECCO 2021 competition on the optimal camera placement problem and the unicost set covering problem [11]. These problem instances introduce two main terms: *candidates* and *samples*. a candidate is a set of camera configurations, which includes location and orientation. A sample is a point in a three-dimensional space. Each sample can be covered by a number of candidates. Likewise, each candidate can cover a number of samples. In this paper, we use 69 problem instances. 32 of which are called academic and 37 are called real-world. We start with describing academic instances.

Each academic instance represents a three-dimensional rectangle (rectangular box), where, in a three-dimensional space, the point coordinates in meters start from (0, 0, 0) and reach up to $(X_{max}, Y_{max}, Z_{max})$. Using a grid that consists of points representing the samples, the space is discretized. In addition, a step size (U) between every neighbouring pair of samples is defined. This means that a unified distance is assigned between each two neighbouring sample points. Regarding the cameras, each one can be distinguished by specific features such as the vertical resolution (V_{res}), the horizontal resolution (H_{res}), and the horizontal field of view (H_{fov}).

Regarding candidates, which represent camera configurations, their locations are defined by point coordinates that go from (0, 0, Z_{cam}) and reach up to $(X_{max}, Y_{max}, Z_{cam})$. Note

TABLE 4.1: Academic problem instances specifications

Name	X_{max}	Y_{max}	Z_{max}	Z_{cam}	Samples	Candidates
AC_01	5	5	2	2.5	605	2904
AC_02	10	10	2	2.5	2205	10584
AC_03	15	15	2	2.5	4805	23064
AC_04	20	20	2	2.5	8405	40344
AC_05	25	25	2	2.5	13005	62424
AC_06	30	30	2	2.5	18605	89304
AC_07	40	40	2	2.5	32805	157464
AC_08	50	50	2	2.5	51005	244824
AC_09	60	60	2	2.5	73205	351384
AC_10	5	5	2	2.5	605	2904
AC_11	10	10	2	2.5	2205	10584
AC_12	15	15	2	2.5	4805	23064
AC_13	20	20	2	2.5	8405	40344
AC_14	25	25	2	2.5	13005	62424
AC_15	30	30	2	2.5	18605	89304
AC_16	40	40	2	2.5	32805	157464
AC_17	50	50	2	2.5	51005	244824
AC_18	60	60	2	2.5	73205	351384
AC_19	70	70	2	2.5	99405	477144
AC_20	80	80	2	2.5	129605	622104
AC_21	90	90	2	2.5	163805	786264
AC_22	100	100	2	2.5	202005	969624
AC_23	110	110	2	2.5	244205	1172184
AC_24	120	120	2	2.5	290405	1393944
AC_25	130	130	2	2.5	340605	1634904
AC_26	140	140	2	2.5	394805	1895064
AC_27	150	150	2	2.5	453005	2174424
AC_28	160	160	2	2.5	515205	2472984
AC_29	170	170	2	2.5	581405	2790744
AC_30	180	180	2	2.5	651605	3127704
AC_31	190	190	2	2.5	725805	3483864
AC_32	200	200	2	2.5	804005	3859224

that 'cam' in Z_{cam} refers to the ceiling, which is the only place cameras can be placed in the academic instances. Each candidate has two types of angles (orientations). The first type concerns the Z-axis rotation and is called the pan angle α; whereas the second type concerns the Y-axis rotation and is called the tilt angle β. Finally, Table 4.1 presents some information regarding the academic problem instances, which provides an idea of the type of data used in this study. This includes the name of each academic instance, the coordinates: X_{max}, Y_{max}, Z_{max}, and Z_{cam}, the total number of samples, and the total number of candidates, respectively. For more details regarding the information from the table, interested readers are advised to refer to [11].

As for the real-world data, urban areas were opted to generate the instances. This was accomplished by using two types of data from the selected locations: maps and elevation data. Unlike the academic instances, real-world instances cannot be represented by a specific shape (e.g., rectangle). Instead, sample points do not follow any specific pattern, as sampling these points relies on the infrastructure of the location that needs to

TABLE 4.2: Real-world problem instance specifications

Name	Samples	Candidates
RW_01	153368	32430
RW_02	285698	56132
RW_03	161099	32040
RW_04	304655	59137
RW_05	206900	34568
RW_06	380420	65691
RW_07	214889	42046
RW_08	382651	77986
RW_09	206816	39003
RW_10	368114	71323
RW_11	82437	15632
RW_12	136555	28109
RW_13	293138	61741
RW_14	81062	14916
RW_15	141309	27008
RW_16	105829	21063
RW_17	180453	35635
RW_18	79947	14423
RW_19	141114	26483
RW_20	332300	50284
RW_21	654068	90050
RW_22	83835	17203
RW_23	142326	31038
RW_24	201967	33880
RW_25	375680	59851
RW_26	105566	18043
RW_27	181090	32669
RW_28	136755	27838
RW_29	273964	49267
RW_30	263518	49354
RW_31	472660	87248
RW_32	124289	30189
RW_33	229231	55000
RW_34	134479	27329
RW_35	238546	47590
RW_36	135043	28162
RW_37	238492	50702

be monitored. Nevertheless, real-world instances also follow a Cartesian coordinate system in a three-dimensional space. More information regarding the real-world instances can be found in [44], where the authors provide a detailed explanation. For instance, the academic and real-world instances were taken from two different sources, namely, OpenStreetMap (OSM) and NASA's Shuttle Radar Topography Mission (SRTM). The former was used for sampling the target location (i.e., surveillance area), while the latter was used to create three-dimensional models from the flat OSM models.

Regarding the modelling of the candidate cameras, the process is identical to the academic instances. However, the way the configurations are presented at the end is quite different. The reason behind this is that for real-world instances, cameras are installed on buildings. This means that, contrary to academic instances, a grid cannot be used in real-world instances. As a result, a candidate camera configuration is defined by the

position (x_p, y_p, z_p) as well as the orientation (x_o, y_o, z_o). Real-world instance specifications are presented in Table 4.2. This includes the name of each real-world instance, the total number of samples, and the total number of candidates, respectively. For more details regarding the information from the table, interested readers are advised to refer to [11]. Also, for more details regarding the sampling process, readers can refer to [44].

To solve both academic and real-world instances, the USCP model (Model 3.2, which was introduced in Chapter 3, is adopted in this chapter as well. The reason behind choosing USCP is the fact that our problem instances do not contain any cost values. This is because that it is assumed that the cost of installing a camera is the same for all cameras. Thus, cost should not be considered when making a decision.

4.4 Reduction Techniques

As can be noted in Table 4.1 and Table 4.2, the size of the majority of the problem instances can be quite large. This means that the period of time needed to solve some of these instances can be very long. Thus, reducing the size of these problems might help in solving them in a significantly shorter period of time. Therefore, in this part of our study, we focus on three reduction techniques that were studied in the SCP literature to reduce the size of different SCP instances. Since OCP can be formulated as an SCP, we can use such techniques to reduce the sizes of our OCP instances.

A study by [63] suggested a number of reduction algorithms that work for SCP. Some of which were discussed in the literature, while others were novel. Three of the reduction techniques discussed in [63] were taken from [64], which are implemented in this study. The first reduction technique is called the *essential site*. This reduction technique examines if a given row can only be covered by one column after reducing the size of the problem. If so, then that column has to be included in the optimal solution and, therefore, all rows covered by said column can be excluded from the problem instance. To demonstrate further, E can be used as an example of an essential site case. Assume that E is a row taken from a matrix for illustrative purposes. In this row, any value of '1' indicates that this row is covered by a specific column; '0' otherwise. It is evident that this row is covered by column 3 only. Thus, column 3 has to be included in the solution, and, therefore, all rows that can be covered by column 3 can be eliminated from the problem.

$$\mathrm{E} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix},$$

The second reduction technique is called *dominated columns*. This method states that if one column covers a number of rows and another column covers the exact same rows and more, then the first column can be removed from the problem instance. In other words, we eliminate the subset. Assume that we are comparing two columns from a matrix, and these columns are presented in Matrix C. Since column 1 is a subset of column 2, it can be excluded from the problem. To illustrate more, both columns cover row 2, in this example. However, column 2 covers row 3 as well. This means that selecting column 2 only would always be the optimal strategy, as selecting column 1 as well would be redundant. As a result, we can eliminate column 1 from the matrix to reduce the size of the problem.

$$\mathrm{C} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix},$$

The third and final reduction technique implemented in this study is called *row domination*. This method can be viewed as an opposite of the dominated columns method. Row domination is used when one row includes a number of columns, and another row includes the same columns and more. Then, the second row can be excluded from the solution. In other words, we eliminate the superset. Assume that we are comparing two rows from a matrix, and these rows are presented in matrix R. Since row 2 is a superset of row 1, it can be excluded from the problem. The reason behind eliminating the superset in this method is that if we decide to keep row 2 and eliminate row 1 instead, column 2 becomes an option and selecting said column would not be optimal as it does not cover the first row. Eliminating the superset, row 2, and keeping row 1 would mean that column 3 is the only choice in this example and it does cover both rows. Therefore, selecting column 3 is the optimal move and having row 2 in the problem instance would be unnecessary.

$$\mathrm{R} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

Based on the previous three reduction methods, a reduction algorithm can be created [64]. The following are the four stages of the algorithm: (1) the algorithm starts with the essential site method, by going through all the rows of the matrix. (2) Then, the dominated columns method is implemented. In an attempt to find and eliminate all the subsets, we compare one column with all the other columns of the matrix, until we go through all the columns. (3) After that, the row domination method takes place. In this step, we compare one row with all the other rows, until we go through all the rows and all the supersets are found and removed. (4) Finally, we repeat the previous three stages until no further reduction can be made.

For our study, we decided to use this reduction algorithm from the SCP literature and apply it to our OCP instances. A significant difference between this and other reduction algorithms used in the OCP literature, such as [62], is that the algorithm we are using goes through three different stages and keeps doing that until no more reduction is possible. This can be useful because, when we go through the first three stages for the first time, a new reduced problem is formed with a possibility to be further reduced. Thus, we go through the first three stages again for as many times as needed, until no further reduction can be done.

One issue with this reduction algorithm is that it can be time consuming. For example, if we decide to use the column domination method, from the discussed reduction algorithm, to reduce the columns (candidates) in instance AC_32 from Table 4.1, we would need to compare each candidate with all the other 3,859,223 candidates. As one can imagine, this would take a significant period of time. The idea of using the reduction algorithm is to reduce the sizes of our problem instances and ultimately to reduce the time consumed to solve these problems; not to add more computational time. If reducing the size of the problem is also time consuming, then we might be better off solving the original problem instances. As a result, we implemented a minor, yet significant, alteration to the reduction algorithm we introduced earlier. This change had a huge impact on the time consumed to reduce the problem instances. As far as we are aware, this small change has not been discussed in the OCP literature.

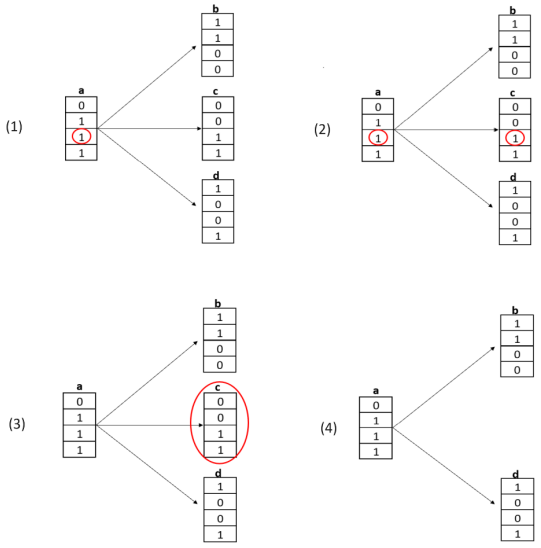

FIGURE 4.2: An example of the improved reduction algorithms

In the new improved reduction algorithm, we eliminate some unnecessary steps. To expand on that, when selecting one column, for example, and comparing it with all the other columns, some of these other columns have nothing in common with the selected column and, therefore, can never be subsets. In order to avoid comparing that column with any of these non-comparable columns, we developed the following algorithm: Firstly, we select a random row from the selected column. Then, we find other columns that cover the randomly selected row. This ensures that the given column and all the other considered columns have that row in common, and, therefore, a subset might be found. Finally, we check if any subsets exist. If so, remove them from the problem. We repeat this process for all the other columns. In this improved algorithm, we remove unnecessary steps as we only compare one column with specific columns instead of comparing it with all the other columns. The impact of the improved algorithm is discussed in Section 3.5. In this study, we employed a random selection decision in step 1, which could be later improved in future studies by employing a more sophisticated, non-random decision. This could yield even better results than the ones that will be discussed later in Section 3.5.

Figure 4.2 provides a visual example of this new and improved algorithm using the dominated columns method. In step 1, a random row is selected in column a. In this case, the third row was selected. In step 2, we check if any of the other columns also cover the third row. Here, column c is selected. In step 3, we check whether column c is a subset of column a. In this case, it is, in fact, a subset. Therefore, we remove column c

from the problem. In step 4, we have a new reduced problem after removing candidate c. In this example, we did not need to compare column a with columns b and d as neither b nor d had row 3 in common with column a. This eliminated two unnecessary steps, and, therefore, saved us some computational time. For the large problem instances, this small change can make a huge difference. The example in Figure 4.2 used columns, but the same concept applies to the rows.

The main conclusion we can make on the new reduction algorithm is that we do not need to compare one column/row with all the other columns/rows. By avoiding that, we obtain the same reduction results in a shorter period of time. The stages of the new and improved reduction algorithms are as follow: In step (1) the algorithm starts, as usual, with the essential site method, by going through all the rows of the matrix. If any essential site is found, that specific column is added to the solution and all the rows covered by that column are removed from the problem. The new changes to the algorithm are applied to the next two steps. In step (2), the dominated column method is implemented, by comparing one column with all the other comparable columns only, until we go through all the columns and all the subsets are found and removed. In step (3) the row domination method is implemented. Similarly, we compare one row with all the other comparable rows, until we go through all the rows and all the supersets are found and removed. In step (4) we go back to the previous three steps until no further reduction can be made. This reduction algorithm is applied to all 69 problem instances, and the results are provided in section 3.5. This includes a comparison between our reduction results and other reduction results from the literature as well as the impact of the reduction algorithm on the OCP problem.

4.5 Results

This section is divided into two parts. The first part is focused on the results of the reduction techniques. This includes both reduction methods that were explained earlier. After comparing the two algorithms, we compare our reduction results with other reduction results from the literature. For the second part of the section, the focus is on the solutions of the 69 problem instances. This includes a summary of the results as well as a comparison between the solutions before and after applying the reduction techniques.

