SET COVER PROBLEMS WITH EXPONENTIALLY MANY CONSTRAINTS: LAGRANGIAN RELAXATION AND LOCAL SEARCH ALGORITHMS WITH APPLICATIONS TO CONNECTED DOMINATING SETS

Vitor Mazal Krauss

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Abilio Pereira de Lucena Filho
Luidi Gelabert Simonetti

Rio de Janeiro
Novembro de 2024

SET COVER PROBLEMS WITH EXPONENTIALLY MANY CONSTRAINTS: LAGRANGIAN RELAXATION AND LOCAL SEARCH ALGORITHMS WITH APPLICATIONS TO CONNECTED DOMINATING SETS

Vitor Mazal Krauss

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Abilio Pereira de Lucena Filho
Luidi Gelabert Simonetti

Aprovada por: Prof. Abilio Pereira de Lucena Filho
Prof. Márcia Rosana Cerioli
Prof. Yuri Abitbol De Menezes Frota

RIO DE JANEIRO, RJ – BRASIL
NOVEMBRO DE 2024

Mazal Krauss, Vitor

Set cover problems with exponentially many constraints: lagrangian relaxation and local search algorithms with applications to connected dominating sets/Vitor Mazal Krauss. – Rio de Janeiro: UFRJ/COPPE, 2024.

XVII, 152 p.: il.; 29, 7cm.

Orientadores: Abilio Pereira de Lucena Filho
Luidi Gelabert Simonetti

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2024.

Referências Bibliográficas: p. 142 – 152.

1. set cover. 2. integer programming. 3. combinatorial optimization. 4. lagrangian relaxation. 5. local search. I. Pereira de Lucena Filho, Abilio *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

To my father, in loving memory.

Agradecimentos

Em primeiro lugar, agradeço aos meus pais, a quem dedico este trabalho. Aqui me reservo de qualquer tentativa mal-sucedida de listar as incontáveis razões, e por isso me resumo a simplesmente agradecer, por tudo. Estendo o agradecimento ao meu irmão, aos familiares e amigos com quem tenho o prazer de compartilhar esta jornada.

Agradeço aos meus orientadores, Prof. Abilio Lucena e Prof. Luidi Simonetti, pela oportunidade e pelo privilégio que tem sido trabalharmos juntos. Além dos tantos ensinamentos e ricas contribuições para este trabalho, agradeço por todo o apoio que me deram durante estes anos.

Agradeço à Prof. Márcia Cerioli e ao Prof. Petrucio Viana por terem me acolhido desde o início da minha trajetória acadêmica. Em especial, agradeço pela enorme dedicação e pela contribuição no meu desenvolvimento como matemático e como profissional. Sou grato pelas inúmeras tardes em que trabalhamos juntos, seja escrevendo uma demonstração, ensaiando para uma apresentação ou revisando pela n-ésima vez o mesmo parágrafo. Todos esses ensinamentos foram fundamentais para a escrita e apresentação deste trabalho, e acredito ter aplicado nele o formalismo matemático, a atenção a detalhes e o capricho que aprendi com vocês.

Este trabalho teve apoio financeiro da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior e do Conselho Nacional de Desenvolvimento Científico e Tecnológico.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

SET COVER PROBLEMS WITH EXPONENTIALLY MANY CONSTRAINTS: LAGRANGIAN RELAXATION AND LOCAL SEARCH ALGORITHMS WITH APPLICATIONS TO CONNECTED DOMINATING SETS

Vitor Mazal Krauss

Novembro/2024

Orientadores: Abilio Pereira de Lucena Filho
Luidi Gelabert Simonetti

Programa: Engenharia de Sistemas e Computação

O problema de cobertura de conjunto, denominado em inglês como set cover problem (SCP), é um conhecido problema de otimização $\mathcal{NP}$-difícil e encontra uma variedade de aplicações. Algumas dessas aplicações levam a formulações de SCP com uma quantidade exponencial de restrições. Um importante exemplo é o problema do conjunto dominante conexo mínimo (PCDCM), que encontra aplicações no design de redes de comunicação.

Ao longo das últimas décadas, algoritmos heurísticos para o SCP foram extensivamente investigados e heurísticas estado-da-arte têm se mostrado muito eficazes em produzir soluções de alta qualidade, até mesmo para instâncias de grande escala do SCP possivelmente com milhões de variáveis. Isso sugere que, para solução de problemas como o PCDCM, considerar uma formulação na forma de SCP pode se beneficiar desses algoritmos. No entanto, por mais promissor que possa parecer, esses algoritmos de SCP não foram projetados para lidar de maneira eficiente com uma quantidade exponencial de restrições.

Embora o SCP tenha sido muito estudado, um tratamento especializado para o SCP com um número exponencial de restrições ainda não foi relatado na literatura. Este trabalho visa ser o primeiro a tratar especificamente este caso. Focamos no desenvolvimento de algoritmos heurísticos que possam tanto lidar de maneira eficiente com o número excessivamente grande de restrições quanto ser eficazes na produção de soluções de alta qualidade. Mais especificamente, propomos um algoritmo relax-and-cut e um algoritmo de busca local para o SCP. Nossos algoritmos propostos incorporam um esquema dinâmico para lidar de maneira eficiente com o

grande número de restrições. Isso é feito definindo convenientemente, a cada iteração, um pequeno subconjunto de restrições que são consideradas ativas e tratadas explicitamente, enquanto as restantes são inativas e tratadas implicitamente. Dessa forma, apesar do número exponencial de restrições, a complexidade de tempo de cada iteração é drasticamente reduzida, e os algoritmos não são apenas viáveis, mas também eficientes.

Como exemplo de aplicação dos nossos algoritmos relax-and-cut e de busca local, consideramos a formulação do SCP para o PCDCM e empregamos nossos algoritmos para resolver instâncias comuns na literatura e comparamos nossos resultados com heurísticas estado-da-arte especificamente projetadas para o PCDCM. De acordo com nossos experimentos computacionais, nosso algoritmo é o mais eficiente na produção de soluções de alta qualidade para o PCDCM, sendo o único a encontrar a melhor solução conhecida na literatura para todas as 41 instâncias consideradas, e em geral fazendo isso em um tempo comparável ou até menor.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SET COVER PROBLEMS WITH EXPONENTIALLY MANY CONSTRAINTS: LAGRANGIAN RELAXATION AND LOCAL SEARCH ALGORITHMS WITH APPLICATIONS TO CONNECTED DOMINATING SETS

Vitor Mazal Krauss

November/2024

Advisors: Abilio Pereira de Lucena Filho
Luidi Gelabert Simonetti

Department: Systems Engineering and Computer Science

The set cover problem (SCP) is a well-known $\mathcal{NP}$-hard optimization problem and finds a wide range of applications. Some of these applications lead to SCP formulations with exponentially many constraints. One relevant example is the min connected dominating set problem (MCDSP), which finds applications in communications network design.

Over the past decades, SCP heuristics have been extensively investigated and state-of-the-art SCP heuristics have been shown to be very effective in producing high-quality solutions, even to large-scale SCP instances with up to millions of variables. This suggests that SCP formulations of problems such as MCDSP can benefit from these algorithms. However, promising as it may seem, these aforementioned state-of-the-art SCP algorithms are not designed to efficiently handle an exponential amount of constraints.

Although SCP has been extensively investigated, a specialized treatment of SCP with exponentially many constraints has not yet been reported in the literature. This work aims to be the first to specifically address this case. We focus on developing heuristic algorithms that can both efficiently handle the exceedingly large number of constraints as well as being effective in producing high-quality solutions. More specifically, we propose a relax-and-cut and a local-search algorithm for SCP. Our proposed algorithms incorporate a dynamic schema to efficiently handle the large amount of constraints. This is done by conveniently defining at each iteration a small subset of constraints that are considered active and are handled explicitly, whereas the remaining ones are inactive and handled implicitly. In this way, despite the

exponentially many constraints, the time complexity of each iteration is drastically reduced, and the algorithms are not only viable but also efficient.

As an application of our proposed relax-and-cut and local-search, we consider the SCP formulation of MCDSP and employ our algorithms to solve benchmark instances from the literature and compare our results with state-of-the-art heuristics specifically designed for MCDSP. According to our computational experiments, our algorithm is the most efficient in producing high-quality MCDSP solutions, being the only one to find the best known solution reported in the literature for every one of the 41 benchmark instances, and generally doing so in comparable or even a shorter amount of time.

Contents

List of Figures

List of Tables

List of Symbols

List of Abbreviations

BB	branch-and-bound . . . 1
BKS	best known solution . . . 2
BP	binary program . . . 11
CDS	connected dominating set . . . 3
DRC	delayed relax-and-cut . . . 75
ET	enumeration tree . . . 18
IP	integer programming . . . 1
LP	linear programming . . . 1
LR	lagrangian relaxation . . . 2
LS	local-search . . . 2
MANET	mobile ad-hoc network . . . 35
MCDSP	min connected dominating set problem . . . 3
MCEDSP	min connected edge-dominating set problem . . . 4
MCEDSP	min tree-cover problem . . . 4
MCVCP	minimum connected vertex cover problem . . . 3
MDSP	min dominating set problem . . . 3
MP	mathematical program . . . 8
MTCP	min tree cover problem . . . 32
MWCDSP	min weight connected dominating set problem . . . 31

MWCEDSP	min weight connected edge-dominating set	32
MWCVCP	min weight connected vertex cover problem	31
MWDSP	min weight dominating set problem	29
MWEDSP	min weight edge-dominating set problem	30
MWTDSP	min weight total dominating set problem	29
MWVCP	min weight vertex cover problem	29
NDRC	non delayed relax-and-cut	75
NM	neighborhood move	88
OR	operations research	13
RC	relax-and-cut	4
SCP	set cover problem	1
SM	subgradient method	58
StandLag	standard lagrangian formulation for SCP	106
StrongLag	strengthened lagrangian formulation for SCP	106
WANET	wireless ad-hoc network	3
WSN	wireless sensor network	35

Chapter 1

Introduction

The *set cover problem* (SCP) is a well-known mathematical optimization problem and finds a wide range of applications. Due to its theoretical and practical relevance, SCP has been extensively investigated and reported in the literature. In its decision version, SCP has been proven to be $\mathcal{NP}$-complete by KARP [4], whereas its optimization version is $\mathcal{NP}$-hard [5, 6]. SCP is most usually formulated and solved by means of *integer programming* (IP). In the context of IP, the so-called *exact* algorithms comprise those that determine a solution with a certificate of optimality, meaning the method provides a solution and proves that no other better solution exists. The standard exact approach for solving SCP is the *branch-and-bound* method (BB), which relies on enumeration. More specifically, these are most usually linear programming (LP) based BB algorithms. On the other hand, given the proven theoretical complexity, as well as the observed practical difficulty of solving SCP instances exactly, *heuristic* algorithms have gained great relevance as alternative methods for producing high-quality SCP solutions in reasonable time. Contrary to an exact algorithm, a heuristic one produces a solution but provides no certificate of optimality. By resigning the proof of optimality, heuristic algorithms require much less computational effort and time to produce a good solution, but the quality gap between that solution to an optimal one is uncertain. Nevertheless, for many optimization problems, and particularly for SCP, many heuristics have been reported and proven to be very effective in producing solutions of good or optimal quality. Additionally, when complicated or large-scale SCP instances are considered, exact algorithms can become impractical and heuristics are the best or even the only alternative.

The development of SCP heuristics dates back to as early as the 1970s, including the greedy heuristic proposed by CHVATAL [7] in 1979. Following his work, many improvements and new ideas have been incorporated, and the some of the currently most effective greedy heuristics for SCP are based on *lagrangian relaxation* (LR). Both CAPRARA *et al.* [8] and CERIA *et al.* [9] consider the LR formula-

tion of SCP and embody the lagrangian information into the greedy heuristic, thus leading to the so-called *lagrangian heuristics*. Additionally, they make use of the lagrangian information in order to set a *core problem*, consisting of a conveniently defined SCP subproblem reduced from the original one. As evidenced by their reported results, working on a core problem lead to a reduction on the running time and effectively improved the quality of their solutions. Another important line of research is on *local-search* (LS). In general, LS algorithms are heuristic procedures that aim to improve a known solution for an optimization problem. In alternative to a greedy heuristic, which builds a solution from scratch, a local-search procedure takes a known solution and modifies it in search of a better one. In the particular case of SCP, LS has been shown to be very effective. We refer specially to the LS algorithms reported by YAGIURA *et al.* [10] and by GAO *et al.* [11]. In fact, YAGIURA *et al.* [10] reported improvements to the *best known solution* (BKS) of some large-scale SCP benchmark instances previously given by the lagrangian heuristics of CAPRARA *et al.* [8] and CERIA *et al.* [9]. In turn, for the particular case of the *unit-cost* SCP, the LS procedure of GAO *et al.* [11] was also successful in improving some of the BKS reported in the literature. In general, LS is most usually combined with a heuristic procedure: first, the heuristic is applied to produce an initial solution; then LS is applied to hopefully improve it.

In practice, many relevant decision problems motivated by real-world applications are formulated as SCP. Some examples arise from crew scheduling, including those for the airline [12], railway [8, 9, 13] and healthcare [14] industries, vehicle scheduling [15], assembly-line balancing [16] and service location [17]. For example, the application of SCP to solve railway crew scheduling problems motivated the italian railway company, *Ferrovie dello Stato SpA*, to host a competition in 1994 called *FASTER* [18], with the goal of promoting the development of algorithms to better handle large-scale instances of SCP which result from crew scheduling problems in the company. This led to the publication of a series of articles on SCP heuristics in the following years [8, 9, 19–21].

Another relevant area of applications includes network design. A *wireless ad-hoc network* (WANET) consists of a decentralized network of autonomous devices that communicate wirelessly. Two devices can communicate directly to each other if they are in each others range. On the other hand, communication between two devices which are not sufficiently close happens by iteratively transmitting information to another device within range in what is known as *multi-hop communication*, thus enabling routing (one-to-one communication), multi-casting (one-to-many) and broadcasting (one-to-all). A *virtual backbone* is a set of devices which are responsible for ensuring efficient multi-hop communication in the network. Usually, the backbone devices are equipped with better processing capabilities than the other ones

and are therefore more costly. Determining the least-cost virtual backbone in a given network is achieved by solving a combinatorial optimization problem called the *min connected dominating set problem* (MCDSP). To do so, the network is represented by an undirected graph, with each vertex corresponding to a device, and an edge between two vertices exists if their corresponding devices are directly connected, i.e., within range. In this way, determining a virtual backbone consists of finding a connected dominating set (CDS) in the corresponding graph. The domination constraints ensure that any device is able to communicate directly with a backbone device. In turn, connectivity constraints ensure that information can be transmitted across the whole network by routing through backbone nodes. When connectivity is not required, the min dominating set problem (MDSP) is easily formulated as SCP, and thus any of the aforementioned methods for SCP can be employed to solve it. However, formulating MCDSP is not as straightforward. MCDSP was originally formulated by integer programming by FUJIE [22] in 2006, and later alternative formulations were reported by LUCENA *et al.* [24] and SIMONETTI *et al.* [2]. All of these formulations lead to an integer program which is not in the form of SCP. It was only in 2015 that BUCHANAN *et al.* [25] proved a characterization of connected dominating sets and proposed an SCP formulation for MCDSP. However, their formulation leads to an integer program with an exponential amount of constraints, and they employ *branch-and-cut* to solve, which is an exact algorithm that adapts the BB method.

Another line of research concerns finding *vertex covers* in the graph. Due to their communication nature, wireless networks are prone to cyberattacks, which can compromise data integrity and confidentiality, thus increasing the importance of monitoring the links in the network. This can be achieved by placing *secure points* on network nodes, which analyze and log traffic through that node and verify its security. Determining a least-cost set of secure points such that every link is monitored requires finding a *vertex cover* in its corresponding graph. A vertex cover ensures that each link between two nodes is monitored. When communication between monitors is also required, connectivity constraints are imposed, leading to *minimum connected vertex-cover problem* (MCVCP) [26]. Our first contribution is to prove that the characterization of BUCHANAN *et al.* [25] is true for connected vertex covers. In addition, we propose a novel SCP formulation for MCVCP.

Other applications include facility location problems. In general, given a transportation network, these problems concern the decision of choosing one or more locations in the network where to place a facility that services demand points. At least originally, facility location problems considered the case that each facility was located on an individual node, until SLATER [27] generalized the idea to allow facilities to take the shape of an edge, a path or a tree in the corresponding graph, which

was later on investigated by MINIEKA [28], RICHEY [29] and HAKIMI *et al.* [30]. Such generalization arises naturally for applications in which geographical distance is directly related to construction or operation costs, such as railways, highways, transmission or pipeline networks [27, 31]. The *min connected edge-dominating set problem* (MCEDSP), also called the *min tree-cover problem* (MTCP), arises when one aims to find a facility of minimum total length. Our second contribution is an original proof of a novel characterization of connected edge-dominating sets. Additionally, we propose a novel SCP formulation for MCEDSP.

The SCP formulation of MCDSP reported by BUCHANAN *et al.* [25], as well as our proposed novel SCP formulations for MCVCP and MCEDSP all share one common characteristic: they have an *exponential amount of constraints*. This means that the number of constraints in their IP formulation is an exponential function of the number of vertices in the graph. Solving such IPs requires the use of specialized methods which are able to handle the exceedingly large amount of constraints. One example is the branch-and-cut algorithm, an exact algorithm, which for example was employed by BUCHANAN *et al.* [25] to solve MCDSP. Alternatively, when heuristics are concerned, lagrangian relaxation is a common approach. The so-called *relax-and-cut* (RC) algorithms are specifically designed to solve LR formulations of integer programs with exponentially many constraints. Instead of working with the exceedingly large number of constraints all at once, which would be impractical, an RC algorithm considers only a small subset of *active* constraints at a time, which are handled *explicitly*, while the remaining constraints are considered *inactive* and handled *implicitly*. RC methods have been successfully applied to a variety of problems, for example by KLOSE [32] for a facility-location problem and by DA CUNHA *et al.* [33] for the Steiner tree problem.

1.1 Motivation

Over the past couple of decades, SCP has been extensively investigated and considerable progress has been made in the development and improvement of SCP heuristics. As a consequence of these advances, current state-of-the-art SCP heuristics are very effective in producing high quality solutions within reasonable amount of time, even when large-scale instances are considered. In addition to its theoretical relevance, SCP finds a wide variety of applications. In fact, many practical problems can be efficiently solved by being formulated as SCP and benefiting from all the advances made in SCP algorithms. Some examples of such problems include the min dominating set problem, the min vertex cover problem and the min edge-dominating set problem. On the other hand, some applications further require a solution to be connected, thus leading to the min connected dominating set problem, the min

connected vertex cover problem and the min connected edge-dominating set problem. These connected variants are not formulated as SCP in such a straightforward way. MCDSP was originally formulated by IP in 2006, but only in 2015 it was formulated as SCP. Additionally, to the best of the author's knowledge, our SCP formulations for MCVCP and MCEDSP are original and first presented here. The purpose of formulating MCDSP, MCVCP and MCEDSP as SCP is to benefit from the advances in SCP algorithms in the solution of these problems. However, the aforementioned state-of-the-art SCP algorithms are not designed to handle the exponential amount of constraints in the SCP formulations of MCDSP, MCVCP and MCEDSP. The main goal of this dissertation is to develop effective SCP heuristics able to handle SCP formulations with exponentially many constraints. Again, to the author's best knowledge, this is the first time that SCP with exponentially many constraints is specifically considered. Considering that state-of-the-art SCP heuristics include lagrangian heuristics [8, 9] and local-search [10, 11, 34], we propose a novel relax-and-cut as well as a novel local-search algorithm for SCP with exponentially many constraints. Our proposed LS procedure handles the exceedingly large number of constraints in a manner similar to a relax-and-cut algorithm, by considering one subset for the active and another for the inactive constraints. Although this idea is part of any RC algorithm, this work is the first one to report the application of this schema to an LS procedure. To evaluate the effectiveness of our proposed algorithms in producing high-quality SCP solutions, we employ them to solve benchmark MCDSP instances and compare our results with state-of-the-art MCDSP algorithms reported in the literature. As presented in Chapter 7, our algorithms are competitive with the best MCDSP algorithms.

1.2 Contributions

The main contribution of this dissertation is treating the set cover problem in a general setting in which the problem possibly has an exponential amount of constraints. Such treatment has not been reported in the literature, at least not specifically for SCP, and we consider this to be the first time that algorithms specifically designed for the SCP with exponentially many constraints have been considered. Since the currently most effective SCP heuristics include lagrangian greedy heuristics [8, 9] and local-search [10, 11, 34] procedures, and since our goal is to develop heuristic algorithms which are effective in producing high-quality solutions, we propose a novel (lagrangian) relax-and-cut and a novel local-search algorithm for SCP which are able to efficiently handle exponentially many constraints. In addition, with the goal of improving the convergence of our relax-and-cut algorithm, we propose a novel strengthened lagrangian relaxation of SCP. Our LR formulation, RC and LS algo-

rithms are employed to solve benchmark MCDSP instances and our shown to be more effective than state-of-the-art MCDSP algorithms in producing high-quality MCDSP in reasonable time.

Our contributions are summarized as follows:

- The proof of a characterization of connected vertex covers and a novel formulation of the min connected vertex cover problem in the form of a set cover problem.
- The proof of a novel characterization of connected edge-dominating sets and a novel formulation of the min connected edge-dominating set problem in the form of a set cover problem.
- A novel lagrangian relaxation formulation for SCP. As demonstrated in Chapter 7, our LR formulation is successful in reducing the zig-zagging behaviour of the subgradient method and successfully improving the its overall convergence when solving the LR formulation of SCP.
- Novel lagrangian greedy heuristic, relax-and-cut and local-search algorithms for the set cover problem with exponentially many constraints.
- Applications of our proposed algorithms to solve the min connected dominating set problem. Computational experiments suggest that our proposed algorithms rank as the most effective heuristic for producing high quality MCDSP solutions, having found the BKS for every one from a set of 41 benchmark instances.

1.3 Outline

The remainder of this document is structured as follows:

- Chapter 2 reviews the necessary background to follow the remainder of this dissertation. It begins with a general review of the theory of mathematical optimization and integer programming. Section 2.1 presents a review of the set cover problem and related work. Section 2.2 reviews some basic graph theory and specially the theory of dominating sets, vertex covers and edge-dominating sets, as well as a literature review.
- Chapter 3 is about imposing connectivity by means of integer programming. Section 3.1 presents a review of the standard approaches for formulating connectivity constraints. In Section 3.2 we present our proposed characterizations of connected vertex covers and connected edge-dominating sets. In Section 3.3

we present the SCP formulation of MCDSP and propose novel SCP formulations of MCVCP and MCEDSP. Section 3.4 describes the so-called *separation procedures*, the class of algorithms used to identify violated constraints in our RC and LS algorithms.

- Chapter 4 is about lagrangian relaxation. Section 4.1 reviews the general theory of lagrangian relaxation. Section 4.2 presents the standard LR formulation for SCP. In Section 4.3 we propose a novel strengthened LR formulation for SCP.

- Chapter 5 presents the relax-and-cut algorithm. Section 5.1 reviews the *subgradient method*, a standard algorithm for solving LR formulations. Section 5.2 reviews the relax-and-cut algorithm in a general setting. In Section 5.3 we propose a novel lagrangian greedy heuristic for SCP with exponentially many constraints. In Section 5.4 we describe our proposed relax-and-cut algorithm for SCP with exponentially many constraints.

- Chapter 6 presents the local-search algorithm. It begins with a review of local-search algorithms in general. In Section 6.1 we propose a novel local-search for SCP with exponentially many constraints.

- Chapter 7 presents the results of our computational experiments.

- In Chapter 8 we present our conclusions and mention some lines of future work.

Chapter 2

Background

The field of *mathematical optimization* draws from applied mathematics, computer science and engineering, and finds many practical applications. Generally speaking, a mathematical *optimization problem*, also called a *mathematical program* (MP), consists of minimizing or maximizing a real-valued function f given a set of constraints. The constraints specify the conditions for an object to be deemed acceptable as a candidate solution for the problem. Formally, the constraints determine a set $\mathcal{P}$ called the *feasible region* of the problem, and any element $y \in \mathcal{F}$ is called a *feasible solution*. In contrary, any $y \notin \mathcal{F}$ is *infeasible*. Furthermore, the function $f : \mathcal{F} \to \mathbb{R}$ is called the *objective value function*, and for each $y \in \mathcal{F}$ assigns a real value $f(y)$, called its corresponding *objective value*. Hence a MP consists of finding the minimum or maximum value of f on the feasible region $\mathcal{F}$. Accordingly, for a minimization problem, a feasible solution $y^* \in \mathcal{F}$ is *optimal* if $f(y^*) \leq f(y)$ for every $y \in \mathcal{F}$. Conversely, for a maximization problem, a feasible solution $y^* \in \mathcal{F}$ is *optimal* if $f(y^*) \geq f(y)$ for every $y \in \mathcal{F}$. Note that a MP might have more than one optimal solution, but all optimal solutions to the same problem have the same objective value. According to the standard notation in the mathematical optimization literature, a minimization problem with objective value function f and feasible region $\mathcal{F}$ is expressed in the following way:

$$\min \quad f(y) \tag{2.1a}$$

$$\text{s.t.} \quad y \in \mathcal{F} \tag{2.1b}$$

Note that (2.1a) states that the objective of the problem is to minimize the value of $f(y)$, while (2.1b) states that the problem is *subject to* the constraint that $y \in \mathcal{P}$ must hold. Similarly, a maximization problem can be expressed as follows:

$$\max \quad f(y) \tag{2.2a}$$

$$\text{s.t.} \quad y \in \mathcal{F} \tag{2.2b}$$

As a simple example, suppose one aims to minimize the value of $f(y) = y^4 - 26y^2 + 48y$ subject to the constraint that y is a non-negative real number. This minimization problem can be formulated as follows:

$$\begin{aligned} \min \quad & y^4 - 26y^2 + 48y \\ \text{s.t.} \quad & y \geq 0 \end{aligned} \tag{2.3}$$

The optimal solution to (2.3) is $y^* = 3$, with corresponding objective value of -9. From the theory of calculus, one can easily find that a local-minimum of f occurs at value -4 and that $f(-4) = -352 < -9$. But since $-4 \geq 0$ does not hold, then -4 is infeasible to (2.3). The graph of f, the feasible region and an optimal solution to (2.3) are presented in Figure 2.1. Note that the feasible region of (2.3) is given by $\mathcal{F} = \mathbb{R}_{\geq 0}$. By highlighting the feasible region in Figure 2.1, it is evident that $y^* = 3$ is the unique optimal solution to (2.3).

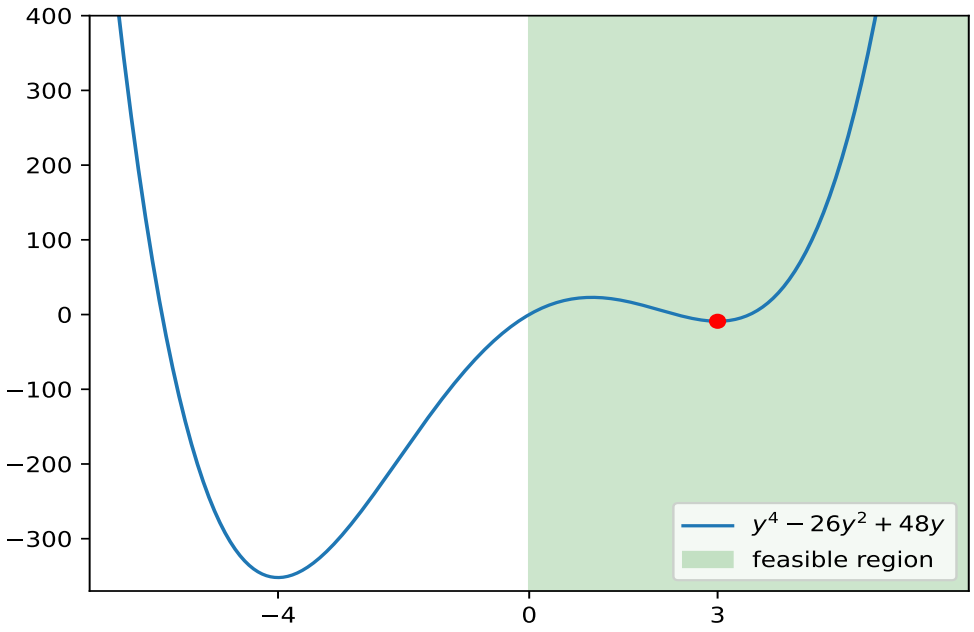

Figure 2.1: The objective value function $f(y) = y^4 - 26y^2 + 48y$, the feasible region $\mathcal{F} = \mathbb{R}_{\geq 0}$ and the optimal solution $y^* = 3$ to (2.3)

Note that in (2.3), instead of denoting the constraint as $y \in \mathbb{R}_{\geq 0}$, the set membership notation is omitted, leading to the more succinct statement that $y \geq 0$

must hold. This is standard in the literature, and constraints are most commonly expressed as an equality or inequality rather than by set membership. Consider the m-th dimensional real space $\mathbb{R}^m$. Given a function $g : \mathbb{R}^m \to \mathbb{R}$ and $b \in \mathbb{R}$, the *less-than* inequality $g(y) \leq b$, the *greater-than* inequality $g(y) \geq b$ and the equality $g(y) = b$ all define subspaces of $\mathbb{R}^m$. Namely, they define the sets $\{y \in \mathbb{R} : g(y) \leq b\}$, $\{y \in \mathbb{R} : g(y) \geq b\}$ and $\{y \in \mathbb{R} : g(y) = b\}$, respectively. In the context of mathematical optimization, the feasible region is most usually defined as the intersection of one or more such regions. Following the standard in the literature, the set membership notation is omitted and the feasible region is expressed by a set of equality and inequality constraints. For example, consider a MP with feasible region corresponding to the 2-dimensional unit square. Hence $\mathcal{F} = [0, 1]^2$. The goal is represent $\mathcal{F}$ by inequality and equality constraints. Naturally, for any $y \in \mathbb{R}^2$, $y \in [0, 1]^2$ if and only if $0 \leq y_1 \leq 1$ and $0 \leq y_2 \leq 1$. Hence $\mathcal{F}$ can be defined by letting: $g_1(y) = y_1$, $b_1 = 0$ and considering $g_1(y) \geq b_1$; $g_2(y) = y_1$, $b_2 = 1$ and considering $g_2(y) \leq b_2$; $g_3(y) = y_2$, $b_3 = 0$ and considering $g_3(y) \geq b_3$; and $g_4(y) = y_2$, $b_4 = 1$ and considering $g_4(y) \leq b_4$. Note that $[0, 1]^2 = \{y \in \mathbb{R}^2 : y_1 \geq 0\} \cap \{y \in \mathbb{R}^2 : y_1 \leq 1\} \cap \{y \in \mathbb{R}^2 : y_2 \geq 0\} \cap \{y \in \mathbb{R}^2 : y_2 \leq 1\}$. Hence, if the objective value function is f, this problem is formulated as follows:

$$\begin{array}{ll} \min & f(y) \\ \text{s.t.} & y_1 \geq 0 \\ & y_1 \leq 1 \\ & y_2 \geq 0 \\ & y_2 \leq 1 \end{array} \tag{2.4}$$

Note that an equality constraint $g(y) = b$ can be equivalently expressed by a pair of inequality constraints: $g(y) \leq b$ and $g(y) \geq b$. Moreover, a less than constraint $g(y) \leq b$ can be equivalently expressed as a greater than constraint $-g(y) \geq -b$. Conversely, a greater than constraint $g(y) \geq b$ can be equivalently expressed as a less than constraint $-g(y) \leq -b$. Finally, note that maximizing f is equivalent to minimizing $-f$. Conversely, minimizing f is equivalent to maximizing $-f$. Hence a mathematical program can be expressed in the so-called *canonical form*:

$$\begin{array}{lll} \min & f(y) & \\ \text{s.t.} & g_i(y) \geq b_i & i \in \{1, \ldots, m\} \\ & y \geq 0 & \end{array} \tag{2.5}$$

y are called the decision variables. From a historical point of view, optimization and OR methods are highly motivated by practical decision-making problems. In those cases, the variables in the MP are associated with some kind of decision related to the application, like a specific quantity of a resource, or a decision from a discrete

set of options. Considering the close relationship between theory and applications, variables in MP have historically been named *decision variables* and this has become the standard nomenclature.