TABLE 4.3: Academic problem instances reduction results

Instance	Samples	Candidates	Reduced Samples	Reduced Candidates	Old Time	New Time
AC_01	605	2904	600	1292	2.28	0.57
AC_02	2205	10584	225	404	11.36	2.84
AC_03	4805	23064	64	92	26.30	14.09
AC_04	8405	40344	513	2580	193.47	46.72
AC_05	13005	62424	1929	6636	558.17	110.13
AC_06	18605	89304	3069	12292	1398.94	239.48
AC_07	32805	157464	5909	28452	6866.49	1189.01
AC_08	51005	244824	9549	51012	108332.65	3156.02
AC_09	73205	351384	-	-	-	-
AC_10	605	2904	605	1672	2.58	0.34
AC_11	2205	10584	2205	7352	39.72	1.47
AC_12	4805	23064	4805	17032	200.22	3.92
AC_13	8405	40344	8405	30712	676.13	7.92
AC_14	13005	62424	13005	48392	1722.56	10.67
AC_15	18605	89304	18605	70072	3619.67	13.69
AC_16	32805	157464	32805	125432	11294.15	24.46
AC_17	51005	244824	51005	196792	29314.98	43.00
AC_18	73205	351384	73205	284152	58306.98	54.61
AC_19	99405	477144	99405	387512	110520.42	82.15
AC_20	129605	622104	129605	506872	184500.38	98.32
AC_21	163805	786264	163805	642232	315057.75	146.97
AC_22	202005	969624	202005	793592	464491.45	194.67
AC_23	244205	1172184	244205	960952	677793.17	256.96
AC_24	290405	1393944	290405	1144312	953569.75	426.80
AC_25	340605	1634904	340605	1343672	1366794.47	688.34
AC_26	394805	1895064	394805	1559032	-	936.63
AC_27	453005	2174424	453005	1790392	-	943.46
AC_28	515205	2472984	515205	2037752	-	1338.29
AC_29	581405	2790744	581405	2301112	-	1657.96
AC_30	651605	3127704	651605	2580472	-	2014.11
AC_31	725805	3483864	725805	2875832	-	2217.76
AC_32	804005	3859224	804005	3187192	-	1231.94

4.5.1 Reduction Results

We start this subsection by summarizing the reduction results of the 69 problem instances. We, then, compare the two reduction methods we applied in terms of time consumption. Finally, we compare our reduction results with other results from the literature.

Table 4.3 provides the reduction results of the academic problem instances. The table includes the instance number, number of samples and candidates, reduced number of samples and candidates, and reduction time using the old algorithm as well as the improved algorithm. For example, problem instance AC_08 originally included 51,005 samples and 244,834 candidates. After applying the reduction algorithm, the new reduced version of problem AC_08 included 9,549 samples and 51,012 candidates. This is about 80% less samples and candidates, which is a significant difference. To achieve this result, the old reduction algorithm took 108,332.65 seconds, while the improved

TABLE 4.4: Real-world problem instances reduction results

Instance	Samples	Candidates	Reduced Samples	Reduced Candidates	Old Time	New Time
RW_01	153368	32430	28675	22754	13600.70	44.89
RW_02	285698	56132	69844	42310	57815.67	178.51
RW_03	161099	32040	27066	19713	13351.54	43.52
RW_04	304655	59137	70727	39639	62152.28	231.89
RW_05	206900	34568	26271	19813	22416.03	45.87
RW_06	380420	65691	80140	45363	88951.40	252.26
RW_07	214889	42046	35373	26965	25519.06	53.06
RW_08	382651	77986	90204	54846	134757.20	389.00
RW_09	206816	39003	31778	24238	22391.55	48.22
RW_10	368114	71323	82511	48189	90996.49	271.56
RW_11	82437	15632	14820	10334	3145.24	21.97
RW_12	136555	28109	37404	20696	14986.07	123.41
RW_13	293138	61741	51908	40503	58412.29	94.97
RW_14	81062	14916	13997	9887	3253.30	20.26
RW_15	141309	27008	37332	20288	13539.79	96.75
RW_16	105829	21063	15225	12726	6052.12	23.57
RW_17	180453	35635	37308	25465	36319.61	129.21
RW_18	79947	14423	13144	9575	4113.01	25.18
RW_19	141114	26483	33426	18976	19479.75	135.97
RW_20	332300	50284	40146	29811	40883.70	63.01
RW_21	654068	90050	105051	61268	313201.40	534.86
RW_22	83835	17203	12162	10673	3632.83	18.58
RW_23	142326	31038	32832	22300	13861.27	73.45
RW_24	201967	33880	27854	21025	17845.31	39.27
RW_25	375680	59851	69866	40434	72299.78	195.82
RW_26	105566	18043	12990	10366	4255.11	20.81
RW_27	181090	32669	36071	22149	27391.31	126.50
RW_28	136755	27838	20317	18239	7937.01	24.65
RW_29	273964	49267	52044	36163	37585.62	114.79
RW_30	263518	49354	48499	34016	48082.67	92.39
RW_31	472660	87248	116789	63266	205780.86	664.57
RW_32	124289	30189	22405	20277	9560.98	31.68
RW_33	229231	55000	56706	41191	52572.57	168.96
RW_34	134479	27329	21793	17419	9326.81	32.78
RW_35	238546	47590	52324	33072	40133.96	152.50
RW_36	135043	28162	22055	17751	10158.12	40.46
RW_37	238492	50702	57305	35563	53716.19	254.11

version of this algorithm took only 3,156.02 seconds. It is quite obvious that the improved reduction algorithm is far more effective than the original one, when it comes to the computational time. This time cut can be extremely valuable when solving a big number of problem instances, especially when those problem instances are not small in size.

For reducing each problem instance, a random time-limit of 16 days was set. As can be seen from Table 4.3, from problem instance AC_26 until AC_32, the reduced version could not be obtained within 16 days using the original reduction algorithm. Whereas, it took around only 25 minutes on average to obtain the reduced problems, using the improved algorithms. As for problem instance AC_09, the file was simply too large to be opened. In other words, the machine we worked on could not access this file due to the size of instance, which was too large for a notepad, where our problem instances

TABLE 4.5: Our reduction results compared with another study [62]

Instance	Original	[62]	Our Work	Difference %
AC_01	2904	1292	1292	0%
AC_02	10584	908	404	56%
AC_03	23064	924	92	90%
AC_04	40344	4572	2580	44%
AC_05	62424	9852	6636	33%
AC_06	89304	16732	12292	27%
AC_07	157464	35291	28452	19%
AC_08	244824	60251	51012	15%
AC_09	-	-	-	-
AC_10	2904	1672	1672	0%
AC_11	10584	7352	7352	0%
AC_12	23064	17032	17032	0%
AC_13	40344	30712	30712	0%
AC_14	62424	48392	48392	0%
AC_15	89304	70072	70072	0%
AC_16	157464	125431	125432	0%
AC_17	244824	193791	196792	-2%
AC_18	351384	284151	284152	0%
AC_19	477144	387511	387512	0%

are stored. This is why we could not obtain any results regarding this instance. Thus, the entries of this respective instance are presented by a dash '-'. In another study that discussed these instances [62], the authors did not provide results regarding instance AC_09; whereas results regarding AC_08 and AC_10 were presented. This shows that others have faced the same issue while accessing this instance.

Table 4.4 provides the reduction results of the real-world problem instances. Similar to Table 4.3, it includes the instance number, original number of samples and candidates, reduced number of samples and candidates, and reduction time using the old method as well as the new method. For example, problem instance RW_08 originally included 382,651 samples and 77,986 candidates. After applying the reduction algorithm, the new reduced version of problem RW_08 included 90,204 samples and 54,846 candidates. This is about 76% less samples and 30% less candidates, which, again, is a significant difference. To achieve this result, the first reduction algorithm took 134,757.20 seconds, while the improved version of this algorithm took only 389.00 second. Again, this big difference can save us a significant amount of computational time, especially when experimenting with larger problem instances.

We can also compare our reduction results with another study from the literature which used the same OCP instances. In their paper, [62] implemented one type of reduction methods, namely, the dominated columns method, on the first 19 academic problem instances. Based on this approach, they produced new versions of the OCP instances with a reduced number of candidates. Therefore, We compare the reduced candidates between the two studies in Table 4.5. The table includes the name of the instance, the original number of candidates, the reduced number of candidates by [62], our reduced number of candidates, and, finally, the difference, in percentage, between the two studies. Our reduction algorithms worked significantly better in 7 out of 19 instances. For the rest, there was almost no difference, except for instance AC_17, where their method worked slightly better. In addition, in instance AC_16, AC_18 and AC_19, there is a difference of just '1' candidate between our results and their results. This is probably a small computational error as this pattern cannot be a coincidence, and both results should be identical in those three instances.

The main reason that our reduction algorithm yielded better results than [62], overall, is that we have used more reduction methods. More specifically, the row domination method helped in creating a new reduced problem each time, which can possibly be reduced again. To illustrate more, a problem is reduced only once when using one reduction method. When adding another reduction method, such as the row domination method, a new problem is generated. Meaning, the column domination method can be implemented once again as more subsets can possibly be found. This way, the reduction algorithm keeps going on until no further reduction can be made. Therefore, we believe that implementing more than one reduction method can be extremely useful and, evidently, produces better results.

4.5.2 Solutions

For solving all 69 problem instances, we used PuLP[1] package in python to build our models, and within python used Gurobi as our commercial Linear Programming solver. For solving each problem instance, we set a three-hours time-limit. The machine used to conduct these experiments is an Intel(R) Core(TM) i5-8500T CPU @ 2.10GHz 2.11GHz with 8.00GB RAM. Table 4.6 and Table 4.7 provide the solutions of the problem instances after applying the reduction algorithms. Both tables include the instance name, number of samples, number of candidates, upper bound, lower bound, gap, and the time

[1] https://pypi.org/project/PuLP/

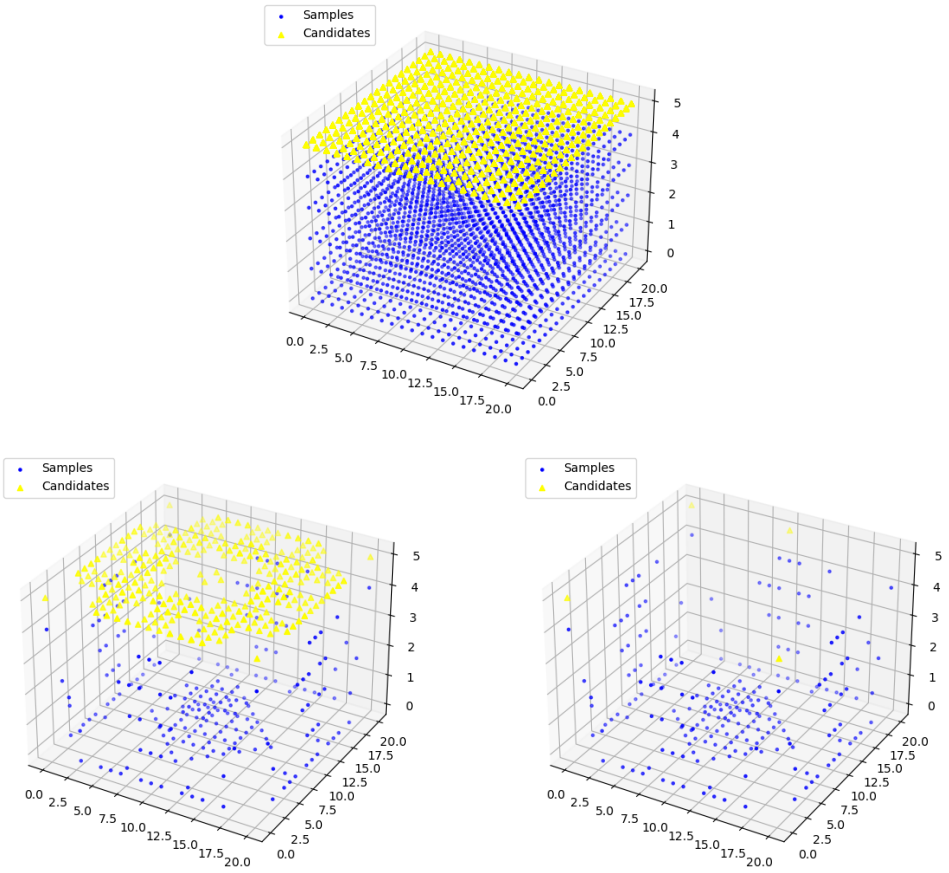

FIGURE 4.3: Visualisation of AC_02 problem instance (top), its reduced version (bottom left), and the solution of the reduced version (bottom right)

needed in seconds. However, Table 4.6 also includes results of another study [62], which includes their UB, LB, and gap.

We can focus on AC_02 from Table 4.6 for illustrative purposes. This academic problem instance contains 2,205 samples and 10,584 candidates. It took only 0.54 seconds to obtain an optimal solution of 4 candidates that cover all samples. The same solution was obtained in 140.30 seconds when solving the problem before implementing the reduction algorithm. The solution is given in Figure 4.3, where the blue points represent the samples and the yellow points represent the candidates. The graph on top of the figure represents the original problem instance. The graph on the lower left represents the reduced problem instance. Finally, the graph on the lower right represents the solution to the reduced problem instance.

We can also compare our results with the work of [62] using Table 4.6, where they used a CPLEX solver to obtain their optimal solutions. Overall, we managed to obtain better

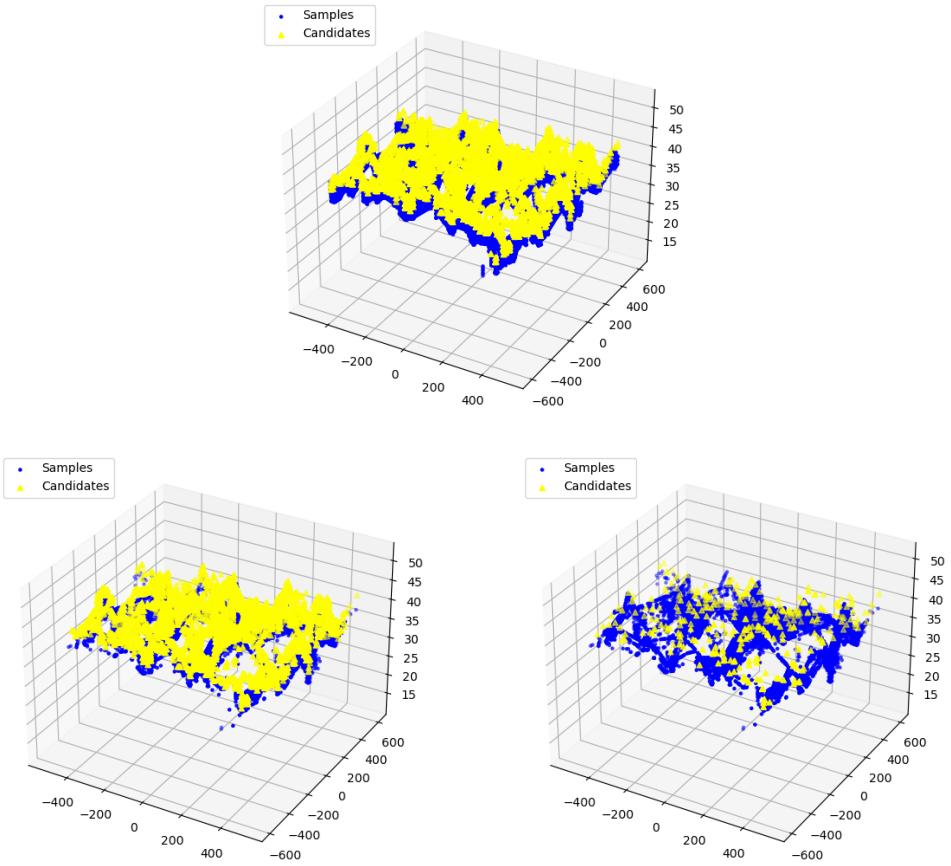

FIGURE 4.4: Visualisation of RW_14 problem instance (top), its reduced version (bottom left), and the solution of the reduced version (bottom right)

results. For the first 5 instances, both this work and their work managed to obtain a gap of 0%. As for instance AC_06, we managed to obtain the optimal solution, where they were far from that with a 100% gap. For AC_07, we managed to obtain a gap of 11.76%, where they have not managed to obtain any results. For AC_10 to AC_15, our gap remained smaller. This, overall, shows that our reduction methods helped us obtain better results than the work of [62].

As for the real-world instances, we can focus on instance RW_14 from Table 4.7. The problem instance originally consisted of 14,916 candidates and 81,062 samples. For this real-world problem, an optimal solution was obtained, with an objective value of 337 candidates to cover all the samples. This problem is also visualised in Figure 4.4. The graph on top represents the original problem of RW_14, the graph on the lower left represents the reduced problem, and the lower right graph represents the solution of the reduced problem. It can be noted that, unlike in instance AC_02, the points are not distributed in a uniform shape. This happens because RW_14 is a real-world problem

TABLE 4.6: Academic problem instances results

Instance	Samples	Candidates	UB [62]	LB [62]	Gap [62]	UB	LB	Gap	Time
AC_01	605	2904	7	7	0.00%	7	7	0.00%	5.15
AC_02	2205	10584	4	4	0.00%	4	4	0.00%	0.54
AC_03	4805	23064	3	3	0.00%	3	3	0.00%	0.05
AC_04	8405	40344	5	5	0.00%	5	5	0.00%	9.92
AC_05	13005	62424	7	7	0.00%	7	7	0.00%	193.16
AC_06	18605	89304	16,732	0	100%	10	10	0.00%	3300.00
AC_07	32805	157464	-	-	-	17	14.95	11.76%	10800.00
AC_08	51005	244824	-	-	-	-	-	-	-
AC_09	73205	351384	-	-	-	-	-	-	-
AC_10	605	2904	21	17.02	18.97%	20	17.63	10.00%	10800.00
AC_11	2205	10584	71	52.21	26.47%	72	52.92	26.38%	10800.00
AC_12	4805	23064	17,032	0	100%	168	109.91	34.52%	10800.00
AC_13	8405	40344	30,712	0	100%	347	187.17	45.82%	10800.00
AC_14	13005	62424	48,392	0	100%	650	286.59	55.84%	10800.00
AC_15	18605	89304	70,072	0	100%	933	407.22	56.27%	10800.00
AC_16	32805	157464	-	-	-	-	-	-	-
AC_17	51005	244824	-	-	-	-	-	-	-
AC_18	73205	351384	-	-	-	-	-	-	-
AC_19	99405	477144	-	-	-	-	-	-	-
AC_20	129605	622104	-	-	-	-	-	-	-
AC_21	163805	786264	-	-	-	-	-	-	-
AC_22	202005	969624	-	-	-	-	-	-	-
AC_23	244205	1172184	-	-	-	-	-	-	-
AC_24	290405	1393944	-	-	-	-	-	-	-
AC_25	340605	1634904	-	-	-	-	-	-	-
AC_26	394805	1895064	-	-	-	-	-	-	-
AC_27	453005	2174424	-	-	-	-	-	-	-
AC_28	515205	2472984	-	-	-	-	-	-	-
AC_29	581405	2790744	-	-	-	-	-	-	-
AC_30	651605	3127704	-	-	-	-	-	-	-
AC_31	725805	3483864	-	-	-	-	-	-	-
AC_32	804005	3859224	-	-	-	-	-	-	-

where the surveillance area represents an actual place. The yellow points in Figure 4.4 are the candidates placed on the walls of different buildings. The blue points, as discussed earlier, are the sample points that must be covered by the candidates.

Looking at tables 4.6 and 4.7, we can observe that the information of some problem instances is not provided. This is due to the fact that we set a three-hour time-limit. This means that, in some cases, three hours were not enough to even build the model, which usually is time consuming due to the size of some problem instances. If the model is not built within three hours, then it cannot be solved. Hence, there are some empty rows in the tables. This computational issue can be dealt with in the future by finding an intelligent way to speed up the model-building process. An easier way to deal with this problem is to increase the time-limit. Of course, the latter will make the overall computational time longer, but it will provide results for more instances and improve the results we have already obtained.

Generally, we believe these results are excellent, especially when we compare them to our previous work [65], In that study, we tried to solve the 69 problem instances without addressing the instance size issue. As a result, we had less optimal solutions and much more gaps in the tables. A comparison between the two studies is summarised in Table

TABLE 4.7: Real-world problem instances results

Instance	Samples	Candidates	UB	LB	Gap	Time
RW_01	153368	32430	665.00	652.00	1.80&	10800.00
RW_02	285698	56132	-	-	-	-
RW_03	161099	32040	745.00	742.43	0.26%	10800.00
RW_04	304655	59137	-	-	-	-
RW_05	206900	34568	934.00	934.00	0.00%	10404.92
RW_06	380420	65691	-	-	-	-
RW_07	214889	42046	989.00	982.96	0.61%	10800.00
RW_08	382651	77986	-	-	-	-
RW_09	206816	39003	966.00	961.58	0.41%	10800.00
RW_10	368114	71323	-	-	-	-
RW_11	82437	15632	315.99	313.20	0.63%	10800.00
RW_12	136555	28109	322.00	313.40	2.48%	10800.00
RW_13	293138	61741	-	-	-	-
RW_14	81062	14916	337.00	337.00	0.00%	6237.76
RW_15	141309	27008	342.00	336.20	1.46%	10800.00
RW_16	105829	21063	508.00	505.39	0.39%	10800.00
RW_17	180453	35635	519.00	507.27	2.11%	10800.00
RW_18	79947	14423	338.00	338.00	0.00%	3548.61
RW_19	141114	26483	349.00	341.37	2.01%	10800.00
RW_20	332300	50284	1467.00	1453.73	0.88%	10800.00
RW_21	654068	90050	-	-	-	-
RW_22	83835	17203	398.00	396.66	0.25%	10800.00
RW_23	142326	31038	417.00	411.07	1.19%	10800.00
RW_24	201967	33880	887.00	877.98	1.01%	10800.00
RW_25	375680	59851	-	-	-	-
RW_26	105566	18043	464.00	461.19	0.43%	10800.00
RW_27	181090	32669	489.00	484.05	0.81%	10800.00
RW_28	136755	27838	648.00	644.76	0.46%	10800.00
RW_29	273964	49267	-	-	-	-
RW_30	263518	49354	-	-	-	-
RW_31	472660	87248	-	-	-	-
RW_32	124289	30189	651.00	646.83	0.61%	10800.00
RW_33	229231	55000	-	-	-	-
RW_34	134479	27329	609.00	605.93	0.49%	10800.00
RW_35	238546	47590	-	-	-	-
RW_36	135043	28162	609.00	606.50	0.32%	10800.00
RW_37	238492	50702	-	-	-	-

4.8. The comparison includes the number of optimal solutions found, the number of instances with no results, and the average gap. It is evident that this study yielded better results in all categories of comparison in both academic and real-world instances. Firstly, the current study found 9 optimal solutions in total, whereas the previous study found 4 only. Secondly, there was a total of 56 instances where no results were obtained in the previous study, compared to 33 in the current study. Finally, the average gap of the current study is noticeably smaller than the average gap of the previous study for both types of problem instances. For a fair comparison, only instances that yielded results in the previous study were used in the average gap comparison.

All of these facts prove that the reduction algorithms that we implemented improve the OCP results. For more details regarding the results before implementing the reduction algorithms, readers are advised to refer to [65]

TABLE 4.8: A comparison between the results of the current study and the previous study [65]

Comparison	Previous Work	Current Work
Optimal solutions found (Academic)	4	6
Optimal solutions found (Real-world)	0	3
No results (Academic)	23	19
No results (Real-world)	33	14
Average Gap (Academic)	24.03%	19.17%
Average Gap (Real-world)	1.07%	0.33%

4.5.3 Further Experimentations, Limitations, and Future Work

There are a couple of limitations that can be discussed in this section. Firstly, starting from the academic instance AC_10 until we reach AC_32 in Table 4.3, the number of samples is not reduced. This cannot be a coincidence, as this issue occurs in more than half the academic problem instances. To address this issue, new reduction methods might be needed to be used. This means that further reduction techniques can be explored from the SCP literature and applied to address the same OCP problem instances. This could reduce the number of samples for instances AC_10 to AC_32, but also might, generally, yield better reduction results. That being said, this might not be an issue at all, and the samples phenomena exists because of the nature of these problem instances. To elaborate more, the academic problem instances are not based on real-world locations, which means they are made up. In addition, these problems seem to be made to be complicated and difficult to address, so having this issue might be because of that. Nevertheless, further experiments should be conducted to prove this theory.

Generally, although our reduction algorithms proved to be effective in improving the OCP results, we wondered if we can further improve them in an easy way. In an attempt to achieve that, we conducted a new simple experiment. In short, we changed the order of our reduction methods in the reduction algorithm. Meaning, instead of starting with the essential site method, then moving to the dominated columns, and finally to the row domination method, we switched the order by starting from the row domination method, moving to the dominated columns, and finally to the essential site method. This was a simple attempt to improve the results. However, the new reduction results were exactly the same in terms of reduced problem size and time consumption.

For future work, there are a few possibilities to address the OCP problem. The first possibility has already been discussed in this paper, which is to simply try new reduction

techniques. If we ever succeed in finding better techniques, especially techniques that reduce the problem instance size even further, the OCP results should be further improved. Another possibility for future work is to apply different optimization methods. Different heuristic techniques can be used to produce more optimal or near-optimal solutions in a reasonable time. Methods, such as greedy algorithms, simulated annealing, genetic algorithms and a few more, can be considered. In a different direction, we could formulate the OCP problem as a bi-objective problem with two objective, namely, maximizing coverage and minimizing cameras. This would be another possibility, where we would focus on offering more realistic results. To illustrate more, instead of one optimal solution, bi-objective OCP would produce multiple efficient solutions. Each efficient solution would represent a combination of the two objectives. For example, one efficient solution could achieve 90% coverage using a number of cameras and another efficient solution could achieve 70% coverage using less cameras. In this case, the second solution offers less coverage, but it is cheaper to implement, whereas the first solution offers better coverage, but it is more expensive. Both efficient solutions are viable as they both offer different combinations, where none of them is better than the other. In real-life, decision makers might not want the optimal solution, as they might prefer something that is good enough at a cheaper price. Using bi-objective techniques, decision makers can have multiple efficient solutions and choose the one that is most suitable to their needs. This is the reason that bi-objective optimization can be more realistic. For future work, there are several bi-objective methods that can be implemented to obtain the efficient solutions of the OCP instances. The epsilon-constraint method as well as the weighted Sum method, two of the most popular multi-objective techniques, can be considered.

4.6 Conclusions

To sum up, OCP is an optimization problem that either maximizes area coverage, such that camera cost is reduced, or minimizes camera cost, such that coverage requirements are achieved. When using the latter objective, OCP can be formulated as an SCP. This means that techniques from the SCP literature can be used to address the OCP problem. Hence, this chapter exploited this relationship, by finding techniques from the literature of SCP to tackle our 69 OCP problem instances. More specifically, we studied a few SCP reduction techniques, improved on them, and then applied them to reduce the sizes of our OCP problem instances. Not only we were able to show that our reduction results

are better than other reduction results from the literature, but also we proved that the OCP problem solutions can be obtained significantly faster when reducing the size of the problems.

For future work, more reduction techniques from the SCP literature can be explored and applied to our OCP problem instances. In addition, different optimization techniques from the SCP literature can be used to solve our instances. This can include multi-objective optimization, where the two main objectives would be to minimize the number of cameras and maximize the area coverage. Heuristic techniques can also be used to obtain near-optimal solutions in a reasonable time. This chapter has mentioned some heuristic techniques that could be used to deal with our problem instances.

Chapter 5

An Effective Approach in Generating the Efficient Frontier of the Bi-Objective Optimal Camera Placement Problem Using the ϵ-Constraint Method

Optimal camera placement (OCP) problem is a modern combinatorial optimization problem, which tries to find the optimal placement of cameras to ensure complete coverage of certain locations, usually for surveillance purposes. Traditionally, OCP is formulated as a single-objective problem. Most studies employ one of the following two formations: minimize the camera cost, such that complete coverage is reached; or maximize coverage such that the cheapest camera plan is utilized. Since there are two main factors that concern this problem, OCP is naturally a bi-objective problem with two contradicting objectives. One objective is to minimize the camera cost, while the other is to maximize the area coverage. Even though this formulation can be obvious, not many studies have tackled the OCP problem as a bi-objective problem. This paper addresses the bi-objective OCP problem using an affective ϵ-constraint approach. In order to accomplish that, academic and real-world instances, published in the GECCO 2021 competition on "the optimal camera placement problem and the unicost set covering problem", are adopted. The results appear to be promising, especially when the complex nature of these problem instances is handled.