A function $f : \mathbb{R}^m \to \mathbb{R}$ is *linear* if there exists $w_1, \ldots, w_m \in \mathbb{R}$ such that $f(y) = w_1 y_1 + \cdots + w_m y_m$ for every $y \in \mathbb{R}^m$. Consequently, by letting $w = (w_1, \ldots, w_m)$, a linear function f can be represented as a linear transformation $f(y) = w \cdot y$, where $\cdot$ denotes the *dot product* of two vectors. An equality constraint $g(y) = b$, a less-than inequality constraint $g(y) \leq b$ or a greater-than constraint $g(y) \geq b$ is *linear* if g is a linear function. A *linear program* consists of a linear objective value function and a set of linear constraints. Hence a linear program can be conveniently expressed in *matrix-form* as follows:

$$\begin{aligned} \min \quad & cy \\ \text{s.t.} \quad & Ay \geq b \\ & y \geq 0 \end{aligned} \tag{2.6}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

Furthermore, if one imposes the constraint that y takes only integer values, then it is called an *linear integer program* (IP). These are called the *integrality constraints*. A general IP formulation in matrix form is as follows:

$$\begin{aligned} \min \quad & cy \\ \text{s.t.} \quad & Ay \geq b \\ & y \geq 0 \\ & y \in \mathbb{Z}^n_{\geq 0} \end{aligned} \tag{2.7}$$

If y is further constrained to assume binary values, then it is called a *linear binary program* (BP), formulated below:

$$\begin{aligned} \min \quad & cy \\ \text{s.t.} \quad & Ay \geq b \\ & y \in \{0,1\}^n \end{aligned} \tag{2.8}$$

Finally, an LP can be obtained from an IP by *relaxing* the integrality constraints, hence allowing y to assume real values. This is called its *linear-programming relaxation*, and as shall be discussed in short, it is very relevant for solving IPs.

Many practical problems can be formulated as mathematical optimization problems. As a simple and intuitive example, let us consider the following situation. Suppose one has a knapsack that can carry up to a certain amount of weight, denoted by $w \in \mathbb{R}_{>0}$. Suppose there is a variety of items that can be carried in the knapsack. Let us also assume that for each variety, there is exactly one item of that kind. Each item has an associated *value*, and items with higher value are worth more. Moreover, each item has an associated *weight*. Due to the weight limit, not

all items can be carried in the knapsack. The value of the knapsack is the sum of values of the items in the knapsack. Similiarly, its weight is the sum of weights of the items in the knapsack. One is interested in carrying in the knapsack a set of items as to maximize the knapsack value, but the knapsack weight must not exceed the weight limit w. Note that there is a trade-off in carrying an item: its value is added, but it also takes up space that could otherwise be filled by other items. At least intuitively, it seems that items with relatively higher values and lower weights are more favorable. We now describe how this can be formulated as a MP. Let $I = \{i_1, \ldots, i_n\}$ be the set of distinct items, and let $N = \{1, \ldots, n\}$ be the set of item indices. Moreover, for each $k \in N$, let $v_k \in \mathbb{R}_{>0}$ and $w_k \in \mathbb{R}_{>0}$ be the value and weight associated with item i_k, respectively. The problem concerns the choice of which items to carry in the knapsack as to maximize value. Note that, for each item, one can choose among two options: to either carry it or not. To formulate this problem mathematically, we need to associate with each decision (to carry the item or not) a variable to represent that choice. For each $k \in N$, let y_k be a variable such that $y_k = 1$ denotes that item i_k is in the knapsack and $y_k = 0$ denote that item i_k is not in the knapsack. Hence $y_k = 1$ denotes the choice of carrying item i_k in the knapsack and $y_k = 0$ denotes otherwise. Accordingly, a binary vector $y = (y_1, \ldots, y_k)$ denotes the choice of which items to carry or not. Note that if $y_k = 1$, then item i_k is in the knapsack and its value v_k should be added to the knapsack value. Conversely, if $y_k = 0$, then item i_k is not in the knapsack and its value v_k should not. Hence, given $y \in \{0, 1\}^n$, the contribution of each item to the knapsack value is

$$v_k y_k = \begin{cases} v_k \text{ if } y_k = 1 \\ 0 \text{ if } y_k = 0 \end{cases}$$

and thus the knapsack value is given by

$$\sum_{k=1}^{n} v_k y_k$$

Similarly, the contribution of each item to the knapsack weight is

$$w_k y_k = \begin{cases} w_k \text{ if } y_k = 1 \\ 0 \text{ if } y_k = 0 \end{cases}$$

and thus the knapsack weight is given by

$$\sum_{k=1}^{n} w_k y_k$$

Recall that a MP is defined by an objective value function, a set of constraints and an objective sense (minimize or maximize). In this case, the objective value function is $f(y) = \sum_{k=1}^{n} v_k y_k$, and the objective is to maximize it. As for the constraints, in order to be feasible, a choice of items must not exceed the weight limit w. This is equivalent to saying that $\sum_{k=1}^{n} w_k y_k \leq w$ must hold. Hence a formulation for this problem is:

$$\begin{aligned} \max \quad & \sum_{k=1}^{n} v_k y_k \\ \text{s.t.} \quad & \sum_{k=1}^{n} w_k y_k \leq w \\ & y_k \in \{0,1\} \qquad k \in \{1,\ldots,n\} \end{aligned} \tag{2.9}$$

MP (2.9) is an example of linear integer program, more specifically a linear binary program. It is called the $0/1$ *knapsack problem*, and is well-known in the fields of mathematical optimization and operations research (OR).

Let $\mathcal{F} = \{y \in \{0,1\}^n : \sum_{k=1}^{n} w_k y_k \leq w\}$ denote the feasible region of (2.9). Let $y^* \in \mathcal{F}$ be an optimal solution to (2.9). Note that, by definition, any feasible solution $y \in \mathcal{F}$ provides a value $\underline{Z} = f(y)$ such that $\underline{Z} \leq f(y^*)$. For a maximization problem, a *primal bound* is a value $\underline{Z} \in \mathbb{R}$ such that $\underline{Z} \leq f(y^*)$. Hence any feasible solution provides a primal bound. Finding an optimal solution to (2.9) requires finding $y^* \in \mathcal{F}$ such that $\sum_{k=1}^{n} v_k y_k^* \geq \sum_{k=1}^{n} v_k y_k$ for every $y \in \mathcal{F}$. One possible approach is to enumerate all the feasible solutions in $\mathcal{F}$ and pick one with greatest corresponding objective value. But note that for a problem such as (2.9), $\mathcal{F}$ has up to 2^n elements, such that enumerating all the exponentially many possibilities is intractable even for small values of n. A more efficient alternative is to prove that there exists a value $\overline{Z} \in \mathbb{R}$ such that $f(y) \leq \overline{Z}$, for every feasible solution $y \in \mathcal{F}$. For a maximization problem, a *dual bound* is a value $\overline{Z} \in \mathbb{R}$ such that $f(y^*) \leq \overline{Z}$. Hence a feasible solution $y \in \mathcal{F}$ is proven optimal if one can determine a dual bound $\overline{Z}$ such that $f(y) = \overline{Z}$.

Now consider a minimization problem. Let $\mathcal{F}$ be its feasible region, let f be its objective value function and let $y^* \in \mathcal{F}$ be an optimal solution to it. A *primal bound* is a value $\overline{Z} \in \mathbb{R}$ such that $f(y^*) \leq \overline{Z}$. Again, any feasible solution provides a primal bound. Additionally, a *dual bound* is a value $\underline{Z} \in \mathbb{R}$ such that $\underline{Z} \leq f(y)$ for every feasible solution $y \in \mathcal{F}$. Accordingly, for a minimization problem, a feasible solution $y \in \mathcal{F}$ is proven optimal if one can determine a dual bound $\underline{Z}$ such that $\underline{Z} = f(y)$.

In summary, solving a MP to proven optimality requires determining a primal bound Z_P and a dual bound Z_D such that $Z_P = Z_D$.

By relaxing the integrality constraints, one obtains the LP-relaxation of (2.9), as follows:

$$
\begin{aligned}
\max \quad & \sum_{k=1}^{n} v_k y_k \\
\text{s.t.} \quad & \sum_{k=1}^{n} w_k y_k \leq w \\
& y_k \in [0,1] \qquad k \in \{1,\ldots,n\}
\end{aligned}
\tag{2.10}
$$

MP (2.10) is an example of a linear program, but it is not integer, since y is allowed to assume non-integral values. Let $\mathcal{F}' = \{y \in [0,1]^n : \sum_{k=1}^{n} w_k y_k \leq w\}$ denote the feasible region of (2.10). Note that $\mathcal{F} \subseteq \mathcal{F}'$, meaning that any feasible solution to (2.9) is also feasible to (2.10). For that reason, (2.10) is said to be a *relaxation* of (2.9). Conversely, (2.9) is said to be *stricter* or *stronger* than (2.10). Let $y^* \in \mathcal{F}$ be an optimal solution to (2.9), and let $x^* \in \mathcal{F}'$ be an optimal solution to (2.10). Since $\mathcal{F} \subseteq \mathcal{F}'$, then y^* is also feasible to (2.10), and it follows by definition that $f(y^*) \leq f(x^*)$. Hence solving (2.10) provides a dual bound on $f(y^*)$. Suppose one knows a feasible solution y to (2.9). If $f(y) = f(x^*)$ holds, then y is guaranteed to be optimal to (2.9).

Although they may appear similar, solving (2.9) and (2.10) requires two different approaches. In fact, linear programs such as (2.10) can be solved in polynomial-time [35]. The standard methods for solving a linear program such as (2.10) are the *simplex algorithm* or the *interior-point method* [35]. On the other hand, many integer programs, including (2.10), belong to the class of NP-hard problems. All NP-hard problems share one common characteristic: no polynomial-time algorithm is known to solve any of them, and many computer scientist believe that one does not exist. For a thorough discussion of NP-hard problems, we refer the reader to GAREY [6]. An *exact algorithm* for an integer program provides an optimal solution with certified optimality, thus guaranteeing that no better solution exists. However, given the difficulty of solving IPs exactly, another relevant line of research are the so-called *heuristic algorithms*. A heuristic algorithm aims to produce a feasible solution of good quality in a reasonable amount of time. But to do so, it compromises the certificate of optimality. Hence there is no guarantee if the solution given by the heuristic is in fact optimal. Even so, in many practical applications, specially the ones leading to large-scale IPs, exact methods are just impractical, and heuristic algorithms are the best (and perhaps only) alternative.

Let us now consider an example. Consider the knapsack problem and suppose that the weight limit is 5, and that there are four items, with values $2, 3.5, 6, 5$ and weights $1, 2, 4, 4$, respectively. Let $w = 5$, $I = \{i_1, i_2, i_3, i_4\}$, $v = (2, 3.5, 6, 5)$ and $w = (1, 2, 4, 4)$. In accordance with (2.9), this problem is formulated as:

$$\begin{array}{lll} \max & 2y_1 + 3.5y_2 + 6y_3 + 5y_4 & \\ \text{s.t.} & y_1 + 2y_2 + 4y_3 + 4y_4 \leq 5 & \\ & y_k \in \{0,1\} & k \in \{1,\ldots,4\} \end{array} \tag{N_0}$$

A feasible solution to (N_0) can be produced by following a simple heuristic procedure described as follows. Sort the items in decreasing order of their value to weight ratio $\frac{v_i}{w_i}$. Starting from an empty knapsack, pick the item corresponding to the greatest ratio and insert it to the knapsack. Accordingly update the knapsack value and weight, remove this item from the set of available items and repeat this choice until no more items can fit in the knapsack. Note that $\frac{v_1}{w_1} = \frac{2}{1} \geq \frac{v_2}{w_2} = \frac{3}{2} \geq \frac{v_3}{w_3} = \frac{6}{4} \geq \frac{v_4}{w_4} = \frac{5}{4}$. Hence the heuristic first picks item i_1, adding 2 to the knapsack value and 1 to the knapsack weight. Since there is still space left in the knapsack, it proceeds by picking item i_2, adding 3.5 to the knapsack value and 2 to the knapsack weight. At this point, the knapsack weight amounts to $1 + 2 = 3$. Since the remaining items i_3 and i_4 both have weight 4 and the weight limit is 5, then they cannot fit in the knapsack, and the heuristic terminates. This produces a feasible solution $y^0 = (1, 1, 0, 0)$ such that $f(y^0) = 2 + 3.5 = 5.5$. In an attempt to check if y^0 is optimal, we consider the LP-relaxation of (N_0), given by:

$$\begin{array}{lll} \max & 2y_1 + 3.5y_2 + 6y_3 + 5y_4 & \\ \text{s.t.} & y_1 + 2y_2 + 4y_3 + 4y_4 \leq 5 & \\ & y_k \in [0,1] & k \in \{1,\ldots,4\} \end{array} \tag{2.11}$$

The optimal solution to (2.11) is given by $x^0 = (1, 1, 0.5, 0)$ with corresponding objective value $f(x^0) = 8.5$. As was previously described, an optimal solution to (2.11) provides a dual bound on an optimal solution to (N_0). For now, we know a feasible solution y^0 to (N_0) with $f(y^0) = 5.5$ and a dual bound of 8.5. Since $8.5 > 5.5$, then the optimality of y^0 is not ensured. Hence in order to find a certified optimal solution to (N_0), we must find a better feasible solution or find a tighter dual bound, perhaps both. This can be done by considering subproblems of (N_0) obtained by specifying a specific value for one or more items. Note that x_3^0 assumes a non-integral value. Naturally, an optimal solution to (N_0) either has item i_3 in the knapsack or not. Let y^* be an optimal solution to (N_0). Then either $y_3^* = 0$ or $y_3^* = 1$. We can then consider two subproblems, one corresponding to the case that $y_3^* = 0$ and the other corresponding to $y_3^* = 1$. Let $\mathcal{F}|_{y_3=0} = \{y \in \mathcal{F} : y_3 = 0\}$ and let $\mathcal{F}|_{y_3=1} = \{y \in \mathcal{F} : y_3 = 1\}$. Accordingly, y^* can be determined in a recursive fashion by noting that

$$y^* = \arg\max \begin{cases} \arg\max \left\{ f(y) : y \in \mathcal{F}|_{y_3=0} \right\} \\ \arg\max \left\{ f(y) : y \in \mathcal{F}|_{y_3=1} \right\} \end{cases} \tag{R_0}$$

This means that an optimal solution to (N_0) can be determined by solving two subproblems obtained from it. One subproblem is obtained by forcing item i_3 to be out of the knapsack, and can be formulated as an IP as follows:

$$\begin{array}{lll} \max & 2y_1 + 3.5y_2 + 5y_4 & \\ \text{s.t.} & y_1 + 2y_2 + 4y_4 \leq 5 & \\ & y_k \in \{0,1\} & k \in \{1,2,4\} \\ & y_3 = 0 & \end{array} \tag{N_1}$$

The other subproblem is obtained by forcing item i_3 to be in the knapsack, and can be formulated as an IP as follows:

$$\begin{array}{lll} \max & 2y_1 + 3.5y_2 + 5y_4 + 6 & \\ \text{s.t.} & y_1 + 2y_2 + 4y_4 \leq 1 & \\ & y_k \in \{0,1\} & k \in \{1,2,4\} \\ & y_3 = 1 & \end{array} \tag{N_2}$$

Note that in this case, since i_3 is forced into the knapsack, its value is added to the objective value function and its weight is discounted from the weight limit constraint.

Following the recursion in (R_0), we must solve subproblems (N_1) and (N_2) to obtain an optimal solution to (N_0).

First consider (N_1). Again, the heuristic procedure can be applied to produce a feasible solution. The heuristic produces the solution $y^1 = (1,1,0,0)$ such that $f(y^1) = 5.5$. Additionally, an optimal solution to the LP-relaxation of (N_1) is given by $x^1 = (1,1,0,0.5)$ and provides a dual bound of 8. Since $8 > 5.5$, then y^2 is not guaranteed to be an optimal solution to (N_1).

Now consider (N_2). It produces the solution $y^2 = (1,0,1,0)$ such that $f(y^2) = 8$. Additionally, an optimal solution to the LP-relaxation of (N_2) is given by $x^2 = (1,0,1,0)$ and provides a dual bound of 8. Hence y^1 is a certified optimal solution to (N_2).

Note that x_4^2 assumes a non-integral value. Let $\mathcal{F}|_{y_4=0}^{y_3=0} = \{y \in \mathcal{F} : y_3 = 0, y_4 = 0\}$ and let $\mathcal{F}|_{y_4=1}^{y_3=0} = \{y \in \mathcal{F} : y_3 = 0, y_4 = 1\}$. Once again, in a recursive fashion, an optimal solution to (N_1) can be determined by noting that

$$y^* = \arg\max \begin{cases} \arg\max \left\{ f(y) : y \in \mathcal{F}|_{y_4=0}^{y_3=0} \right\} \\ \arg\max \left\{ f(y) : y \in \mathcal{F}|_{y_4=1}^{y_3=0} \right\} \end{cases} \tag{R_1}$$

This means that an optimal solution to (N_1) can be determined by solving two subproblems obtained from it. One subproblem is obtained by forcing item i_4 to be out of the knapsack, and can be formulated by as an IP as follows:

$$
\begin{array}{lll}
\max & 2y_1 + 3.5y_2 & \\
\text{s.t.} & y_1 + 2y_2 \leq 5 & \\
& y_k \in \{0,1\} & k \in \{1,2\} \\
& y_3 = 0 & \\
& y_4 = 0 &
\end{array}
\qquad (N_3)
$$

The other subproblem is obtained by forcing item i_4 to be in the knapsack, and can be formulated as an IP as follows:

$$
\begin{array}{lll}
\max & 2y_1 + 3.5y_2 + 5 & \\
\text{s.t.} & y_1 + 2y_2 \leq 1 & \\
& y_k \in \{0,1\} & k \in \{1,2,4\} \\
& y_3 = 0 & \\
& y_4 = 1 &
\end{array}
\qquad (N_4)
$$

Following the recursion in (R_1), we must solve subproblems (N_3) and (N_4) to obtain an optimal solution to (N_1).

First consider (N_3). The heuristic produces the solution $y^3 = (1,1,0,0)$ with objective value $f(y^3) = 5.5$. Additionally, an optimal solution to the LP-relaxation of (N_3) is given by $x^3 = (1,1,0,0)$ and provides a dual bound of 5.5. Hence y^3 is a certified optimal solution to (N_3).

Now consider (N_4). The heuristic produces the solution $y^4 = (1,0,0,1)$ with objective value $f(y^4) = 7$. Additionally, an optimal solution to the LP-relaxation of (N_4) is given by $x^4 = (1,0,0,1)$ and provides a dual bound of 7. Hence y^4 is a certified optimal solution to (N_4).

Now that optimal solutions to both (N_3) and (N_4) have been determined, an optimal solution to (N_1) can be determined by recursion, according to (R_1). Since $f(y^3) = 5.5 < 7 = f(y^4)$, then y^4 is an an optimal solution to (N_1).

In a similar manner, now that optimal solutions to both (N_1) and (N_2) have been determined, an optimal solution to (N_0) can be determined by recursion, according to (R_0). Since $f(y^4) = 7 < 8 = f(y^2)$, then y^2 is an an optimal solution to (N_0).

In conclusion, finding a certified optimal solution to (N_0) required solving a series of conveniently defined subproblems of (N_0). There is a recursive relationship between the subproblems, which can be represented by tree, called an *enumeration tree* (ET). The root node of ET corresponds to (N_0). Each of the remaining nodes correspond to one of the subproblems (N_1), (N_2), (N_3) and (N_4). If a subproblem is not solved optimally, the procedure assorts to recursion. From that subproblem, new subproblems are defined by splitting the feasible region, or the *search space*, in a process called *branching*. For example, (N_1) and (N_2) were obtained from (N_0) by branching on variable y_3. Similarly, (N_3) and (N_4) were obtained from (N_1) by

branching on variable y_4. Branching on a node, called the parent node, generates one or more new nodes, which are called its *children*. Then an optimal solution to the parent node is given according to the recursive relationship between the parent and its child nodes. The search-tree for the knapsack example is presented in Figure 2.2.

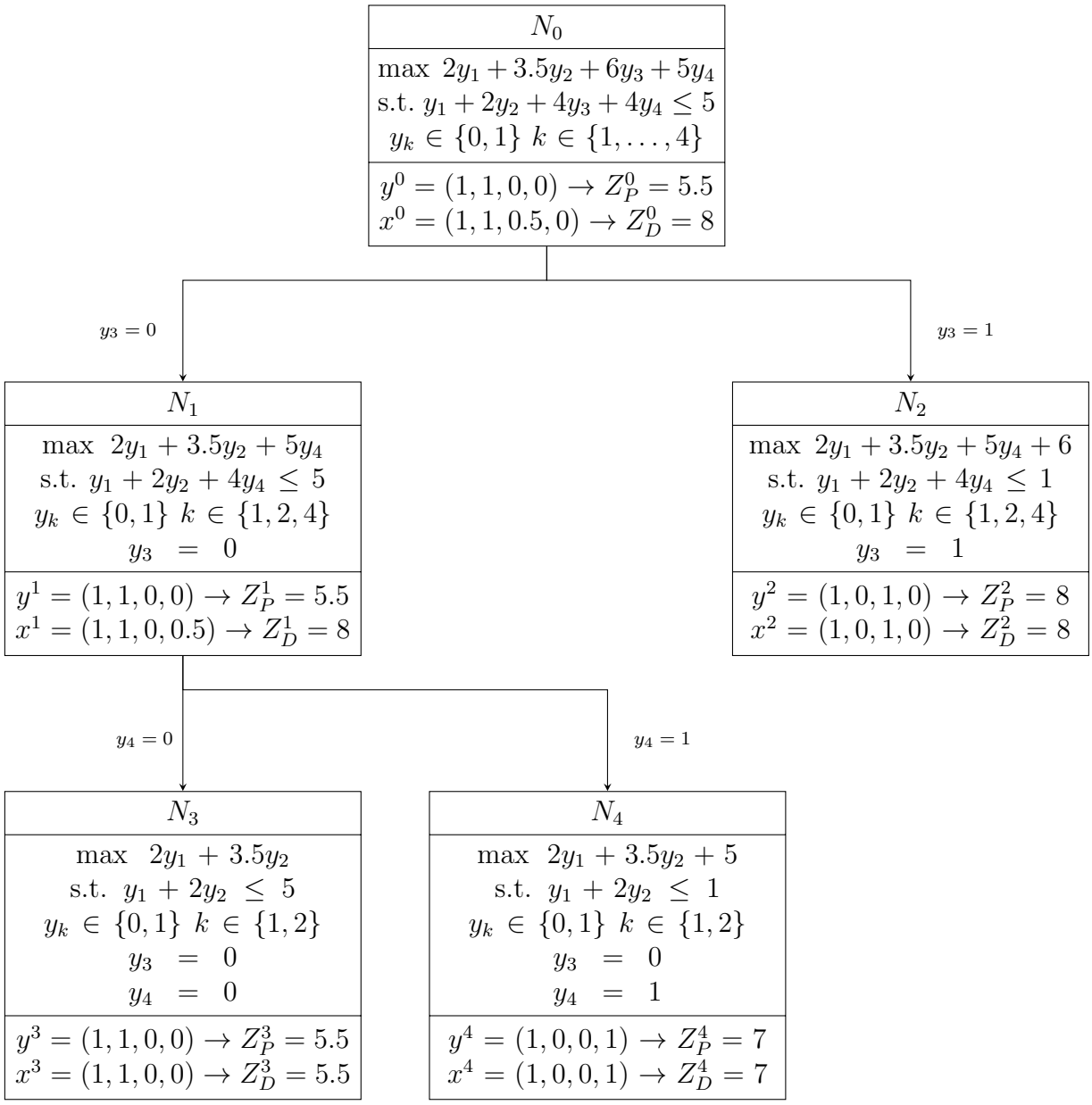

Figure 2.2: Enumeration-tree for the knapsack problem example. Each node corresponds to a subproblem. A primal and a dual bound are obtained for each node.

The procedure just described is an example of the *branch-and-bound algorithm* (BB). In this specific case, since LP-relaxations were employed to obtain dual bounds, it is called an *LP-based* branch-and-bound. In alternative, other methods can be used to determine dual bounds, such as *lagrangian relaxation*, which is discussed in detail in Chapter 4. This leads to a lagrangian-relaxation based BB.

LP-based BB is the standard method for solving IPs with proven optimality. Starting from an IP, BB finds a certified optimal solution of IP by recursively splitting the IP feasible region, in a process called *branching*. Each branching produces new subproblems conveniently derived from IP. The recursive relationship between the subproblems is usually represented by an enumeration tree. Hence each node of ET corresponds to one subproblem. At the root node of ET, the original IP is considered. A heuristic procedure is applied to produce a feasible solution $y^0 \in \mathcal{F}$ and a corresponding primal bound $Z_P^0 = f(y^0)$. Additionally, a dual bound Z_D^0 is also determined. In an LP-based BB, this dual bounds is obtained by solving the LP-relaxation of IP. If Z_D^0 does not guarantee the optimality of y^0, then the root node is branched, creating new nodes. Whenever a node is branched, this original node is called the *parent* node, and the new nodes created by branching from it are called its *children*. Each time a node is branched, the child nodes are associated with a reduced subproblem obtained from the parent node. Since the problem is reduced, each child node is associated with a stricter subproblem than its parent. In particular, the dual bound of the parent node is also a valid dual bound for its children. For example, in Figure 2.2, nodes N_1 and N_2 are children of N_0, and nodes N_3 and N_4 are children of N_1. After enough branching, a subproblem may become infeasible or have a trivial solution. For example, node N_4 had a trivial solution. It follows from the recursive relationship that an optimal solution to the parent node is given by the best solution among its child nodes. In this way, as nodes from lower levels of the tree are solved or deemed infeasible, solutions to nodes at higher levels are obtained by recursion. Aside from branching, the other most important mechanism of BB is *pruning*. Instead of simply enumerating all of the exponentially many possibilities, BB eliminates much of the redundancy by keeping track of primal and dual bounds. The bounds are used to conveniently avoid exploring branches of the ET which are guaranteed to lead to sub-optimal solutions. Each node of ET is associated with a reduced subproblem obtained from the original IP. At a given node k, a primal bound Z_P^k and a dual bound Z_D^k are determined for that subproblem. The primal bound is usually given by a heuristic algorithm. As for the dual bound, for an LP-based BB, it is given by solving the LP-relaxation of that subproblem. If $Z_P^k = Z_D^k$, then that node is solved optimally, and no further branching is required. On the other, if the optimality is not guaranteed for that node, BB resorts to branching. However, in some cases, branches of ET are known to contain only sub-optimal solutions. In that case, BB *prunes* those branches, meaning they are not even explored. To do so, BB keeps track of the best feasible solution found so far, i.e, the one with best corresponding objective value. By best, we mean maximum for a maximization problem, and minimum for a minimization problem. Denote it by y^* and let $Z_P = f(y^*)$ be the corresponding primal bound.

Let Z_D^k is the dual bound obtained for a given node k. For a maximization problem, if $Z_D^k \leq Z_P$, since Z_D^k is a valid dual bound for the child nodes, then the branch rooted at node k will certainly not produce a feasible solution better than y^*. Accordingly that branched is pruned, meaning it is no longer enumerated. Similarly, for a minimization problem, the branch rooted at node k is pruned if $Z_P \leq Z_D^k$. Consider the example from Figure 2.2. Node N_1 was branched, generating N_3 and N_4 as its children. The dual bound at N_1 was 8. Suppose that a solution with objective value 8 was known when solving N_1. In that case, branching from N_1 would be unnecessary, since that branch is guaranteed to not produce a solution withj objective value greater than 8. Hence this branch is pruned, and the exploration of nodes N_3 and N_4 is avoided. In fact, pruning is the main difference between BB and exhaustive enumeration, and is essential in assuring that BB is viable even for large-scale IPs. In pratice, due to significant advances in LP-based BB in the last few decades, they have become the standard method for solving IPs with proven optimality. In fact, a variety of publicly and commercially distributed LP-based BB solvers are available. These solvers are able to solve IPs with up to thousands and even million of variables and constraints. A detailed comparison of LP solvers was reported by MEINDL and TEMPL [36]. Note that the performance of BB is heavily influenced by the efficacy of the heuristic in finding good solutions and the tightness (strenght) of dual bounds. The tighter the gap between the primal and dual bounds, pruning is more likely. Consider Figure 2.2 for example. A total of five nodes were required to find the optimal solution. At the root node, the heuristic produced a solution with objective value 5.5. As already mentioned, if a solution with objective value 8 had been known from the start, finding the optimal solution would require enumerating only three nodes, namely N_0, N_1 and N_2. For that reasons, developing effective heuristics is a very relevant line of research in mathematical optimization and OR. In a similar way, improving the dual bounds is also favorable in reducing the enumeration tree. In that direction, lines of research include investigating stronger IP formulations.

In Section 2.1 we present the *set cover problem*, which is another well-known optimization problem and the main motivation of this work. In Section 2.2 we describe some basic concepts from graph theory and present a class of optimization problems which arise from graphs, generally called domination and vertex-cover problems.

2.1 Set Cover Problem

In order to define the *set cover problem*, or simply SCP: let $\mathcal{U}$ be a finite set called the *universe*; let $\mathcal{S} \subseteq 2^{\mathcal{U}}$ be a family of subsets of $\mathcal{U}$; and for each $S \in \mathcal{S}$, let

$c_S \in \mathbb{R}_{>0}$ be a *cost* associated with S. A set $\mathcal{C} \subseteq \mathcal{S}$ *covers* an element $u \in \mathcal{U}$ if there exists an $S \in \mathcal{C}$ such that $u \in S$. Furthermore, a set $\mathcal{C} \subseteq \mathcal{S}$ is a *cover* of $\mathcal{U}$ if every element $u \in \mathcal{U}$ is covered by $\mathcal{C}$. The cost $C(\mathcal{C})$ of a cover $\mathcal{C}$ is defined as $C(\mathcal{C}) = \sum_{S \in \mathcal{C}} c_S$. The set cover problem asks for a cover of minimum cost, i.e, it asks for a cover $\mathcal{C}^* \subseteq \mathcal{S}$ such that $C(\mathcal{C}^*) \leq C(\mathcal{C})$ for every cover $\mathcal{C} \subseteq \mathcal{S}$. Note that as long as $\bigcup_{S \in \mathcal{S}} S = \mathcal{U}$, then $\mathcal{S}$ is a cover of $\mathcal{U}$, and therefore SCP has at least one solution. Finally, an *instance* of SCP is one particular example of it, and is defined by a triple $(\mathcal{U}, \mathcal{S}, c)$, where $\mathcal{U}$ is a finite set, $\mathcal{S} \subseteq 2^{\mathcal{U}}$ and $c \in \mathbb{R}_{>0}^{|\mathcal{S}|}$. Note that there are infinitely many SCP instances. In the case that all costs are equal, all costs can be assumed to be 1, without loss of generality. These instances are called *unit-cost*.

In the context of linear and integer programming, SCP is usually cast in terms of an $m \times n$ binary *incidence matrix* $A = [a_{ij}] \in \{0,1\}^{m \times n}$ and an n-dimensional vector of costs $c \in \mathbb{R}_{>0}^n$. In this case, SCP is formulated as the following integer program:

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j y_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} y_j \quad \geq 1 \quad i \in \mathbf{M} \\ & y \in \{0,1\}^n \end{aligned} \tag{SCP-IP}$$

For the sake of notational convenience, let:

- Let $\mathbf{M} = \{1, \ldots, m\}$ denote the set of row indices of A.
- Let $\mathbf{N} = \{1, \ldots, n\}$ denote the set of column indices of A.

For each $j \in \mathbf{N}$, variable y_j is called a *decision variable*. Thus each column of A is associated with a decision variable, whereas each row is associated with a set cover constraint. A column $j \in \mathbf{N}$ is said to *cover* a row $i \in \mathbf{M}$ if $a_{ij} = 1$ holds. Additionally, a binary vector $y = (y_1, \ldots, y_n)$ is *feasible* for (SCP-IP) if for every row $i \in \mathbf{M}$, y satisfies that $\sum_{j=1}^{n} a_{ij} y_j \geq 1$. Equivalently, a binary vector $y \in \{0,1\}^n$ is feasible if for every row $i \in \mathbf{M}$ there exists a $j \in \mathbf{N}$ such that $a_{ij} = 1$ and $y_j = 1$ hold. In the particular case of the SCP, any feasible solution of (SCP-IP) is said to be a *cover*. For each column $j \in \mathbf{N}$, there is an associated *cost* $c_j \in \mathbb{R}_{>0}$. Accordingly, for any $y \in \{0,1\}^n$, its *cost* $C(y)$ is defined as $C(y) = \sum_{j=1}^{n} c_j y_j$. Finally, let $\mathcal{P}_{SCP} = \{y \in \{0,1\}^n : \sum_{j=1}^{n} a_{ij} y_j \geq 1\}$ be the set of all feasible solutions of (SCP-IP). Hence (SCP-IP) asks for a cover of minimum cost, i.e, a cover $y^* \in \mathcal{P}_{SCP}$ such that $C(y^*) \leq C(y)$ for every cover $y \in \mathcal{P}_{SCP}$. Note that as long as for each row $i \in \mathbf{M}$ there is a column $j_i \in \mathbf{N}$ which covers row i, then the n-th dimensional unit vector

1 is a feasible solution to (SCP-IP) and therefore (SCP-IP) has at least one feasible solution. Finally, an *instance* of SCP is defined in matrix form by a tuple (A, c), where $A \in \{0,1\}^{m \times n}$ and $c \in \mathbb{R}^n_{>0}$.