5.1 Introduction

Various real-life optimization problems are addressed using different techniques for obtaining optimal or near-optimal solutions. Often, the focus is on the quality of the solution and the time needed to obtain that solution. In single objective problems, researchers aim to find optimal solutions in a reasonable period of time. If the latter is difficult to achieve, especially when dealing with $\mathcal{NP}$-hard problems, some tend to utilize different heuristic and metaheuristic techniques to obtain optimal or near-optimal solutions in a shorter time period. For many real-life optimization problems, however, having multiple efficient solutions is more desirable than one single optimal solution, as this provides the decision makers the option to choose the solution that best suits their needs. Therefore, this study converts the traditional single-objective OCP problem into a bi-objective problem. OCP is an $\mathcal{NP}$-hard combinatorial optimization problem, and its goal is to find the cheapest camera placement plan that fully monitors a target area for several purposes, including monitoring and surveillance purposes. More formally, single-objective OCP has two main formulations: one is to maximize area coverage such that camera cost is reduced, while the other is to minimize camera cost such that full coverage is achieved. In bi-objective OCP, which is the main topic of this study, the goal is to simultaneously maximize area coverage and minimize camera cost. The idea of OCP was first implemented in the 1970s, when [1] attempted to fully monitor an art gallery using a minimal number of guards. To achieve that, the author introduced the Art Gallery Problem (AGP) to minimize the number of guards, such that the gallery is fully monitored. This idea has then influenced the OCP field, where guards are substituted by cameras. In recent years, it was pointed out that the OCP problem can be formulated as the popular set covering problem [9], and because of that, SCP techniques can be used to address OCP problems. This paper exploits this feature and formulates the bi-objective OCP problem as a bi-objective SCP. This study employs an effective approach to solve the bi-objective OCP problem and generate the full efficient frontier using the ϵ-constraint method, which is a well-known approach in the multi-objective optimization field.

Most researchers have addressed the OCP problem using single-objective optimization techniques; only a few have adopted multi-objective techniques (e.g., [66]; [67]; [46]). Before jumping into some of these applications, a brief introduction to the relevant multi-objective terms might be needed. Firstly, instead of a single optimal solution, a multi-objective problem produces a set of alternatives, known as the *feasible set*. Feasible sets consist of different combinations of the objective values. For example, an OCP problem

that has two objectives, such as maximizing coverage and minimizing camera cost, would produce a feasible set where each feasible solution represents a combination of coverage and cost values. From those feasible solutions, *efficient solutions* are identified, which brings us to the second point. A solution is considered as efficient if there are no other solutions offering 'better' values. Going back to the OCP example, no other solution would have a better combination of coverage and cost values than an efficient solution. This means we could have several efficient solutions with different combinations. For instance, one efficient solution could have better coverage but worse cost value than another efficient solution which has worse coverage but better cost. Correspondingly, if a solution is worse in both coverage and cost, then it would not be identified as an efficient solution. After all the efficient solutions have been obtained, the next step is to plot those solution to present the *efficient frontier*. Any point that lies on an efficient frontier is considered as an efficient solution. Finally, and after all the efficient solutions have been obtained and the efficient frontier has been generated, decision makers can select the efficient solution that works best for their specific problem. For more details regarding multi-objective optimization, readers can refer to [15].

[67] is an example of a study that used multiple objectives to deal with the OCP problem. The authors aimed to use sensor cameras to monitor 3-d surveillance areas. Their model consisted of two contradicting objectives: minimizing the cost of the camera network and maximizing the coverage quality of the network. The first objective can be achieved by selecting the needed camera sensors and choosing their optimal positions, whereas the second objective concerns the quality of the coverage, which is assessed by the importance of the covered points. To illustrate more on the latter, points on the surveillance area are given a specific score that reflects their importance. For instance, a more important point would be given a higher score than a less important one. These are called the weighted points. Therefore, the goal here is to maximize the sum of the weighted covered points, which can provide a better network coverage quality. For this problem, the authors added a security constraint, which means that all cameras from the camera network must be observed by other cameras (at least by one camera). To deal with this OCP problem, the authors formulated a bi-objective binary integer programming model. After formulating the bi-objective problem in [67], the authors used three different bi-objective methods to produce the efficient frontier. These three methods are: the weighted sum scalarization, ϵ-constraint, and a two-phase method. In another study, [66] used OCP to maximize the visibility from heavy machinery, while also minimizing the number of cameras needed. Using these two objectives, they formulated their problem as a bi-objective OCP problem. The authors addressed this problem using

the lexicographic method. This method solves the bi-objective problem in a hierarchical manner. To elaborate more, one objective can be set as the priority objective, and, then, is optimized. The solution produced from solving this objective is, then, set as a fixed value, and based on that value, the other objective is solved. For example, if the priority objective is minimizing the number of cameras, that objective is solved then is fixed. After that, the other objective, i.e., maximizing coverage, is solved by searching for the maximum coverage value from all the minimal cost solutions. Finally, yet another study employed the typical two OCP objectives to formulate a bi-objective OCP problem: minimizing camera cost and maximizing coverage. [46] used bi-objective OCP to find several camera placement plans (i.e., efficient solutions) for surveillance purposes. The authors addressed their bi-objective OCP problem by using a multi-objective genetic algorithm, called NSGA-II [48], to obtain the complete efficient set.

Since the OCP problem is formulated as a set covering problem in our study, a few relevant multi-objective SCP studies are discussed next. a study by [26] used SCP to address a transit route network problem with multiple objectives. Before formulating their problems, the authors implemented a Route Constructive Genetic Algorithm (RCGA), which was tailored to their defined objectives, to generate a big set of candidates (transit routes). Then, they applied a heuristic algorithm, called the Meta-heuristic for Randomized Priority Search (Meta-RaPS), to generate the feasible set. From that feasible set, efficient solutions were identified. In a study by [68], the authors adopted a combined multi-objective SCP and knapsack problem to deal with database queries. To solve this problem, Hill Climbing and Genetic algorithms were implemented. In another study, [69] focused on bi-objective SCP and used ten multi-objective metaheuristic methods on the problem to evaluate their performances. Nine of these multi-objective metaheuristic methods have been used in the literature, and they are mainly based on Simulated Annealing, Genetic Algorithms, Hybrid Genetic Algorithms, and Multiple Start Local Search. The remaining method is a novel algorithm based on Hybrid Genetic Algorithm. Finally, a study by [70] focused on generating all the efficient solutions for two multi-objective problems, one of which was a set covering problem. To obtain the efficient solutions, the authors applied a method called AUGMECON2, which is an enhanced version of the original augmented ϵ-constraint method [71]. There are other studies that worked with multi-objective SCP and are worth mentioning in this paper such as [72], [73], and [74]. Each one of these multi-objective SCP studies, mentioned and discussed in this paper, offers a glimpse of what can be done to address our bi-objective OCP problem.

TABLE 5.1: Academic problem instances specifications

Name	X_{max}	Y_{max}	Z_{max}	Z_{cam}	Samples	Candidates
AC_01	5	5	2	2.5	605	2904
AC_02	10	10	2	2.5	2205	10584
AC_03	15	15	2	2.5	4805	23064
AC_04	20	20	2	2.5	8405	40344
AC_05	25	25	2	2.5	13005	62424
AC_06	30	30	2	2.5	18605	89304
AC_07	40	40	2	2.5	32805	157464
AC_08	50	50	2	2.5	51005	244824
AC_09	60	60	2	2.5	73205	351384
AC_10	5	5	2	2.5	605	2904
AC_11	10	10	2	2.5	2205	10584
AC_12	15	15	2	2.5	4805	23064
AC_13	20	20	2	2.5	8405	40344
AC_14	25	25	2	2.5	13005	62424
AC_15	30	30	2	2.5	18605	89304
AC_16	40	40	2	2.5	32805	157464
AC_17	50	50	2	2.5	51005	244824
AC_18	60	60	2	2.5	73205	351384
AC_19	70	70	2	2.5	99405	477144
AC_20	80	80	2	2.5	129605	622104
AC_21	90	90	2	2.5	163805	786264
AC_22	100	100	2	2.5	202005	969624
AC_23	110	110	2	2.5	244205	1172184
AC_24	120	120	2	2.5	290405	1393944
AC_25	130	130	2	2.5	340605	1634904
AC_26	140	140	2	2.5	394805	1895064
AC_27	150	150	2	2.5	453005	2174424
AC_28	160	160	2	2.5	515205	2472984
AC_29	170	170	2	2.5	581405	2790744
AC_30	180	180	2	2.5	651605	3127704
AC_31	190	190	2	2.5	725805	3483864
AC_32	200	200	2	2.5	804005	3859224

The rest of the paper is structured as follows: Section 5.2 provides a summary of the problem instances used in this study. Section 5.3 introduces all the relevant mathematical models, including the bi-objective OCP formulation. Section 5.4 discusses the first set of results, and is followed by Section 5.5 which introduces techniques from the SCP literature to improve the initial results. Section 5.6, then, discusses the improved results, which is followed by further experimentations in Section 5.7. Finally, Section 5.8 reflects on and concludes this paper.

5.2 Problem Description

Each problem instance used in this study consists of two main components. The first component is a set of candidates, where each candidate represents a possible camera position as well orientation (i.e., angle and tilt). This means, multiple candidates can share the same position but have different angles and/or tilts. The second component is a set of samples, where each sample represents a 3-dimensional point in a 3-dimensional

TABLE 5.2: Real-world problem instance specifications

Name	Samples	Candidates
RW_01	153368	32430
RW_02	285698	56132
RW_03	161099	32040
RW_04	304655	59137
RW_05	206900	34568
RW_06	380420	65691
RW_07	214889	42046
RW_08	382651	77986
RW_09	206816	39003
RW_10	368114	71323
RW_11	82437	15632
RW_12	136555	28109
RW_13	293138	61741
RW_14	81062	14916
RW_15	141309	27008
RW_16	105829	21063
RW_17	180453	35635
RW_18	79947	14423
RW_19	141114	26483
RW_20	332300	50284
RW_21	654068	90050
RW_22	83835	17203
RW_23	142326	31038
RW_24	201967	33880
RW_25	375680	59851
RW_26	105566	18043
RW_27	181090	32669
RW_28	136755	27838
RW_29	273964	49267
RW_30	263518	49354
RW_31	472660	87248
RW_32	124289	30189
RW_33	229231	55000
RW_34	134479	27329
RW_35	238546	47590
RW_36	135043	28162
RW_37	238492	50702

space. Samples can be covered by candidates and candidates can cover samples. The goal of a bi-objective OCP problem would be to simultaneously minimize the cost of candidates and maximize the coverage of the sample points.

69 problem instances are used in this paper. These problem instances are divided into two parts. First, 32 made-up instances, called *academic*. Each academic instance is represented by a 3-dimensional grid, which is supposed to sample a rectangular room. Moreover, each room is modelled with a different size and different camera placement possibilities. In these rooms, cameras can only be placed on the ceilings. The second part of these problem instances consists of 37 instances, which are called *real-world*. Real-world instances are modelled after real urban locations, where cameras can be placed on different spots, such as walls of buildings. All the problem instances that are utilized in this study are published online by the GECCO 2021 competition on the optimal camera placement problem and the unicost set covering problem [11]. Table

5.1 and Table 5.2 summarize all the information regrading the academic and real-world instances, respectively. Some of this information is discussed next, but for a more detailed discussion regarding the creation of these problem instances, readers can refer to [11] and [44].

Table 5.1 and Table 5.2 provide all the information regrading the academic and real-world instances, respectively. For instance, in the second, third, and fourth columns of Table 5.1, the values X_{max}, Y_{max}, and Z_{max} represent the maximum width, depth, and height of each sample point, respectively. In other words, each sample can be located anywhere between (0, 0, 0) and $(X_{max}, Y_{max}, Z_{max})$ in any 3-dimensional grid. In addition, Z_{cam}, in the fifth column of Table 5.1, is the unified height for all candidates. This would be the ceiling of a room. The numbers of samples and candidates for both academic and real-world instances are also given in the last two columns of both tables.

Other relevant information from the two tables could be the characteristics of the candidates, and they are: (V_{res}) representing the vertical resolution, (H_{res}) representing the horizontal resolution, and (H_{fov}) representing the horizontal field of view. Figure 5.1 provides a visual illustration of two problem instances: an academic instance on the left and a real-world instance on the right. In both graphs, the yellow points represent the candidates, whereas the blue points represent the samples that must be covered by a subset of those candidates.

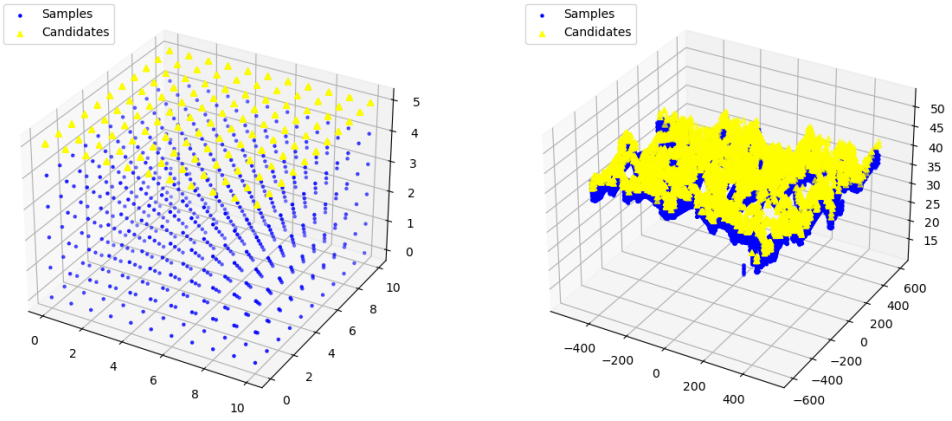

FIGURE 5.1: Two visual examples of two OCP instances: acadmeic instances (AC_01) on the left and real-world instance (RW_14) on the right

For this study, we believe that the details that have been discussed in this section should be sufficient to understand the idea of these problem instances. However, for a more in-depth discussion, including a complete description of the information in Table 5.1 and Table 5.2, readers are encouraged to refer to [11].

5.3 Mathematical Models

In this section, we will introduce several mathematical models in regards to OCP. Each model will serve a different purpose and will build onto the next one. This section is divided into three subsections. The first subsection will introduce and briefly discuss the single objective formulation of the OCP problem. The second subsection will introduce and discuss the bi-objective formulations of the OCP problem. The third and final subsection will focus on the ϵ-constraint formulations, which will be used to solve the bi-objective OCP problem.