Note that the assumption that costs are positive is made without loss of generality. In fact, consider an SCP instance (A, c) and suppose that column $k \in \mathbf{N}$ is associated with a non-positive cost $c_k \leq 0$. Then there is an optimal solution y^* for this instance such that $y^*_k = 1$. Consider any cover $y \in \mathcal{P}_{SCP}$ and let y' be such that $y'_j = y_j$ for $j \neq k$ and $y'_k = 1$. Since y is a cover, then y' is also a cover. Furthermore, note that $C(y') = C(y) + c_{y_k}(1 - y_k) \leq C(y)$ since $c_k \leq 0$. Thus for every cover y there is a cover y' such that $y'_k = 1$ and $C(y') \leq C(y)$. Therefore the instance (A, c) can be mapped into an equivalent instance (A', c'), obtained by removing column k and all the rows that it covers. Conversely, an optimal solution y' of (A', c') can be mapped to an optimal solution y^* of (A, c) by considering $y^*_k = 1$ and $y^*_j = y'_j$ for $j \neq k$. This concludes the proof that the assumption is made with no loss of generality.

For the sake of notational convenience, given an SCP instance (A, c):

- For each $i \in \mathbf{M}$, let $J_i = \{j \in \mathbf{N} : a_{ij} = 1\}$ be the set of columns which cover row i, called its *column-set*.
- For each $j \in \mathbf{N}$, let $I_j = \{i \in \mathbf{M} : a_{ij} = 1\}$ be the set of rows which are covered by column j, called its *row-set*.
- For each $j \in \mathbf{N}$, let $N_j = \{j' \in \mathbf{N} : I_j \cap I_{j'} \neq \emptyset\}$ denote the set of *neighbors* of j.

In the linear and integer programming literature, any constraint of the form $\vec{a} \cdot \vec{y} \geq 1$ is said to be a *set cover constraint* if $\vec{a} \in \{0,1\}^k$ is a binary row vector and $\vec{y} \in \{0,1\}^k$ is a vector of binary decision variables. Therefore an integer program might have set cover constraints, even if the model is meant to solve some problem other than SCP. As shall be discussed in subsequent sections, many practical problems can be cast either as a set cover problem or as a mixed-integer program with set cover constraints.

2.1.1 Literature Review

SCP is well-known and has been thoroughly studied and extensively reported in the operations research literature. In its decision version, SCP has been proven to be $\mathcal{NP}$-complete by KARP [4], whereas its optimization version is $\mathcal{NP}$-hard [5, 6]. Its computational difficulty and relevance for practical applications has motivated the development of many exact, approximation and heuristic algorithms over the past

few decades. Considering that no polynomial-time algorithm is known to solve SCP exactly, i.e, with certified optimality – and since SCP is NP-complete, the existence of such an algorithm would imply that P = NP – most of the exact algorithms for SCP rely on enumeration, most commonly in the form of a branch-and-bound (BB) procedure [35, 37]. In the case of SCP, BB algorithms are most usually based on linear programming (LP), meaning that they rely on the LP-relaxation of an integer program such as (SCP-IP) in order to generate dual bounds for SCP. More specifically, at each node of the enumeration tree, the algorithm solves the LP-relaxation of a conveniently defined subproblem of (SCP-IP) in order to obtain a dual bound. Once an enumeration node is solved, two child nodes are created by *branching* on a chosen column $j \in \mathbf{N}$. One of the child nodes includes the additional constraint that $y_j = 0$, while the other child node includes the additional constraint that $y_j = 1$. Branching guarantees that the dual bound for the child nodes is at least as great as, and hopefully, greater than the dual bound for its parent node. By succesively branching, the dual bounds get tighter and BB gets closer to proving optimality. CHRISTOFIDES and KORMAN [38] survey some of the earliest exact algorithms for SCP, including the LP-based BB algorithms of LEMKE *et al.* [39], PIERCE [40] and PIERCE and LASKY [41]. Alternatively, dual bounds can be obtained by *lagrangian relaxation* (LR). In the LR approach, the set cover constraints are *dualized* and a corresponding *lagrangian dual problem* (LDP) is formulated. In turn, an optimal solution to LDP provides the best possible lagrangian dual bound. In LR-based BB algorithms, at each node of the enumeration tree a correspoding LDP is formulated and attempted to be solved in order to generate a valid lagrangian dual bound. Throughout the BB enumeration tree, each LDP is most typically solved by means of subgradient optimization [42]. Some examples of LR-based BB algorithms for SCP were proposed by BALAS and HO [43], BEASLEY [44], BALAS and CARRERA [45] and BEASLEY and JÖRNSTEN [46]. NOBILI and SASSANO [47] attempted to obtain tighter dual bounds than those attained by the LP-relaxation of (SCP-IP) by reinforcing it with cutting planes. In general terms, with the advances in LP theory and the improving performance of general-purpose LP-solvers, observed specially from the 1990's onwards, the use of LP-based BB algorithms seems to have been established as the overall best approach for solving SCP with proven optimality. In fact, a more recent survey by CAPRARA *et al.* [48] compares the performance of different exact algorithms for SCP, namely: the LR-based ones of BEASLEY [44], BEASLEY and JÖRNSTEN [46] and BALAS and CARRERA [45]; and LP-based ones, offered by the use of the general-purpose LP solvers IBM-CPLEX 4.0.8 and MINTO 2.3 [49]. According to the authors, CPLEX had the overall best performance across different classes of SCP benchmark instances. On the other hand, given the observed practical difficulty of solving SCP instances to optimality, the

development of effective heuristic algorithms proved to be of great importance and has been widely investigated. One of the earliest developments in this direction was the greedy heuristic proposed by CHVATAL [7] in 1979. The greedy heuristic begins with an empty solution and, at each iteration, inserts a new column into the current solution which is chosen based on a specific *greedy score function*. BALAS and CARRERA [45] and FEO and RESENDE [50] proposed alternative greedy score functions. However, it seems that the most effective SCP heuristics, at least so far, are those that employ LR and make use of the corresponding lagrangian information as part of the greedy heuristic. This idea was originally proposed by BALAS and HO [43] and subsequently many SCP heuristics based on LR have been proposed, with novel ideas incorporated to them. Probably the most effective contribution of the lagrangian approach for SCP heuristics is to define the greedy score as a function of the lagrangian multipliers. This was first proposed by BALAS and HO [43] and considered later by BALAS and CARRERA [45]. Afterwards, new lagrangian greedy score functions were proposed by CERIA *et al.* [9], and the most effective overall seems to be the one proposed by CAPRARA *et al.* [8]. BEASLEY [44] proposed variable fixing based on *lagrangian reduced costs*. CAPRARA *et al.* [51] and CERIA *et al.* [9] proposed working with a reduced *core problem* which considers at each iteration only a subset out of all the columns. CERIA *et al.* [9] also consider the simultaneous LR of both SCP and its dual problem, thus obtaining two LDPs. In doing so, the authors then perform subgradient optimization, in an attempt to obtain optimal lagrangian multipliers for both LDPs. Furthermore,they make use of the lagrangian information obtained on heuristics for the two lagrangian problems at hand. Another relevant line of research is the development of local-search algorithms for SCP. Given a known feasible solution for an optimization problem, the goal of a local-search algorithm is to search for solutions with better objective value by iteratively modifying the current solution and moving to another (possibly partial) solution whithin a formally defined *neighborhood* of the current solution. Some examples of local-search procedures for SCP are: those of YAGIURA *et al.* [10] and of GAO *et al.* [34]; the GRASP procedure of BAUTISTA and PEREIRA [52]; the GRASP and path-relinking procedures of PESSOA *et al.* [53]; and the local-search procedures for the unit-cost set cover problem of MUSLIU [54], of NAJI-AZIMI *et al.* [55] and of GAO *et al.* [11]. Other lines of research include: the genetic algorithm of BEASLEY and CHU [56]; and the simulated annealing based algorithms of BRUSCO *et al.* [57] and of JACOBS and BRUSCO [58].

In practice, many relevant combinatorial optimization problems are formulated either as an SCP or as a mixed-integer program that includes set cover constraints. Some examples of this which originate from practical applications are crew scheduling, including those for the airline [12], railway [8, 9, 13] and healthcare [14] indus-

tries, vehicle scheduling [15], assembly-line balancing [16], service location [17] and placement of surveillance cameras [59, 60]. For instance, the application of SCP to railway crew scheduling motivated the italian railway company, *Ferrovie dello Stato SpA*, to host a competition in 1994, called *FASTER* [18], with the goal of promoting the development of algorithms to better tackle large-scale instances of SCP which result from crew scheduling problems in the company. This, in turn, lead to the publication of a series of papers in the following years [8, 9, 19–21]. Moreover, the instances proposed during the FASTER competition have become a standard benchmark for large-scalle SCP algorithms in the literature up to this day.

2.2 Graphs

A *simple graph* is a pair $G = (V, E)$ where V is a finite set and $E \subseteq \{\{u, v\} : u, v \in V \text{ and } u \neq v\}$. Each element $v \in V$ is called a *vertex* and each element $e \in E$ is called an *edge*. An edge $\{u, v\} \in E$ is said to *join* vertices u and v, which are called its *endpoints*. Note that in the case of simple graphs, the edges are not ordered, and therefore $\{u, v\} = \{v, u\}$. Note as well that a simple graph does not contain *loops*, i.e, it does not contain edges of the kind $\{v, v\}$, for any $v \in V$. Additionally, they do not contain multiple edges between two vertices.

For the sake of notational convenience, given a simple graph $G = (V, E)$:

- For each vertex $v \in V$, let $N(v) = \{u \in V : \{v, u\} \in E\}$ be its *open neighborhood*;
- For each vertex $v \in V$, let $N[v] = \{v\} \cup N(v)$ be its *closed neighborhood*;
- For each vertex $v \in V$, let $\delta(v) = \{\{v, u\} \in E : u \in N(v)\}$ be the set of edges *incident* to v;
- For each edge $e = \{u, v\} \in E$, let $N(e) = \{e' \in E \setminus \{e\} : u \in e' \vee v \in e'\}$ be its *open neighborhood*;
- For each edge $e = \{u, v\} \in E$, let $N[e] = N(e) \cup \{e\}$ be its *closed neighborhood*;
- Given $V' \subseteq V$, let $E[V'] = \{\{u, v\} \in E : u \in V' \wedge v \in V'\}$ be the set of edges *induced* by V', i.e, the set of edges with both endpoints in V';
- Given $V' \subseteq V$, let $G[V'] = (V', E[V'])$ be the *induced subgraph* of V', i.e, the graph with vertex set V' and edge set $E[V']$;
- Given $E' \subseteq E$, let $V[E'] = \{v \in V : \exists e \in E' \wedge v \in e\}$ be the set of vertices *induced* by E', i.e, the set of vertices which are an endpoint of at least one edge in E';

- Given $E' \subseteq E$, let $G[E'] = (V[E'], E')$ be the *induced subgraph* of E', i.e, the graph with vertex set $V[E']$ and edge set E'.

Given a simple graph $G = (V, E)$ and two vertices $s, t \in V$, an (s,t)-*walk* in G is a finite non-null sequence of vertices $w_{st} = (v_0, \ldots, v_\ell)$ such that $v_0 = s$ and $v_\ell = t$ and $\{v_k, v_{k+1}\} \in E$ for every $k \in \{0, \ldots, \ell-1\}$. Hence a walk is a sequence of vertices such that two consecutive vertices are joined by an edge. In this case, ℓ is called the *length* of walk w_{st}. Note that $w_{s,s} = (s)$ is an (s,s)-walk of length zero. Given a walk w_{st}, let $E[w_{st}] = \{\{v_k, v_{k+1}\} \in E : k = 0, \ldots, \ell - 1\}$ be the set of edges induced by w_{st}. A *trail* is a walk with no repeated edges, i.e, it is a walk $t_{v_0,v_\ell} = (v_0, \ldots, v_\ell)$ such that $\{v_k, v_{k+1}\} \neq \{v_l, v_{l+1}\}$ for all $k, l \in \{0, \ldots, \ell\}, k \neq l$. In turn, a *path* is a trail with no repeated vertices, i.e, it is a trail $p_{v_0,v_\ell} = (v_0, \ldots, v_\ell)$ such that $v_k \neq v_l$ for all $k, l \in \{0, \ldots, \ell\}, k \neq l$. Note that $p_{s,s} = (s)$ is an (s,s)-path of length zero. Two vertices s and t are *connected* if there exists at least one (s,t)-path in G. Otherwise s and t are *disconnected*. Note that the connectivity relation in a simple graph is *symmetric*. Accordingly, if s and t are connected, then there exists an (s,t)-path $p_{s,t} = (s, v_0, \ldots, v_\ell, t)$ and a companion (t,s)-path $p_{t,s} = (t, v_\ell, \ldots, v_0, s)$. Any vertex s is connected to itself, since $p_{s,s} = (s)$ is an (s,s)-path. Hence the connectivity relation in a simple graph is *reflexive*. Let $\mathcal{P}(G, s, t) = \{p_{st} : p_{st} \text{ is an } (s,t)\text{-path}\}$ be the set of all (s,t)-paths in G.

Lemma 1. *Let $G = (V, E)$ be a simple graph and $s, t \in V$. For every (s,t)-walk there is an (s,t)-path.*

Proof. If $s = t$, then for any (s,s)-walk w_{st} there is $p_{s,s} = (s)$. Assume $s \neq t$ and let $w_{st} = (v_0, \ldots, v_\ell)$ be such that $s = v_0$ and $t = v_\ell$. We proceed by induction on ℓ. Base: if $\ell = 1$, then $w_{st} = (s,t)$ and w_{st} is a path. Induction hypothesis: suppose the claim is true for every $\ell \leq n$ for some $n > 1$. Induction step: consider $\ell = n + 1$. If all the vertices in w_{st} are distinct, then w_{st} is a path. Assume otherwise. Then, without loss of generality, there are $k, l \in \{0, \ldots, \ell\}$ such that $k \neq l$ and $v_k = v_l$. And therefore $w_{st} = (v_0, \ldots, v_{k-1}, v_k, v_{k+1}, \ldots, v_{l-1}, v_l, v_{l+1}, \ldots, v_\ell)$. Consider the (s,t)-walk $w'_{st} = (v_0, \ldots, v_{k-1}, v_k, v_{l+1}, \ldots, v_\ell)$ obtained by removing vertices $v_{k+1}, \ldots, v_{l-1}, v_l$ from w_{st}. Since G is a simple graph, then $\{k, l\} \notin E$ and therefore without loss of generality $l > k + 1$. Hence w_{st} has length at most $\ell - 1$, since at least one vertex has been removed. And since $\ell - 1 \leq n$, it follows from the the induction hypothesis that there is an (s,t)-path p'_{st}. □

Lemma 2. *Let $G = (V, E)$ be a simple graph and $v_1, v_2, v_3 \in V$. If v_1 and v_2 are connected and v_2 and v_3 are connected, then v_1 and v_3 are connected.*

Proof. Since v_1 and v_2 are connected, then there is a (v_1, v_2)-path $p_{v_1,v_2} = (v_1, u_0, \ldots, u_{\ell_1}, v_2)$. And since v_2 and v_3 are connected, then there is a (v_2, v_3)-path $p_{v_2,v_3} = (v_2, w_0, \ldots, w_{\ell_2}, v_3)$. Hence $w_{v_1,v_3} = (v_1, u_0, \ldots, u_{\ell_1}, v_2, w_0, \ldots, w_{\ell_2}, v_3)$

is a (v_1, v_3)-walk. It then follows from Lemma 1 that there is a (v_1, v_3)-path and therefore v_1 and v_3 are connected. □

Lemma 2 proves that the connectivity relation in a simple graph is also *transitive*, and thus it is an *equivalence relation*. A subset of vertices $V' \subseteq V$ is *connected* if s and t are connected for every $s, t \in V'$. Otherwise V' is *disconnected*. Additionally, a simple graph $G = (V, E)$ is *connected* if s and t are connected for every $s, t \in V$. For each $v \in V$, let $C[v] = \{v' \in V : v \text{ and } v' \text{ are connected in } G\}$ be the set of all vertices which are connected to v. Since every vertex $v \in V$ is connected to itself, then $v \in C[v]$ and therefore $C[v] \neq \emptyset$.

Lemma 3. *Let $G = (V, E)$ be a simple graph and $s, t \in V$. Vertices s and t are connected if and only if $C[s] = C[t]$.*

Proof. ($\rightarrow$): Suppose s and t are connected. Consider any $v \in C[s]$. It follows from the definition of $C[s]$ that s and v are connected. It then follows from Lemma 2 that t and v are connected. Hence $v \in C[t]$ and we conclude that $C[s] \subseteq C[t]$. In a similar way we conclude that $C[t] \subseteq C[s]$. Thus $C[s] = C[t]$. ($\leftarrow$): Suppose $C[s] = C[t]$. Since $s \in C[s]$, then $s \in C[t]$ and thus, by definition, s and t are connected. □

Lemma 4. *Let $G = (V, E)$ be a simple graph and $s, t \in V$. Vertices s and t are disconnected if and only if $C[s] \neq C[t]$.*

Proof. Follows from the negation of Lemma 3. □

Lemma 5. *Let $G = (V, E)$ be a simple graph and $s, t \in V$. $C[s] \neq C[t]$ if and only $C[s] \cap C[t] = \emptyset$.*

Proof. Since $C[s]$ and $C[t]$ are non-empty, it suffices to prove that $C[s] \cap C[t] = \emptyset$. Suppose, for the sake of contradiction, that there is $v \in C[s] \cap C[t]$. Then by definition s and v are connected and t and v are connected. It then follows from Lemma 2 that s and t are connected, a contradiction. □

Lemmas 3, 4 and 5 guarantee that there is a partition of V into pairwise disjoint non-empty maximal connected subsets $C[v_1], \ldots, C[v_m]$. A non-empty subset of vertices $V' \subseteq V$ is a *connected component* of G if V' is a maximal (with respect to inclusion) connected subset of vertices, i.e, if V' is connected and $V' \cup \{u\}$ is disconnected for every $u \in V \setminus V'$. Hence a simple graph G can be expressed as a disjoint union of the connected subgraphs $G[C[v_1]], \ldots, G[C[v_m]]$ induced by its connected components.

Given $v \in V$, let $C[v]$ be the connected component of v.

Lemma 6. *Let $G = (V, E)$ be a simple graph. For every $v \in V$, there is a unique connected component $C[v] \in C[G]$ such that $v \in C[v]$.*

Proof. Consider any $v \in V$ and $C[v] = \{v' \in V : v \text{ and } v' \text{ are connected in } G\}$. It follows from Lemma 3 that $C[v]$ is connected. Consider any $u \in V \setminus C[v]$. It follows from Lemma 4 that v and u are disconnected. And thus $C[v] \cup \{u\}$ is disconnected. Hence $C[v]$ is a maximal connected subset and thus a connected component. Suppose there are two connected components C_1 and C_2 such that $v \in C_1$ and $v \in C_2$. Consider any $u \in C_1$. Since C_1 is connected, then v and u are connected. Then, since $v \in C_2$ and C_2 is a maximal connected subset, then $u \in C_2$. Hence $u \in C_2$ and thus $C_1 \subseteq C_2$. In a similar way we prove that $C_2 \subseteq C_1$. And thus $C_1 = C_2$. □

Lemma 6 guarantees that for every vertex $v \in V$, there is a unique connected component containing v. Given $v \in V$, let $C[v] = \{v' \in V : v \text{ and } v' \text{ are connected}\}$ be the unique connected component containing v. Lemma 3 guarantees that two vertices are connected if and only if they are in the same connected component, two vertices are disconnected if they are in different connected components. Let $C[G] = \{C \subseteq V : C \text{ is a connected component of } G\}$ be the set of all connected components of G.

Lemma 7. *Let $G = (V, E)$ be a simple graph such that $V \neq \emptyset$. If G is connected, then $|C[G]| = 1$.*

Proof. Since G is connected, it follows by definition that V is connected. And since V is maximal with respect to inclusion, then V is a connected component. Any other connected subset $V' \subseteq V$ is not a maximal connected subset and therefore is not a connected component. Hence $C[G] = \{V\}$ and thus $|C[G]| = 1$. □

Lemma 8. *Let $G = (V, E)$ be a simple graph. If $|C[G]| = 1$, then G is connected.*

Proof. Consider any $s, t \in V$. Since $|C[G]| = 1$, then $C[s] = C[t]$. It then follows from Lemma 3 that s and t are connected. □

Lemma 9. *Let $G = (V, E)$ be a simple graph such that $V \neq \emptyset$. G is connected if and only if $|C[G]| = 1$.*

Proof. ($\rightarrow$): Follows from Lemma 7. ($\leftarrow$): Follows from Lemma 8. □

Lemma 10. *Let $G = (V, E)$ be a simple graph such that $V \neq \emptyset$. G is disconnected if and only if $|C[G]| > 1$.*

Proof. Since $V \neq \emptyset$, then G has at least one connected component and thus $|C[G]| \geq 1$. The result then follows from the negation of Lemma 9. □

Hence a non-empty simple graph is connected if and only if it has exactly one connected component, and it is disconnected if and only if it has at least two components. Note that since a connected component is by definition non-empty, then the

empty graph $(\emptyset, \emptyset)$ has zero connected components and thus is the only connected graph G such that $C[G] \neq 1$.

2.2.1 Optimization Problems in Graphs

For many problems and applications, *weights* are associated to the vertices and/or edges of the graph. Given a weight $w_v \in \mathbb{R}$ for each vertex $v \in V$, the weight $w(V')$ of a subset of vertices $V' \subseteq V$ is defined as $W(V') = \sum_{v \in V'} w_v$. Similarly, given a weight $w_e \in \mathbb{R}$ for each edge $e \in E$, the weight $W(E')$ of a subset of edges $E' \subseteq E$ is defined as $W(E') = \sum_{e \in E'} w_e$. Note that when all weights are equal to 1, then the weight of any subset corresponds to its cardinality. This means that when $w_v = 1$ for every $v \in V$, then $W(V') = |V'|$ for every $V' \subseteq V$. Similarly, when $w_e = 1$ for every $e \in E$, then $W(E') = |E'|$ for every $E' \subseteq E$.

A set $\mathcal{D} \subseteq V$ is a *dominating set* if $N[v] \cap \mathcal{D} \neq \emptyset$ for every $v \in V$. Hence $\mathcal{D} \subseteq V$ is a dominating set if for every vertex $v \in V$, either $v \in \mathcal{D}$ or there is $u \in N(v)$ such that $u \in \mathcal{D}$. The *min weight dominating set problem* (MWDSP) asks for a dominating set $\mathcal{D}^* \subseteq V$ such that $W(\mathcal{D}^*) \leq W(\mathcal{D})$ for every dominating set $\mathcal{D} \subseteq V$. Accordingly, the *min dominating set problem* (MDSP) asks for a dominating set of minimum cardinality. Note that V is a dominating set and therefore the MWDSP and MDSP have at least one solution.

A set $\mathcal{D} \subseteq V$ is a *total dominating set* if $N(v) \cap \mathcal{D} \neq \emptyset$ for every $v \in V$. Hence $\mathcal{D} \subseteq V$ is a total dominating set if for every vertex $v \in V$ there is $u \in N(v)$ such that $u \in \mathcal{D}$. The *min weight total dominating set problem* (MWTDSP) asks for a total dominating set $\mathcal{D}^* \subseteq V$ such that $W(\mathcal{D}^*) \leq W(\mathcal{D})$ for every total dominating set $\mathcal{D} \subseteq V$. Accordingly, the *min total dominating set problem* (MTDSP) asks for a total dominating set of minimum cardinality. Note that as long as $N(v) \neq \emptyset$ for every $v \in V$, then V is a total dominating set and therefore MWTDSP and MTDSP have at least one solution.

A set $\mathcal{C} \subseteq V$ is a *vertex-cover* if $u \in \mathcal{C}$ or $v \in \mathcal{C}$ for every edge $\{u, v\} \in E$. Hence $\mathcal{C} \subseteq V$ is a set cover if for every edge $e \in E$ at least one of its endpoints is part of $\mathcal{C}$. The *min weight vertex cover problem* (MWVCP) asks for a vertex-cover $\mathcal{C}^* \subseteq V$ such that $W(\mathcal{C}^*) \leq W(\mathcal{C})$ for every vertex-cover $\mathcal{C} \subseteq V$. Accordingly, the *min vertex-cover problem* (MVCP) asks for a vertex-cover of minimum cardinality. Note that V is a vertex-cover and therefore MWVCP and MVCP have at least one solution.

A set $\mathcal{D} \subseteq E$ is an *edge-dominating set* if $N[e] \cap \mathcal{D} \neq \emptyset$ for every edge $e \in E$. Hence $\mathcal{D}$ is an edge-dominating set if for every edge $e \in E$, either $e \in \mathcal{D}$ or there is $e' \in N(e)$ such that $e' \in \mathcal{D}$. The *min weight edge-dominating set problem* (MWEDSP) asks for an edge-dominating set $\mathcal{D}^* \subseteq E$ such that $W(\mathcal{D}^*) \leq W(\mathcal{D})$

for every edge-dominating set $\mathcal{D} \subseteq E$. Accordingly, the *min edge-dominating set problem* (MEDSP) asks for an edge-dominating set of minimum cardinality. Note that E is an edge-dominating set and therefore MWEDSP and MEDSP have at least one solution.

We denote an instance of any of these problems by (G, w), where $G = (V, E)$ is a simple graph and $w \in \mathbb{R}^{|V|}_{>0}$. Note that in the case of dominating sets, total dominating sets and vertex covers, if $V = \emptyset$, then any of these problems only contains the trivial solution $\emptyset$, which vacuously satisfies the conditions for domination or coverage, as directly implied by the fact that $V = \emptyset$. The same is true for MWEDSP if $E = \emptyset$. Hence from this point forward we assume that $V \neq \emptyset$ and $E \neq \emptyset$. Moreover, for all these problems, one can assume without loss of generality that G is connected, because solving the problem is equivalent to solving an instance of the same problem for each connected component of the graph. In fact, consider an instance (G, w) and suppose G is not connected. Then G has at least two connected components. For each connected component $C \in C[G]$, consider the instance $(G[C], w^C)$ obtained from (G, w) by considering the subgraph $G[C]$ induced by C and defining $w^C_v = w_v$ for $v \in C$. Let $S^* \subseteq V$ be an optimal solution of (G, w) and let $S^C \subseteq C$ be an optimal solution of $(G[C], w^C)$. We first consider the case of MWDSP, MWTDSP and MWVC. Note that $W(S^*) = W(S^* \cap C) + W(S^* \setminus C)$. Since S^C is an optimal solution to $(G[C], w^C)$, then $W(S^C) \leq W(S \cap C)$. Suppose, for the sake of contradiction, that $W(S^C) < W(S \cap C)$. Consider $S' \subseteq V$ such that $S' = S^C \cup (S^* \setminus C)$. Then $W(S') = W(S^C) + W(S^* \setminus C) < W(S^* \cap C) + W(S^* \setminus C) = W(S^*)$, a contradiction, since S^* is an optimal solution. Hence $W(S^C) = W(S \cap C)$ for every connected component $C \in C[G]$. Therefore, if G is not connected, then solving instance (G, w) is equivalent to solving instance $(G[C], w^C)$ for each connected component $C \in C[G]$. For each $C \in C[G]$, let S^C be an optimal solution to $(G[C], w^C)$. Then $S^* = \bigcup_{C \in C[G]} S^C$ is an optimal solution to (G, w). The proof for the case of MWEDSP is obtained along similar lines, but considering edges instead of vertices.

The assumption that the weights are positive is also made without loss of generality, in a manner similar to the one presented in Section 2.1 for the set cover problem.

It is straightforward to cast as SCP any of the domination and cover problems discussed in this subsection. In particular, this was originally done for MDSP by SLATER [61]. The general approach consists of appropriately defining an incidence matrix A and a vector of costs c in such a way to express any instance of these problems as an SCP instance.

- For MWDSP and MDSP, let $A \in \{0, 1\}^{|V| \times |V|}$ be such that for each $u, v \in V$, $a_{uv} = 1$ if $v \in N[u]$ and $a_{uv} = 0$ otherwise. And let $c_v = w_v$ for every $v \in V$.

- For MWTDSP and MTDSP, let $A \in \{0,1\}^{|V|\times|V|}$ be such that for each $u, v \in V$, $a_{uv} = 1$ if $v \in N(u)$ and $a_{uv} = 0$ otherwise. And let $c_v = w_v$ for every $v \in V$.

- For MWVCP and MVCP, consider the indexed edge set $E = \{e_1, \ldots, e_{|E|}\}$ and let $A \in \{0,1\}^{|E|\times|V|}$ be such that for each vertex $v \in V$ and each edge $e_i \in E$, $a_{iv} = 1$ if $v \in e_i$ and $a_{iv} = 0$ otherwise. And let $c_v = w_v$ for every $v \in V$.

- For MWEDSP and MEDSP, consider the indexed edge set $E = \{e_1, \ldots, e_{|E|}\}$ and let $A \in \{0,1\}^{|E|\times|E|}$ be such that for each $e_i, e_j \in E$, $a_{ij} = 1$ if $e_j \in N[e_i]$ and $a_{ij} = 0$ otherwise. And let $c_j = w_{e_j}$ for every $j \in \{1, \ldots, |E|\}$.

Note that for MWDSP, MWTDSP and MWVCP, the costs of the corresponding SCP are equal to the weights associated with the vertices. For MWEDSP, these costs are equal to the weights associated with the edges. Finally, for mininimum cardinality problems, any cost is equal to 1, or more generally, all costs have the same value.

The so-called *connected variants* of the problems mentioned in the previous paragraph arise when one further imposes the constraint that the subgraph induced by the solution must be connected.

A set $\mathcal{D} \subseteq V$ is a *connected dominating set* if $\mathcal{D}$ is a dominating set and $G[\mathcal{D}]$ is connected. The *min weight connected dominating set problem* (MWCDSP) asks for a connected dominating set $\mathcal{D}^* \subseteq V$ such that $W(\mathcal{D}^*) \leq W(\mathcal{D})$ for every connected dominating set $\mathcal{D} \subseteq V$. Similarly, the *min connected dominating set problem* (MCDSP) asks for a connected dominating set of minimum cardinality. Note that as long as G is connected, then V is a connected dominating set and therefore MWCDSP and MCDSP have at least one solution. On the other hand, if G is disconnected, then MWCDSP and MCDSP are infeasible in G and have no solution. In fact, any dominating set $\mathcal{D} \subseteq V$ satisfies that $\mathcal{D} \cap C \neq \emptyset$ for every connected component $C \in C[G]$. Thus, if G is disconnected, then $|C[G]| \geq 2$ and $\mathcal{D}$ is not connected. Note that if $|V| \geq 2$, then a connected dominating set is a total dominating set, since if $v \in \mathcal{D}$, then v cannot be an isolate vertex in $G[\mathcal{D}]$.

A set $\mathcal{C} \subseteq V$ is a *connected vertex-cover* if $\mathcal{C}$ is a vertex cover and $G[\mathcal{C}]$ is connected. The *min weight connected vertex cover problem* (MWCVCP) asks for a connected vertex-cover $\mathcal{C}^* \subseteq V$ such that $W(\mathcal{C}^*) \leq W(\mathcal{C})$ for every connected vertex cover $\mathcal{C} \subseteq V$. The *min connected vertex cover problem* (MCVCP) asks for a connected vertex-cover of minimum cardinality. Note that as long as G is connected, then V is a connected vertex-cover and therefore MWCVCP and MCVCP have at least one solution. On the other hand, if G is disconnected, then MWCVCP and MCVCP are infeasible in G and have no solution. In fact, any vertex cover $\mathcal{C} \subseteq V$ satisfies that $\mathcal{C} \cap C \neq \emptyset$ for every connected component $C \in C[G]$.

A set $\mathcal{D} \subseteq E$ is a *connected edge-dominating set* if $\mathcal{D}$ is an edge-dominating set and $G[\mathcal{D}]$ is connected. The *min weight connected edge-dominating set* (MWCEDSP) asks for a connected-edge dominating set $\mathcal{D}^* \subseteq E$ such that $W(\mathcal{D}^*) \leq W(\mathcal{D})$ for every connected-edge dominating set $\mathcal{D} \subseteq E$. Alternatively, a connected edge-dominating set is might also be referred to as a *tree-cover*. This follows from the fact that the vertices induced by a connected edge-dominating set define a vertex-cover. Moreover, if all edge weights are positive, then the optimal solution to MWCEDSP is a tree, i.e, a connected set with no cycles. Accordingly, a set $\mathcal{T} \subseteq E$ is a *tree-cover* if $\mathcal{T}$ is an edge-dominating set and $G[\mathcal{T}]$ is a tree. The *min weight tree-cover problem* (MWTCP) asks for a tree-cover $\mathcal{T}^* \subseteq E$ such that $W(\mathcal{T}^*) \leq W(\mathcal{T})$ for every tree-cover $\mathcal{T} \subseteq E$. The *min tree cover problem* (MTCP) asks for a tree-cover of minimum cardinality. Note that if G has at least two distinct connected components $C_1, C_2 \in C[G]$ such that $E[C_1] \neq \emptyset$ and $E[C_2] \neq \emptyset$, then MWCEDSP and MWTCP have no solution and hence are infeasible. Otherwise, E is an edge-dominating set and a tree-cover can be obtained from E by iteratively removing edges until it contains no cycles. Hence, as long as the edges of G induce a connected subset, then MWCEDSP and MWTCP have at least one solution. Moreover, one can assume without loss of generality that G has no isolate vertices, since isolate vertices do not have any incident edges. In fact, consider an instance (G, w) and suppose there is v' such that v' is isolate. Consider the instance (G', w) obtained from (G, w) by considering the subgraph $G' = (V \setminus \{v'\}, E)$ obtained by removing v'. Since both instances are defined on the same edge set as well as the same edge weights, then clearly any optimal to (G', w) corresponds exactly to an optimal solution of (G, w) and vice-versa.

2.2.2 Literature Review

The problems presented in Subsection 2.2.1 have been extensively studied and find a wide variety of applications, specially in network design problems. The vertex-cover, connected vertex-cover, dominating set, total dominating set, connected dominating set and edge-dominating set problems have been proven to be NP-complete by GAREY [6]. The connected edge-dominating set problem and tree-cover problem were originally considered by ARKIN *et al.* [62] and proven to be NP-hard. Since these problems can be formulated as set cover problems, much of the work and progress made for SCP also applies to them. However, each one of these problems has also been investigated individually and in greater detail. Similarly to SCP, due to the computational difficulty and practical relevance of these problems, much of the work reported on the literature focus on exact, approximation and heuristic approaches, as well as applications. The books by HAYNES *et al.* [63] and HAYNES

[64] provide a thorough report on theoretical and computational aspects of the dominating set problem, as well as the surveys by COCKAYNE and HEDETNIEMI [65], COCKAYNE [66] and CHANG [67]. Both FOMIN *et al.* [68] and VAN ROOIJ and BODLAENDER [69] propose exact algorithms specifically for DSP, which rely on enumeration, however also take into consideration the graph structure in order to reduce the proven complexi1ty. As for heuristic approaches for DSP, WANG *et al.* [70] propose a local-search and ROMANIA [71] an ant-colony optimization approach for the weighted variant, and ABED *et al.* [72] a local-search approach for the unit-weight variant. MTDSP was originally introduced by COCKAYNE *et al.* [73]. An extensive study on the theory of total dominating sets is presented in HENNING and YEO [74]. ÁLVAREZ-MIRANDA and SINNL [75] propose mixed-integer programming formulations for MWTDSP and a class of valid inequalities. As for heuristic approaches, HU *et al.* [76] propose a local-search method and YUAN *et al.* [77] propose a hybrid approach involving a genetic algorithm and local-search. Applications of dominating, total dominating and connected dominating sets have been reported to graph mining [78], social network analysis [79] and specially to network design [80–84]. Dominating sets, total dominating sets and connected dominating sets find a wide variety of applications specially in wireless networks. A *mobile ad-hoc network* (MANET) or *wireless ad-hoc network* (WANET) consists of a decentralized network of autonomous mobile nodes that communicate wirelessly. Instead of having a fixed infrastructure, such as routers and access-points that receive and transmit data to nodes, the network is ad-hoc because of its self-configurable nature. For economic or other practical reasons, the nodes usually have limited capabilities. Additionally, due to limited radio range, each node is only able to communicate with other nodes within a certain range. Finally, due to limited battery storage or limited processing capacity, nodes might not be available at all times. For such reasons, the network configuration varies over time. Two nodes which are sufficiently close to each other are able to communicate directly by radio. On the other hand, communication between two nodes which are not within each other's range happens by iteratively transmitting information to another node within range in what is known as *multi-hop communication*, thus allowing routing(one-to-one communication), multi-casting(one-to-many) and broadcasting(one-to-all). Such networks can be represented by a *unit-disk graph*. Given a finite set of points V in the euclidean plane and a range $r > 0$, the unit-disk graph $G_U = (V, E)$ corresponding to V and r is the graph such that $\{u, v\} \in E$ if $d(u, v) \leq r$ and $\{u, v\} \notin E$ otherwise. Hence the unit-disk graph contains an edge $\{u, v\}$ for every pair of nodes which are within a distance of at most r of each other. One important application of dominating and connected dominating sets is determining a *virtual backbone* for an ad-hoc network, i.e, determine a set of nodes which are assigned to pass information along the network

in a multi-hop fashion. The nodes which are part of the backbone need to be better equipped in order to posses routing features and have higher energy and processing capabilities and thus are more costly than other nodes in the network. Therefore designing a smaller backbone is crucial in terms of energy efficiency and the total cost of the network. Considering the unit-disk graph G_U corresponding to an ad-hoc network, by selecting a dominating set of nodes in G_U to be part of the backbone, one ensures that every node is connected to at least one node in the backbone. An example of this process is presented in Figure 2.3. By further imposing the backbone to be a connected dominating set, then every two nodes can communicate with each other by iteratively transmitting information along the nodes in the backbone. Determining a connected dominating set of minimum weight or cardinality has advantages such as reducing routing and broadcast overhead by minimizing redundant transmission of data and thus making communication more energy-efficient. A survey of applications of dominating sets to MANET is presented in YU *et al.* [85]. A common example of MANET are *wireless sensor networks* (WSN). A wireless sensor network consists of autonomous smart sensor devices which are usually spread across a geographical area in order to monitor physical or enviromental conditions and report that data to a central storage or processing unit. Generally the sensors are able to communicate whithin a certain range by radio. Similarly to MANET, wireless sensor networks can be modelled as unit-disk graphs. One example of application of vertex-covers is in WSN security. Due to its communication nature, WSN are prone to cyberattacks which possibly compromise data integrity and confidentiality, thus increasing the importance of monitoring the links in the network. This can be achieved by placing *secure points* on network nodes, which analyze and log traffic through that node and verify its security. A vertex-cover ensures that each link between two nodes is monitored in at least one of its participating nodes, and thus finds applications such as reported by SAFAR *et al.* [86], DAGDEVIREN [87] and DAGDEVIREN [88]. When communication between monitors is required, connectivity must be imposed in order to determine a backbone for multi-hop communication, and thus one aims for a minimum connected vertex-cover [26]. The tree cover problem was originally reported by ARKIN *et al.* [62] and proven to be NP-hard in its optimization version. The problem is motivated by the *watchman route problem*, and the goal is to find a shortest path in a graph such that each edge can be monitored [89]. Other applications include facility location problems as well as transportation network design problems. In general, facility location problems concern the decision of choosing one or more locations in a given network where to place a facility that services other demand points. At least originally, these facility location problems considered the case that each facility was located on an individual node, but SLATER [27] generalized the idea to allow facilities to take the shape

of a path or a tree, which was later investigated by MINIEKA [28], RICHEY [29] and HAKIMI *et al.* [30]. Such generalization arises naturally for applications such as railroad lines, highways, transmission or pipeline networks [27, 31], in which geographical distance is directly related to construction or operation costs. The min tree-cover arises when one aims to find a central facility of minimum total length, and other classes of similar problems arise when other objectives are considered, such as maximum distance or eccentricity. MÖLLE *et al.* [90] study in greater detail the complexity and approximability of the connected vertex-cover and tree-cover problems.

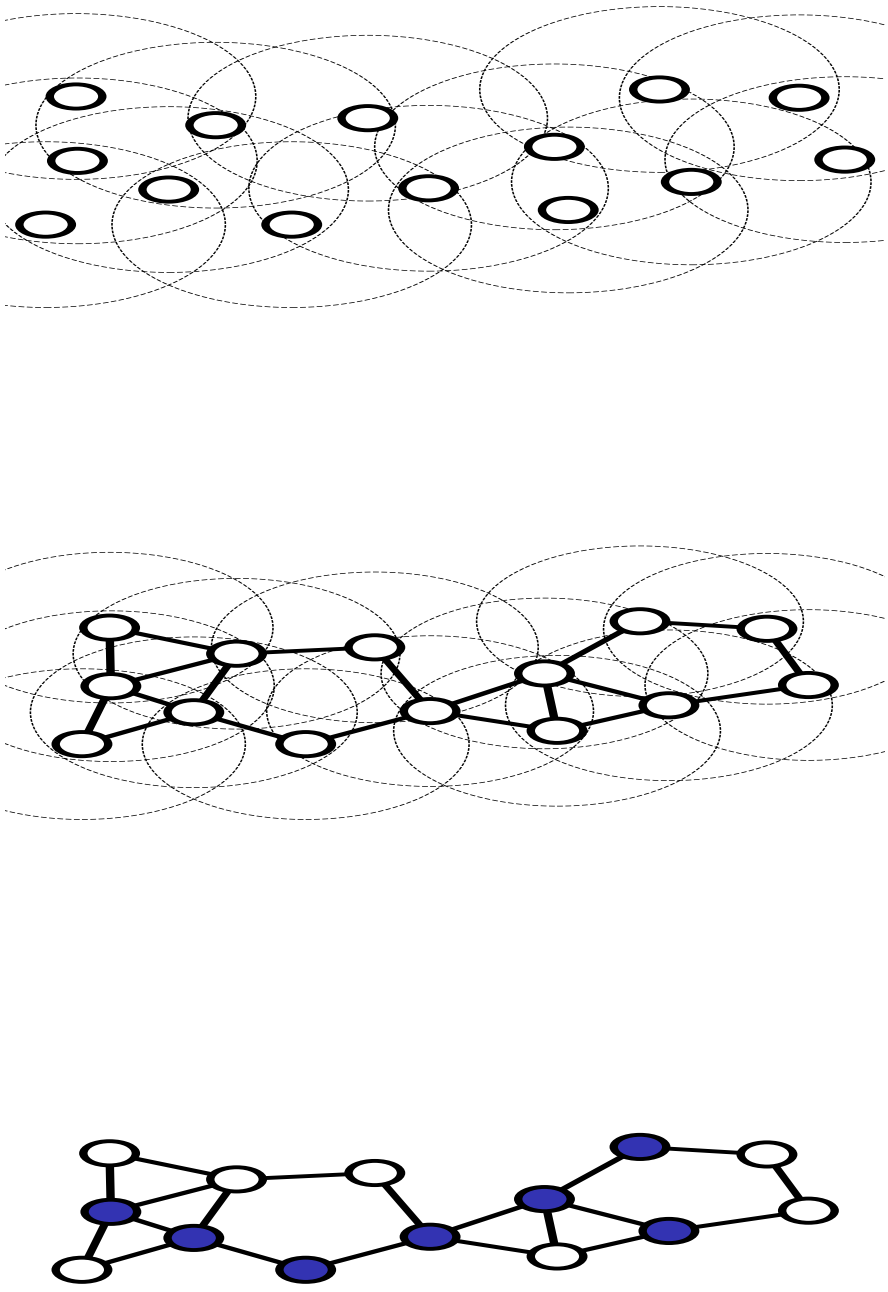

Figure 2.3: Determining a virtual backbone for a wireless network. Each device has a position on the plane, and can communicate by radio with other devices within a certain radius. The network is represented by a unit-disk graph G_U. A connected dominating set on G_U defines a virtual backbone. Information can be transmitted between any two nodes in the network by multi-hopping information through backbone nodes.

Chapter 3

Imposing Connectivity by Integer Programming

In Section 3.1 we consider some fundamental results from graph theory concerning connectivity and the standard methods for imposing connectivity constraints in integer programming. In Section 3.3 we present characterizations of connected dominating sets, connected vertex-covers and connected edge-dominating sets and formulate these problems as SCP. Finally, in Section 3.4 we describe the separation of vertex-cut constraints.

3.1 Theory

Similar to these connected variants, many other combinatorial optimization problems which arise in the context of graphs require the subgraph induced by the solution to be connected. A common approach is to model these problems as integer programs. For this reason, different ways to impose connectivity by means of integer programming have been studied and reported in the literature. Both WANG *et al.* [91] and REHFELDT *et al.* [92] survey different classes of linear integer programming constraints that impose the solution to induce a connected subgraph. In general, in the case of simple graphs, these classes can be classified as: *vertex-based* formulations, which consider variables defined over the vertices of the graph and are based on the concepts of vertex-cuts and vertex-separators; or *edge-based* formulations, which consider variables defined over the edges of the graph and are based on the concept of edge-cuts or subtour elimination.

Given a connected simple graph $G = (V, E)$, a non-empty subset of vertices $S^V \subset V$ is a *vertex-cut* if $G[V \setminus S^V]$ is disconnected. Hence a vertex-cut is a set $S^V \subset V$ such that the subgraph obtained from G by removing the vertices in S^V is disconnected. Let $\mathcal{S}^V(G) = \{S^V \subset V : S^V \text{ is a vertex-cut}\}$ be the set of all vertex-

cuts in G. If one is given two connected vertices $s, t \in V$, a non-empty subset of vertices $S^V_{st} \subset V$ is an (s,t)-*separator* if $s, t \notin S^V_{st}$ and s and t are not connected in $G[V \setminus S^V_{st}]$. Hence an (s,t)-separator is a set $S^V_{st} \subset V$ such that the subgraph obtained from G by removing the vertices in S^V_{st} does not contain an (s,t)-path. Let $\mathcal{S}^V(G,s,t) = \{S^V_{st} \subset V : S^V_{st}$ is an (s,t)-separator$\}$ be the set of all (s,t)-separators in G. Note that $\mathcal{S}^V(G,v,v) = \emptyset$ for every vertex $v \in V$ and $\mathcal{S}^V(G,u,v) = \emptyset$ for every edge $\{u,v\} \in E$.

Lemma 11. *Let $G = (V,E)$ be a connected simple graph. If $S^V \in \mathcal{S}^V(G)$, then $S^V \in \mathcal{S}^V(G,s,t)$ for some $s,t \in V$.*

Proof. Consider any vertex-cut $S^V \in \mathcal{S}^V(G)$. Since $G[V \setminus S^V]$ is disconnected and $V \setminus S^V \neq \emptyset$, it then follows from Lemma 9 that $G[V \setminus S^V]$ has at least two connected components. Let C_s, C_t be two distinct connected components of $G[V \setminus S^V]$. Consider any $s \in C_s$ and any $t \in C_t$. Since s and t are in different connected components, it follows from Lemma 4 that s and t are not connected in $G[V \setminus S^V]$. Hence S^V is an (s,t)-separator. □

Lemma 12. *Let $G = (V,E)$ be a connected simple graph. If $S^V_{st} \in \mathcal{S}^V(G,s,t)$, then $S^V_{st} \in \mathcal{S}^V(G)$.*

Proof. Consider any (s,t)-separator $S^V_{st} \in \mathcal{S}^V(G,s,t)$. Since S^V_{st} is an (s,t)-separator, then by definition s and t are disconnected in $G[V \setminus S^V_{st}]$. Thus $G[V \setminus S^V_{st}]$ is disconnected. Hence S^V_{st} is a vertex-cut. □

Lemma 13. *Let $G = (V,E)$ be a connected simple graph. $S^V \in \mathcal{S}^V(G)$ if and only if $S^V \in \mathcal{S}^V(G,s,t)$ for some $s,t \in V$.*

Proof. ($\rightarrow$): Follows from Lemma 11. ($\leftarrow$): Follows from Lemma 12. □

A non-empty subset of edges $S^E \subset E$ is an *edge-cut* if the subgraph $(V, E \setminus S^E)$ is disconnected. Hence $S^E \subset E$ is an edge-cut if the subgraph obtained from G by removing the edges in S^E is disconnected. Let $\mathcal{S}^E(G) = \{S^E \subset E :$ S^E is an edge-cut in $G\}$ be the set of all edge-cuts in G. If one is given two connected vertices $s, t \in V$, a subset of edges $S^E_{st} \subset E$ is an (s,t)-*cut* if $s, t \in V[E \setminus S^E_{st}]$ and s and t are disconnected in $G[E \setminus S^E_{st}]$. Hence $S^E_{st} \subseteq E$ is an (s,t)-cut if the subgraph $G[E \setminus S^E_{st}]$ obtained from G by removing the edges in S^E_{st} contains vertices s and t and does not contain an (s,t)-path. Let $\mathcal{S}^E(G,s,t) = \{S^E_{st} \subseteq E : S^E_{st}$ is an (s,t)-cut$\}$ be the set of all (s,t)-cuts in G.

Lemma 14. *Let $G = (V,E)$ be a connected simple graph. If $S^E \in \mathcal{S}^E(G)$, then $S^E \in \mathcal{S}^E(G,s,t)$ for some $s,t \in V$.*

Proof. Consider any edge-cut $S^E \in \mathcal{S}^E(G)$. Since $G[E \setminus S^E]$ is not connected and $E \setminus S^E \neq \emptyset$, it then follows from Lemma 9 that $G[E \setminus S^E]$ has at least two connected components. Let C_s, C_t be two distinct connected components of $G[E \setminus S^E]$. Consider any $s \in C_s$ and any $t \in C_t$. Since s and t are in different connected components, it follows from Lemma 4 that s and t are not connected in $G[E \setminus S^E]$. Hence S^E is an (s,t)-cut. □

Lemma 15. *Let $G = (V, E)$ be a connected simple graph. If $S^E_{st} \in \mathcal{S}^E(G, s, t)$, then $S^E_{st} \in \mathcal{S}^E(G)$.*

Proof. Consider any (s,t)-cut $S^E_{st} \in \mathcal{S}^E(G, s, t)$. Since S^E_{st} is an (s,t)-cut, then s and t are disconnected in $G[E \setminus S^E_{st}]$. Thus $G[E \setminus S^E_{st}]$ is disconnected. Hence S^E_{st} is an edge-cut. □

Lemma 16. *Let $G = (V, E)$ be a connected simple graph. $S^E \in \mathcal{S}^E(G)$ if and only if $S^E \in \mathcal{S}^E(G, s, t)$ some $s, t \in V$.*

Proof. ($\rightarrow$): Follows from Lemma 14. ($\leftarrow$): Follows from Lemma 15. □

In what follows, let $G = (V, E)$ be a connected simple graph and suppose we aim to formulate a linear integer program to solve a combinatorial optimization problem in G and impose connectivity constraints on the subgraph induced by the solution. First, we consider vertex-based connectivity constraints. For each vertex $v \in V$, let $y_v \in \{0, 1\}$ be a binary decision variable associated with vertex v such that $y_v = 1$ if v is part of the solution and $y_v = 0$ otherwise. For any $y \in \{0, 1\}^{|V|}$, let $V[y] = \{v \in V : y_v = 1\}$ be the set of vertices induced by the solution y and $G[y] = G[V[y]]$ be the subgraph induced by y.

The *vertex-separator constraints* are defined in the following way:

$$\sum_{v \in S^V_{st}} y_v \quad \geq \quad y_s + y_t - 1 \quad \text{for every } s, t \in V, S^V_{st} \in \mathcal{S}^V(G, s, t) \tag{VSC}$$

Note that (VSC) ensure that if $s, t \in V[y]$, then y satisfies that $V[y] \cap S^V_{st} \neq \emptyset$ for every (s,t)-separator $S^V_{st} \in \mathcal{S}^V(G, s, t)$. Hence if two vertices s and t are induced by the solution y, then $V[y]$ intersects every (s,t)-separator. The following results state that this condition is necessary and sufficient to guarantee the connectivity of $G[y]$.

Since any path $p = (v_0, \ldots, v_\ell)$ in a simple graph $G = (V, E)$ defines a subset of vertices $\{v_0, \ldots, v_\ell\}$ induced by p, we introduce the following notation.

Given $v \in V$, $V' \subseteq V$ and a path $p = (v_0, \ldots, v_\ell)$:

- Let $V[p] = \{v_0, \ldots, v_\ell\}$ be the set of vertices induced by p.

- Let $v \in p$ if $v \in V[p]$.
- Let $p \cap V' = V[p] \cap V'$.
- Let $p \subseteq V'$ if $V[p] \subseteq V'$.

Lemma 17. *Let $G = (V, E)$ be a connected simple graph and $s, t \in V$. If $p_{st} \in \mathcal{P}(G, s, t)$, then $p_{st} \cap S^V_{st} \neq \emptyset$ for every $S^V_{st} \in \mathcal{S}^V(G, s, t)$.*

Proof. Consider any $p_{st} \in \mathcal{P}(G, s, t)$. Consider any (s, t)-separator $S^V_{st} \in \mathcal{S}^V(G, s, t)$. By definition, $G[V \setminus S^V_{st}]$ does not contain any (s, t)-path. Thus $p_{st} \not\subseteq V \setminus S^V_{st}$. Since $V[p_{st}]$ and $V \setminus S^V_{st}$ are non-empty, this implies that there is $v \in p_{st}$ such that $v \notin V \setminus S^V_{st}$. Thus $v \in S^V_{st}$ and since $v \in p_{st}$, it then follows that $p_{st} \cap S^V_{st} \neq \emptyset$. □

Lemma 18. *Let $G = (V, E)$ be a connected simple graph, $V' \subseteq V$ and $s, t \in V'$. If s and t are connected in $G[V']$, then $V' \cap S^V_{st} \neq \emptyset$ for every $S^V_{st} \in \mathcal{S}^V(G, s, t)$.*

Proof. If $\mathcal{S}^V(G, s, t) = \emptyset$, then the claim is vacuously true. Assume $\mathcal{S}^V(G, s, t) \neq \emptyset$. This implies that $s \neq t$. Since s and t are connected in $G[V']$, then there is an (s, t)-path $p_{st} \subseteq V'$. Consider any $S^V_{st} \in \mathcal{S}^V(G, s, t)$. It follows from Lemma 17 that $p_{st} \cap S^V_{st} \neq \emptyset$ and since $p_{st} \subseteq V'$, it then follows that $V' \cap S^V_{st} \neq \emptyset$. □

Lemma 19. *Let $G = (V, E)$ be a connected simple graph, $V' \subseteq V$ and $s, t \in V'$. If $V' \cap S^V_{st} \neq \emptyset$ for every $S^V_{st} \in \mathcal{S}^V(G, s, t)$, then s and t are connected in $G[V']$.*

Proof. If $|V'| \leq 1$, then the claim is vacuously true, since $\mathcal{S}^V(G, s, t) = \emptyset$. Assume $|V'| \geq 2$. Since G is connected, then there is at least one (s, t)-path in G. Suppose, for the sake of contradiction, that s and t are not connected in $G[V']$. Therefore there is no (s, t)-path in $G[V']$. And thus for every (s, t)-path $p_{st} \in \mathcal{P}(G, s, t)$, there is a vertex $u \in p_{st}$ such that $u \notin V'$. Let $S^V_{st} = \bigcup_{p_{st} \in \mathcal{P}(G,s,t)} \{u \in p_{st} : u \notin V'\}$. Note that by definition $V' \cap S^V_{st} = \emptyset$. And note that by definition $p_{st} \cap S^V_{st} \neq \emptyset$ for every $p_{st} \in \mathcal{P}(G, s, t)$. Hence $p_{st} \not\subseteq V \setminus S^V_{st}$ for every $p_{st} \in \mathcal{P}(G, s, t)$. Therefore there is no (s, t)-path in $G[V \setminus S^V_{st}]$ and thus S^V_{st} is an (s, t)-separator. But then S^V_{st} is an (s, t)-separator such that $G[V'] \cap S^V_{st} = \emptyset$, a contradiction. □

Lemma 20. *Let $G = (V, E)$ be a connected simple graph, $V' \subseteq V$ and $s, t \in V'$. Vertices s and t are connected in $G[V']$ if and only if $V' \cap S^V_{st} \neq \emptyset$ for every $S^V_{st} \in \mathcal{S}^V(G, s, t)$.*

Proof. ($\rightarrow$): Follows from Lemma 18. ($\leftarrow$): Follows from Lemma 19. □

From Lemma 20 we conclude that V' is a connected subset if and only if V' has a non-empty intersection with every (s, t)-separator for $s, t \in V'$. In turn, this demonstrates that constraints (VSC) are necessary and sufficient for y to induce a connected subgraph.

Considering constraints (VSC), one may devise a linear integer formulation for MWCDSP and MCDSP. Also note that if $|V| \leq 1$, then V is the only solution to MWCDSP and MCDSP. Conversely, if $|V| \geq 2$, then any $\mathcal{D}$ is a total dominating set. In this way, given a vector of weights $w \in \mathbb{R}^{|V|}_{\geq 0}$ associated with the vertices of G, a linear integer programming formulation for MWCDSP based on constraints (VSC) is defined as:

$$\begin{array}{lll} \min & \displaystyle\sum_{v \in V} w_v y_v & \\ \text{s.t.} & \displaystyle\sum_{u \in N(v)} y_u \quad \geq 1 & v \in V \\ & \displaystyle\sum_{v \in S^V_{st}} y_v \quad \geq y_s + y_t - 1 & s,t \in V, S^V_{st} \in \mathcal{S}^V(G,s,t) \\ & y_v \in \{0,1\}^{|V|} & \end{array} \tag{3.1}$$

Note that constraints $\sum_{u \in N(v)} y_u \geq 1$ for every $v \in V$ ensure that $V[y]$ is a total dominating set. Aditionally, note that vertex-separator constraints ensure that $V[y]$ is connected. Hence $V[y]$ is a connected dominating set.

Once again considering the vertex-separator constraints, one may formulate the MWCVCP and MCVCP as the following integer program:

$$\begin{array}{lll} \min & \displaystyle\sum_{v \in V} w_v y_v & \\ \text{s.t.} & y_u + y_v \quad \geq 1 & \{u,v\} \in E \\ & \displaystyle\sum_{v \in S^V_{st}} y_v \quad \geq y_s + y_t - 1 & s,t \in V, S^V_{st} \in \mathcal{S}^V(G,s,t) \\ & y_v \in \{0,1\}^{|V|} & \end{array} \tag{3.2}$$

Note that constraints $y_u + y_v \geq 1$ for every $e \in E$ ensure that $V[y]$ is a vertex-cover. Note as well that the vertex-separator constraints ensure that $V[y]$ is connected. Hence $V[y]$ is a connected vertex-cover.

Now we consider edge-based formulations. For each edge $e \in E$, let $x_e \in \{0,1\}$ be a binary decision variable associated with edge e such that $x_e = 1$ if e is part of the solution and $x_e = 0$ otherwise. Before presenting the so-called *generalized subtour-elimination constraints*, we first present some important properties of graphs and trees on which this class of constraints are based.

Lemma 21. *Let $G = (V,E)$ be a simple graph. If $|N(v)| \geq 2$ for every $v \in V$, then G contains a cycle.*

Proof. Since $|N(v)| \geq 2$ for every $v \in V$, then every vertex has at least one incident edge. Furthermore, since each edge defines a path, then the graph has at least one path. Let $p = (v_0, \ldots, v_\ell)$ be a maximum length path in G. Since $|N(v_\ell)| \geq 2$, then there must exist $v \in N(v_\ell)$ such that $v \neq v_{\ell-1}$. Suppose, for the sake of

contradiction, that $v \notin p$. Then $p' = (v_0, \ldots, v_\ell, v)$ is a path with length $\ell + 1$. This is a contradiction, since p has length ℓ and p is a maximum length path. Hence $v \in p$ and thus $v = v_k$ for some $k \in \{1, \ldots, \ell - 1\}$. Therefore $(v_k, \ldots, v_\ell, v_k)$ is a cycle. □

Lemma 22. *Let $G = (V, E)$ be a simple graph such that $V \neq \emptyset$. If $|E| \geq |V|$, then G contains a cycle.*

Proof. We proceed by induction on $|V|$. Base: if $|V| = 3$, then $|E| \geq |V|$ if and only if $|E| = 3$, since G is a simple graph. In this case, G clearly contains a cycle. Induction hypothesis: suppose the claim is true for every graph with $|V| = n$ for some $n > 3$. Induction step: suppose that $|V| = n + 1$ for some $n > 3$. If $|N(v)| \geq 2$ for every $v \in V$, it immediately follows from Lemma 21 that G contains a cycle. Suppose there exists $v \in V$ such that $|N(v)| < 2$. First, further suppose that $|N(v)| = 1$ and consider the graph G' obtained by removing v and its single incident edge from G. Since $|V| = n + 1$ and $|E| \geq |V|$, then G' has $(n + 1) - 1 = n$ vertices and at least $(n + 1) - 1 = n$ edges. It follows from the induction hypothesis that G' contains a cycle. Moreover, since G' is a subgraph of G, then G contains a cycle. Now suppose that $|N(v)| = 0$. Consider the graph G' obtained by removing v from G. Since $|V| = n + 1$ and $|E| \geq |V|$, then G' has $(n + 1) - 1 = n$ vertices and at least $n + 1$ edges. It follows from the induction hypothesis that G' contains a cycle. And since G' is a subgraph of G, then G contains a cycle. □

Lemma 23. *Let $G = (V, E)$ be a simple graph. If G is connected, then $|E| \geq |V| - 1$.*

Proof. We proceed by induction on $|V|$. Base: if $|V| = 0$, then G is connected and $|E| = 0 = |V|$. Induction hypothesis: suppose the claim is true for every graph with $|V| \leq n$ for some $n > 0$. Induction step: suppose that $|V| = n + 1$. If $|V| = 1$, then the claim is clearly true. Assume $|V| \geq 2$. Consider any $v \in V$. Since G is connected, then $|N(v)| \geq 1$. Let G' be the graph obtained by removing v and all its incident edges from G. Note that G' has $(n + 1) - 1 = n$ vertices and let $C_1, \ldots, C_m$ be the corresponding connected components of G'. For each $k \in \{1, \ldots, m\}$, note that there must exist an edge $e_k \in \delta(v)$ such that $e_k \cap C_k \neq \emptyset$ and $e_k \cap C_l = \emptyset$ for every $l \in \{1, \ldots, m\}$ with $l \neq k$. This follows from the definition of connected components, which ensures that they are maximally connected subsets of vertices, and thus implies that two distinct connected components can not be connected by an edge. By the induction hypothesis, it follows that for each $k \in \{1, \ldots, m\}$, the induced subgraph $G[C_k]$ has at least $|C_k| - 1$ edges. Therefore, by considering the number of edges in each connected component of G', we conclude that G has at least $\sum_{k \in \{1, \ldots, m\}} |C_k| - 1 = |V| - 1 - m$ edges. Moreover, since for each $k \in \{1, \ldots, m\}$ there is a unique edge $e_k \in \delta(v)$ such that $e_k \notin E[C_k]$ and $e_k \in E$, then it follows that G has at least $(|V| - 1 - m) + m = |V| - 1$ edges. □

Lemma 24. *Let $G = (V, E)$ be a simple graph. If $|E[V']| \leq |V'| - 1$ for every $V' \subseteq V$ such that $V' \neq \emptyset$, then G contains no cycles.*

Proof. If $V = \emptyset$, then the claim is vacuously true. Moreover, since G is a simple graph, it contains no cycle if $|V| \in \{1, 2\}$. Assume $|V| \geq 3$. By definition, any cycle $c = (v_0, \ldots, v_\ell, v_0)$ induces $\ell + 1$ edges. Suppose, for the sake of contradiction, that there is a cycle $c = (v_0, \ldots, v_\ell, v_0)$ in G. Consider $C = \{v_0, \ldots, v_\ell\} \subseteq V$. Note that $|C| = \ell + 1$. Then $E[C] \geq \ell + 1 = |C| > |C| - 1$, a contradiction. □

Lemma 25. *Let $G = (V, E)$ be a simple graph. If G contains no cycles, then $|E[V']| \leq |V'| - 1$ for every $V' \subseteq V$ such that $V' \neq \emptyset$.*

Proof. Follows from the negation of Lemma 22. □

Lemma 26. *Let $G = (V, E)$ be a simple graph. G contains no cycles if and only if $|E[V']| \leq |V'| - 1$ for every $V' \subseteq V$ such that $V' \neq \emptyset$.*

Proof. ($\rightarrow$): Follows from Lemma 25. ($\leftarrow$): Follows from Lemma 24. □

Lemma 27. *Let $G = (V, E)$ be a simple graph. If $|E| \geq |V| - 1$ and G contains no cycle, then G is connected.*

Proof. If $|V| \leq 1$, then the claim is clearly true. Assume $|V| \geq 2$. Suppose, for the sake of contradiction, that G is not connected. Hence there are $s, t \in V$ such that $s \neq t$ and s and t are not connected. Thus $\{s, t\} \notin E$. Let G' be the graph obtained by inserting edge $\{s, t\}$ in G. Note that G' has at least $(|V| - 1) + 1 = |V|$ edges. It follows from Lemma 22 that G' contains a cycle $c = (v_0, \ldots, v_\ell, v_0)$. Let $C = \{v_0, \ldots, v_\ell\}$. Suppose $\{s, t\}$ is an edge induced by c. Assume, without loss of generality, that $c = (s, \ldots, v_{\ell-1}, t, s)$. This implies that $(s, \ldots, v_{\ell-1}, t)$ is an (s, t)-path in G, a contradiction. Now suppose $\{s, t\}$ is not an edge induced by c. This implies that G contains a cycle, a contradiction. □

Lemma 28. *Let $G = (V, E)$ be a simple graph. The following statements are equivalent:*

1. *G is a tree.*
2. *$|E| = |V| - 1$ and G contains no cycle.*
3. *$|E| = |V| - 1$ and $|E[V']| \leq |V'| - 1$ for every $V' \subseteq V$ such that $V' \neq \emptyset$.*

Proof. The equivalence of 2 and 3 follows from Lemma 26. ($1 \rightarrow 2$): Since G is a tree, by definition, G contains no cycles and is connected. Since G is connected, it follows from Lemma 23 that $|E| \geq |V| - 1$. Since G contains no cycles, it follows from Lemma 22 that $|E| \leq |V| - 1$. So we conclude that $|E| = |V| - 1$. ($2 \rightarrow 1$): Since $|E| = |V| - 1$ and G contains no cycle, it follows from Lemma 27 that G is connected. And since it has no cycles, then G is a tree. □

The so called *generalized subtour elimination constraints* date back to as early as 1986 in the work of LAPORTE [93] and were later considered by GOEMANS [94] in 1994. They are defined in the following way:

$$\sum_{e \in E[V']} x_e \;\leq\; \sum_{v \in V' \setminus \{v'\}} y_v \quad \text{for every } V' \subseteq V, v' \in V' \tag{GSEC}$$

Note that constraints (GSEC) ensure that for each subset of vertices $V' \subseteq V$, solution y is such that $|E[V' \cap V[y]]| \leq |V' \cap V[y]| - 1$. It then follows from Lemma 24 that $G[V' \cap V[y]]$ contains no cycles.