5.3.1 OCP Single Objective Mathematical Models

When formulating the OCP problem using the minimization objective, the nature of the problem becomes identical to the set covering problem (SCP). Given that a binary matrix A consists of a number of rows m, a number of columns n, and a cost c_j associated to each column j, SCP aims to find a subset of columns that minimizes the total cost while ensuring each row i is covered by at least one column j. This process can be applied to OCP as well. Given that a coverage matrix A consists of a number of samples m, a number of candidates n, and a cost c_j associated to each candidate j, OCP aims to find a subset of candidates that minimizes the total cost, such that each sample i is covered by at least one candidate j. In this paper, we formulate our OCP problem as an SCP. First, we introduce a binary decision variable x_j, where $x_j = 1$ if candidate j is selected; Otherwise, $x_j = 0$. The complete OCP formulation can be found in Model 5.1.

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad \forall i \in \{1, \ldots, m\} \\ & x_j \in \{0, 1\}, \qquad j \in \{1, \ldots, n\} \end{aligned} \tag{5.1}$$

There is one issue with the formulation in Model 5.1. As discussed in Section 5.2, our problem instances contain a number of samples and a number of candidates, and there is no cost associated with each candidate. This means that our objective must be to

minimize the number of candidates, instead of the cost. To address this issue, we resort to a variant of SCP, called the Unicost Set Covering Problem (USCP). In USCP, the cost is not a factor as it is assumed to be unified for all candidates. Thus, our objective would be to minimize the number of candidates, such that each sample is covered by at least one candidate. The updated OCP model can be referred to in Model 5.2. The only difference between Model 5.1 and Model 5.2 is that we do not include the cost parameter c_j in Model 5.2.

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad \forall i \in \{1, \ldots, m\} \\ & x_j \in \{0, 1\}, \qquad j \in \{1, \ldots, n\} \end{aligned} \tag{5.2}$$

We can use client C as a hypothetical example for illustrative purposes. Client C is a small company that wants to install some cameras in the CEO's office to monitor the CEO's desk as it contains some confidential documents that should not be seen by anyone else. The client shared a camera placement plan, which includes the possible positions of the cameras (candidates) and the areas that they want them to be fully monitored (samples). In addition, the client informed us that the cost of placing cameras in any of the suggested locations is always the same. Based on the given camera placement plan, a visibility matrix, called matrix V, is created. In matrix V, columns represent candidates, and rows represent the samples that must be covered by a subset of those candidates. In this example, cost is unified; hence, we can use the USCP formulation from Model 5.2 to solve this problem.

$$V = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Before solving this problem, we can analyze this visibility matrix first. Candidate 1 (column 1) covers sample 1 (row 1) and sample 3 (row 3), candidate 2 (column 2) covers

sample 2 (row 2) only, candidate 3 (column 3) covers sample 1 (row 1) and sample 4 (row 4), and candidate 4 (column 4) covers sample 2 (row 2) and sample 3 (row 3). Likewise, sample 1 (row 1) can be covered by candidate 1 (column 1) and candidate 3 (column 3), sample 2 (row 2) can be covered by candidate 2 (column 2) and candidate 4 (column 4), sample 3 (row 3) can be covered by candidate 1 (column 1) and candidate 4 (column 4), and sample 4 (row 4) can be covered by candidate 3 (column 3) only. Solving this optimization problem using model 5.2 gives the following result: The minimum number of candidates that can be used to cover all the samples is two. These candidates are: candidate 3 (column 3) and candidate 4 (column 4). Looking back at matrix V, there is no other similar or better solutions, which means this one indeed is the optimal solution and client C should use it. However, there could other feasible solutions that are not optimal. For instance, selecting candidate 1 (column 1), candidate 2 (column 2), and candidate 3 (column 3) achieves full coverage of the samples (rows) of matrix V. This solution is feasible as all the samples are covered. That being said, this solution costs more that the first solution since three candidates are used to achieve the same coverage outcome, which would not make sense for the client to adopt. Thus, we can conclude that the solution of using two candidates, namely candidates 3 and 4, is the only optimal solution.

5.3.2 OCP Bi-Objective Mathematical Models

In this subsection, we transform Model 5.2 into a bi-objective model. Firstly, a binary matrix A, which consists of a number of samples m and a number of candidates n, is given. Secondly, two binary decision variables, namely, x_j and y_i, are given. Similar to models 5.1 and 5.2, $x_j = 1$ if candidate j is selected; otherwise, $x_j = 0$. y_i, on the other hand, represents the coverage decision variable. This means if sample i is covered by at least one candidate, y_i would be 1; otherwise, y_i would be 0. Therefore, the bi-objective OCP problem aims to minimize the number of cameras and maximize the coverage, simultaneously, while ensuring that if a sample i is selected ($y_i = 1$), all the relevant candidates are also selected. The bi-objective OCP problem can be refereed to in Model 5.3. The next subsection explains how this bi-objective OCP problem is tackled.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} x_j \\
\text{maximize} \quad & \sum_{i=1}^{m} y_i \\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq y_i, \quad \forall i \in \{1, \ldots, m\} \\
& x_j \in \{0, 1\}, \qquad j \in \{1, \ldots, n\} \\
& y_i \in \{0, 1\}, \qquad i \in \{1, \ldots, m\}
\end{aligned}
\tag{5.3}
$$

5.3.3 OCP ϵ-Constraint Mathematical Models

In this subsection, we demonstrate how the ϵ-constraint is utilized to address the bi-objective OCP that is represented in Model 5.3. Before jumping into the mathematical formulations, a brief introduction to the ϵ-constraint method is given. ϵ-constraint is one of, if not the most, well-known techniques used to tackle general multi-objective problems. What makes this method different from other popular multi-objective methods, such as the weighted sum method, is that ϵ-constraint transforms all objectives, except for one, into constraints. The bound of each constraint is represented by a value of epsilon ϵ. Since all objectives except for one are transformed into constraints, the bi-objective problem is converted into several single objective sub-problems. These sub-problems are solved by systematically changing the values of ϵ until it reaches a certain limit. Both the starting value of ϵ as well as the limit of the bound are selected based on the nature of each problem instance. Each sub-problem's output represents a feasible solution, which consists of the selected value ϵ and the objective value. Once all the sub-problems are solved and the feasible set is obtained, all the efficient solutions are identified and the complete efficient frontier is generated.

One of the main reasons the ϵ-constraint method was selected for this study is that it guarantees that the complete efficient frontier will be generated, unlike other popular methods that were considered, such as the weighted sum method. In the weighted sum method, both objectives would be combined into one objective, where each objective would be assigned a specific weight. This would result in a new single-objective problem that consists of both objectives. Despite the fact that the weighted sum method may seem simpler to implement, it does not guarantee a complete efficient frontier, which

is one of the reasons ϵ-constraint method was adopted in this study, as stated earlier. Another reason why the ϵ-constraint method was selected instead of the weighted sum method, is the fact that it is easy to assign a value to ϵ for the former, where as it is a challenge to determine the weights for the latter. To elaborate on that, the value of ϵ can be easily assigned for our specific problem, which will be explained more later; while the weights can be subjective or case-dependent. For more details regarding the ϵ-constraint method, readers are advised to refer to [15].

After we have introduced the ϵ-constraint method, now we can discuss how it is going to be used to address our bi-objective OCP problem instances. Since our OCP problem is bi-objective, which means that it contains only two objectives, we only need to transform one objective into a constraint; Consequently, the other objective will serve as the main objective. As a result, our bi-objective OCP problem can be formulated in two opposite ways when using the ϵ-constraint method. One way with transforming the coverage objective into a constraint, while keeping the candidates objective as the main and only objective. The other way with transforming the candidates objective into a constraint, while keeping the coverage objective as the main and only objective. Both formulations are presented by models 5.4 and 5.5, respectively.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} x_j \\
\text{subject to} \quad & \sum_{i=1}^{m} y_i \geq \epsilon \\
& \sum_{j=1}^{n} a_{ij} x_j \geq y_i, \quad \forall i \in \{1, \ldots, m\} \\
& x_j \in \{0, 1\}, \qquad j \in \{1, \ldots, n\} \\
& y_i \in \{0, 1\}, \qquad i \in \{1, \ldots, m\}
\end{aligned}
\tag{5.4}
$$

Model 5.4 represents the first ϵ-constraint formulation, where the main objective is to minimize the number of cameras. This means that the coverage objective is transformed into a constraint. In Model 5.4, the first constraint represents the coverage objective. This constraint is given a bound value of ϵ. Once the value of ϵ is set, the problem is solved several times until ϵ reaches the limit. If we take problem instance AC_01 as an example, the limit of ϵ can be set to 605, which is the total number of samples

available. Once the limit of ϵ is set, the first sub-problem can be solved by setting $\epsilon = 1$, producing the first feasible solution. Once the first feasible solution is obtained, a new sub-problem can be solved. For the next sub-problem, we increase the ϵ value by 1 and set $\epsilon = 2$. This new sub-problem is then solved, and a new feasible solution is obtained. This process continues until $\epsilon = 605$ and all the sub-problems are solved, obtaining all the feasible solutions. From those feasible solutions, efficient solutions are identified and the complete efficient frontier is generated.

$$\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{m} y_i \\
\text{subject to} \quad & \sum_{j=1}^{n} x_j \leq \epsilon \\
& \sum_{j=1}^{n} a_{ij} x_j \geq y_i, \quad \forall i \in \{1, \ldots, m\} \\
& x_j \in \{0, 1\}, \qquad j \in \{1, \ldots, n\} \\
& y_i \in \{0, 1\}, \qquad i \in \{1, \ldots, m\}
\end{aligned} \tag{5.5}$$

The second ϵ-constraint formulation is represented by Model 5.5. In this model, the main objective is to maximize the area coverage, while the number of cameras objective is transformed into the first constraint. Similar to Model 5.4, the value of ϵ is set using the number of candidates available in the given problem instance. For example, the limit of ϵ for problem instance AC_01 would be set to 2904. Once this is set, all the sub-problems are solved starting from $\epsilon = 1$ until the value of ϵ reaches the limit ($\epsilon = 2904$), obtaining all the feasible solutions. Similar to Model 5.4, the efficient solutions are filtered out and the complete efficient frontier is found.

After introducing the bi-objective OCP model and the two ϵ-constraint formulations, we can, again, take client C, as an example. Assume that client C informed us that they decided to be more flexible and are willing to to accept a solution that does not guarantee that all the spots on their CEO's desk are monitored, if it meant they can use less cameras, and therefore pay less money. In other words, they want to receive other camera placement solutions, and then make a decision on which one they are willing to adopt. This means we can solve this problem as as a bi-objective problem. Looking

TABLE 5.3: A list of the alternatives produced by solving the bi-objective OCP problem represented by matrix V (Efficient solutions are in bold)

Solution	Candidates	Uncovered Samples	ϵ Value
a	1	3	1
b	**1**	**2**	2
c	2	1	3
d	**2**	**0**	4

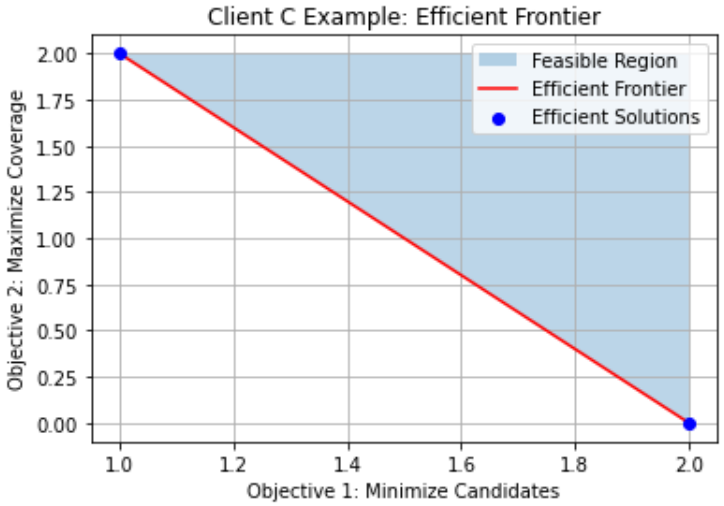

FIGURE 5.2: Efficient frontier produced by solving the bi-objective OCP problem represented by matrix V

back at matrix V, we have 4 samples and 4 candidates available and we want to find all the efficient solutions of this problem using the ϵ-constraint method. The first step would be to select which formulation of ϵ-constraint to use. In this case, we are going to select model 5.4, where the coverage objective is transformed into a constraint. The second step would be to select the limit of ϵ. In this case, we can set it as $\epsilon = 4$, which is the maximum number of samples. The third step would be to systematically solve all the sub-problems, starting from $\epsilon = 1$ and then adding 1 to the value of ϵ each time we solve a new sub-problem. We stop when ϵ reaches 4, which is the predetermined upper limit. To elaborate more, the first time we solve the problem, the value of ϵ will be 1 and the solution of this problem would be added to the feasible set. In this case, the solution is 1 and the first alternative is to use 1 candidates to have only 1 sample covered (3 samples uncovered). After that, we set the value of ϵ to 2, solve the new problem, and add the new results to the feasible set. This process continues until ϵ reaches and solves the upper limit (i.e, $\epsilon = 4$). Table 5.3 lists all the alternatives obtained from solving this

example. Once the feasible set has been obtained, the efficient solutions can be identified and the efficient frontier can be generated. This would be the fourth and final step. The efficient solutions are formatted in bold in Table 5.3 and the efficient frontier is presented in Figure 5.2. The highlighted blue region in the latter contains feasible solutions, while the two dark blue points represent the efficient solutions. Based on these results, client C can now make a decision based on their priorities. If a complete coverage of the desk is essential no matter what the cost is, then efficient solution d would be their choice. If they want to save some money and are willing to adopt a camera placement plan that does not cover every spot on the CEO's desk, then efficient solution b would be their choice. In the next sections, this process will be applied to much bigger problems and the results will be presented and discussed.

5.4 Initial Results

Up to this point, this paper has introduced the camera placement problem, the bi-objective version of this problem, and how to solve the latter using the ϵ-constraint method. This section discusses the first set of results obtained by using the ϵ-constraint method to address some of the OCP problem instances introduced in Section 5.2. The main solver which was used to obtain the results was Gurobi within Python. In addition, the following machine was used for solving all the problem instances: Intel(R) Core(TM) i5-8500T CPU @ 2.10GHz 2.11GHz with 8.00GB RAM.

TABLE 5.4: A summary of the optimal results obtained from solving the bi-objective OCP problem instances using the first formulation of the ϵ-constraint method

Instance	Samples	Candidates	Efficient solutions	Time in seconds (minutes)
AC_01	605	2904	7	5571.55 (92.85)
AC_02	2205	10584	4	344842.71 (5747.37)
AC_03	4805	23064	-	-
AC_04	8405	40344	-	-

The first set of results can be summarized in Table 5.4. The table contains the instance name, number of samples, number of candidates, number of efficient solutions, and the time needed to obtain those solution. The results were obtained by solving the problem instances using Model 5.4. Initially, a time limit of two weeks was enforced when solving each instance. We ran the experiment for the first 4 academic problem instances as they are the smallest and least complicated to solve, according to our first OCP paper [65].