By further imposing the following constraint

$$\sum_{e \in E} x_e \;=\; \sum_{v \in V} y_v - 1 \tag{3.3}$$

one ensures that $G[y]$ has exactly $|V[y]| - 1$ edges. It then follows from Lemma 28 that $G[y]$ is a tree and thus $G[y]$ is connected. Therefore, constraints (GSEC) together with (3.3) ensure that $G[y]$ is connected.

In fact, the following integer program for MCDSP was originally proposed by SIMONETTI *et al.* [2] and later considered by GENDRON *et al.* [95]:

$$\begin{aligned}
\min \quad & \sum_{v \in V} y_v & & \\
\text{s.t.} \quad & \sum_{u \in N(v)} y_u & \geq 1 \qquad & v \in V \\
& \sum_{e \in E} x_e & = \sum_{v \in V} y_v - 1 & \\
& \sum_{e \in E[V']} x_e & \leq \sum_{v \in V' \setminus \{v'\}} y_v \qquad & V' \subseteq V, v' \in V' \\
& y_v \in \{0,1\}^{|V|} & & \\
& x_e \in \{0,1\}^{|E|} & &
\end{aligned} \tag{3.4}$$

Note that constraints $\sum_{u \in N(v)} y_u \geq 1$ for every $v \in V$ ensure that $V[y]$ is a total dominating set. In turn, the generalized subtour elimination constraints together with constraint $\sum_{e \in E} x_e = \sum_{v \in V} y_v - 1$ ensure that $G[y]$ is a tree. Hence $V[y]$ is a connected dominating set.

Considering the generalized subtour elimination constraints, we propose the following integer programming formulation for MWTCP:

$$
\begin{aligned}
\min \quad & \sum_{e\in E} w_e x_e & & \\
\text{s.t.} \quad & \sum_{e\in N[e]} x_e & \geq 1 \qquad & e \in E \\
& \sum_{e\in E} x_e & = \sum_{v\in V} y_v - 1 & \\
& \sum_{e\in E[V']} x_e & \leq \sum_{v\in V'\setminus\{v'\}} y_v \quad & V' \subseteq V, v' \in V' \\
& y_v \in \{0,1\}^{|V|} & & \\
& x_e \in \{0,1\}^{|E|} & &
\end{aligned}
\tag{3.5}
$$

Note that constraints $\sum_{e'\in N[e]} x_{e'} \geq 1$ for every $e \in E$ ensure that $G[y]$ is an edge-dominating set. And the generalized generalized subtour elimination constraints together with constraint $\sum_{e\in E} x_e = \sum_{v\in V} y_v - 1$ ensure that $G[y]$ is a tree. Hence $V[y]$ is a tree cover.

3.2 Characterizations

Note that the integer programs presented here for MWCDSP, MWCVCP and MTCP include: set cover constraints to ensure domination, in the case of MWCDSP, and coverage, in the case of MWCVCP and MTCP; and connectivity constraints, either in the form of vertex-separator constraints or generalized subtour elimination constraints. Either way, note that both these classes of connectivity constraints are not in the form of set cover constraints as presented in 2.1. We now present a series of results which prove that in the case of connected dominating sets, connected vertex-covers and tree-covers, one can impose connectivity by employing constraints which are in the form of set cover constraints.

Lemma 29. *Let $G = (V, E)$ be a connected simple graph. If $\mathcal{D} \subseteq V$ is a connected dominating set, then $\mathcal{D} \cap S^V \neq \emptyset$ for every vertex-cut $S^V \in \mathcal{S}^V(G)$.*

Proof. If $\mathcal{S}^V(G) = \emptyset$, then the claim is vacuously true. Assume $\mathcal{S}^V(G) \neq \emptyset$. Consider any $S^V \in \mathcal{S}^V(G)$. Since S^V is a vertex-cut, it follows from Lemma 11 that there are $s_0, t_0 \in V$ such that $s_0 \neq t_0$ and S^V is an (s_0, t_0)-separator. And since $\mathcal{D}$ is a dominating set, then there is $s \in N[s_0]$ such that $s \in \mathcal{D}$ and there is $t \in N[t_0]$ such that $t \in \mathcal{D}$. If $s \in S^V$ or $t \in S^V$, then clearly $\mathcal{D} \cap S^V \neq \emptyset$. Assume $s, t \notin S^V$. And since $G[\mathcal{D}]$ is connected, then there is an (s, t)-path $p_{st} \subseteq \mathcal{D}$. If $s = s_0$ and $t \neq t_0$, note that $p_{s_0,t_0} = (s_0, \ldots, t, t_0)$ is an (s_0, t_0)-path. If $s \neq s_0$ and $t = t_0$, note that $p_{s_0,t_0} = (s_0, s, \ldots, t)$ is an (s_0, t_0)-path. If $s \neq s_0$ and $t \neq t_0$, note that $p_{s_0,t_0} = (s_0, s, \ldots, t, t_0)$ is an (s_0, t_0)-path. In all three cases, since p_{s_0,t_0} is an (s_0, t_0)-path, then it follows from Lemma 17 that $p_{s_0,t_0} \cap S^V \neq \emptyset$. Since $s_0, s, t_0, t \notin S^V$, then $p_{st} \cap S^V \neq \emptyset$. And since $p_{st} \subseteq \mathcal{D}$, then $\mathcal{D} \cap S^V \neq \emptyset$. □

Lemma 30. *Let $G = (V, E)$ be a connected simple graph and $V' \subseteq V$. If $V' \cap S^V \neq \emptyset$ for every vertex-cut $S^V \in \mathcal{S}^V(G)$, then V' is connected.*

Proof. Suppose, for the sake of contradiction, that V' is not connected. Then there are $s, t \in V'$ such that $s \neq t$ and there is no (s,t)-path in V'. Since G is connected, then there is at least one (s,t)-path p_{st} in G and thus $\mathcal{P}(G,s,t) \neq \emptyset$. Let $S^V = \bigcup_{p_{st} \in \mathcal{P}(G,s,t)} \{u \in p_{st} : u \notin V'\}$. Note that, by definition, $V' \cap S^V = \emptyset$. We argue that S^V is a vertex-cut. Note that, by definition, $p_{st} \cap S^V \neq \emptyset$ for every $p_{st} \in \mathcal{P}(G,s,t)$. Hence $p_{st} \not\subseteq V \setminus S^V$ for every $p_{st} \in \mathcal{P}(G,s,t)$. Thus there is no (s,t)-path in $G[V \setminus S^V]$. Hence S^V is an (s,t)-separator and thus a vertex-cut. So S^V is a vertex-cut and $V' \cap S^V = \emptyset$, a contradiction. □

Theorem 1. *Let $G = (V, E)$ be a connected simple graph and $\mathcal{D} \subseteq V$ a dominating set. $\mathcal{D}$ is connected if and only if $\mathcal{D} \cap S^V \neq \emptyset$ for every vertex-cut $S^V \in \mathcal{S}^V(G)$.*

Proof. ($\rightarrow$): Follows from Lemma 29. ($\leftarrow$): Follows from Lemma 30. □

Lemma 31. *Let $G = (V, E)$ be a simple graph. If $\mathcal{C} \subseteq V$ is a vertex-cover, then $\mathcal{C}$ is a dominating set.*

Proof. Consider any $v \in V$. Since G is connected, then there is some $u \in N(v)$ and thus $\{v, u\} \in E$. And since $\mathcal{C}$ is a vertex-cover, then $v \in \mathcal{C}$ or $u \in \mathcal{C}$. And thus $\mathcal{C} \cap N[v] \neq \emptyset$. Hence $\mathcal{C}$ is a dominating set. □

Theorem 2. *Let $G = (V, E)$ be a connected simple graph and $\mathcal{C} \subseteq V$ a vertex-cover. $\mathcal{C}$ is connected if and only if $\mathcal{C} \cap S^V \neq \emptyset$ for every vertex-cut $S^V \in \mathcal{S}^V(G)$.*

Proof. ($\rightarrow$): Since $\mathcal{C}$ is a vertex-cover, it follows from Lemma 31 that $\mathcal{C}$ is a dominating set. The result follows from Theorem 1. □

Lemma 32. *Let $G = (V, E)$ be a simple graph. If $\mathcal{D}$ is an edge-dominating set, then $V[\mathcal{D}]$ is a vertex-cover.*

Proof. Consider any edge $e = \{u, v\} \in E$. Since $\mathcal{D}$ is edge-dominating, then there is $e' \in N[e]$ such that $e' \in \mathcal{D}$. Since $e' \in N[e]$, by definition, $u \in e'$ or $v \in e'$. Then, by the definition of $V[\mathcal{D}]$, $u \in V[\mathcal{D}]$ or $v \in V[\mathcal{D}]$. □

Lemma 33. *Let $G = (V, E)$ be a connected simple graph. If $p_{st} \in \mathcal{P}(G,s,t)$, then $E[p_{st}] \cap S^E_{st} \neq \emptyset$ for every (s,t)-cut $S^E_{st} \in \mathcal{S}^E(G,s,t)$.*

Proof. Consider any (s,t)-path $p_{st} \in \mathcal{P}(G,s,t)$. Consider any (s,t)-cut $S^E_{st} \in \mathcal{S}^E(G,s,t)$. Since S^E_{st} is an (s,t)-cut, then s and t are disconnected in $G[E \setminus S^E_{st}]$. And since p_{st} is an (s,t)-path in G, then $E[p_{st}] \not\subseteq E \setminus S^E_{st}$. Hence $E[p_{st}] \cap S^E_{st} \neq \emptyset$. □

Lemma 34. *Let $G = (V, E)$ be a connected simple graph and $\mathcal{D} \subseteq E$ an edge-dominating set. If $\mathcal{D}$ is connected, then $\mathcal{D} \cap S^E \neq \emptyset$ for every edge-cut $S^E \in \mathcal{S}^E(G)$.*

Proof. If $\mathcal{S}^E(G) = \emptyset$, then the claim is vacuously true. Assume $\mathcal{S}^E(G) \neq \emptyset$. Consider any $S^E \in \mathcal{S}^E(G)$. Since S^E is an edge-cut, it then follows from Lemma 15 that there are $s_0, t_0 \in V[E \setminus S^E]$ such that $s_0 \neq t_0$ and S^E is an (s_0, t_0)-cut. First, assume $s_0, t_0 \in V[\mathcal{D}]$. Since $G[\mathcal{D}]$ is connected, then there is an (s_0, t_0)-path p_{s_0,t_0} in $G[\mathcal{D}]$. It then follows from Lemma 33 that $E[p_{s_0,t_0}] \cap S^E \neq \emptyset$. And since $E[p_{s_0,t_0}] \subseteq \mathcal{D}$, then $\mathcal{D} \cap S^E \neq \emptyset$. Now assume that $s_0 \notin V[\mathcal{D}]$ or $t_0 \notin V[\mathcal{D}]$. Since $s_0, t_0 \in V[E \setminus S^E]$, then there is $s \in N(s_0)$ such that $\{s_0, s\} \notin S^E$ and there is $t \in N(t_0)$ such that $\{t_0, t\} \notin S^E$. And since $\mathcal{D}$ is edge-dominating, then there is $e_s \in N[\{s_0, s\}]$ such that $e_s \in \mathcal{D}$ and there is $e_t \in N[\{t_0, t\}]$ such that $e_t \in \mathcal{D}$. And it follows from Lemma 32 that $\{s_0, s\} \cap V[\mathcal{D}] \neq \emptyset$ and $\{t_0, t\} \cap V[\mathcal{D}] \neq \emptyset$. If $s_0 \in V[\mathcal{D}]$, let $s = s_0$. Otherwise, let $s = s$. Similarly, if $t_0 \in V[\mathcal{D}]$, let $t = t_0$. Otherwise, let $t = t$. Since $s, t \in V[\mathcal{D}]$ and since $G[\mathcal{D}]$ is connected, then there is an (s, t)-path $p_{st} = (s, \ldots, v_\ell, t)$ in $G[\mathcal{D}]$. And thus $E[p_{st}] \subseteq \mathcal{D}$. If $s = s_0$ and $t \neq t_0$, note that $p_{s_0,t_0} = (s_0, \ldots, t, t_0)$ is an (s_0, t_0)-path. If $s \neq s_0$ and $t = t_0$, note that $p_{s_0,t_0} = (s_0, s, \ldots, t)$ is an (s_0, t_0)-path. If $s \neq s_0$ and $t \neq t_0$, note that $p_{s_0,t_0} = (s_0, s, \ldots, t, t_0)$ is an (s_0, t_0)-path. In all three cases, since p_{s_0,t_0} is an (s_0, t_0)-path, then it follows from Lemma 33 that $E[p_{s_0,t_0}] \cap S^E \neq \emptyset$. And since $\{s_0, s\}, \{t_0, t\} \notin S^E$, then $E[p_{s_0,t_0}] \cap S^E \neq \emptyset$. And since $E[p_{s_0,t_0}] \subseteq \mathcal{D}$, then $\mathcal{D} \cap S^E \neq \emptyset$. □

Lemma 35. *Let $G = (V, E)$ be a connected simple graph and $E' \subseteq E$. If $E' \cap S^E \neq \emptyset$ for every edge-cut $S^E \in \mathcal{S}^E(G)$, then E' is connected.*

Proof. If $\mathcal{S}^E(G) = \emptyset$, then the claim is vacuously true. Suppose, for the sake of contradiction, that E' is not connected. Then there are $s, t \in V[E']$ such that $s \neq t$ and there is no (s, t)-path in $G[E']$. Since G is connected, then there is at least one (s, t)-path p_{st} in G and thus $\mathcal{P}(G, s, t) \neq \emptyset$. Let $S^E = \bigcup_{p_{st} \in \mathcal{P}(G,s,t)} \{e \in E[p_{st}] : e \notin E'\}$. Note that by definition $E' \cap S^E = \emptyset$. And therefore $s, t \in V[E \setminus S^E]$. We argue that S^E is an edge-cut. Note that by definition $p_{st} \cap S^E \neq \emptyset$ for every $p_{st} \in \mathcal{P}(G, s, t)$. Hence $E[p_{st}] \not\subseteq E \setminus S^E$ for every $p_{st} \in \mathcal{P}(G, s, t)$. Thus there is no (s, t)-path in $G[E \setminus S^E]$. Hence S^E is an (s, t)-cut. So S^E is a edge-cut and $E' \cap S^E \neq \emptyset$, a contradiction. □

Theorem 3. *Let $G = (V, E)$ be a connected simple graph and $\mathcal{D} \subseteq E$ an edge-dominating set. $\mathcal{D}$ is connected if and only if $\mathcal{D} \cap S^E \neq \emptyset$ for every edge-cut $S^E \in \mathcal{S}^E(G)$.*

Proof. ($\rightarrow$): Follows from Lemma 34. ($\leftarrow$): Follows from Lemma 35. □

3.3 Formulations

The lemmas and theorems presented in Section 3.2 ensure that MWCDSP, MWCVCP and MTCP can be formulated as set cover problems, in alternative to

other formulations which consider constraints other than set cover constraints. As mentioned in Section 2.1, the advantage of this alternative is that SCP has been widely investigated and many effective algorithms for SCP have been developed. For this reason, employing SCP algorithms in the solution of problems such as MWCDSP, MWCVCP and MTCP seems to be a promising alternative. We now present the SCP formulations of MWCDSP, MWCVCP and MTCP.

We consider a connected simple graph $G = (V, E)$, and for each vertex $v \in V$, we let $y_v \in \{0, 1\}$ be a binary decision variable such that $y_v = 1$ denotes that v is part of the solution and $y_v = 0$ otherwise. Moreover, given $y \in \{0, 1\}^{|V|}$, we let $V[y] = \{v \in V : y_v = 1\}$ denote the subset of vertices induced by y and we let $G[y] = G[V[y]]$ denote the subgraph induced by y.

We begin by considering the min weight connected dominating set problem. To ensure that y induces a dominating set, we impose *domination constraints* $\sum_{u \in N[v]} y_u \geq 1$ for every vertex $v \in V$. Additionally, it follows from Theorem 1 that connectivity of a dominating set is guaranteed by imposing a *vertex-cut constraint* $\sum_{v \in S^V} y_v \geq 1$ for every vertex-cut $S^V \in \mathcal{S}^V(G)$. Accordingly, given a positive weight w_v for each vertex $v \in V$, the SCP formulation of MWCDSP is as follows:

$$\begin{aligned} \min \quad & \sum_{v \in V} w_v y_v & & \\ \text{s.t.} \quad & \sum_{u \in N[v]} y_u & \geq 1 \quad & v \in V \\ & \sum_{v \in S^V} y_v & \geq 1 \quad & S^V \in \mathcal{S}^V(G) \\ & y_v \in \{0, 1\} & & v \in V \end{aligned} \tag{SC-WCDS}$$

Suppose $y \in \{0, 1\}^{|V|}$ is a feasible solution to (SC-WCDS). Then for each $v \in V$, there is $u \in N[v]$ such that $y_u = 1$. Therefore $V[y]$ is a dominating set. Moreover, for each vertex-cut $S^V \in \mathcal{S}^V(G)$, there is $v \in S^V$ such that $y_v = 1$. Hence $S^V \cap V[y] \neq \emptyset$ for every vertex-cut $S^V \in \mathcal{S}^V(G)$. It follows from Theorem 1 that $V[y]$ is a connected dominating set.

MCDSP is a particular case of MCWDSP when all weights are equal and without loss of generality assumed to be equal to 1. Even though SC-WCDS is a valid formulation for MCDSP, we now argue that for MCDSP in non-trivial cases, the domination constraints can be lifted to a stronger class of constraints. Suppose there is a vertex $v_0 \in V$ such that $N[v_0] = V$. In this case, solving MCDSP is trivial and its optimal solution is the one-element set $\{v_0\}$. On the other hand, let us assume that $N[v] \neq V$ for every $v \in V$. Let $\mathcal{D} \subseteq V$ be a connected dominating set in G. Consider any $v \in \mathcal{D}$. Since $N[v] \neq V$, then there must exist $u \in \mathcal{D}$ such that $v \neq u$, otherwise $\mathcal{D}$ would not be a dominating set. Since $\mathcal{D}$ is a CDS, then there is a (v, u)-path $p_{v,u}$ in $\mathcal{D}$. And since $p_{v,u}$ is a (v, u)-path, then there is $w_0 \in p_{v,u}$ such

that $w_0 \in N(v)$. Hence there is $w_0 \in \mathcal{D}$ such that $w_0 \in N(v)$. We conclude that for every $v \in \mathcal{D}$ there is $w_0 \in N(v) \cap \mathcal{D}$. Hence $\mathcal{D}$ is a (connected) total dominating set. Finally, we conclude that when there is $v_0 \in V$ such that $N[v_0] = V$, then MCDSP is trivial. On the other hand, by assuming that $N[v] \neq V$ for every $v \in V$, then for each vertex $v \in V$, the *domination* constraint $\sum_{u \in N[v]} y_u \geq 1$ can be lifted to the *total-domination* constraint $\sum_{u \in N(v)} y_u \geq 1$. Accordingly, an SCP formulation for MCDSP in the non-trivial case is as follows:

$$\begin{aligned} \min \quad & \sum_{v \in V} y_v & & \\ \text{s.t.} \quad & \sum_{u \in N(v)} y_u & \geq 1 \quad & v \in V \\ & \sum_{v \in S^V} y_v & \geq 1 \quad & S^V \in \mathcal{S}^V(G) \\ & y_v \in \{0,1\} & & v \in V \end{aligned} \tag{SC-CDS}$$

Suppose $y \in \{0,1\}^{|V|}$ is a feasible solution to (SC-CDS). Then for each $v \in V$, there is $u \in N(v)$ such that $y_u = 1$. Therefore $V[y]$ is a total-dominating set, which implies it is also dominating. Moreover, for each vertex-cut $S^V \in \mathcal{S}^V(G)$, there is $v \in S^V$ such that $y_v = 1$. Hence $S^V \cap V[y] \neq \emptyset$ for every vertex-cut $S^V \in \mathcal{S}^V(G)$. It follows from Theorem 1 that $V[y]$ is a connected dominating set.

We note that formulation SC-CDS is the same as proposed by BUCHANAN *et al.* [25]. Other integer programming formulations for MCDSP which are not in the form of SCP were considered by FUJIE [22] and by SIMONETTI *et al.* [2].

We now formulate the connected vertex-cover problem. To ensure that y induces a vertex-cover, we impose *vertex-cover* constraints $y_u + y_v \geq 1$ for every edge $\{u, v\} \in E$. Additionally, it follows from Theorem 2 that connectivity of a vertex-cover is guaranteed by imposing a *vertex-cut constraint* $\sum_{v \in S^V} y_v \geq 1$ for every vertex-cut $S^V \in \mathcal{S}^V(G)$. Accordingly, given a positive weight w_v for each vertex $v \in V$, the SCP formulation of MWCVCP is as follows:

$$\begin{aligned} \min \quad & \sum_{v \in V} w_v y_v & & \\ \text{s.t.} \quad & y_u + y_v & \geq 1 \quad & \{u, v\} \in E \\ & \sum_{v \in S^V} y_v & \geq 1 \quad & S^V \in \mathcal{S}^V(G) \\ & y_v \in \{0,1\}^{|V|} & & \end{aligned} \tag{SC-CVCP}$$

Suppose $y \in \{0,1\}^{|V|}$ is a feasible solution to (SC-CDS). Then for every edge $\{u, v\} \in E$, $y_v = 1$ or $y_u = 1$, or both. Therefore $V[y]$ is a vertex-cover. Moreover, for every vertex-cut $S^V \in \mathcal{S}^V(G)$, there is $v \in S^V$ such that $y_v = 1$. Hence $S^V \cap V[y] \neq \emptyset$ for every vertex-cut $S^V \in \mathcal{S}^V(G)$. It follows from Theorem 2 that $V[y]$ is a connected vertex-cover.

Finally, we formulate the min connected edge-dominating set problem. We consider a connected simple graph $G = (V, E)$, and for each edge $e \in E$, we let $x_e \in \{0, 1\}$ be a binary decision variable such that $x_e = 1$ denotes that e is part of the solution and $x_e = 0$ otherwise. Moreover, given $x \in \{0, 1\}^{|E|}$, we let $E[x] = \{e \in E : x_e = 1\}$ denote the subset of edges induced by x, $V[x] = V[E[x]]$ denote the subset of vertices induced by x and $G[x] = (V[x], E[x])$ denote the subgraph induced by x. To ensure that x induces an edge-dominating set, we impose *edge-domination* constraints $\sum_{e \in N[e]} x_e \geq 1$ for every edge $e \in E$. Additionally, it follows from Theorem 3 that connectivity of an edge-dominating set can be guaranteed by imposing an *edge-cut constraint* $\sum_{e \in S^E} x_e \geq 1$ for every edge-cut $S^E \in \mathcal{S}^E(G)$. Accordingly, given a positive weight w_e for each edge $e \in E$, the SCP formulation of MWCEDSP or MWTCP is as follows:

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e x_e & & \\ \text{s.t.} \quad & \sum_{e \in N[e]} x_e & \geq 1 \quad & e \in E \\ & \sum_{e \in S^E} x_e & \geq 1 \quad & S^E \in \mathcal{S}^E(G) \\ & x_e \in \{0, 1\}^{|E|} & & \end{aligned} \tag{SC-TCP}$$

Suppose $x \in \{0, 1\}^{|E|}$ is a feasible solution to (SC-TCP). Then for every edge $e \in E$, there is $e' \in N[e]$ such that $x_{e'} = 1$. Hence $E[x]$ is an edge-dominating set. Moreover, for every edge-cut $S^E \in \mathcal{S}^E(G)$, there is $e \in S^E$ such that $x_e = 1$. Hence $S^E \cap E[x] \neq \emptyset$ for every edge-cut $S^E \in \mathcal{S}^E(G)$. It follows from Theorem 3 that $E[x]$ is a connected edge-dominating set.

To the best of the author's knowledge, both formulations (SC-CVCP) as well as (SC-TCP) are novel. MÖLLE *et al.* [90] propose an enumeration procedure for connected vertex-covers and tree covers, but is it not based on integer programming. Both FUJITO [23] and NGUYEN [96] consider an integer programming formulation for tree covers by mapping the original instance defined on an undirected graph to a corresponding rooted directed graph. Connectivity is then imposed by *rooted edge-cut* constraints. This formulation is also in the form of a set cover problem, but has twice as many decision variables as (SC-TCP), since for each edge $e \in E$ there are two corresponding arcs in the directed graph.

It is important to remark that the number of elements in the sets $\mathcal{S}^V(G)$ and $\mathcal{S}^E(G)$ are an exponential function of the number of vertices $|V|$. Consequently, formulations (SC-CDS), (SC-CVCP) and (SC-TCP) have exponentially many set cover constraints. A *separation procedure* is a polynomial-time algorithm that identifies among the exponentially many constraints the ones that are violated (not satisfied) by a given tentative solution. By employing polynomial-time separation procedures,

one can efficiently solve integer programs despite the exponential amount of constraints. The general idea is to begin with a subproblem obtained from the integer program by considering only a *small* subset of constraints, called the *active* constraints, and while the algorithm is being executed, more constraints are activated *on the fly*. More specifically, every time that a solution to the current subproblem is found, the separation procedure is applied to identify the constraints violated by it. If there are violated constraints, they are activated and from that point on effectively take part on the current subproblem. Otherwise, if no constraints are violated, then a feasible solution to the original integer program was found. Since the amount of constraints is finite, at some point no more constraints are violated. The advantage of this approach is that usually the majority of the exponentially many constraints are naturally satisfied by good or optimal solutions, and for this reasons most of the constraints end up not being violated during the whole procedure. As a result, when the algorithm terminates, only a small portion of the exponentially many constraints are active and actually took part in the problem solution. Some examples of such methods include: for linear-programming branch-and-bound procedures, the standard approach is to include constraints during the enumeration procedure, leading to the so-called *branch-and-cut* algorithm; for lagrangian relaxation methods, the so-called relax-and-cut methods are a common approach [97], which incorporate the dynamic dualization of constraints into a lagrangian relaxation procedure.

3.4 Separation Procedures

Note that the integer programs SC-CDS and SC-CVCP have as many as $|\mathcal{S}^V(G)|$ constraints, whereas SC-TCP has as many as $|\mathcal{S}^E(G)|$. For a graph with vertex set V, both $|\mathcal{S}^V(G)|$ and $|\mathcal{S}^E(G)|$ are in $\mathcal{O}(2^{|V|-1})$. Consequently, even for small values of $|V|$, the number of elements in $\mathcal{S}^V(G)$ or $\mathcal{S}^E(G)$ is exceedingly large. As a result, even enumerating all of the cuts is impractical, not to mention that solving a linear program with this many constraints is intractable. For this reason, in the practice of OR, methods have been developed in order to deal with integer programs with an exceptionally large amount of constraints. When IP is solved by LP-based branch-and-bound, the standard approach is to consider the so-called *lazy constraints*. The guiding principle is to start the BB with an IP containing only a subset $\mathcal{A} \subset \mathbf{M}$ of all the possible constraints. $\mathcal{A}$ is called the set of *active* constraints, whereas $\mathcal{I} = \mathbf{M} \setminus \mathcal{A}$ is the set of *inactive* constraints. To ensure the solution given by BB is feasible, meaning that it satisfies all the constraints in $\mathbf{M}$, inactive constraints are incorporated into the IP when they are violated by an integer solution found during the procedure. More specifically, during BB, whenever an integer solution y satisfying all of the currently active constraints is found, a procedure is applied to

y to identify the set $\mathcal{V}$ of inactive constraints which are violated by y. This is called the *separation procedure*. If $\mathcal{V} \neq \emptyset$, then y is deemed infeasible, and these violated constraints are activated. On the other hand, if $\mathcal{V} = \emptyset$ holds, then y is feasible and the best solution is updated. The constraints which are initially inactive are called *lazy* since they only take effective part in the problem if they are violated at some point. Note that a lazy constraint might end up not being activated at all. In fact, in many cases, most of the lazy constraints are not violated along the BB algorithm, and in the end only a small portion of the exceedingly many constraints are in fact activated. This ensures the computational viability of the BB algorithm with the lazy constraint approach, resulting in linear programs with a much smaller number of constraints. Other kinds of methods also rely on a separation procedure, such as the relax-and-cut algorithm described in Chapter 5 and the local-search algorithm described in Chapter 6.

Considering the formulations presented in Section 3.3 for MCDSP, MCVCP and MTCP, we now describe separation procedures for identifying violated vertex-cuts as well as edge-cuts.

We begin by describing the separation procedure for vertex cuts. It takes a subset of vertices $V' \subseteq V$ and returns a vertex-cut $S \in \mathcal{S}^V(G)$ such that $S \cap V' = \emptyset$ if one exists. When solving an IP, if $y \in \{0,1\}^{|V|}$ is a binary vector where $y_v = 1$ denotes that vertex v is part of the solution and $y_v = 0$ otherwise, then a vertex cut can be separated by letting $V' = \{v \in V : y_v = 1\}$. Consider any minimal vertex cut $S^V \in \mathcal{S}^V(G)$. Since S^V is a minimal vertex-cut, it follows that $G[V \setminus S^V]$ has exactly two connected components. Hence S^V defines two disconnected sets $C_1, C_2 \subset V$ such that every vertex $v \in S^V$ has at least one neighbor in C_1 and at least one neighbor in C_2. Conversely, if two disjoint sets $C_1, C_2 \subset V$ satisfy that every vertex $v \in V \setminus (C_1 \cup C_2)$ has at least one neighbor in C_1 and at least one neighbor in C_2, then C_1 and C_2 define a unique minimal vertex cut $S^V = V \setminus (C_1 \cup C_2)$. Therefore, given a subset of vertices $V' \subset V$, in order to find a vertex-cut violated by V', it suffices to find a partition S, C_1, C_2 of V such that $V' \cap S = \emptyset$ and that every vertex $v \in S$ has at least one neighbor in C_1 and at least one neighbor in C_2. If no such partition exists, then V' is connected. The separation procedure works by maintaining a set S containing the vertices which are part of the vertex cut, and another set $C = V \setminus S$ containing vertices which are not in the cut. Initially $S = V \setminus V'$. Note that at any iteration, $G[C]$ has one or more connected components. If V' is a connected subset, then $G[C]$ has exactly one component and the procedure can stop. Otherwise, $G[C]$ has at least two components. For each vertex $v \in V$, an integer value is stored and denotes the label of the connected component of v. Moreover, a *disjoint-set* (DS) data structure is used to keep track of the connected components. In accordance with the DS implementation, each connected component is represented

by its *root*. Hence for each $v \in V$, the connected component of v is given by the root of the set containing v. The separation procedure begins by labeling the connected components of $G[C]$ by breadth-first search. To this end, a queue Q_{label} stores the vertices to be labeled and is initially empty. During the labeling phase, vertices are popped from Q_{label} and visited. More specifically, whenever v is visited, first it is assigned its component label and then its neighbors are traversed. Each $u \in N(v) \cap V'$ is pushed to Q_{label}. On the other hand, each $u \in N(v) \setminus V'$ is pushed to a queue Q_{merge} of merging candidates. To produce a minimal vertex cut, vertices are removed from S in what is known as *merging*. Merging a vertex v means removing v from S and inserting it to C. Note that after v is merged, all of the connected component of $G[C]$ containing at least one neighbor of v become connected and thus belong to the same component. Accordingly, the roots of these connected component must be merged. This means that for every $u \in N(v) \cap C$, we merge the roots of the two components containing v and u, respectively. Meanwhile, each $u \in N(v) \cap S$ is pushed to Q_{merge} if it was not already. A vertex $v \in S$ is merged only if it does not have a neighbor in every connected component of $G[C]$. Otherwise merging v would make S no longer a vertex cut. During the *merging phase*, at each iteration a vertex is popped from the queue and merged if possible. Each vertex in $V \setminus V'$ is considered for merging once, and the procedure terminates as soon as Q_{merge} is empty. If V' is not connected, then at termination $G[C]$ has exactly two connected components and S is a minimal vertex cut. The labeling phase and the merging phase are described in Algorithms 1 and 2, respectively. The separation of vertex cuts is described in Algorithm 3.

Algorithm 1 Labeling the connected components

```
function LABEL COMPONENTS(V')
    ℓ ← 0
    Q_label ← ∅
    Q_merge ← ∅
    for r ∈ V' do
        if label[r] = 0 then
            ℓ ← ℓ + 1
            push queue(Q_label, r)
            visited[r] ← true
            while Q_label ≠ ∅ do
                v ← pop queue(Q_label)
                label[v] ← ℓ
                for u ∈ N(v) do
                    if u ∈ V' and visited[u] = false then
                        push queue(Q_label, u)
                    else if u ∉ V' and visited[u] = false then
                        push queue(Q_merge, u)
                    visited[u] ← true
    components ← disjoint set({{1}, ..., {ℓ}})
```

Algorithm 2 Merging vertices to obtain minimal vertex cut

```
function MERGE VERTICES(V')
    while Q_merge ≠ ∅ do
        v ← pop queue(Q_merge)
        k ← |{component[u] : u ∈ N(v)}|
        if k < |{component[u] : u ∈ V}| then
            S ← S \ {v}
            C ← C ∪ {v}
            for u ∈ N(v) do
                if u ∈ C then
                    merge roots(components, component[v], component[u])
                else if u ∈ S and visited[u] = false then
                    push queue(Q_merge, u)
                    visited[u] ← true
```

Algorithm 3 Separation of vertex cuts

function SEPARATE VERTEX CUT(V', G)
    LABEL COMPONENTS(V')
    MERGE VERTICES(V')
    return S

Chapter 4

Lagrangian Relaxation

In Section 4.1 we present the basic theory of lagrangian relaxation, a general approach for obtaining dual bounds for an integer program. In Section 4.2 we describe the standard way of applying lagrangian relaxation to SCP. In Section 4.3 we propose an alternative lagrangian relaxation formulation for SCP that leads to a stronger dual problem.

4.1 Theory

A *minimization problem* (MinP) $Z = \min\{C(y) : y \in \mathcal{P}\}$ is defined by its *feasible region* $\mathcal{P}$ and by its *objective value function* $C : \mathcal{P}' \to \mathbb{R}$ where $\mathcal{P} \subseteq \mathcal{P}'$. Problem (MinP) is *feasible* if $\mathcal{P} \neq \emptyset$. Additionally, any point $y \in \mathcal{P}$ is said to be a *feasible solution* of (MinP). An *optimal solution* is a feasible solution of minimum objective value, i.e, it is a feasible solution $y^* \in \mathcal{P}$ such that $C(y^*) \leq C(y)$ for every feasible solution $y \in \mathcal{P}$. Therefore, if the problem is feasible, then $Z = C(y^*)$ for every optimal solution y^*. A *primal bound* of Z is a value $\overline{Z} \in \mathbb{R}$ such that $Z \leq \overline{Z}$. Any feasible solution $y \in \mathcal{P}$ provides a primal bound $\overline{Z} = C(y)$ on the optimal objective value. A *dual bound* of Z is a value $\underline{Z} \in \mathbb{R}$ such that $\underline{Z} \leq Z$. Consider a *maximization problem* (MaxP) $W = \max\{C'(x) : x \in \mathcal{D}\}$. Problems (MinP) and (MaxP) form a *weak dual pair* if $C'(x) \leq C(y)$ for every $y \in \mathcal{P}$ and $x \in \mathcal{D}$. In this case, (MinP) is called the *primal problem* and (MaxP) is called the *dual problem*. If $W = Z$, then the problems form a *strong dual pair*. Since any point $x \in \mathcal{D}$ is said to be a feasible solution of the dual problem, then in order to make a clearer distinction between feasible solutions to the primal and to the dual problem, any point $y \in \mathcal{P}$ is said to be *primal feasible* and any point $x \in \mathcal{D}$ is said to be *dual feasible*. A primal feasible solution $y \in \mathcal{P}$ is proven to be optimal if there is a dual bound $\underline{Z}$ such that $\underline{Z} = C(y)$. Therefore, when trying to solve an optimization problem up to proven optimality, it is helpful to define a pair of primal and dual problems and iteratively search for improving solutions to both the primal and dual

problem. The primal solutions lead to a non-increasing sequence of primal bounds $\overline{Z}_1 \geq \overline{Z}_2 \geq \cdots \geq Z$, while the dual solutions lead to a non-decreasing sequence of dual bounds $\underline{Z}_1 \leq \underline{Z}_2 \leq \cdots \leq Z$. As better solutions are found for both the primal and dual problem, the *optimality gap* $\overline{Z}_k - \underline{Z}_k$ decreases and one gets closer to proving optimality.

Consider the following integer program:

$$\begin{array}{rlll} Z = & \min & C(y) & \\ & \text{s.t} & f_i(y) & \geq a_i \quad i \in \{1, \ldots, m\} \\ & & y \in \mathcal{P} & \end{array} \tag{IP}$$

We assume that constraints $f_i(y) \geq a_i$ are *complicating constraints*, meaning that the problem $\min\{C(y) : y \in \mathcal{P}\}$ obtained by removing these constraints from (IP) can be solved in polynomial time, while the original problem (IP) cannot and is possibly NP-hard. *Lagrangian relaxation* (LR) is a standard procedure for defining a weak dual pair for an integer program such as (IP). The guiding principle of lagrangian relaxation is to *dualize* the complicating constraints. In order to obtain the *lagrangian dual problem* (LDP) for (IP), let $\lambda_i \in \mathbb{R}_{\geq 0}$, $i \in \{1, \ldots, m\}$, be a non-negative value associated with constraint $f_i(y) \geq a_i$ called its *lagrangian multiplier*. Given $\lambda \in \mathbb{R}^m_{\geq 0}$, the *lagrangian subproblem* associated with λ is obtained from (IP) by dualizing the complicating constraints. Each constraint $f_i(y) \geq a_i$ is dualized in the following way: it is relaxed, meaning that it is not a constraint in the lagrangian subproblem; additionally, a *penalty term* $\lambda_i\big(a_i - f_i(y)\big)$ is associated with it and brought to the objective value function of the lagrangian subproblem.

Thus, given $\lambda \in \mathbb{R}^m_{\geq 0}$, the lagrangian subproblem associated with λ is:

$$\begin{array}{rlll} Z(\lambda) = & \min & C(y) + \displaystyle\sum_{i=1}^{m} \lambda_i\big(a_i - f_i(y)\big) & \\ & \text{s.t} & y \in \mathcal{P} & \end{array} \tag{IPLagSp}$$

Let $\mathcal{P}_{f,a} = \{y \in \mathcal{P} : f_i(y) \geq a_i,\ i \in \{1, \ldots, m\}\}$ be the feasible region of (IP). Note that $\mathcal{P}_{f,a} \subseteq \mathcal{P}$ and that any $y \in \mathcal{P}_{f,a}$ is also feasible to (IPLagSp).

Given $\lambda \in \mathbb{R}^m_{\geq 0}$, let $C^\lambda(y) = C(y) + \sum_{i=1}^m \lambda_i\big(a_i - f_i(y)\big)$ be the corresponding *lagrangian cost* of y.

Lemma 36. *Given $\lambda \in \mathbb{R}^m_{\geq 0}$, $C^\lambda(y) \leq C(y)$ for every $y \in \mathcal{P}_{f,a}$.*

Proof. Consider any $y \in \mathcal{P}_{f,a}$. Since $y \in \mathcal{P}_{f,a}$, then $(a_i - f_i(y)) \leq 0$ for every $i \in \{1, \ldots, m\}$. And since $\lambda \in \mathbb{R}^m_{\geq 0}$, then $\lambda_i(a_i - f_i(y)) \leq 0$ for every $i \in \{1, \ldots, m\}$. Hence $\sum_{i=1}^m \lambda_i(a_i - f_i(y)) \leq 0$ and $C^\lambda(y) = C(y) + \sum_{i=1}^m \lambda_i(a_i - f_i(y)) \leq C(y)$. □

Lemma 37. *Given $\lambda \in \mathbb{R}^m_{\geq 0}$, $Z(\lambda) \leq Z$.*

Proof. Let y^* be an optimal solution of (IP). Hence $C(y^*) = Z$. Note that since $y^* \in \mathcal{P}_{f,a}$, then $y^* \in \mathcal{P}$ and hence y^* is feasible to the lagrangian subproblem. Additionally, it follows from Lemma 36 that $C^\lambda(y^*) \leq C(y^*)$. And thus $Z(\lambda) \leq C^\lambda(y^*) \leq C(y^*) = Z$. □

It follows from Lemma 37 that for any $\lambda \in \mathbb{R}^m_{\geq 0}$, the optimal solution to (IPLagSp) provides a dual bound $Z(\lambda)$ on Z. Searching for $\lambda \in \mathbb{R}^m_{\geq 0}$ which provides the largest dual bound implied by (IPLagSp) leads to the *lagrangian dual problem*:

$$\begin{array}{rll} W_{LR} = & \max & Z(\lambda) \\ & \text{s.t} & \lambda \in \mathbb{R}^m_{\geq 0} \end{array} \qquad \text{(IPLagDual)}$$

It follows from Lemma 37 that $Z(\lambda) \leq C(y)$ for any $y \in \mathcal{P}_{f,a}$ and $\lambda \in \mathbb{R}^m_{\geq 0}$. Hence problems (IP) and (IPLagDual) form a weak dual pair and thus $W_{LR} \leq Z$. The *subgradient method* (SM) is a standard algorithm for solving a problem such as (IPLagDual) and is discussed in detail in Section 5.1.

4.2 Lagrangian Relaxation for the Set Cover Problem

Now consider the particular case of SCP. Consider the following integer programming formulation for it:

$$\begin{array}{rlll} Z_{SC} = & \min & \displaystyle\sum_{j \in \mathbf{N}} c_j y_j & \\ & \text{s.t} & \displaystyle\sum_{j \in J_i} y_j & \geq 1 \quad i \in \mathbf{M} \\ & & y \in \{0,1\}^n & \end{array} \qquad \text{(SCP-IP)}$$

In the case of SCP, all of the set cover constraints are the complicating constraints, i.e, all of $\sum_{j \in J_i} y_j \geq 1, i \in \mathbf{M}$. In accordance with the procedure just described, a lagrangian relaxation for SCP is obtained in the following way: for each $i \in \mathbf{M}$, let $\lambda_i \in \mathbb{R}_{\geq 0}$ be the lagrangian multiplier associated with the set cover constraint indexed by i. Each set cover constraint is dualized, and, in this way, given $\lambda \in \mathbb{R}^m_{\geq 0}$, the objective value function of the lagrangian subproblem associated with λ is:

$$\begin{aligned}\sum_{j\in\mathbf{N}} c_j y_j + \sum_{i\in\mathbf{M}} \lambda_i\big\{1-\sum_{j\in J_i} y_j\big\} &= \sum_{j\in\mathbf{N}} c_j y_j + \sum_{i\in\mathbf{M}} \lambda_i\big\{1-\sum_{j\in J_i} y_j\big\}\\ &= \sum_{j\in\mathbf{N}} c_j y_j + \sum_{i\in\mathbf{M}} \lambda_i - \sum_{i\in\mathbf{M}} \lambda_i\big\{\sum_{j\in J_i} y_j\big\}\\ &= \sum_{j\in\mathbf{N}} c_j y_j + \sum_{i\in\mathbf{M}} \lambda_i - \sum_{j\in\mathbf{N}} y_j\big\{\sum_{i\in I_j} \lambda_i\big\}\\ &= \sum_{i\in\mathbf{M}} \lambda_i + \sum_{j\in\mathbf{N}} \big\{c_j - \sum_{i\in I_j} \lambda_i\big\} y_j\end{aligned}$$

For the sake of notational convenience, given $\lambda \in \mathbb{R}^m_{\geq 0}$ and $j \in \mathbf{N}$, let $c^\lambda_j = c_j - \sum_{i\in I_j} \lambda_i$ be the *lagrangian cost* of column j. Thus $C^\lambda(y) = \sum_{i\in\mathbf{M}} \lambda_i + \sum_{j\in\mathbf{N}} c^\lambda_j y_j$. Accordingly, the lagrangian subproblem for SCP associated with λ is:

$$\begin{aligned} Z_{SC}(\lambda) = \quad & \min \quad \sum_{i\in\mathbf{M}} \lambda_i + \sum_{j\in\mathbf{N}} c^\lambda_j y_j \\ & \text{s.t.} \quad y \in \{0,1\}^n \end{aligned} \tag{SCLagSp}$$

Given $\lambda \in \mathbb{R}^m_{\geq 0}$, let $y^*(\lambda)$ be an optimal solution of the SCP lagrangian subproblem (SCLagSp).

Lemma 38. *Let $\lambda \in \mathbb{R}^m_{\geq 0}$ and $j \in \mathbf{N}$. If $c^\lambda_j < 0$, then $y^*_j(\lambda) = 1$ for any optimal solution $y^*(\lambda)$ of* (SCLagSp).

Proof. Consider any $y \in \{0,1\}^n$ such that $y_j = 0$. Let $y' \in \{0,1\}^n$ be such that $y'_j = 1$ and $y'_k = y_k$ for every $k \neq j$. Then $C^\lambda(y') = C^\lambda(y) - c^\lambda_j$. Since $c^\lambda_j < 0$, then $C^\lambda(y') = C^\lambda(y) - c^\lambda_j < C^\lambda(y)$. Thus any optimal solution $y^*(\lambda)$ satisfies that $y^*_j(\lambda) = 1$, otherwise it would not be optimal. □

Lemma 39. *Let $\lambda \in \mathbb{R}^m_{\geq 0}$ and $j \in \mathbf{N}$. If $c^\lambda_j > 0$, then $y^*_j(\lambda) = 0$ for any optimal solution $y^*(\lambda)$ of* (SCLagSp).

Proof. Consider any $y \in \{0,1\}^n$ such that $y_j = 1$. Let $y' \in \{0,1\}^n$ be such that $y'_j = 0$ and $y'_k = y_k$ for every $k \neq j$. Then $C^\lambda(y') = C^\lambda(y) + c^\lambda_j$. Since $c^\lambda_j > 0$, then $C^\lambda(y') = C^\lambda(y) + c^\lambda_j > C^\lambda(y)$. Thus any optimal solution $y^*(\lambda)$ satisfies that $y^*_j(\lambda) = 0$, otherwise it would not be optimal. □

Lemma 40. *Let $\lambda \in \mathbb{R}^m_{\geq 0}$ and $j \in \mathbf{N}$. A feasible solution y is optimal for* (SCLagSp) *if and only if $y_j = 1$ for every $j \in \mathbf{N}$ such that $c^\lambda_j < 0$ and $y_j = 0$ for every $j \in \mathbf{N}$ such that $c^\lambda_j > 0$.*

Proof. ($\rightarrow$): Follows from Lemma 38 and Lemma 39. ($\leftarrow$): Let $\mathbf{N}_- = \{j \in \mathbf{N} : c^\lambda_j < 0\}$, $\mathbf{N}_+ = \{j \in \mathbf{N} : c^\lambda_j > 0\}$ and $\mathbf{N}_0 = \{j \in \mathbf{N} : c^\lambda_j = 0\}$. Note that $C^\lambda(y') = \sum_{j\in\mathbf{N}_-} c^\lambda_j y'_j + \sum_{j\in\mathbf{N}_+} c^\lambda_j y'_j + \sum_{j\in\mathbf{N}_0} 0 = \sum_{j\in\mathbf{N}_-} c^\lambda_j y'_j + \sum_{j\in\mathbf{N}_+} c^\lambda_j y'_j$ for

any $y' \in \{0,1\}^n$. Hence $C^\lambda(y') \geq \sum_{j \in \mathbf{N}_-} c_j^\lambda$ for any $y' \in \{0,1\}^n$ and threfore $Z_{SC}(\lambda) \geq \sum_{j \in \mathbf{N}_-} c_j^\lambda$. Now suppose $y' \in \{0,1\}^n$ is such that $y_j = 1$ holds for every $j \in \mathbf{N}$ such that $c_j^\lambda < 0$ and $y_j = 0$ otherwise applies for every $j \in \mathbf{N}$ such that $c_j^\lambda > 0$. Then $C^\lambda(y) = \sum_{j \in \mathbf{N}_-} c_j^\lambda$. And since $Z_{SC}(\lambda) \geq \sum_{j \in \mathbf{N}_-} c_j^\lambda$, then $Z_{SC}(\lambda) = C^\lambda(y)$ results and y is thus optimal. □

Lemma 40 guarantees that an optimal solution $y^*(\lambda)$ to (SCLagSp) can be determined in $\mathcal{O}(n)$ time in the following way: for each column $j \in \mathbf{N}$, set $y_j^*(\lambda) = 1$ if $c_j^\lambda < 0$ and set $y_j^*(\lambda) = 0$ otherwise. Although the values assigned to columns with null lagrangian cost is arbitrary, setting their values to zero can be advantageous as it reduces the corresponding subgradient, as discussed in Section 5.1.

Finally, we close this part of the text with the following formulation for the *lagrangian dual problem* for SCP:

$$\begin{aligned} W_{SC} = \quad & \max \quad Z_{SC}(\lambda) \\ & \text{s.t} \quad \lambda \in \mathbb{R}^m_{\geq 0} \end{aligned} \tag{SCLagDual}$$

4.3 Strengthening the lagrangian subproblem

Now suppose that a primal bound $\overline{Z}$ as well as a dual bound $\underline{Z}$ are known for an SCP instance. This information can be taken into account in order to strengthen the lagrangian subproblem. Since $\underline{Z}$ is a dual bound, then any feasible solution y to (SCP-IP) satisfies that $\sum_{j \in \mathbf{N}} c_j y_j \geq \underline{Z}$. And since $\overline{Z}$ is a primal bound, then any optimal solution y^* to (SCP-IP) satisfies that $\sum_{j \in \mathbf{N}} c_j y_j^* \leq \overline{Z}$. Note that the lagrangian subproblem (SCLagSp) is unconstrained and thus a feasible solution to (SCLagSp) possibly violates these conditions. On the other hand, taking into consideration that any optimal solution y^* of Z_{SC} satisfies that $\underline{Z} \leq \sum_{j \in \mathbf{N}} c_j y_j^* \leq \overline{Z}$, we propose a novel strengthened version of the lagrangian subproblem. This strengthened version appends two valid inequalities to the original lagrangian subproblem formulation, in order to enforce the previous conditions, and is thus more likely to return stronger SCP dual bounds than its standard, unrestricted, counterpart. The strengthened version of the lagrangian subproblem is derived in the following way: starting from the lagrangian subproblem in (SCLagSp), we introduce the following two constraints: $\underline{Z} \leq \sum_{j \in \mathbf{N}} c_j y_j$, in order to ensure that every solution of the lagrangian subproblem respects the lower bound provided by the dual bound $\underline{Z}$; and $\sum_{j \in \mathbf{N}} c_j y_j \leq \overline{Z}$, in order to ensure that every solution of the lagrangian subproblem respects the upper bound provided by the primal bound $\overline{Z}$. Accordingly, the strengthened version of the lagrangian subproblem for SCP is given by:

$$
\begin{aligned}
\min \quad & \sum_{i \in \mathbf{M}} \lambda_i + \sum_{j \in \mathbf{N}} c_j^{\lambda} y_j \\
\text{s.t.} \quad & \\
& \sum_{j \in \mathbf{N}} c_j y_j \quad \geq \underline{Z} \\
& \sum_{j \in \mathbf{N}} c_j y_j \quad \leq \overline{Z} \\
& y \in \{0,1\}^n
\end{aligned}
\tag{SCIntLagSp+}
$$

However, the problem described in (SCIntLagSp+) formulates a restricted 0-1 knapsack problem which turns out to be NP-hard in the general case. However, by further relaxing the integrality constraints in (SCIntLagSp+), one thus obtains the following *strengthened lagrangian subproblem* for SCP:

$$
\begin{aligned}
Z_{SC}^{+}(\lambda) = \quad \min \quad & \sum_{i \in \mathbf{M}} \lambda_i + \sum_{j \in \mathbf{N}} c_j^{\lambda} y_j \\
\text{s.t.} \quad & \\
& \sum_{j \in \mathbf{N}} c_j y_j \quad \geq \underline{Z} \\
& \sum_{j \in \mathbf{N}} c_j y_j \quad \leq \overline{Z} \\
& y \in [0,1]^n
\end{aligned}
\tag{SCLagSp+}
$$

Given $\lambda, \underline{Z}$ and $\overline{Z}$, let $y^*(\lambda, \underline{Z}, \overline{Z})$ be an optimal solution to (SCLagSp+). Accordingly, likewise any optimal solution to SCP, $y^*(\lambda, \underline{Z}, \overline{Z})$ must satisfy that $\underline{Z} \leq \sum_{j \in \mathbf{N}} c_j y_j^*(\lambda, \underline{Z}, \overline{Z})$ and $\sum_{j \in \mathbf{N}} c_j y_j^*(\lambda, \underline{Z}, \overline{Z}) \leq \overline{Z}$. By contrast, the optimal solution $y^*(\lambda)$ of (SCLagSp) does not necessarily satisfy them. Therefore, by further imposing these constraints, an optimal solution to (SCLagSp+) is a better estimate of an optimal SCP solution than its (SCLagSp) counterpart. In fact, as demonstrated by the following results, $Z_{SC}^{+}(\lambda) \geq Z_{SC}(\lambda)$ for every $\lambda \in \mathbb{R}_{\geq 0}^m$, and we prove that in some cases $Z_{SC}^{+}(\lambda) > Z_{SC}(\lambda)$.

In the general case, let $v \in \mathbb{R}^n$, $w \in \mathbb{R}_{>0}^n$ and $\underline{w}, \overline{w} \in \mathbb{R}_{>0}$ such that $\underline{w} \leq \overline{w}$ and $\underline{w} \leq \sum_{j=1}^n w_j$. Consider the following restricted 0-1 knapsack problem:

$$
\begin{aligned}
\max \quad & \sum_{j=1}^{n} v_j y_j \\
\text{s.t.} \quad & \sum_{j=1}^{n} w_j y_j \quad \geq \quad \underline{w} \\
& \sum_{j=1}^{n} w_j y_j \quad \leq \quad \overline{w} \\
& y \in \{0,1\}^n
\end{aligned}
\tag{IntKp}
$$

and its linear programming relaxation

$$
\begin{aligned}
Z_{KP} = \quad & \max \quad \sum_{j=1}^{n} v_j y_j & & \\
& \text{s.t.} \quad \sum_{j=1}^{n} w_j y_j & \geq & \underline{w} \\
& \qquad \sum_{j=1}^{n} w_j y_j & \leq & \overline{w} \\
& \qquad y \in [0,1]^n & &
\end{aligned}
\tag{FracKp}
$$

For each $j \in \{1, \ldots, n\}$, j is called an *item*, v_j is called its *value* and w_j is called its *weight*. Given $y \in [0,1]^n$, let $V(y) = \sum_{j=1}^{n} v_j y_j$ be the *value* of y and let $W(y) = \sum_{j=1}^{n} w_j y_j$ be its *weight*. Note that (IntKp) and (FracKp) are generalizations of the well-known *integer knapsack problem* and the *fractional knapsack problem*, respectively. The two differences are: the standard knapsack problems only consider an upper bound $\overline{w}$, but no lower bound; the standard knapsack problems usually assume without loss of generality that all item values are positive, whereas (IntKp) and (FracKp) allow item values to be zero or negative as well. Finally, note that (SCIntLagSp+) and (SCLagSp+) are particular cases of (IntKp) and (FracKp), respectively, when $v = -c^{\lambda}$, $w = c$, $\underline{w} = \underline{Z}$ and $\overline{w} = \overline{Z}$. An instance of problem (IntKp) or (FracKp) is defined by $(v, w, \underline{w}, \overline{w})$. We argue that an optimal solution of (FracKp) can be determined in polynomial time by a greedy algorithm. Assume, without loss of generality, that $\frac{v_1}{w_1} > \frac{v_2}{w_2} > \cdots > \frac{v_n}{w_n}$. To prove that the assumption that $\frac{v_k}{w_k} \neq \frac{v_l}{w_l}$ for every $k \neq l$ is made without loss of generality, assume that there are two items $k, l \in \{1, \ldots, n\}$ such that $\frac{v_k}{w_k} = \frac{v_l}{w_l}$. Then an equivalent instance can be obtained by *merging* items k and l in the following way: remove both k and l from the current instance and introduce a new item with value $\frac{v_k}{w_k}(w_k + w_l)$ and weight $w_k + w_l$. Note that $\frac{\frac{v_k}{w_k}(w_k+w_l)}{w_k+w_l} = \frac{v_k}{w_k} = \frac{v_l}{w_l}$, thus the merged item has the same value to weight ratio as the two original items. It follows from Lemma 51 that both instances have the same optimal objective value. The assumption that items are ordered by their value to weight ratio is made without of loss generality because if that was not the case, then one can simply reorder the items. The greedy algorithm begins with a null solution and iterates over the items in non-increasing order of their value to weight ratio. For the sake of notational convenience, given $x \in \mathbb{R}$, let $\phi(x) = \min\{1, \max\{0, x\}\}$ be the truncation of x to the interval $[0,1]$. At each iteration $t \geq 0$, let y^t denote the partially constructed knapsack solution at iteration t. At each iteration $t \geq 1$, the algorithm defines a new partial solution y^t from y^{t-1} by changing the solution value only for item t. Start by setting $t = 0$ and $y_j^0 = 0$ for every $j \in N$. Note that at iteration $t \geq 1$, $y_j^{t-1} = 0$ for every $j \geq t$. And therefore $W(y^{t-1}) = \sum_{l<t} w_l y_l^{t-1}$ and $W(y^t) = W(y^{t-1}) + w_t y_t^t$. At iteration $t \geq 1$, the greedy algorithm considers item t. If $v_t > 0$, then inserting this item

to the current solution increases the objective value, which is favorable for the sake of maximization. But in order to satisfy the knapsack upper bound constraint, at most $\frac{\overline{w}-W(y^{t-1})}{w_t}$ can be inserted. And since $W(y^{t-1}) = \sum_{l<t} w_l y_l^{t-1}$, then at most $\frac{\overline{w}-\sum_{l<t} w_l y_l^{t-1}}{w_t}$ can be inserted. And because of constraint $y \in [0,1]^n$, the algorithm sets $y_t^t = \phi\Big(\frac{\overline{w}-\sum_{l<t} w_l y_l^{t-1}}{w_t}\Big)$ and $y_j^t = y_j^{t-1}$ for every $j \neq t$. On the other hand, if $v_t \leq 0$, then inserting this item to the current solution does not increase the objective value, which is not favorable for the sake of maximization. But in order to satisfy the knapsack lower bound constraint, at least $\frac{\underline{w}-W(y^{t-1})}{w_t}$ must be inserted. And since $W(y^{t-1}) = \sum_{l<t} w_l y_l^{t-1}$, then at least $\frac{\underline{w}-\sum_{l<t} w_l y_l^{t-1}}{w_t}$ must be inserted. Furthermore, due to constraint $y \in [0,1]^n$, the algorithm sets $y_t^t = \phi\Big(\frac{\underline{w}-\sum_{l<t} w_l y_l^{t-1}}{w_t}\Big)$ and $y_j^t = y_j^{t-1}$ for every $j \neq t$. Let y^g be the solution given by the greedy algorithm at termination. Note that for every $j \in N$, $y_j^t = y_j^j$ for every $t \geq j$, meaning that once a value is assigned to an item j at the j-th iteration, then its value is no longer modified. Moreover, if at some iteration t, $v_t \leq 0$ and $W(y^{t-1}) \geq \underline{w}$, then we conclude that $y_j^{t'} = 0$ for every $j \geq t$ and every $t' \geq t$. Therefore the greedy algorithm can stop, since no more modifications to the current solution are going to be made. So we conclude that $y_j^g = \phi\Big(\frac{\overline{w}-\sum_{l<j} w_l y_l^g}{w_j}\Big)$ if $v_j > 0$ and $y_j = \phi\Big(\frac{\underline{w}-\sum_{l<j} w_l y_l^g}{w_j}\Big)$ if $v_j \leq 0$. We begin by proving that y^g is a feasible solution to (FracKp).

Given w and $\underline{w}$, let $\tilde{k} = \max\{1 \leq k \leq n : \sum_{l\leq k} w_l \leq \underline{w}\}$.

Lemma 41. *Let y^g be the solution given by the greedy algorithm. Then $y_k^g = 1$ for every $k \leq \tilde{k}$.*

Proof. Consider any $k \leq \tilde{k}$. It follows from the definition of $\tilde{k}$ that $\sum_{l\leq k} w_l = \sum_{l<k} w_l + w_k \leq \underline{w}$. And thus $1 \leq \frac{\underline{w}-\sum_{l<k} w_l}{w_k}$ holds. Since $\sum_{l<k} w_l y_l^g \leq \sum_{l<k} w_l$, then $1 \leq \frac{\underline{w}-\sum_{l<k} w_l}{w_k} \leq \frac{\underline{w}-\sum_{l<k} w_l y_l^g}{w_k}$. Hence $y_k^g = \phi\Big(\frac{\underline{w}-\sum_{l<k} w_l y_l^g}{w_k}\Big) = 1$. □

Lemma 42. *Let y^g be the solution given by the greedy algorithm. Then $W(y^g) \geq \underline{w}$.*

Proof. It follows from Lemma 41 that $W(y^g) = \sum_{j\leq\tilde{k}} w_j + \sum_{j>\tilde{k}} w_j y_j^g$. And therefore $W(y^g) \geq \sum_{j\leq\tilde{k}} w_j + w_{\tilde{k}+1} y_{\tilde{k}+1}^g$. Additionally from the definition of $\tilde{k}$ it follows that $0 \leq \frac{\underline{w}-\sum_{l\leq\tilde{k}} w_l}{w_{\tilde{k}+1}} \leq 1$. Therefore $y_{\tilde{k}+1}^g \geq \frac{\underline{w}-\sum_{l\leq\tilde{k}} w_l}{w_{\tilde{k}+1}}$, and thus $W(y^g) \geq \sum_{j\leq\tilde{k}} w_j + w_{\tilde{k}+1}\frac{\underline{w}-\sum_{l\leq\tilde{k}} w_l}{w_{\tilde{k}+1}} = \sum_{j\leq\tilde{k}} w_j + \underline{w} - \sum_{l\leq\tilde{k}} w_l = \underline{w}$. □

Lemma 43. *Let y^g be the solution given by the greedy algorithm. Then $W(y^g) \leq \overline{w}$.*

Proof. We prove by induction that $\sum_{l\leq j} w_l y_l^g \leq \overline{w}$ for every $j \geq 1$. Base: immediate since $y_1^g \leq \phi\Big(\frac{\overline{w}-w_1}{w_1}\Big)$. Induction hypothesis: suppose the claim is true for some $k > 1$. Induction step: Consider $\sum_{l\leq k+1} w_l y_l^g$. It follows from the induction hypothesis that $\sum_{l\leq k} w_l y_l^g \leq \overline{w}$, and thus $0 \leq \frac{\overline{w}-\sum_{l\leq k} w_l y_l^g}{w_k}$. And hence $y_k^g \leq \frac{\overline{w}-\sum_{l\leq k} w_l y_l^g}{w_k}$. Therefore $\sum_{l\leq k+1} w_l y_l^g = \sum_{l\leq k} w_l y_l^g + w_k y_k^g \leq \sum_{l\leq k} w_l y_l^g + w_k \frac{\overline{w}-\sum_{l\leq k} w_l y_l^g}{w_k} = \overline{w}$. □

It follows from Lemma 42 and 43 that the greedy algorithm produces a feasible solution of (FracKp).

Given a feasible solution y of (FracKp), $k, l \in N$ and $\delta > 0$, let $y'(y, k, l, \delta)$ be such that $y'_k(y, k, l, \delta) = y_k + \delta$ and $y'_l(y, k, l, \delta) = y_l - \frac{w_k \delta}{w_l}$ and $y'_j(y, k, l, \delta) = y_j$ for every $j \neq k, l$. Note that $W(y'(y, k, l, \delta)) = W(y) + w_k\delta - w_l(\frac{w_k\delta}{w_l}) = W(y) + w_k\delta - w_k\delta = W(y)$. Note as well that if $\delta \in [-y_k, 1 - y_k]$, then $y'_k(y, k, l, \delta) \in [0, 1]$. And if $\delta \in [\frac{w_l(y_l-1)}{w_k}, \frac{w_l y_l}{w_k}]$, then $y'_l(y, k, l, \delta) \in [0, 1]$. Thus if $\max\{-y_k, \frac{w_l(y_l-1)}{w_k}\} \leq \delta \leq \min\{1 - y_k, \frac{w_l y_l}{w_k}\}$, then $y'(y, k, l, \delta)$ is a feasible solution of (FracKp). And note that $V(y'(y, k, l, \delta)) = V(y) + v_k\delta + v_l(\frac{w_l y_l - w_k\delta}{w_l} - y_l) = V(y'(y, k, l, \delta)) + v_k\delta + v_l y_l - \frac{v_l w_k \delta}{w_l} - v_l y_l = W(y) + v_k\delta - \frac{v_l w_k \delta}{w_l} = W(y) + \delta w_k(\frac{v_k}{w_k} - \frac{v_l}{w_l})$.