TABLE 5.5: A list of the efficient solutions produced by solving AC_01

Solution	Candidates	Uncovered Samples
1	7	0
2	6	1
3	5	26
4	4	56
5	3	127
6	2	224
7	1	394

TABLE 5.6: A list of the efficient solutions produced by solving AC_02

Solution	Candidates	Uncovered Samples
1	4	0
2	3	9
3	2	200
4	1	826

In that study, we tried to solve all the 69 single objective OCP problem instances using exact methods with a time-limit of 3 hours. The only optimal solutions obtained from that study were from AC_01, AC_02, AC_03, and AC_04; thus, they were also used for experimentation in the current study.

What can be observed from Table 5.4 is that only AC_01 and AC_02 have managed to produce some results in the table. The efficient frontiers of both problem instances can be referred to in Figure 5.3, where the highlighted blue regions contains feasible solutions, while the dark blue points represent the efficient solutions. To obtain all the efficient solutions, and to produce the respective efficient frontiers, AC_01 and AC_02 needed about 92 minutes and 5,747 minutes, respectively. We can focus on AC_01 for illustrative purposes. In AC_01, 7 efficient solutions were obtained, which can be found in Table 5.6. From those results, decision makers can select the plan (i.e., efficient solution) that suits their specific case. If they do not care about complete coverage and do not want to use more than 4 cameras, then solution number 4 or even 5 could be their choice. An interesting observation from Table 5.6 is that efficient solution 2 is only one sample short from achieving complete coverage with one less candidate than efficient solution 1. If the spot represented by that specific uncovered sample is not too important to the clients, then they can adopt efficient solution 2 as it is cheaper than solution 1 and offers almost the same coverage quality as efficient solution 1. This is something that

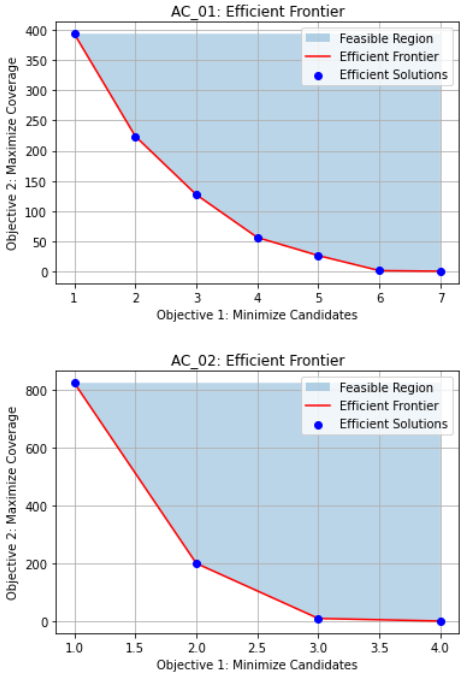

FIGURE 5.3: The graph on top of this figure represents the efficient frontier produced by solving AC_01, while the graph on the bottom represents the efficient frontier produced by solving AC_02

would not appear when solving this problem instance as a single-objective problem that produces one optimal solution. This is why, generally, multi-objective optimization can be more useful when making real-life business decisions. The efficient frontier, generated by solving AC_01, is represented by the top graph in Figure 5.3, where the highlighted area contains feasible solutions.

As for instances AC_03 and AC_04, two weeks time-limit for each was not enough to provide all the results. We believe that the cause of the time consumption issue is the size of each of these problem instances. Moreover, each problem has to be solved multiple times, which can be time-consuming as well. To elaborate more, in the ϵ-constraint method, each problem is converted into a number of single-objective sub-problems based on the starting value of ϵ, which is also the number of samples in the case of Model 5.4. For example, AC_01, AC_02, AC_03 and AC_04 were converted into 605, 2205, 4805, and 8405 sub-problems, respectively. If AC_02 needed 5,747 minutes to be solved, then AC_03 and AC_04 might need much more time as they are bigger in size and contain more samples. An even bigger and potentially much more time-consuming example is instance AC_32. From Table 5.1, we can see that instance AC_32 contains 804,005, which means we would need to solve 804,005 sub-problems. For an instance as big as

this one, solving even just one sub-problem can take weeks or even months. As a results, we decide to adopt a few techniques to reduce the size of these problems instance.

5.5 Addressing the Size of the Problem Instances

In Section 5.4, we talked about our first OCP study, where we experimented in solving all the 69 problem instances using single-objective exact methods with a time-limit of 3 hours. We also mentioned that the only optimal solutions found from in that study were from AC_01, AC_02, AC_03, and AC_04. To deal with this issue, we conducted a second study, where we addressed the large size of most of these instances by reducing the size of all 69 problems. To achieve that, we employed and improved on a three instance reduction techniques, which were introduced by [64] and were cited in [63]. These techniques were created to address set covering problems. Since our bi-objective OCP problem was formulated as a bi-objective set covering problem, these techniques were adopted and they are: the essential site method, the dominated columns method, and the row domination method. These reduction techniques are explained next.

Firstly, essential site states that if there exists a row in a matrix where that row is covered by one column only, then said column is forced to be part of the solution and every row that is covered by this column is removed from the matrix. Secondly, dominated columns is a reduction method that eliminates all subsets in a matrix. In other words, it removes any column that is dominated by other columns. This means if a certain column covers a number of rows, and said column is compared with another column that covers the same rows in addition to a few more, then the former column can removed from the matrix. Lastly, row domination acts exactly as the opposite of dominated columns, where supersets are eliminated from a matrix. To elaborate more, if a certain row is covered by a number of columns, and it was compared to another row that is covered by the same columns in addition to a few more, then the latter row is removed from the matrix. In other words, we eliminate the dominant rows.

Using these three techniques, the reduction algorithm works as follows: We start by initializing the reduction algorithm. Then, we check if any essential site exists in a given matrix. If yes, then the essential site method is activated; otherwise, no changes occur. Subsequently, we check if there are any subsets. If yes, the dominated columns method is activated; otherwise, no changes occur. After that, we check if there are any supersets. If yes, the row domination method is activated; otherwise, no changes occur. Finally, we

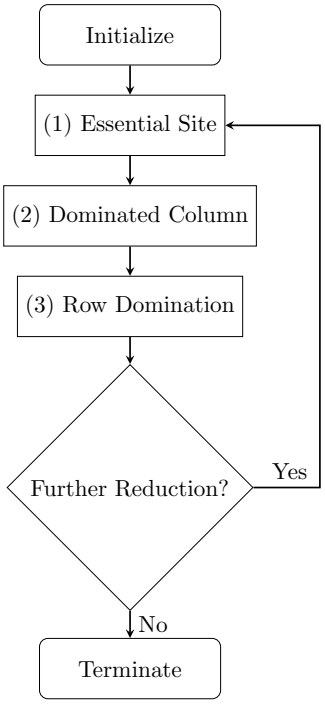

FIGURE 5.4: A summary of the reduction algorithm used in Chapter 4

check if any further reduction can be made. If yes, we repeat the previous three steps. If no, the algorithm is terminated, and the reduced problem instance is provided. This process keeps going until no further reduction is possible. A summary of the reduction algorithm is provided in Figure 5.4.

We can go back to the example of client C and use matrix V to apply those reduction techniques. First, we check for any essential sites. In this case, row 4 is covered by column 3 only. As a result, column 3 is added to the solution of the single-objective set covering problem, and rows 1 and 4 are removed. To make this example easier to follow, we can convert all the values in row 1 and 4 to '0' instead of removing them from the matrix. This creates a new matrix, which we can call matrix $v1$.

$$v1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Now we can move on to the next step, which is investigating if there are any subsets, using the dominated columns method. In matrix $v1$, we can see that columns 1 and 2 are each subsets of column 4. Therefore, we should remove columns 1 and 2 from the matrix. Again, instead of doing that, we will convert all the values in said columns to '0'. This creates another matrix, which we can call matrix $v2$.

$$v2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

In the next step, we will use the row domination method and investigate if there are any supersets in the matrix. Here, rows 2 and 3 are both supersets to each other, which means we can remove either of them. For this example, we will remove row 3 by converting all the value to 0. The new matrix can be called $v3$.

$$v3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

The final step would be to check if the matrix can be further reduced. In this case, the answer is yes. Row 2 is covered by only one column, which is column 4. This means we can use the essential site method to add column 4 to the solution and remove row 2 from the matrix. The final matrix can be called matrix $v4$.

$$v4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

TABLE 5.7: Academic problem instances reduction results

Instance	Samples	Candidates	Reduced Samples	Reduced Candidates	Time in seconds
AC_01	605	2904	600	1292	0.57
AC_02	2205	10584	225	404	2.84
AC_03	4805	23064	64	92	14.09
AC_04	8405	40344	513	2580	46.72
AC_05	13005	62424	1929	6636	110.13
AC_06	18605	89304	3069	12292	239.48
AC_07	32805	157464	5909	28452	1189.01
AC_08	51005	244824	9549	51012	3156.02
AC_09	73205	351384	-	-	-
AC_10	605	2904	605	1672	0.34
AC_11	2205	10584	2205	7352	1.47
AC_12	4805	23064	4805	17032	3.92
AC_13	8405	40344	8405	30712	7.92
AC_14	13005	62424	13005	48392	10.67
AC_15	18605	89304	18605	70072	13.69
AC_16	32805	157464	32805	125432	24.46
AC_17	51005	244824	51005	196792	43
AC_18	73205	351384	73205	284152	54.61
AC_19	99405	477144	99405	387512	82.15
AC_20	129605	622104	129605	506872	98.32
AC_21	163805	786264	163805	642232	146.97
AC_22	202005	969624	202005	793592	194.67
AC_23	244205	1172184	244205	960952	256.96
AC_24	290405	1393944	290405	1144312	426.8
AC_25	340605	1634904	340605	1343672	688.34
AC_26	394805	1895064	394805	1559032	936.63
AC_27	453005	2174424	453005	1790392	943.46
AC_28	515205	2472984	515205	2037752	1338.29
AC_29	581405	2790744	581405	2301112	1657.96
AC_30	651605	3127704	651605	2580472	2014.11
AC_31	725805	3483864	725805	2875832	2217.76
AC_32	804005	3859224	804005	3187192	1231.94

Now we check matrix $v4$ for any dominated column cases then we move to the row domination method. In this case, neither cases occur. Finally, we check again if any more reduction can be made. Here, matrix $v4$ only contains zeros, so no further reduction can be made. Since matrix $v4$ is empty, this means we do not need to solve the problem. Moreover, the solution has already been obtained during both essential site stages. The two columns that were added to the solution were columns 3 and 4. To prove that this reduction method works perfectly, we can go back to matrix V in Section 5.3, where we have already established that the optimal solution to this problem was to select candidate 3 (column 3) and candidate 4 (column 4) to achieve full coverage.

This reduction algorithm was applied to the 69 problem instances and a summary of the reduction results can be viewed in Table 5.7 for the academic instances and Table 5.8 for the real-world instances. Both tables contain the original size, the reduced size,

TABLE 5.8: Real-world problem instances reduction results

Instance	Samples	Candidates	Reduced Samples	Reduced Candidates	Time in seconds
RW_01	153368	32430	28675	22754	44.89
RW_02	285698	56132	69844	42310	178.51
RW_03	161099	32040	27066	19713	43.52
RW_04	304655	59137	70727	39639	231.89
RW_05	206900	34568	26271	19813	45.87
RW_06	380420	65691	80140	45363	252.26
RW_07	214889	42046	35373	26965	53.06
RW_08	382651	77986	90204	54846	389
RW_09	206816	39003	31778	24238	48.22
RW_10	368114	71323	82511	48189	271.56
RW_11	82437	15632	14820	10334	21.97
RW_12	136555	28109	37404	20696	123.41
RW_13	293138	61741	51908	40503	94.97
RW_14	81062	14916	13997	9887	20.26
RW_15	141309	27008	37332	20288	96.75
RW_16	105829	21063	15225	12726	23.57
RW_17	180453	35635	37308	25465	129.21
RW_18	79947	14423	13144	9575	25.18
RW_19	141114	26483	33426	18976	135.97
RW_20	332300	50284	40146	29811	63.01
RW_21	654068	90050	105051	61268	534.86
RW_22	83835	17203	12162	10673	18.58
RW_23	142326	31038	32832	22300	73.45
RW_24	201967	33880	27854	21025	39.27
RW_25	375680	59851	69866	40434	195.82
RW_26	105566	18043	12990	10366	20.81
RW_27	181090	32669	36071	22149	126.5
RW_28	136755	27838	20317	18239	24.65
RW_29	273964	49267	52044	36163	114.79
RW_30	263518	49354	48499	34016	92.39
RW_31	472660	87248	116789	63266	664.57
RW_32	124289	30189	22405	20277	31.68
RW_33	229231	55000	56706	41191	168.96
RW_34	134479	27329	21793	17419	32.78
RW_35	238546	47590	52324	33072	152.5
RW_36	135043	28162	22055	17751	40.46
RW_37	238492	50702	57305	35563	254.11

and the time needed to reduce each problem. In most of these instances, the difference between the size of the original problem and the reduced problem is quite significant. In addition, most instances were successfully reduced within seconds or a few minutes in the worst case scenarios. For more details regarding the reduction algorithm, how it works, and the reduction results, readers are advised to refer to Chapter 4.

5.6 Improved Results

After reducing our OCP problem instances, we can try again solving the problem instances discussed in Section 5.4. The new results are presented in Table 5.9. We can first focus on instance AC_01. Before applying the reduction techniques on this instance, obtaining all the efficient solutions took more than 90 minutes (see Table 5.4). After

TABLE 5.9: A summary of the post-reduction optimal results obtained from solving the bi-objective OCP problem instances using the first formulation of the ϵ-constraint method

Instance	Samples	Candidates	Efficient solutions	Time in seconds (minutes)
AC_01	605	2904	7	2638.03 (43.96)
AC_02	2205	10584	4	23393.83 (389.89)
AC_03	4805	23064	3	190314.49 (3171.90)
AC_04	8405	40344	5	2460077 (41001.28)

applying the reduction algorithms, it took less than 45 minutes, which is more the half the original time, to achieve the same results. This is a noticeable improvement in terms of time-consumption. Now we can move on to instance AC_02. Originally, AC_02 took more than 5,700 minutes to obtain all the efficient solutions, whereas now it takes only 389 minutes after having applied the reduction algorithm. In other words, it used to take us about four days and now it takes us only six and a half hours to achieve the same result. The difference here is much more significant. This was achieved because instance AC_02 moved from containing 2205 samples and 10,584 candidates to 225 samples and 404 candidates. The difference in size here is tremendous, and that is why it takes much less time now to solve this instance. This is because each time we solve a sub-problem in AC_02, we only have to navigate through 225 samples and 404 candidates instead of 2205 samples and 10,584 candidates. It is important to note that the reduction algorithm does not remove any efficient solutions. This is true because the reduction algorithm eliminates unnecessary columns only, which would never be part of the efficient frontier. The results, thus far, prove that, as we managed to produce the same exact efficient frontiers for AC_01 and AC_02 before and after applying the reduction algorithm. The only change here is the time needed to obtain the same results (i.e., efficient frontiers).

In Section 5.4, we only managed to obtain results for two instances. After we reduced the size of the problem instances, we managed to obtain all the efficient solutions for instance AC_03 in less than two weeks. In fact, it took about two days to produce those results. This could not be achieved within the two-weeks time-limit using the original problem. As for AC_04, we managed to obtain all the efficient solutions in around 28 days. This exceeds the time-limit we set before implementing the reduction techniques; however, considering the significant improvements in terms of time needed to solve the first three academic instances, solving AC_04 without implementing the reduction techniques should take much longer than 28 days. All these facts prove that our reduction algorithm transformed ϵ-constraint into a more effective multi-objective

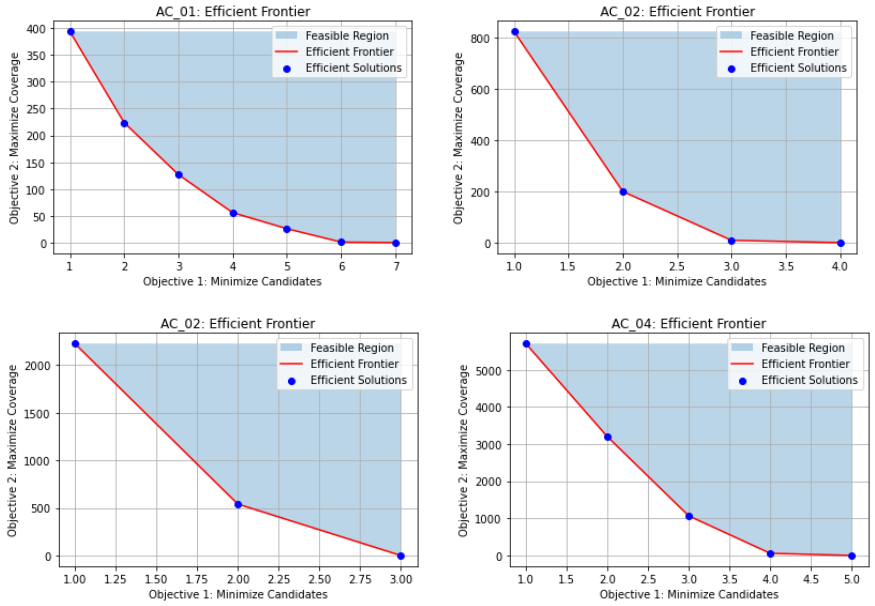

FIGURE 5.5: Efficient frontiers of the reduced instances, using the first formulation of the ϵ-constraint method

technique. The efficient frontier of each one of these reduced instances is given in Figure 5.5, where the highlighted blue regions contains feasible solutions, while the dark blue points represent the efficient solutions. It is worth pointing out that both efficient frontiers of the reduced problems AC_01 and AC_02 are identical to the efficient frontiers obtained before reducing the problem instances.

5.7 Further Experimentations

After proving that our reduction algorithm significantly reduces the time needed to obtain the complete efficient frontier, the next step is to solve our OCP instances using the second formulation of the ϵ-constraint method. Presented in Model 5.5, the main objective of the second formulation is to maximize the area coverage, while the number of candidates objective is transformed into the first constraint. The results of solving the same academic instances are given in Table 5.10.

As can be observed in Table 5.10, the time needed to obtain the same efficient solutions has dramatically dropped, both before and after reducing the instances. To elaborate more, before implementing the reduction algorithms, the time needed to obtain all the

TABLE 5.10: A summary of the results obtained from solving the bi-objective OCP problem instances using the second formulation of the ϵ-constraint method

Instance	Samples	Candidates	Efficient Solutions	Before Reduction Time (seconds)	After Reduction Time (seconds)
AC_01	605	2904	7	48.50	42.08
AC_02	2205	10584	4	76.71	39.74
AC_03	4805	23064	3	368.48	130.86
AC_04	8405	40344	5	2071.54	1215.92

efficient solutions of instance AC_01 for the first formulation is about an hours and half (5,571.55 seconds), compared to 48.5 seconds for the second formulation. Similarly, the time needed to obtain all the efficient solutions of instance AC_02 using the first formulation is almost 4 days (344,842.71 seconds), compared to 76.71 seconds using the second formulation. As for AC_03 and AC_04, we could not even obtain the efficient solutions within two weeks; whereas, it only took 368.48 seconds and 2,071.54 seconds, respectively, using the second ϵ-constraint formulation. After implementing the reduction algorithms, the time needed to obtain all the efficient solutions of AC_01, AC_02, AC_03, and AC_04 using the the first formulation of the ϵ-constraint method dropped when using the second formulation. More specifically, the time (in seconds) dropped from 2,638.03, 23,393.83, 190,314.49, and 2,460,077 to 42.08, 39.74, 130.86, and 1,215.92, respectively.

The difference in time is quite significant, but it should not be a surprise. To elaborate on that, in the second formulation, we convert the candidates number objective into a constraint. This means that we solve this problem systematically based on the values of the candidates (ϵ). When we start with $\epsilon = 1$, the problem will maximize the coverage using 1 candidate only, which would make that solution efficient as it achieves that maximum coverage possible using 1 candidate. Then, we continue the process by increasing the value of ϵ by 1, and solving the new sub-problem with $\epsilon = 2$. Similarly, the new solution is also efficient because it achieves the maximum coverage given the selected number of candidates. This process continues until the coverage objective cannot be improved any further. In other words, we can terminate the algorithm when we reach the maximum coverage possible (i.e., number of samples). This is because all the solutions of the remaining sub-problems will have the same exact coverage objective value, but the number of candidate will be higher. This means all the remaining solutions are not efficient (i.e., alternatives). Therefore, the algorithm can be terminated, and the efficient solutions can be obtained. This is the main reason why the time needed to

solve our OCP instances using the second formulation is much lower; we do not need to solve all the sub-problems.

Lastly, it is worth noting that even when using the second ϵ-constraint formulation, solving the problems after implementing the reduction algorithms is less time consuming, especially when the instances sizes get larger (see Table 5.10). This proves the effectiveness of our reduction algorithm on both formulations of the ϵ-constraint method.

5.8 Conclusions and Recommendations

This paper introduced an effective ϵ-constraint approach to solve 69 bi-objective OCP problem instances. Firstly, an introduction to as well as the background of camera placement was given. This was followed by a brief description of the problem instances used in this study. Then, all the relevant OCP formulations were introduced, where each formulation was accompanied by a detailed description. After that, the initial results were provided and discussed. This was followed by a discussion on using a few reduction techniques to address the size of the problem instances, which proved to be affective later in the paper.

Even though this paper proved the impact of the reduction techniques on the bi-objective OCP problem, more experiments should be considered. The reason behind this is that it still takes a significant amount of time for most instances. This is because of the nature of these instances, as we believe they were created to be challenging. This could be a good thing, as researches could get inspired by trying to come up with novel ways to improve the quality of the results of these instances. For future work, other multi-objective techniques could be considered, such as the weighted sum method, goal programming, and lexicographic ordering, as they might be more effective in obtaining the results, especially when it comes to the time consumed. Generally, since time consumption is the main concern, multi-objective heuristic and metaheuristic techniques could be helpful in obtaining more results.

Chapter 6

Conclusion

In this chapter, a summary of the thesis and a discussion of the limitations as well as possibilities for future work are given.

6.1 Thesis Summary

This thesis tackled the OCP problem using various optimization techniques. OCP is an $\mathcal{NP}$-hard combinatorial optimization problem, where the goal is to find the cheapest camera placement plan, such that the given location is completely covered by said plan. OCP can be formulated in two possible ways. The first one is to maximize the area coverage, given that a minimal camera cost is used; while the second one is to minimize the cost of cameras, given that complete coverage is achieved. When adopting the latter, the OCP problem can be formulated as a set covering problem, which is another $\mathcal{NP}$-hard combinatorial optimization problem. This work exploited the connection between OCP and SCP, as not many studies have made use of that connection. It adopted techniques from the SCP literature to address the given OCP problem. Those techniques were employed to tackle 69 OCP instances available at the GECCO 2021 competition on "the optimal camera placement problem and the unicost set covering problem".

6.1.1 Chapter 1 Summary

This thesis started with an introduction to the main idea of the OCP problem in Chapter 1. This included the main research motivation and contribution, which was mainly

inspired by exploiting the connection between OCP and SCP and finding techniques from the latter to solve the former and address its $\mathcal{NP}$-hard nature. This introductory chapter also discussed the the nature of the problem instances used in this study. Firstly, OCP problem did not consider the camera cost as a factor in the decision making process, which is why the OCP problem was formulated a special type of SCP, namely the unicost SCP (USCP). Secondly, there were two types of problem instances to deal with: one is called academic which was created based on made-up data, while the other it called real-world which was created based on real data. Thirdly, each problem instance, whether it is academic or real-world, consisted of two main terms: one is called candidates which represents the cameras (including location and orientation), while the other is called samples which represents 3-dimensional sample points that can be covered by those candidates.

6.1.2 Chapter 2 Summary

In Chapter 2, we provided an extensive introduction to operations research, generally, and optimization, specifically. Various relevant and crucial concepts were discussed in this chapter. After we introduced operations research in general, we introduced the concept of optimization, which included definitions, general formulation of single-objective problems, and common ways to solve it. In the next part of this chapter, we introduced multi-objective optimization (MO), discussed its general formulation, and explained a couple of common MO methods. After that, we discussed the concept of combinatorial optimization and time complexity, respectively, as in this thesis, we are dealing with a classic combinatorial optimization problem, which is called as $\mathcal{NP}$-hard in terms of time complexity. Finally, we introduced the topic of SCP, and briefly discussed the different formulations that were used in this study.

6.1.3 Chapter 3 Summary

In Chapter 3, the main focus was to address the first two research questions, which were introduced in Chapter 1: "Can we formulate our specific OCP problem using SCP?" and "Can we solve the OCP problem using classic exact methods?". We first started this part of the thesis by understanding the relationship between OCP and SCP, which involved studying both concepts individually and finding the connection between them. In this chapter, we formulated OCP as a single-objective USCP, which is a special variant of the well-known SCP. The goal of this problem was to minimize the number of candidates,

such that a full coverage of the samples is achieved. This problem was then solved using classic exact methods and the results obtained showed some promise. However, the results could have been better; therefore, we proposed a few possible reasons behind this. The main issue was the size of our problem instances. As the problems grew in size, it became much more computationally expensive. This is because of the $\mathcal{NP}$-hard nature of this problem. We, then, discussed a few possible ways to approach this problem in future work.

6.1.4 Chapter 4 Summary

In Chapter 4, the main focus was to address the second two research questions: "Does reducing the size of our problem instances help in obtaining the solutions faster?" and "Will the reduction algorithm itself be time consuming? If so, is there a way to fix this issue?". We started this chapter by we resuming studying the connection between OCP and SCP. However, the main goal of this chapter was to improve the results obtained in Chapter 3. This was achieved by addressing the size of the given OCP problem instances. Moreover, classic SCP problem reduction techniques were implemented to address the size issue. The first set of results were very promising as we managed to greatly reduce the size of our problem instances. One issue regarding this reduction algorithm was that it was time consuming to reduce the problem itself. Therefore, a small yet effective modification to the reduction algorithm was implemented. The new results were much more promising as the time needed for reduction was basically negligible. Solving those reduced versions of the OCP problem instance was also significantly faster compared to Chapter 3. This proved that our reduction algorithm does not only work better than other reduction algorithms, but it also helped in providing better OCP results.

6.1.5 Chapter 5 Summary

Finally, in Chapter 5, the focus was to address the last research question: "Can we formulate our OCP problem as a bi-objective OCP and use SCP techniques to tackle it? If so, does the reduction algorithm have an impact on the bi-objective OCP problem?". First, we formulated our OCP problem as a bi-objective problem. Similar to the previous chapters, we used USCP to formulate this problem, where the two objective were to minimize the number of candidates and maximize the number of samples. To address this bi-objective problem, we adopted a popular multi-objective optimization method, namely the ϵ-constraint method. From the first set of results, we noticed how time

consuming it was to solve each problem instance. This makes sense, since using this method means solving the problem multiple times based on the value of ϵ. To address this issue, we used the reduction algorithm introduced in Chapter 4. The new results showed much more promise, as the time needed to solve each problem and obtain all the efficient solutions was dramatically reduced.

Overall, in this thesis, different techniques inspired from the SCP literature were implemented to address our 69 OCP problem instances. Even though the results showed great promise, there are still some limitations in this thesis that must be discussed. In addition, possible ways to approach these OCP problems should also be discussed for future work.

6.2 Limitations and Future Work

As mentioned earlier this thesis has some limitations that must be addressed. Each chapter discussed some specific issues, thus, the overall limitations are discussed here and ways to address them for future work is discussed.

1. Solving the single-objective OCP problem is still time consuming, even after having the problem instances reduced: Since the main issue stems from the fact that this problem is $\mathcal{NP}$-hard, which means the problem is hard to solve optimally using classic exact methods, heuristic and metaheuristic techniques, such as greedy algorithms or genetic algorithms, can be adopted, where near-optimal or even optimal solutions can be obtain in a reasonable time.

2. Solving the multi-objective OCP problem is still time consuming, even after having the problem instances reduced: Multi-objective heuristic and metaheuristic techniques can be adopted to address the bi-objective OCP problem. That being said, the main reason that the bi-objective OCP problem is still time consuming to solve, is that we used the ϵ-constraint method to address it. To elaborate more, using this method means solving the problem multiple times based on the value of ϵ, which naturally is time consuming. For future work, other MO techniques, such as the weighted sum method, can be adopted.

3. Only three reduction techniques are used in this study to create the reduction algorithm: There are many other reduction techniques discussed in the SCP literature

that could potentially reduce our problem instances even more. For future work, these techniques can be experimented with.