Lemma 44. *Let y^* be an optimal solution of* (FracKp)*. If $v_1 > 0$, then $y^*_1 = \phi\Big(\frac{\overline{w}}{w_1}\Big)$.*

Proof. Note that any feasible solution y satisifies that $y_1 \leq \phi\Big(\frac{\overline{w}}{w_1}\Big)$, otherwise y would be infeasible. Assume $v_1 > 0$ and suppose, for the sake of contradiction, that there is an optimal solution y^* such that $y^*_1 < \phi\Big(\frac{\overline{w}}{w_1}\Big)$. Let $\epsilon = \phi\Big(\frac{\overline{w}}{w_1}\Big) - y^*_1$. Since $v_1 > 0$, we argue that $W(y^*) = \overline{w}$. Suppose $W(y^*) < \overline{w}$. In this case, let $\delta = \min\{\epsilon, \frac{\overline{w} - W(y^*)}{w_1}\}$. Let y such that $y_1 = y^*_1 + \delta$ and $y_j = y^*_j$ for every $j > 1$. Note that $W(y) = W(y^*) + w_1\delta \leq W(y^*) + w_1\frac{\overline{w} - W(y^*)}{w_1} = W(y^*) + \overline{w} - W(y^*) = \overline{w}$. Thus y is feasible. And note that $V(y) = V(y^*) + v_1\delta > V(y^*)$, a contradiction, since y^* is optimal. Hence $W(y^*) = \overline{w}$. Since $\epsilon > 0$ and $W(y^*) = \overline{w}$, then there is $l > 1$ such that $y^*_l > 0$. Let $\delta_1 = \min\{\epsilon, \frac{w_l y^*_l}{w_1}\} > 0$. Then $y'(y^*, 1, k, \delta_1)$ is a feasible solution. Since $\delta_1 > 0$ and $\frac{v_1}{w_1} > \frac{v_l}{w_l}$, then $V(y'(y^*, 1, k, \delta_1)) = V(y^*) + \delta_1 w_1(\frac{v_1}{w_1} - \frac{v_l}{w_l}) > V(y^*)$, a contradiction, since y^* is optimal. □

Lemma 45. *Let y^* be an optimal solution of* (FracKp)*. If $v_1 < 0$, then $y^*_1 \geq \phi\Big(\frac{\underline{w}}{w_1}\Big)$.*

Proof. Suppose, for the sake of contradiction, that there is an optimal solution y^* such that $y^*_1 < \phi\Big(\frac{\underline{w}}{w_1}\Big)$. Let $\epsilon = \phi\Big(\frac{\underline{w}}{w_1}\Big) - y^*_1$. Since $v_1 < 0$, we argue that $W(y^*) = \underline{w}$. Suppose $W(y^*) > \underline{w}$. In this case, let $\delta = \min\{\epsilon, \frac{W(y^*) - \underline{w}}{w_1}\}$. Let y such that $y_1 = y^*_1 - \delta$ and $y_j = y^*_j$ for every $j > 1$. Note that $W(y) = W(y^*) - w_1\delta \geq W(y^*) - w_1\frac{W(y^*) - \underline{w}}{w_1} = \overline{w}$. And since $W(y) = W(y^*) - w_1\delta \leq W(y^*) \leq \overline{w}$, then y is feasible. And note that $V(y) = V(y^*) - v_1\delta > V(y^*)$, a contradiction, since y^* is optimal. Hence $W(y^*) = \underline{w}$. Since $\epsilon > 0$ and $W(y^*) = \underline{w}$, then there is $l > 1$ such that $y^*_l > 0$. Let $\delta_1 = \min\{\epsilon, \frac{w_l y^*_l}{w_1}\} > 0$. Then $y'(y^*, 1, l, \delta_1)$ is a feasible solution. Since $\delta_1 > 0$ and $\frac{v_1}{w_1} > \frac{v_l}{w_l}$, then $V(y'(y^*, 1, k, \delta_1)) = V(y^*) + \delta_1 w_1(\frac{v_1}{w_1} - \frac{v_l}{w_l}) > V(y^*)$, a contradiction, since y^* is optimal. □

Lemma 46. *Let y^* be an optimal solution of* (FracKp)*. If $v_1 < 0$, then $y^*_1 \leq \phi\Big(\frac{\underline{w}}{w_1}\Big)$.*

Proof. Suppose, for the sake of contradiction, that there is an optimal solution y^* such that $y^*_1 > \phi\Big(\frac{\underline{w}}{w_1}\Big)$. Let $\epsilon = y^*_1 - \phi\Big(\frac{\underline{w}}{w_1}\Big)$. Note that $W(y^*) > w_1\frac{\underline{w}}{w_1} = \underline{w}$. Let

$\delta = \min\{\epsilon, \frac{W(y^*)-\underline{w}}{w_1}\}$. Let y such that $y_1 = y_1^* - \delta$ and $y_j = y_j^*$ for $j > 1$. Note that $W(y) = W(y^*) - w_1\delta = W(y^*) - w_1 \min\{\epsilon, \frac{W(y^*)-\underline{w}}{w_1}\} \geq W(y^*) - w_1\frac{W(y^*)-\underline{w}}{w_1} = \underline{w}$. And since $W(y) \leq W(y^*) \leq \overline{w}$, then y is feasible. But since $v_1 < 0$, then $V(y) = V(y^*) - v_1\delta > V(y^*)$, a contradiction, since y^* is optimal. □

Lemma 47. *Let y^* be an optimal solution of* (FracKp). *If $v_1 = 0$ and $w_1 \leq \underline{w}$, then $y_1^* = \phi\Big(\frac{\underline{w}}{w_1}\Big)$.*

Proof. It follows from Lemma 45 that any optimal solution y^* satisfies that $y_1^* \geq \phi\Big(\frac{\underline{w}}{w_1}\Big)$. Since $w_1 \leq \underline{w}$, then $\phi\Big(\frac{\underline{w}}{w_1}\Big) = 1$. And since $y_1^* \leq 1$, then $y_1^* = 1 = \phi\Big(\frac{\underline{w}}{w_1}\Big)$. □

Lemma 48. *Let y^* be an optimal solution of* (FracKp). *If $w_1 \leq \underline{w}$, then $y_1^* = y_1^g$.*

Proof. If $v_1 > 0$, it follows from Lemma 44. If $v_1 \leq 0$ and $w_1 \leq \underline{w}$, it follows from Lemma 45 and 46 and 47. □

Lemma 49. *Let y^g be the solution given by the greedy algorithm. If $v_1 = 0$ and $w_1 > \underline{w}$, then y^g is optimal.*

Proof. Since $w_1 > \underline{w}$, then $\phi\Big(\frac{\underline{w}}{w_1}\Big) < 1$. Note that $y_1^g = \phi\Big(\frac{\underline{w}}{w_1}\Big)$ and $y_j^* = 0$ for every $j > 1$. Note that $V(y^g) = 0$. Since $v_1 = 0$, then $v_j < 0$ for every $j > 1$. And thus $V(y) \leq 0$ for any feasible solution y. Hence y^g is optimal. □

Lemma 50. *Let y^* be an optimal solution of instance $\big(v, w, \underline{w}, \overline{w}\big)$ of* (FracKp) *and $k \geq 1$. Then $\{y_j^*\}_{l\geq k}$ is an optimal solution of instance $\big(\{v\}_{l\geq k}, \{w\}_{l\geq k}, \underline{w} - \sum_{l<k} w_l y_l^*, \overline{w} - \sum_{l<k} w_l y_l^*\big)$.*

Proof. Suppose, for the sake of contradiction, that $\{y_j^*\}_{l\geq k}$ is not an optimal solution of instance $\big(\{v\}_{l\geq k}, \{w\}_{l\geq k}, \underline{w} - \sum_{l<k} w_l y_l^*, \overline{w} - \sum_{l<k} w_l y_l^*\big)$. Then there is $y \in [0,1]^{n-k+1}$ such that $V(y) > V(\{y_j^*\}_{l\geq k})$. Let $y' \in [0,1]^n$ such that $y'_j = y_j^*$ for $j < k$ and $y'_j = y_j$ for $j \geq k$. Note that $W(y') = W(y) + \sum_{l<k} w_l y_l^*$. And thus $W(y') \geq (\underline{w} - \sum_{l<k} w_l y_l^*) + \sum_{l<k} w_l y_l^* = \underline{w}$ and $W(y') \leq (\overline{w} - \sum_{l<k} w_l y_l^*) + \sum_{l<k} w_l y_l^* = \overline{w}$. Hence y' is feasible for $\big(v, w, \underline{w}, \overline{w}\big)$. And note that $V(y') = V(y) + \sum_{l<k} v_l y_l^* > V(\{y_j^*\}_{l\geq 1}) + \sum_{l<k} v_l y_l^* = V(y^*)$. This is a contradiction, since y^* is an optimal solution of of (FracKp). □

Lemma 51. *Let y^g be the solution given by the greedy algorithm. Then y^g is an optimal solution of* (FracKp).

Proof. If $v_1 = 0$ and $w_1 > \underline{w}$, it follows from Lemma 49 that y^g is optimal. Assume otherwise. Let y^* be an optimal solution. The proof is by induction. Base: it follows from Lemma 48 that there is an optimal solution y^* such that $y_1^* = y_1^g$. Induction hypothesis: suppose the claim is true for every $l \leq k$ for some $k > 1$. Induction step:

it follows from the induction hypothesis that $y^*_j = y^g_j$ for every $j \leq k$. Consider the reduced instance $\big(\{v\}_{l\geq k}, \{w\}_{l\geq k}, \underline{w} - \sum_{l\leq k} w_l y^*_l, \overline{w} - \sum_{l\leq k} w_l y^*_l\big)$. We consider two cases. If $w_{k+1} \leq \underline{w} - \sum_{l\leq k} w_l y^*_l$, then it follows from Lemma 48 that $y^*_{k+1} = y^g_{k+1}$. If $\underline{w} - \sum_{l\leq k} w_l y^*_l < w_{k+1}$, then it follows from Lemma 49 that $\{y^g\}_{l\geq k+1}$ is optimal for the reduced instance. And it follows from Lemma 50 that $\{y^*\}_{l\geq k+1}$ is also optimal. Hence $V(\{y^g\}_{l\geq k+1}) = V(\{y^*\}_{l\geq k+1})$. And thus $V(y^g) = \sum_{j\leq n} v_j y^g_j = \sum_{j\leq k} v_j y^*_j + V(\{y^g\}_{l\geq k+1}) = \sum_{j\leq k} v_j y^*_j + V(\{y^*\}_{l\geq k+1}) = V(y^*)$. Hence y^g is an optimal solution. □

It follows from Lemma 51 that an optimal solution $y^*(\lambda, \underline{Z}, \overline{Z})$ to (SCLagSp+) can be determined in the following way: given λ, let $\mathbf{N}(\lambda) = (j_1, \ldots, j_n)$ be a permutation of $\mathbf{N}$ such that $\frac{c^\lambda_{j_1}}{c_{j_1}} \leq \cdots \leq \frac{c^\lambda_{j_n}}{c_{j_n}}$. At each iteration $t \geq 1$, let:

$$y^*_{j_t}(\lambda, \underline{Z}, \overline{Z}) = \begin{cases} \phi\Big(\frac{\overline{Z} - \sum_{l<t} c_{j_l} y^*_{j_l}(\lambda, \underline{Z}, \overline{Z})}{w_{j_t}}\Big) \text{ if } c^\lambda_{j_t} < 0 \\ \phi\Big(\frac{\underline{Z} - \sum_{l<t} c_{j_l} y^*_{j_l}(\lambda, \underline{Z}, \overline{Z})}{w_{j_t}}\Big) \text{ if } c^\lambda_{j_t} \geq 0 \end{cases}$$

Given a primal bound $\overline{Z}$ on the optimal objective value Z_{SC} and since the column costs are assumed to be positive, the maximum cardinality $\overline{K}$ of an optimal set cover solution can be determined in the following way: let $j_1, j_2, \ldots, j_n$ be a permutation of $\mathbf{N}$ such that $c_{j_1} \leq c_{j_2} \leq \ldots \leq c_{j_n}$. Let $\overline{j} = \max\{k \in \{1, \ldots, n\} : \sum_{l\leq \overline{j}} c_{j_l} \leq \overline{Z}\}$. Thus $\overline{K}$ is determined by starting from an empty set S and inserting columns into S in non-decreasing order of cost while $\sum_{j\in S} c_j \leq \overline{Z}$. When the procedure stops, S satisfies that $|S| \geq |S'|$ for any $S' \subseteq \mathbf{N}$ such that $\sum_{j\in S} c_j \leq \overline{Z}$. Let $\overline{K} = |S|$. And thus, given any $\lambda \in \mathbb{R}^m_{\geq 0}$, y^g has at most $\overline{K}$ non-zero values. Therefore when solving (SCLagSp+), it is only necessary to sort the set of $\overline{K}$ columns with smaller $\frac{c^\lambda}{c}$ This can be done in $\mathcal{O}(n \log \overline{K})$ time in the following way: initialize an empty max-heap H which will store pairs $(j, \frac{c^\lambda_j}{c_j})$ consisting of a column $j \in \mathbf{N}$ and its corresponding lagrangian cost ratio $\frac{c^\lambda_j}{c_j}$. Two pairs $(k, \frac{c^\lambda_k}{c_k})$ and $(l, \frac{c^\lambda_l}{c_l})$ are compared based on $\frac{c^\lambda_k}{c_k}$ and $\frac{c^\lambda_l}{c_l}$, and therefore the max-heap root node stores a pair containing the greatest lagrangian cost ratio. Insert the pairs corresponding to columns $1, \ldots, \overline{K}$ into H, i.e, the pairs corresponding to the first $\overline{K}$ columns in lexicographical order. For each $k \in \{\overline{K}+1, \ldots, n\}$, insert the pair $(k, \frac{c^\lambda_k}{c_k})$ into H and then remove the root node from H. When this procedure stops, each column has been inserted into the max-heap exactly once and only the $\overline{K}$ columns with smaller lagrangian cost ratio remain in the heap. Each time an element is inserted, H has at most $\overline{K}$ nodes, and thus each insertion is done in $\mathcal{O}(\log \overline{K})$ time. Each time the root is removed, H has at most $\overline{K} + 1$ nodes, and thus each removal is done in $\mathcal{O}(\log \overline{K})$ time. Hence the procedure can be done in $\mathcal{O}(n \log \overline{K})$ time.

Since (SCLagSp) is a relaxation of (SCLagSp+), then it is clear that $Z_{SC}(\lambda) \leq$

$Z^+_{SC}(\lambda)$ for every $\lambda \in \mathbb{R}^m_{\geq 0}$. However, we now compare $Z_{SC}(\lambda)$ and $Z^+_{SC}(\lambda)$ in greater detail and demonstrate the cases in which $Z_{SC}(\lambda) < Z^+_{SC}(\lambda)$. Given $\lambda \in \mathbb{R}^m_{\geq 0}$, assume, without loss of generality, that $\frac{c^\lambda_1}{c_1} \leq \cdots \leq \frac{c^\lambda_n}{c_n}$.

- Let $k_- = \max\{1 \leq k \leq n : c^\lambda_k < 0\}$ denote the greatest index associated with a negative lagrangian cost. Note that $y^*_j(\lambda) = 1$ for every $l \leq k_-$.
- Let $k_+ = \min\{1 \leq k \leq n : c^\lambda_k > 0\}$ denote the smallest index associated with a positive lagrangian cost. Note that $y^*_j(\lambda) = 0$ for every $l \geq k_+$.
- Let $\underline{k} = \min\{1 \leq k \leq n : \underline{Z} < \sum_{l<k} c_l\}$ denote the smallest index to respect the lower lower bound constraint. Note that $y^*_l(\lambda, \underline{Z}, \overline{Z}) = 1$ for every $l < \underline{k}$ and $y^*_{\underline{k}}(\lambda, \underline{Z}, \overline{Z}) > 0$.
- Let $\overline{k} = \max\{1 \leq k \leq n : \sum_{l<k} c_l < \overline{Z}\}$ denote the greatest index to respect the upper bound constraint. Note that $y^*_l(\lambda, \underline{Z}, \overline{Z}) = 0$ for every $l > \overline{k}$.

Note that

$$\begin{aligned} C^\lambda(y^*(\lambda)) &= \sum_{l \leq n} c^\lambda_l y^*_l(\lambda) \\ &= \sum_{l \leq k_-} c^\lambda_l \end{aligned}$$

and

$$\begin{aligned} C^\lambda(y^*(\lambda, \underline{Z}, \overline{Z})) &= \sum_{l \leq n} c^\lambda_l y^*_l(\lambda, \underline{Z}, \overline{Z}) \\ &= \sum_{l < \underline{k}} c^\lambda_l y^*_l(\lambda, \underline{Z}, \overline{Z}) + \sum_{\underline{k} \leq l \leq \overline{k}} c^\lambda_l y^*_l(\lambda, \underline{Z}, \overline{Z}) \\ &+ \sum_{l > \overline{k}} c^\lambda_l y^*_l(\lambda, \underline{Z}, \overline{Z}) \\ &= \sum_{l < \underline{k}} c^\lambda_l \cdot 1 + \sum_{\underline{k} \leq l \leq \overline{k}} c^\lambda_l y^*_l(\lambda, \underline{Z}, \overline{Z}) \\ &+ \sum_{l > \overline{k}} c^\lambda_l \cdot 0 \\ &= \sum_{l < \underline{k}} c^\lambda_l + \sum_{\underline{k} \leq l \leq \overline{k}} c^\lambda_l y^*_l(\lambda, \underline{Z}, \overline{Z}) \end{aligned}$$

First, suppose that $\sum_{l \leq k_-} c_l < \underline{Z}$ and $k_+ \leq \underline{k}$. Then $c^\lambda_{\underline{k}} > 0$ and therefore

$$\begin{aligned} Z^+_{SC}(\lambda) - Z_{SC}(\lambda) &= C^\lambda(y^*(\lambda, \underline{Z}, \overline{Z})) - C^\lambda(y^*(\lambda)) \\ &= \sum_{k_+ \leq l < \underline{k}} c^\lambda_l + c^\lambda_{\underline{k}} y^*_{\underline{k}}(\lambda, \underline{Z}, \overline{Z}) \\ &> 0 \end{aligned}$$

This is the case in which the lower bound constraint forces $y^*(\lambda, \underline{Z}, \overline{Z})$ to have positive values corresponding to columns with positive lagrangian cost. In comparison with $y^*(\lambda)$, for which only the columns with negative lagrangian cost have a corresponding positive value, a positive term is added to the corresponding lagrangian subproblem objective function, and therefore the inclusion of the lower bound constraint ensures that (SCLagSp+) provides a greater dual bound than than (SCLagSp).

Now suppose that $\sum_{l \leq k_-} c_l > \overline{Z}$. Then $k_- > \overline{k}$ and hence $y^*_{\overline{k}}(\lambda, \underline{Z}, \overline{Z}) < 1$ and $c^\lambda_{\overline{k}} < 0$. And thus

$$\begin{aligned} Z^+_{SC}(\lambda) - Z_{SC}(\lambda) &= C^\lambda(y^*(\lambda, \underline{Z}, \overline{Z})) - C^\lambda(y^*(\lambda)) \\ &= -\big(1 - y^*_{\overline{k}}(\lambda, \underline{Z}, \overline{Z})\big) c^\lambda_{\overline{k}} - \sum_{\overline{k} < l \leq k_-} c^\lambda_l \\ &> 0 \end{aligned}$$

This is the case in which the upper bound constraint forces $y^*(\lambda, \underline{Z}, \overline{Z})$ to have zero values corresponding to columns with negative lagrangian cost. In comparison with $y^*(\lambda)$, for which all the columns with negative lagrangian cost have a corresponding positive value, a negative term is subtracted from the corresponding lagrangian subproblem objective function, and therefore the inclusion of the upper bound constraint ensures that (SCLagSp+) provides a greater dual bound than than (SCLagSp).

In practice, for values of lagrangian multipliers for which both the standard and strengthened lagrangian subproblems lead to the same solution and bound, partially sorting the columns is not necessary. These cases can easily be identified without any additional increase in complexity. When solving the strengthened lagrangian subproblem, an $\mathcal{O}(n)$ pass through the columns is necessary to update the lagrangian cost ratios. During this pass, one can keep track of the sum of costs of all columns with negative lagrangian cost. If this sum satisfies the lower and upper bound cosntraints, we can conclude that both lagrangian subproblem formulations lead to the same solution and there is no need to solve the strengthened problem. However, our experiments suggest that this is rarely the case.

Considering the strengthened lagrangian subproblem described in (SCLagSp+), we propose the *strengthened lagrangian dual* for SCP, defined by:

$$\begin{array}{rll} W_{SC}^{+} = & \max & Z_{SC}^{+}(\lambda) \\ & \text{s.t} & \lambda \in \mathbb{R}_{\geq 0}^{m} \end{array} \qquad \text{(SCLagDual+)}$$

Chapter 5

Relax and Cut

In Section 5.1 we describe the subgradient method for solving the lagrangian dual of an integer program. In Section 5.2 we describe the relax-and-cut algorithm, which adapts the subgradient method to incorporate dynamic inclusion of constraints. In Section 5.3 we present a lagrangian-based primal heuristic for SCP, and propose adaptations to handle the exponentially many constraints. Finally, in Section 5.4 we describe our proposed relax-and-cut algorithm for SCP.

5.1 Subgradient Method

Let $f : \mathbb{R}^m \to \mathbb{R}$ be a concave function. A *subdifferential* or *subgradient* of f at point u is a vector $\partial f(u)$ such that $f(v) \leq f(u) + (v - u) \cdot \partial f(u)$ for every $v \in \mathbb{R}^m$. Note that if f is a continuously differentiable function, then the gradient $\nabla f(u)$ is the subgradient of f at point u. The *gradient method* is an iterative method for solving an unconstrained optimization problem of a concave and continuously differentiable objective value function. In turn, the *subgradient method* was originally proposed by POLYAK [98] and is an analogous version of the gradient method but for solving unconstrained optimization problems with a concave but not continuously differentiable objective value function, such as (SCLagDual) and (SCLagDual+). Consider the following unconstrained maximization problem:

$$\begin{array}{rll} f^* = & \max & f(u) \\ & \text{s.t} & u \in \mathbb{R}^m_{\geq 0} \end{array} \tag{5.1}$$

where $f(u)$ is a concave not continuously differentiable function.

The guiding principle of the subgradient method is to start at a point u^0 and at each iteration $t \geq 0$ generate a new point u^{t+1} by taking a *subgradient step* in the direction of $\partial f(u^t)$. Accordingly, given an appropriate *step size* $\epsilon_t > 0$, a new point u^{t+1} is determined by the following rule: $u_i^{t+1} = \max\{0, u_i^t - \epsilon_t \partial f(u^t)\}$ for each

$i \in \{1, \ldots, m\}$. The general idea is that by iteratively taking steps of conveniently defined length in the opposite direction of the subgradient, the sequence $\{u^t\}_{t\geq 0}$ gets closer to the optimal solution of (5.1) and $\{f(u^t)\}_{t\geq 0}$ gets closer to the optimal objective value f^*.

Consider a general linear integer programming problem

$$\begin{array}{rlll} Z = & \min & C(y) & \\ & \text{s.t} & a_i \cdot y & \geq b_i \quad i \in \{1, \ldots, m\} \\ & & y \in \{0,1\}^n & \end{array} \tag{IP}$$

Additionally, for a given $\lambda \in \mathbb{R}^m_{\geq 0}$, consider the lagrangian subproblem

$$\begin{array}{rll} Z(\lambda) = & \min & C(y) + \sum_{i=1}^{m} \lambda_i \big(b_i - a_i \cdot y\big) \\ & \text{s.t} & y \in \{0,1\}^n \end{array} \tag{IPLagSp}$$

And the corresponding lagrangian dual problem that results from (IPLagSp)

$$\begin{array}{rll} W_{LR} = & \max & Z(\lambda) \\ & \text{s.t} & \lambda \in \mathbb{R}^m_{\geq 0} \end{array} \tag{IPLagDual}$$

The subgradient method is probably the most common method for solving a lagrangian dual problem such as (IPLagDual). In the case of a general linear integer program (IP) and its corresponding lagrangian dual problem (IPLagDual), the subgradient vector $\partial f(\lambda)$ for a given vector of lagrangian multipliers λ is such that

$$\partial f_i(\lambda) = b_i - a_i \cdot y^*$$

for each $i \in \{1, \ldots, m\}$, where y^* is the optimal solution of the lagrangian subproblem (IPLagSp).

The success of the subgradient method in finding an optimal solution or near-optimal solution relies on an appropriate choice of step sizes and an appropriate stopping criteria. According to [35], it has been proven that if $\lim_{t\to\infty} \epsilon_t = 0$ and $\sum_{t=0}^{\infty} \epsilon_t = \infty$, then $\lim_{t\to\infty} f(u^t) = f^*$. However, such a sequence is considered to have very slow convergence for practical purposes. Another result described in [35] is that if $\underline{f}$ is such that $\underline{f} \leq f^*$ and $\epsilon_t = \alpha_t \frac{f(u^t) - \underline{f}}{||\partial f(u^t)||}$ where $0 < \alpha_t < 2$, then $\lim_{t\to\infty} f(u^t) = \underline{f}$ or there is t' such that $\underline{f} \leq f(u^{t'}) \leq f^*$. Hence the success of the subgradient method with this choice of step sizes relies on knowing a tight lower bound $\underline{f}$. In practice however, $\underline{f}$ is typically unkown or $f^* - \underline{f} > 0$ and thus $f(u^t) \nrightarrow f^*$. A well-accepted compromise solution for is described by FISHER [42]: suppose that $\overline{f}, f^* \leq \overline{f}$ is a known upper bound on f^*. Let u^* be the point associated with the currently known greatest value of f. Hence $u^* = u^k$ for some $k \geq 0$

such that $f(u^l) \leq f(u^k)$ for every $l \neq k$. Let $\alpha_0 = 2$. At each iteration t, the step size is given by $\epsilon_t = \alpha_t \frac{\overline{f} - f(u^t)}{||\partial f(u^t)||^2}$. Given $\tilde{\eta}$, at each iteration t, if u^* was not updated in the last $\tilde{\eta}$ iterations, then α_t is halved, meaning that $\alpha_{t+1} = \frac{\alpha_t}{2}$. Otherwise, $\alpha_{t+1} = \alpha_t$. Therefore $\{\alpha_t\}_{t \geq 0}$ is a non-increasing sequence. Given $\tilde{\alpha} > 0$, the subgradient method stops as soon as $\alpha_t < \tilde{\alpha}$. Note that if $u_i^t = 0$ and $\partial f_i(u^t) < 0$, then $u_i^t = u_i^{t+1} = 0$ and thus the subgradient step thus have no effect. Accordingly, it was suggested by BEASLEY [99] to arbitrarily set $\partial f_i(u^t) = 0$ if $u_i^t = 0$ and $\partial f_i(u^t) < 0$. By doing so, when computing the norm $||\partial f(u^t)||^2$, those subgradient indices which do not lead to an actual update on the corresponding lagrangian multiplier are not taken into account. As reported in BEASLEY [99], in practice this leads to a better convergence of the subgradient method. The convergence of the subgradient method to a near-optimal solution with this step size rule depends on the quality of the upper bound $\overline{f}$, and ideally $\overline{f}$ should be equal to f^*. When solving the lagrangian dual problem for an integer program, such as the case of (IPLagDual), $\overline{f}$ is usually given by a primal feasible solution. For integer programming and combinatorial optimization problems, finding a primal solution of good or optimal quality may require the exploration of a diverse set of feasible solutions. For this reason, it is common to employ a lagrangian heuristic algorithm recurrently during the subgradient procedure, with the goal of producing diverse primal solutions and hopefully improving the upper bound $\overline{f}$. The solution produced by the lagrangian heuristic at an iteration t depends on the value of u^t and thus distinct solutions are generated along the subgradient procedure. This idea is discussed in detail in Section 5.3.

5.2 Relax and Cut

If the number of constraints m is a polynomial function of n, then the subgradient method exactly as described in Section 5.1 can be used to solve the lagrangian dual problem and in practice shows good convergence to near-optimal lagrangian multipliers, as reported by FISHER [42]. On the other hand, suppose that m is an exponential function of n and thus the integer program has exponentially many constraints. In this case, the subgradient method needs to be adapted in order to deal with the exceedingly large number of constraints. Accordingly, *relax-and-cut* algorithms are lagrangian relaxation methods designed to solve the lagrangian dual of a primal problem with exponentially many constraints. The development of such lagrangian techniques dates to as early as 1981, including, in chronological order, the works of BALAS and CHRISTOFIDES [100] in 1981 for the travelling salesman problem, of GAVISH [101] in 1985 for a network design problem and of LUCENA [102] in 1992 for the steiner tree problem. The works by LUCENA [102] in 1992 and 1993 were the first ones to dualize violated constraints "on the fly", mean-

ing that constraints are dynamically dualized during the relax-and-cut procedure whenever they are violated by the solution of the current lagrangian subproblem. The term *relax-and-cut* was originally proposed in 1994 by ESCUDERO *et al.* [104] in their work for a sequential ordering problem. Following the suggestion of LUCENA [97] in 2005, we adopt the term relax-and-cut to refer to a broad class of lagrangian relaxation based algorithms with dynamic inclusion of constraints. Furthermore, LUCENA [97] makes a distinction between two classes of relax-and-cut algorithms: the so-called *delayed relax-and-cut* (DRC), which performs a sequence of consecutive rounds of the subgradient method and delays the inclusion of violated constraints until the end of each subgradient round; and *non-delayed relax-and-cut* (NDRC), which performs a single round of the subgradient method and includes violated constraints at each iteration of the subgradient method. The general procedure of a non-delayed relax and cut is described as follows. In accordance with the standard procedure of lagrangian relaxations methods, as described in Chapter 4, for each constraint i there is a corresponding lagrangian multiplier $\lambda_i \geq 0$, and the lagrangian multipliers are optimized by means of an adapted subgradient method. More specifically, at each iteration $t \geq 0$ of the relax-and-cut algorithm, let λ^t denote the current lagrangian multipliers and $\bar{y}^t$ denote the solution of the corresponding lagrangian subproblem. Note that at each iteration, only the lagrangian multipliers with positive value effectively take part on the current lagrangian subproblem, since a lagrangian multiplier with null value has no effective contribution to its objective value function. At each iteration $t \geq 0$, the set of constraints can be grouped into three classes. The first one, denoted by CV^t, consists of all constraints which are violated by $\bar{y}^t$. The second class, denoted by CE^t, contains every constraint $i \in \mathbf{M}$ such that $\lambda_i^t > 0$. And the third class, denoted by CI^t, contains every constraint $i \in \mathbf{M}$ such that $\lambda_i^t = 0$ and $b_i - a_i \cdot \bar{y}^t \leq 0$, i.e, which is not violated by $\bar{y}^t$. Note that at any iteration $t \geq 0$, only the constraints in CE^t effectively contribute to the current lagrangian subproblem, since only positive lagrangian multipliers make an effective contribution to it. For this reason, we refer to CE^t as the *currently effective set*. Moreover, note that for each constraint $i \in \mathrm{CV}^t$, since i is violated by $\bar{y}^t$, then $\partial f_i(\lambda^t) = b_i - a_i \cdot \bar{y}^t > 0$ and thus, because of the subgradient step, it follows that $\lambda_i^{t+1} > 0$. Hence $\mathrm{CV}^t \subseteq \mathrm{CE}^{t+1}$ and thus the constraints in CV^t will have an effective contribution in the next iteration. We refer to CV^t as the *currently violated set*. Finally, note that for each constraint $i \in \mathrm{CI}^t$, $\lambda_i^t = \lambda_i^{t+1} = 0$ holds, since $\lambda_i^t = 0$ and $\partial f_i(\lambda^t) = b_i - a_i \cdot \bar{y}^t \leq 0$. Therefore the constraints in CI^t have no effective contribution to both the current and the next lagrangian subproblem. For this reason, we refer to CI^t as the *currently ineffective set*. Even though they have no effective contribution to the current lagrangian subproblem, the constraints in CI^t do take part in the computation of the step size given by $\epsilon_t = \alpha_t \frac{\overline{f} - f(\lambda^t)}{||\partial f(\lambda^t)||^2}$.