4. Generally, not enough results are obtained across the three main studies: As mentioned in the first and second point, we can use other optimization techniques, such as heuristic and metaheuristic, to obtain more results, even if those results may not actually be optimal. But the main issue here is the time factor. These studies had to be conducted within a certain time-frame, which is why we had to set a time-limit for solving these studies. For future work, time might not be a factor, and conducting studies such as this one would get enough time to obtain most, if not all, of the results.

Bibliography

[1] Vasek Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39–41, 1975.

[2] Bill Triggs and Christian Laugier. Automatic camera placement for robot vision tasks. In *Proceedings of 1995 Ieee International Conference on Robotics and Automation*, volume 2, pages 1732–1737. IEEE, 1995.

[3] Stefanos Nikolaidis, Ryuichi Ueda, Akinobu Hayashi, and Tamio Arai. Optimal camera placement considering mobile robot trajectory. In *2008 IEEE international conference on robotics and biomimetics*, pages 1393–1396. IEEE, 2009.

[4] Xing Chen and James Davis. Camera placement considering occlusion for robust motion capture. *Computer Graphics Laboratory, Stanford University, Tech. Rep*, 2 (2.2):2, 2000.

[5] Pooya Rahimian and Joseph K Kearney. Optimal camera placement for motion capture systems. *IEEE transactions on visualization and computer graphics*, 23(3): 1209–1221, 2016.

[6] Azeddine Aissaoui, Abdelkrim Ouafi, Philippe Pudlo, Christophe Gillet, Zine-Eddine Baarir, and Abdelmalik Taleb-Ahmed. Designing a camera placement assistance system for human motion capture based on a guided genetic algorithm. *Virtual reality*, 22:13–23, 2018.

[7] Robert Bodor, Andrew Drenner, Paul Schrater, and Nikolaos Papanikolopoulos. Optimal camera placement for automated surveillance tasks. *Journal of Intelligent and Robotic Systems*, 50:257–295, 2007.

[8] Junbin Liu, Sridha Sridharan, and Clinton Fookes. Recent advances in camera planning for large area surveillance: A comprehensive review. *ACM Computing Surveys (CSUR)*, 49(1):1–37, 2016.

[9] Julien Kritter, Mathieu Brévilliers, Julien Lepagnot, and Lhassane Idoumghar. On the optimal placement of cameras for surveillance and the underlying set cover problem. *Applied Soft Computing*, 74:133–153, 2019. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2018.10.025. URL https://www.sciencedirect.com/science/article/pii/S1568494618305829.

[10] Michael R Gary and David S Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[11] Mathieu Brévilliers, Julien Lepagnot, and Lhassane Idoumghar. Gecco 2021 competition on the optimal camera placement problem (ocp) and the unicost set covering problem (uscp), 2021.

[12] Frederick S Hillier and Gerald J Lieberman. *Introduction to Operations Research*. McGraw-Hill Education, New York, USA, 11th edition, 2021. ISBN 978-1-260-57863-2.

[13] Fotios Petropoulos, Gilbert Laporte, Emel Aktas, Sibel A Alumur, Claudia Archetti, Hayriye Ayhan, Maria Battarra, Julia A Bennell, Jean-Marie Bourjolly, John E Boylan, et al. Operational research: methods and applications. *Journal of the Operational Research Society*, 75(3):423–617, 2024.

[14] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004.

[15] Matthias Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

[16] Christos H Papadimitriou. On the complexity of integer programming. *Journal of the ACM (JACM)*, 28(4):765–768, 1981.

[17] Jerrold Rubin. A technique for the solution of massive set covering problems, with application to airline crew scheduling. *Transportation Science*, 7(1):34–48, 1973.

[18] Alberto Caprara, Matteo Fischetti, Paolo Toth, Daniele Vigo, and Pier Luigi Guida. Algorithms for railway crew management. *Mathematical programming*, 79:125–141, 1997.

[19] Ferdinando Pezzella and Enrico Faggioli. Solving large set covering problems for crew scheduling. *Top*, 5(1):41–59, 1997.

[20] Richard Church and Charles R Velle. The maximal covering location problem. *Papers in regional science*, 32(1):101–118, 1974.

[21] Charles ReVelle, Constantine Toregas, and Louis Falkson. Applications of the location set-covering problem. *Geographical analysis*, 8(1):65–76, 1976.

[22] Rashed Sahraeian and Mohammad Sadeq Kazemi. A fuzzy set covering-clustering algorithm for facility location problem. In *2011 IEEE International Conference on Industrial Engineering and Engineering Management*, pages 1098–1102. IEEE, 2011.

[23] Reza Zanjirani Farahani, Masoud Hekmatfar, Alireza Boloori Arabani, and Ehsan Nikbakhsh. Hub location problems: A review of models, classification, solution techniques, and applications. *Computers & industrial engineering*, 64(4):1096–1109, 2013.

[24] Guy Even, Guy Kortsarz, and Wolfgang Slany. On network design problems: fixed cost flows and the covering steiner problem. *ACM Transactions on Algorithms (TALG)*, 1(1):74–101, 2005.

[25] Dimitrios Zorbas, Dimitris Glynos, Panayiotis Kotzanikolaou, and Christos Douligeris. Solving coverage problems in wireless sensor networks using cover sets. *Ad Hoc Networks*, 8(4):400–415, 2010.

[26] Mahmoud Owais, Mostafa K Osman, and Ghada Moussa. Multi-objective transit route network design as set covering problem. *IEEE Transactions on Intelligent Transportation Systems*, 17(3):670–679, 2015.

[27] Víctor Bucarey, Bernard Fortz, Natividad González-Blanco, Martine Labbé, and Juan A Mesa. Benders decomposition for network design covering problems. *Computers & Operations Research*, 137:105417, 2022.

[28] Alexandre Drouin, Sébastien Giguere, Vladana Sagatovich, Maxime Déraspe, François Laviolette, Mario Marchand, and Jacques Corbeil. Learning interpretable models of phenotypes from whole genome sequences with the set covering machine. *arXiv preprint arXiv:1412.1074*, 2014.

[29] Songjian Lu and Xinghua Lu. An exact algorithm for the weighed mutually exclusive maximum set cover problem. *arXiv preprint arXiv:1401.6385*, 2014.

[30] Yichao Li, Yating Liu, David Juedes, Frank Drews, Razvan Bunescu, and Lonnie Welch. Set cover-based methods for motif selection. *Bioinformatics*, 36(4):1044–1051, 2020.

[31] Egon Balas and Manfred W Padberg. On the set-covering problem. *Operations Research*, 20(6):1152–1161, 1972.

[32] John E Beasley. An algorithm for set covering problem. *European Journal of Operational Research*, 31(1):85–93, 1987.

[33] Giuseppe Lancia and Paolo Serafini. A set-covering approach with column generation for parsimony haplotyping. *INFORMS Journal on Computing*, 21(1):151–166, 2009.

[34] Marcos Antonio Pereira, Luiz Antonio Nogueira Lorena, and Edson Luiz França Senne. A column generation approach for the maximal covering location problem. *International Transactions in Operational Research*, 14(4):349–364, 2007.

[35] Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[36] Laurence A Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

[37] Mutsunori Yagiura, Masahiro Kishida, and Toshihide Ibaraki. A 3-flip neighborhood local search for the set covering problem. *European Journal of Operational Research*, 172(2):472–499, 2006.

[38] Chao Gao, Xin Yao, Thomas Weise, and Jinlong Li. An efficient local search heuristic with row weighting for the unicost set covering problem. *European Journal of Operational Research*, 246(3):750–761, 2015.

[39] John E Beasley and Paul C Chu. A genetic algorithm for the set covering problem. *European journal of operational research*, 94(2):392–404, 1996.

[40] Uwe Aickelin. An indirect genetic algorithm for set covering problems. *Journal of the Operational Research Society*, 53(10):1118–1126, 2002.

[41] Marco Caserta. Tabu search-based metaheuristic algorithm for large-scale set covering problems. *Metaheuristics: Progress in complex systems optimization*, pages 43–63, 2007.

[42] Gary W Kinney, J Wesley Barnes, and Bruce W Colletti. A reactive tabu search algorithm with variable clustering for the unicost set covering problem. *International Journal of Operational Research*, 2(2):156–172, 2007.

[43] Mathieu Brévilliers, Julien Lepagnot, Julien Kritter, and Lhassane Idoumghar. Parallel preprocessing for the optimal camera placement problem. *International Journal of Modeling and Optimization*, 8(1):33–40, 2018.

[44] Julien Kritter, Mathieu Brévilliers, Julien Lepagnot, and Lhassane Idoumghar. On the real-world applicability of state-of-the-art algorithms for the optimal camera placement problem. In *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 1103–1108. IEEE, 2019.

[45] Sungbum Jun, Tai-Woo Chang, and Hyuk-Jin Yoon. Placing visual sensors using heuristic algorithms for bridge surveillance. *Applied Sciences*, 8(1):70, 2018.

[46] Xincong Yang, Heng Li, Ting Huang, Ximei Zhai, Fenglai Wang, and Chen Wang. Computer-aided optimization of surveillance cameras placement on construction sites. *Computer-Aided Civil and Infrastructure Engineering*, 33(12):1110–1126, 2018.

[47] V Anirudh Puligandla and Sven Lončarić. A multiresolution approach for large real-world camera placement optimization problems. *IEEE Access*, 10:61601–61616, 2022.

[48] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[49] Boyu Zhang, Xuebo Zhang, Xiang Chen, and Yongchun Fang. A differential evolution approach for coverage optimization of visual sensor networks with parallel occlusion detection. In *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1246–1251. IEEE, 2016.

[50] Andre L Maravilha, Jaime A Ramirez, and Felipe Campelo. A new algorithm based on differential evolution for combinatorial optimization. In *2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, pages 60–66. IEEE, 2013.

[51] Altahir Abdalla Altahir, Vijanth Sagayan Asirvadam, Nor Hisham B Hamid, Patrick Sebastian, Nordin B Saad, Rosdiazli B Ibrahim, and Sarat C Dass. Optimizing visual surveillance sensor coverage using dynamic programming. *IEEE Sensors Journal*, 17(11):3398–3405, 2017.

[52] MS Sumi Suresh, Athi Narayanan, and Vivek Menon. Maximizing camera coverage in multicamera surveillance networks. *IEEE Sensors Journal*, 20(17):10170–10178, 2020.

[53] Jun-Woo Ahn, Tai-Woo Chang, Sung-Hee Lee, and Yong Won Seo. Two-phase algorithm for optimal camera placement. *Scientific Programming*, 2016, 2016.

[54] Martin Desrochers and François Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation science*, 23(1):1–13, 1989.

[55] Sávio SV Vianna. The set covering problem applied to optimisation of gas detectors in chemical process plants. *Computers & Chemical Engineering*, 121:388–395, 2019.

[56] W Walker. Application of the set covering problem to the assignment of ladder trucks to fire houses. *Operations Research*, 22:275–277, 1974.

[57] Xin-Yuan Zhang, Jun Zhang, Yue-Jiao Gong, Zhi-Hui Zhan, Wei-Neng Chen, and Yun Li. Kuhn–munkres parallel genetic algorithm for the set cover problem and its application to large-scale wireless sensor networks. *IEEE Transactions on Evolutionary Computation*, 20(5):695–710, 2015.

[58] Alberto Caprara, Paolo Toth, and Matteo Fischetti. Algorithms for the set covering problem. *Annals of Operations Research*, 98(1):353–371, 2000.

[59] Zhi-Gang Ren, Zu-Ren Feng, Liang-Jun Ke, and Zhao-Jun Zhang. New ideas for applying ant colony optimization to the set covering problem. *Computers & Industrial Engineering*, 58(4):774–784, 2010.

[60] S Indu, Shreya Srivastava, and Vasundhara Sharma. Optimal camera placement and orientation of a multi-camera system for self driving cars. In *Proceedings of the 2020 4th International Conference on Vision, Image and Signal Processing*, pages 1–5, 2020.

[61] Zhouxing Su, Qingyun Zhang, Zhipeng Lü, Chu-Min Li, Weibo Lin, and Fuda Ma. Weighting-based variable neighborhood search for optimal camera placement. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12400–12408, 2021.

[62] Mathieu Brévilliers, Julien Lepagnot, Lhassane Idoumghar, Maher Rebai, and Julien Kritter. Hybrid differential evolution algorithms for the optimal camera placement problem. *Journal of Systems and Information Technology*, 2018.

[63] Richard L Church and Ross A Gerrard. The multi-level location set covering model. *Geographical Analysis*, 35(4):277–289, 2003.

[64] Justin C Williams and Charles S ReVelle. Applying mathematical programming to reserve selection. *Environmental Modeling & Assessment*, 2:167–175, 1997.

[65] Malek Almousa, Matthias Ehrgott, and Ahmed Kheiri. Exploring the optimal camera placement problem and its relationship with the set covering problem. In *International Conference on Business Analytics in Practice*, pages 295–306. Springer, 2024.

[66] V Anirudh Puligandla and Sven Lončarić. Optimal camera placement to visualize surrounding view from heavy machinery. In *Proceedings of the 2020 2nd Asia Pacific Information Technology Conference*, pages 52–59, 2020.

[67] Maher Rebai, Matthieu Le Berre, Faicel Hnaien, and Hichem Snoussi. Exact biobjective optimization methods for camera coverage problem in three-dimensional areas. *IEEE Sensors Journal*, 16(9):3323–3331, 2016.

[68] Sean A Mochocki, Gary B Lamont, Robert C Leishman, and Kyle J Kauffman. Multi-objective database queries in combined knapsack and set covering problem domains. *Journal of big data*, 8(1):1–21, 2021.

[69] Andrzej Jaszkiewicz. A comparative study of multiple-objective metaheuristics on the bi-objective set covering problem and the pareto memetic algorithm. *Annals of Operations Research*, 131(1):135–158, 2004.

[70] Kostas Florios and George Mavrotas. Generation of the exact pareto set in multi-objective traveling salesman and set covering problems. *Applied Mathematics and Computation*, 237:1–19, 2014.

[71] George Mavrotas and Kostas Florios. An improved version of the augmented ε-constraint method (augmecon2) for finding the exact pareto set in multi-objective integer programming problems. *Applied Mathematics and Computation*, 219(18): 9652–9669, 2013.

[72] SR Arora, Ravi Shanker, and Neelam Malhotra. A goal programming approach to solve linear fractional multi-objective set covering problem. *Opsearch*, 42(2): 112–125, 2005.

[73] Andrzej Jaszkiewicz. Do multiple-objective metaheuristics deliver on their promises? a computational experiment on the set-covering problem. *IEEE Transactions on Evolutionary Computation*, 7(2):133–143, 2003.

[74] Ryan J Marshall, Lakmali Weerasena, and Anthony Skjellum. A parallel meta-solver for the multi-objective set covering problem. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 529–538. IEEE, 2021.