Because of the exceedingly large number of constraints, there are possibly too many non-zero subgradient entries, leading to a possibly too large value of $||\partial f(\lambda^t)||^2$ and consequently a virtually zero value of ϵ_t. As a result, when all subgradient entries are considered, the step sizes are virtually zero, which in turn causes the lagrangian multipliers to stay virtually unchanged after each subgradient update, and hence compromise the convergence of the subgradient method. One solution to this problem is proposed by BEASLEY [99] and consists of projecting the subgradient into a smaller dimension by arbitrarily setting to zero the subgradient entries corresponding to the constraints in CI^t. Recall that these constraints have no actual change in their corresponding lagrangian multipliers for the next iteration. When m is not too large when compared to n, BEASLEY [99] reported satisfactory convergence of the subgradient method by applying this kind of projection. Nevertheless, another challenge remains: since the complexity of each subgradient step is $\mathcal{O}(nm)$, then explicitly storing in memory and computing subgradient values and multiplier updates for an exponentially large number of constraints would require an overwhelming computational endeavor to perform a series of subgradient steps. The relax-and-cut algorithm overcomes this difficulty by considering that only a subset of constraints are *active*, whereas the remaining constraints are *inactive*. Each active constraint is associated with an active lagrangian multiplier which is updated by the subgradient step at each iteration. By contrast, each inactive constraint is associated with an inactive lagrangian multiplier which has its value set to 0 and is not updated. In practice, values such as the corresponding lagrangian multiplier and subgradient are only stored in memory and computed for active constraints, meaning that the active constraints are handled *explicitly*. On the other hand, these values are not stored for inactive constraints, meaning that the inactive constraints are handled *implicitly*. This ensures that less computational effort is required to perform each subgradient iteration and enables the procedure to be computationally viable. Since a constraint in CV^t should have a corresponding non-zero lagrangian multiplier in the next iteration because of the subgradient step, then inactive constraints which are violated must be *activated* in order to allow that lagrangian multiplier to change value. Accordingly, an inactive constraint i is activated at the first iteration $t \geq 0$ such that i is violated by $\bar{y}^t$. From that point on, its corresponding lagrangian multiplier is active and gets updated by the subgradient step. At each iteration $t \geq 0$, let $\mathcal{A}^t \subseteq \mathbf{M}$ denote the set of active constraints and let $\mathcal{I}^t \subseteq \mathbf{M}$ denote the set of inactive constraints. Note that $\mathcal{A}^t \cup \mathcal{I}^t = \mathbf{M}$. Additionally, at each iteration $t \geq 0$, the inactive constraints which are not violated, i.e, which belong to $\mathcal{I}^t \cap \mathrm{CI}^t$, have a corresponding null lagrangian multiplier in the current and the next iteration. Again, following the suggestion of BEASLEY [99], the relax-and-cut algorithm projects the subgradient vector into a smaller dimension by only considering those

indices which lead to an actual change in the corresponding lagrangian multiplier. Let g^t denote the projected subgradient at iteration t. Therefore, at each iteration $t \geq 0$, the projected subgradient vector g^t is such that:

$$g_i^t = \begin{cases} 0 \text{ if } \lambda_i^t = 0 \text{ and } b_i - a_i \cdot \bar{y}^t \leq 0 \\ b_i - a_i \cdot \bar{y}^t \text{ otherwise} \end{cases}$$

for each active constraint $i \in \mathcal{A}^t$, and $g_i^t = 0$ for each inactive constraint $i \in \mathcal{I}^t$. Consequently, the projected subgradient vector only contains non-zero entries corresponding to the active constraints that will have a non-zero update on their lagrangian multiplier. In comparison with the subgradient vector, the projected subgradient possibly has much fewer non-zero entries. As a consequence, $||g^t||$ is much smaller than $||\partial f(\lambda^t)||$, and thus the step size given by $\alpha_t \frac{\overline{f} - f(\lambda^t)}{||g^t||^2}$ is much larger than the one given by $\alpha_t \frac{\overline{f} - f(\lambda^t)}{||\partial f(\lambda^t)||^2}$. Accordingly, the projection of the subgradient vector is providential in avoiding virtually zero step sizes and consequently in the successful convergence of the relax-and-cut. As reported by LUCENA [97], the relax-and-cut algorithm shows good convergence to near-optimal lagrangian multipliers. The identification of violated constraints is done by the so-called *separation procedure*. In general, given a tentative primal solution, the separation procedure identifies constraints which are violated by that solution. In the particular case of the non-delayed relax-and-cut, at each iteration, the separation procedure is applied to the current lagrangian solution $\bar{y}^t$ and CV^t is identified. In practice, it is customary for the relax-and-cut algorithm to start with a non-empty set of active constraints, and during its execution, as violated constraints are identified by the separation procedure, the amount of active constraints is non-decreasing. Even if the problem contains an exponential amount of constraints m, by starting with a number of active constraints which is a polynomial function of n, the amount of constraints which are activated tends to be a polynomial function of n and thus much smaller than m. This ensures that the relax-and-cut algorithm is not only viable but also efficient, as reported, for example, by DA CUNHA *et al.* [33] for the prize-collecting steiner tree problem, by CAVALCANTE *et al.* [105] for the set partitioning problem, by KLIEWER and TIMAJEV [106] for a network design problem and by KLOSE [32] for the capacitated facility location problem.

5.3 Lagrangian Heuristic

For practical purposes, when solving an optimization problem, one is usually interested in obtaining a primal as well as a dual bound. As discussed in Chapter 4 and in Section 5.1, a dual bound can be obtained by solving the lagrangian dual problem

with the subgradient method. In turn, a primal bound can be obtained from a primal feasible solution. However, for optimization problems in general, and specially for integer and combinatorial optimization problems, there might be many distinct feasible solutions with near or equal objective values. Moreover, the difference between an optimal and a suboptimal solution can be very slight. For these reasons, in order to find a feasible solution of good or optimal quality, it is usually necessary to explore a diverse set of feasible solutions. Heuristic algorithms are a standard way of constructing feasible solutions for an optimization problem. In general, a heuristic algorithm is a polynomial time algorithm that produces a feasible solution but with no guarantee of optimality. In particular, a *greedy algorithm* builds a primal solution by starting from an empty solution and by growing it into a feasible solution by iteratively making a local-optimal choice at each iteration. The choice at each iteration is made based on a *score function*, which assigns an estimated value for each possible choice, and the choice at each iteration is said to be *greedy* because it chooses the one which optimizes the score function. In the case of SCP, a greedy algorithm was originally proposed by CHVATAL [7] in 1979 and subsequently many improvements have been incorporated. The common guideline is to embody a diversifying strategy to the original greedy algorithm so that a wide variety of feasible solutions are explored. Lagrangian relaxation based heuristics incorporate the information provided by the lagrangian multipliers into the heuristic as part of the score function. Combined with the subgradient method, a lagrangian heuristic explores a diverse set of feasible solutions by applying the lagrangian greedy heuristic at different iterations of the subgradient method. Since the lagrangian multipliers are modified at each subgradient iteration, this produces many distinct solutions, and the one with best corresponding objective value is stored. A lagrangian heuristic was originally applied to SCP by BALAS and HO [43] in 1980 and, in fact, some of the presently most effective heuristics for SCP as far as solution quality is concerned are probably the lagrangian based heuristics of CAPRARA *et al.* [51] and of CERIA *et al.* [9], as well as the local-search procedure of YAGIURA *et al.* [10].

We now describe the lagrangian greedy heuristic for the set cover problem with exponentially many constraints. Similarly to what is done in a relax-and-cut algorithm, as described in Section 5.2, the usual greedy procedure is adapted to allow the dynamic inclusion of set cover constraints. Assume that the set cover problem possibly contains exponentially many constraints. Again, storing and handling values associated with an exponential number of constraints would require an overwhelming computational effort. Thus, in order to deal with the exceedingly large number of constraints, the adaptation is to consider that only a subset of constraints are *active*, while the remaining constraints are *inactive*. At any iteration of the lagrangian heuristic, let $\mathcal{A}$ denote the set of active constraints and let $\mathcal{I}$ denote the

set of inactive constraints. Each active constraint has an effective contribution to the score function and thus effectively takes part at the current step and in the greedy choice. In contrary, inactive constraints have their lagrangian multipliers arbitrarily set to zero, and thus have no contribution to the lagrangian score function and do not take part in the greedy choice. However, in order for the heuristic to produce a feasible solution, this solution must satisfy all of the exponentially many constraints. Therefore, inactive constraints must be activated if they are violated by the current solution. An inactive constraint is actived if at some iteration the *separation procedure* identifies its violation by the current solution. From that point on, it gets actived and from that point on is taken into consideration in the greedy choice. When a constraint i is activated, if a non-zero lagrangian multiplier was initially associated with it, then its value, previously set to zero, is adjusted accordingly. On the other hand, if a value for the lagrangian multiplier was not given, then its value is set to zero upon activation. This type of situation might happen if the lagrangian heuristic is inbuilt to a relax-and-cut procedure and at some iteration the heuristic solution violates a constraint which is inactive for the relax-and-cut. In this case, since the constraint is inactive, then its current lagrangian multiplier is zero. We recall that the lagrangian heuristic here proposed is specifically designed for use during a relax-and-cut algorithm. Additionally, we suggest that if at iteration $t \geq 0$ of the relax-and-cut algorithm the lagrangian heuristic is to be called, then one should set $\mathcal{A} = \mathcal{A}^t$. This means that $\mathcal{A}$, the set of active constraints for the heuristic, should correspond to $\mathcal{A}^t$, the set of currently active constraints for the relax-and-cut.

Let $\lambda \in \mathbb{R}^m_{\geq 0}$ be a vector of lagrangian multipliers. For the sake of notational convenience:

- Let $y \in \{0,1\}^n$ be the current partial SCP solution and let $C^y = \sum_{j \in \mathbf{N}} c_j y_j$ be its corresponding objective value.
- For each $i \in \mathcal{A}$, let $\gamma_i = \sum_{j \in J_i} y_j$ denote the number of times that constraint i is covered by the current solution.
- For each $j \in \mathbf{N}$ and $k \in \{0,1\}$, let $\theta^k_j = \{i \in I_j \cap \mathcal{A} : \gamma_i = k\}$ be the subset of active constraints in I_j which are covered exactly k times by the current solution.
- For each $j \in \mathbf{N}$, let $\Lambda_j = \sum_{i \in \theta^0_j} \lambda_i$.
- For each $j \in \mathbf{N}$, let $\hat{c}_j = c_j - \Lambda_j$ be the *lagrangian reduced cost* of j.
- For each $j \in \mathbf{N}$, let σ_j be the greedy score of j.

- Let $\theta^0 = \{i \in \mathcal{A} : \gamma_i = 0\}$ be the set of active constraints which are not covered by the current solution.

Note that the values of γ_i and λ_i are actually stored in memory and considered for computation only if i is active. Again, because of the exceedingly large number of constraints, this is made to ensure that the lagrangian heuristic is not only computationally viable but also efficient.

The greedy algorithm starts with a null solution $y = 0$ and at each iteration selects the column $j^* \in \mathbf{N}$ which minimizes the greedy score to be inserted into the current solution. The lagrangian heuristic has three phases. The first phase is dedicated to building a tentative SCP solution. During the first phase, the set of active constraints remains the same, and the solution is incremented until all of the initially active constraints are satisfied. The goal of the second phase is to ensure that the solution satisfies all of the exponentially many constraints. During the second phase, at each iteration a *separation procedure* is applied to identify the constraints violated by the current solution. If at least one such constraint is found, then these violated constraints are activated and a column is chosen to be inserted into the current solution. If no such constraints are found, then the current solution covers all of the exponentially many constraints and hence is feasible. At this point the second phase ends. The third phase is dedicated to *pruning* the solution. Given a feasible solution y' and $j \in \mathbf{N}$ such that $y'_j = 1$, column j is *redundant* if $\sum_{j' \in J_i} y'_{j'} \geq 2$, for each $i \in I_j$. Hence a column j is not redundant if and only if there is $i \in I_j$ such that $y'_j = 1$ and $y'_{j'} = 0$ for every $j' \in J_i \setminus j$. In that case, we say that i is uniquely covered by j. A redundant column does not uniquely cover any constraint. Consequently, a redundant column can be removed without compromising the feasibility of the solution. Additionally, a feasible solution y' is *minimal* if it has no redundant columns. Since the column costs are positive, it is clear that an optimal solution must be minimal. The pruning phase is dedicated to removing redundant columns from the heuristic solution as to make it minimal. At each iteration, let $\mathcal{R} = \{j \in \mathbf{N} : y_j = 1 \wedge \theta^1_j = \emptyset\}$ be the set of currently redundant columns. If $\mathcal{R} \neq \emptyset$ holds, then the redundant column $j^- \in \mathcal{R}$ with greatest cost is chosen to be removed from the current solution. After its removal, the separation procedure is applied to verify whether the pruned solution is feasible. If no constraints are violated, then the pruned solution is feasible and the pruning procedure continues to another iteration. Otherwise, the pruned solution is infeasible. In that case, the violated constraints are activated and j^- is inserted back into the solution. Note that the activation of the violated constraints ensures that j^- is no longer redundant in the next iteration and thus will not be a candidate for pruning again. The pruning phase ends when there are no redundant columns in the current solution. Note that values such as $C^y, \gamma, \theta^k, \hat{c}$ need not be recomputed

at every iteration. These values need only be initialized at the start and be updated with each insertion, removal or activation. More specifically, if column j is inserted to or removed from the solution, these values are updated for the active constraints in I_j and for the columns in N_j. The insertion and removal procedures are described in Algorithms 4 and 5, respectively. Similarly, whenever a constraint i is activated, values associated with i and with the columns in J_i are updated. The procedure for activating a constraint is described in Algorithm 6. The first, second and third phases are described in Algorithms 7, 8 and 9, respectively. Finally, the lagrangian heuristic for the set cover problem with exponentially many constraints is described in Algorithm 10.

Different greedy scores have been proposed in the literature. It was originally proposed by FISHER [42] in 1981 to consider $\sigma_j = \hat{c}_j$, and later considered by CERIA *et al.* [9], for example. Alternatively, the following score function was proposed by CAPRARA *et al.* [51] in 1999:

$$\sigma_j = \begin{cases} \frac{\hat{c}_j}{|\theta_j^0|} \text{ if } \hat{c}_j \leq 0 \\ \hat{c}_j |\theta_j^0| \text{ if } \hat{c}_j > 0 \end{cases}$$

It was reported by CAPRARA *et al.* [51] that this score function proved to be more effective than the one of FISHER [42].

In the present work, we further propose the following score function:

$$\sigma_j = \frac{\sum_{i \in \theta_j^0} (1 + \lambda_i)}{c_j} = \frac{|\theta_j^0| + \Lambda_j}{c_j}$$

According to our computational experiments, this score function performs slightly better than the one in 51 when (SCLagDual+) is considered. On the other hand, these two score functions perform very similarly when (SCLagDual) is considered.

The intuition that guides these score functions is the following. Suppose that for each constraint $i \in \mathbf{M}$, λ_i is a measure of priority that is placed on i, such that constraints with higher lagrangian multipliers are placed a higher priority than those with smaller lagrangian multipliers. In this way, for each $j \in \mathbf{N}$, Λ_j measures the sum of priorities placed on the constraints that would be covered with the insertion of j into the current solution. Note that the insertion of a column j with a relatively higher value of Λ_j into the current solution means that j will cover a set of constraints with relatively higher priority. However, because of the costs, when inserting a column into the solution, there is a tradeoff between covering constraints and increasing the corresponding objective value. Note that $\hat{c}_j$ is the difference between the cost of a column and the sum of priorities of the constraints that it covers. Furthermore, $|\theta_j^0|$ is the number of constraints that will be covered with the

insertion of j to the current solution. At least intuitively, a column with relatively higher Λ_j and $|\theta_j^0|$ seems to be a better choice for insertion, since its insertion would cover a set of constraints with larger cardinality and which is placed higher priority. All in all, these score functions attempt to estimate the value of inserting a column into the current solution by considering the tradeoff between its "gains" from covering constraints and the "loss" incurred by adding its cost to the corresponding objective value.

Algorithm 4 Insert column j into an SCP solution

```
function INSERT COLUMN(j)
    y_j ← 1
    C^y ← C^y + c_j
    for i ∈ I_j ∩ A do
        if γ_i = 0 then
            θ^0 ← θ^0 \ {i}
            for j' ∈ J_i do
                θ^0_{j'} ← θ^0_{j'} \ {i}
                θ^1_{j'} ← θ^1_{j'} ∪ {i}
                Λ_{j'} ← Λ_{j'} − λ_i
                ĉ_{j'} ← ĉ_{j'} + λ_i
        else if γ_i = 1 then
            for j' ∈ J_i do
                θ^1_{j'} ← θ^1_{j'} \ {i}
        γ_i ← γ_i + 1
```

Algorithm 5 Remove column j from an SCP solution

function REMOVE COLUMN(j)
- $y_j \leftarrow 0$
- $C^y \leftarrow C^y - c_j$
- **for** $i \in I_j \cap \mathcal{A}$ **do**
  - **if** $\gamma_i = 1$ **then**
    - $\theta^0 \leftarrow \theta^0 \cup \{i\}$
    - **for** $j' \in J_i$ **do**
      - $\theta^0_{j'} \leftarrow \theta^0_{j'} \cup \{i\}$
      - $\theta^1_{j'} \leftarrow \theta^1_{j'} \setminus \{i\}$
      - $\Lambda_{j'} \leftarrow \Lambda_{j'} + \lambda_i$
      - $\hat{c}_{j'} \leftarrow \hat{c}_{j'} - \lambda_i$
  - **else if** $\gamma_i = 2$ **then**
    - **for** $j' \in J_i$ **do**
      - $\theta^1_{j'} \leftarrow \theta^1_{j'} \cup \{i\}$
  - $\gamma_i \leftarrow \gamma_i - 1$

Algorithm 6 Activate a violated constraint during the SCP lagrangian heuristic

function ACTIVATE CONSTRAINT(i)
- $\mathcal{A} \leftarrow \mathcal{A} \cup \{i\}$
- $\mathcal{I} \leftarrow \mathcal{A} \setminus \{i\}$
- $\theta^0 \leftarrow \theta^0 \cup \{i\}$
- $\lambda_i \leftarrow 0$
- $\gamma_i \leftarrow 0$
- **for** $j \in J_i$ **do**
  - $\theta^0_j \leftarrow \theta^0_j \cup \{i\}$

Algorithm 7 Build a tentative SCP solution

function BUILD SOLUTION(λ, θ^0)
- **while** $\theta^0 \neq \emptyset$ **do**
  - $j^* \leftarrow \arg\min\{\sigma_j : j \in \mathbf{N}\ ,\ y_j = 0\}$
  - INSERT COLUMN(j^*)

Algorithm 8 Ensure an SCP solution satisfies all the exponentially many constraints

```
function SATISFY INACTIVE CONSTRAINTS(y)
    V ← SEPARATION PROCEDURE(y)
    while V ≠ ∅ do
        for i ∈ V do
            ACTIVATE CONSTRAINT(i)
        j* ← arg min{σ_j : j ∈ N , y_j = 0}
        INSERT COLUMN(j*)
        V ← SEPARATION PROCEDURE(y)
```

Algorithm 9 Prune an SCP solution

```
function PRUNE SOLUTION(y)
    while R ≠ ∅ do
        j* ← arg max{c_j : j ∈ R}
        REMOVE COLUMN(j*)
        V ← SEPARATION PROCEDURE(y)
        if V ≠ ∅ then
            for i ∈ V do
                ACTIVATE CONSTRAINT(i)
            INSERT COLUMN(j*)
```

Algorithm 10 Lagrangian heuristic for SCP with exponentially many constraints

```
function LAGRANGIAN GREEDY HEURISTIC(λ, θ^0)
    y ← BUILD SOLUTION(λ, θ^0)
    SATISFY INACTIVE CONSTRAINTS(y)
    PRUNE SOLUTION(y)
    return y, C^y
```

5.4 Non-delayed relax-and-cut for SCP

We now describe in detail the proposed novel non-delayed relax-and-cut algorithm for the set cover problems with exponentially many constraints. Recall that in Chapter 4 two different lagrangian dual problems for SCP were described.

The *standard lagrangian dual* problem for SCP is defined as:

$$W_{SC} = \begin{array}{ll} \max & Z_{SC}(\lambda) \\ \text{s.t} & \lambda \in \mathbb{R}^m_{\geq 0} \end{array} \qquad \text{(SCLagDual)}$$

where

$$\begin{aligned} Z_{SC}(\lambda) = \quad & \min \quad \sum_{i \in \mathbf{M}} \lambda_i + \sum_{j \in \mathbf{N}} c_j^{\lambda} y_j \\ & \text{s.t.} \quad y \in \{0,1\}^n \end{aligned} \tag{SCLagSp}$$

In turn, the *strengthened lagrangian dual* problem for SCP is defined as:

$$\begin{aligned} W_{SC}^{+} = \quad & \max \quad Z_{SC}^{+}(\lambda) \\ & \text{s.t} \quad \lambda \in \mathbb{R}_{\geq 0}^{m} \end{aligned} \tag{SCLagDual+}$$

where

$$\begin{aligned} Z_{SC}^{+}(\lambda) = \quad & \min \quad \sum_{i \in \mathbf{M}} \lambda_i + \sum_{j \in \mathbf{N}} c_j^{\lambda} y_j \\ & \text{s.t.} \quad \sum_{j \in \mathbf{N}} c_j y_j \quad \geq \underline{Z} \\ & \qquad \sum_{j \in \mathbf{N}} c_j y_j \quad \leq \overline{Z} \\ & \qquad y \in [0,1]^n \end{aligned} \tag{SCLagSp+}$$

Note that regardless of the choice between (SCLagDual) or (SCLagDual+), a subgradient vector at a point λ for both $Z_{SC}(\lambda)$ and $Z_{SC}^{+}(\lambda)$ is given by the vector g such that:

$$g_i = 1 - \sum_{j \in J_i} \bar{y}_j$$

for each $i \in \mathbf{M}$, and where $\bar{y} = y^*(\lambda)$ for $Z_{SC}(\lambda)$ and $\bar{y} = y^*(\lambda, \underline{Z}, \overline{Z})$ for $Z_{SC}^{+}(\lambda)$.

In accordance with the subgradient method described in Section 5.1 and the relax-and-cut procedure described in Section 5.2, at each iteration $t \geq 0$ of the non-delayed relax-and-cut algorithm for SCP: let λ^t denote the current lagrangian multipliers; let $\bar{y}^t$ denote the solution of the current lagrangian subproblem; let g^t denote the current projected subgradient; let $\overline{Z}$ denote the best primal bound available so far; and let $\underline{Z}$ denote the best dual bound available so far. The relax-and-cut procedure starts with a non-empty set of active constraints $\mathcal{A}^0$ and a set of inactive constraints $\mathcal{I}^0$. Following the suggestion of CAPRARA *et al.* [51], the lagrangian multipliers are initialized in the following way:

$$\lambda_i^0 = \min \left\{ \frac{c_j}{|I_j \cap \mathcal{A}^0|} : j \in J_i \right\}$$

for each active constraint $i \in \mathcal{A}^0$, whereas $\lambda_i^0 = 0$ for each inactive constraint $i \in \mathcal{I}^0$. Recall that in practice the lagrangian multipliers and corresponding subgradient are only stored for active constraints. On the other hand, inactive constraints are handled implicitly until their activation. Once the multipliers are initialized, the

lagrangian heuristic as described in Section 5.3 is applied to λ^0 in order to determine an initial primal bound $\overline{Z}$. Although this first application of the heuristic likely produces a sub-optimal solution, it is important to initialize $\overline{Z}$ with a reasonably tight value as it prevents the first subgradient step size from being too large. In turn, $\underline{Z}$ is initialized with value zero. At this point the subgradient phase starts. At each iteration, the solution of the current lagrangian subproblem is computed, as well as its corresponding projected subgradient g^t. The projected subgradient g^t is determined in the following way:

$$g_i^t = \begin{cases} 0 \text{ if } \lambda_i^t = 0 \text{ and } 1 - \sum_{j \in J_i} \bar{y}_j^t \leq 0 \\ 1 - \sum_{j \in J_i} \bar{y}_j^t \text{ otherwise} \end{cases}$$

for each active constraint $i \in \mathcal{A}^t$, whereas $g_i^t = 0$ for each inactive constraint $i \in \mathcal{I}^t$. Additionally, the step size ϵ_t is given by

$$\epsilon_t = \alpha_t \frac{\overline{Z} - \underline{Z}^t}{||g^t||^2}$$

Thus each active lagrangian multiplier is updated by a subgradient step in the following way:

$$\lambda_i^{t+1} = \max\left\{0, \lambda_i^t + \epsilon_t g_i^t\right\}$$

While the inactive lagrangian multipliers are not updated and have their value set to zero.

Recall from Section 5.2 that the relax-and-cut algorithm relies on a separation procedure in order to identify constraints violated by $\bar{y}^t$. For the sake of efficiency, the separation procedure is most usually applied over integral solutions, since this is generally faster than separating from fractional solutions. However, in the present work, we propose an alternative to applying the separation procedure on the current lagrangian subproblem solution, as is customary in relax-and-cut algorithms [97]. Instead, we propose that the constraints to be activated at each iteration of the relax-and-cut should correspond to the constraints activated during the lagrangian heuristic. Suppose that at some iteration $t \geq 0$ of the relax-and-cut, the lagrangian heuristic is employed and takes as initial set of active constraints $\mathcal{A}^t$. According to Section 5.3, the first phase of the heuristic produces a tentative solution which covers all the constraints in $\mathcal{A}^t$. Moreover, during the second and third phases, there is a possibly non-empty set $\mathcal{V} \subseteq \mathbf{M} \setminus \mathcal{A}^t$ of constraints activated during these phases. In this work we propose to set CV^t as $\mathcal{V}$. In doing so, the separation procedure is applied to tentative solutions given by the lagrangian heuristic, in alternative to being

applied to the lagrangian subproblem solution. We argue that this leads to general improvements in the relax-and-cut, specially in the primal and dual bound as well as its convergence. First of all, some of our experiments with the SCP benchmark instances indicated that the lagrangian subproblem solution is a poor estimate of an optimal or even a primal feasible solution. The same was reported by CERIA *et al.* [9] in their lagrangian relaxation approach for SCP. Consequently, the constraints activated during the relax-and-cut add little information to the lagrangian subproblem and heuristic. In particular, because the lagrangian subproblem solution is very different from optimal and near-optimal primal solutions, the activated constraints likely are not violated by tentative solutions in subsequent applications of the lagrangian heuristic. Hence these activated constraints have limited effect in forcing the heuristic to explore different solutions and therefore the diversification strategy is compromised. Alternatively, by activating the constraints violated by the previous tentative solution, subsequent applications of the heuristic are guaranteed to not violate these constraints again, and hence subsequent tentative solutions are more likely to be different from previous ones. Consequently a more diverse set of primal feasible solutions is explored and more likely a better solution is found. As a result, there is a significant improvement in the quality of the primal bound. Accordingly, because of the reduction in the optimality gap $\overline{Z} - \underline{Z}$, the step sizes are more appropriate and overall the convergence improves. More specifically, not only this alternative leads to an improvement in the primal and dual bound at termination, but the amount of subgradient steps required to find a near-optimal lagrangian multiplier is reduced and the algorithm is computationally faster. Furthermore, considering that the application of the lagrangian heuristic is already customary part of the relax-and-cut, this alternative does not lead a significant increase in time or complexity. Additionally, it ensures that the time and effort spent in the separation procedure is more valuable. Finally, note that when (SCLagSp+) is considered, the lagrangian subproblem solution is not guaranteed to be integral. Since the separation procedure is faster for integral solutions, this is another reason to apply it to tentative feasible solutions instead of the lagrangian subproblem solution.

As a final remark, note that the frequency at which the lagrangian heuristic is employed during the relax-and-cut has a direct impact on its running time. On the one hand, applying the heuristic more frequently might lead to a better primal bound at termination, but on the other hand it tends to increase the convergence time. According to our experiments, the appropriate frequency depends mostly on the difficulty of each instance. In fact, the most difficult instances usually require the exploration of a large number of primal solutions in order to find one of acceptable quality. For these hard instances, we suggest that the frequency should be higher, regardless of the running time increase it implies. We propose a simple rule

for determining the iterations one should call the heuristic. At each relax-and-cut iteration, the heuristic is called if at least one of the following two conditions are met. First, if the best dual bound so far attained is improved. In this case, a better lagrangian multiplier vector has been found and the heuristic should be called with this new and better lagrangian information. Secondly, given a predetermined parameter $p \in [0, 1]$, a Bernoulli trial with probability p of success is made. The reason for this randomized condition is to allow the exploration of a broader region of lagrangian multipliers and hence contribute to the diversification strategy. Note that higher values of p lead to more frequent application of the lagrangian heuristic.

The relax-and-cut algorithm for the set cover problem with exponentially many constraints is described in Algorithm 11.

Algorithm 11 Relax-and-cut for SCP with exponentially many constraints

function SCP RELAX AND CUT
- **for** $i \in \mathcal{A}^t$ **do** $\lambda_i^0 \leftarrow \min\left\{\frac{c_j}{|I_j \cap \mathcal{A}^0|} : j \in J_i\right\}$
- $\overline{Z} \leftarrow$ LAGRANGIAN HEURISTIC$(\lambda^0, \mathcal{A}^0)$
- $\underline{Z} \leftarrow 0$, $\alpha_0 \leftarrow 2$
- **while** $\alpha_t > \tilde{\alpha}$ **do**
  - $\bar{y}^t \leftarrow \arg\min Z(\lambda^t), \underline{Z}^t \leftarrow C^\lambda(\bar{y}^t)$
  - **if** $\underline{Z}^t > \underline{Z}$ **then**
    - $\underline{Z} \leftarrow \underline{Z}^t$, $\lambda^* \leftarrow \lambda^t$, $\eta \leftarrow 0$
  - **else**
    - $\eta \leftarrow \eta + 1$
    - **if** $\eta > \tilde{\eta}$ **then**
      - $\alpha_t \leftarrow \frac{\alpha_t}{2}$, $\eta \leftarrow 0$
  - **if** heuristic should be called **then**
    - $y^t, \overline{Z}^t, \mathcal{V} \leftarrow$ LAGRANGIAN HEURISTIC$(\lambda^t, \mathcal{A}^t)$
    - **if** $\overline{Z}^t < \overline{Z}$ **then** $y^* \leftarrow y^t$, $\overline{Z} \leftarrow \overline{Z}^t$
  - **for** $i \in \mathcal{A}^t$ **do**
    - **if** $\lambda_i^t = 0$ and $1 - \sum_{j \in J_i} \bar{y}_j^t \leq 0$ **then**
      - $g_i^t \leftarrow 0$
    - **else**
      - $g_i^t \leftarrow 1 - \sum_{j \in J_i} \bar{y}_j^t$
  - $\epsilon_t \leftarrow \alpha_t \frac{\overline{Z} - \underline{Z}^t}{||g^t||^2}$
  - **for** $i \in \mathcal{A}^t$ **do** $\lambda_i^{t+1} \leftarrow \max\left\{0, \lambda_i^t + \epsilon_t g_i^t\right\}$
  - $\alpha_{t+1} \leftarrow \alpha_t$
  - $t \leftarrow t + 1$
- **return** $\underline{Z}, \overline{Z}, \lambda^*, y^*$

Chapter 6

Local Search

Due to their combinatorial nature, integer and combinatorial optimization problems often have a large number of feasible solutions. Additionally, the difference between an optimal and a sub-optimal solution can be very slight. As a result, in practice, finding an optimal solution usually requires the exploration of a diverse set of feasible ones. As discussed in Section 5.3, a common approach to achieve diversification is to incorporate randomness into a heuristic algorithm and apply it systematically. Every run of the heuristic is initialized with the null solution which is then enlarged until it becomes feasible. Due to its use of randomness, most likely distinct solutions are generated. Even though this approach has been successfully applied quite frequently, it presents some limitations. First of all, depending on the complexity of the heuristic, applying it multiple times might be too time-consuming. Secondly, in spite of resorting to the use of randomness, the heuristic might still fail to produce a diverse enough set of feasible solutions. In alternative to building solutions from scratch, different methods have been proposed which iteratively modify a feasible or not yet feasible partial solution in search for better one. In this process, every operation over a given solution (resp. expanding solution) is restricted to be local and should result in a close *neighbor* to what is currently in hand. By doing so, various tentative solutions, in a corresponding predefined solution neighborhood, are allowed to be efficiently explored. Ideally, these *local search* (LS) procedures must also include mechanisms to prevent the repetition of previously visited solutions, hence favoring diversification. Under these guidelines, LS procedures manage to explore a wider area of the feasible region and in many applications are more successful in finding better feasible solutions than the alternatives available. Examples of them include local-search, tabu search, variable-neighborhood search and simulated annealing. In summary, the guiding principle of LS is to modify the current (possibly partial) solution by moving to another solution in its *neighborhood*. Given $y \in \{0,1\}^n$, the *neighborhood* of y is a suitably defined set $N(y) \subseteq \{0,1\}^n$. The specific definition of $N(y)$ depends on the particular optimization problem one

is attempting to solve. Moreover, alternative descriptions might be available for the same problem, leading to different LS algorithms. Nevertheless, in general, the neighborhood is defined so that the difference between y and any of its neighbors is slight. At every iteration, given a partial or feasible solution y, LS attempts to modify y to y', where y' is an appropriately chosen neighbor $y' \in N(y)$. This is called a *neighborhood move* (NM). In particular, $N(y)$ is established so that it is computationally easy to go from y to any $y' \in N(y)$, i.e., to carry out a NM. This enables LS to explore many distinct solutions in an acceptable amount of time. The particular choice of NM takes into account the constraints and objective value function one is subject to and is established with the aim of finding a better feasible solution in subsequent iterations. By iteratively moving within neighborhoods, LS ends up exploring a diverse set of tentative solutions. Whenever better solutions are obtained in a given NM round, y and $N(y)$ should be updated with the best of them. In the case of SCP, different LS algorithms have been investigated, such as the ones reported by YAGIURA *et al.* [10] and by GAO *et al.* [34] for general instances of the problems, as well as those of MUSLIU [54], of NAJI-AZIMI *et al.* [55] and of GAO *et al.* [11] for the unit-cost case. Among the SCP benchmark instances in the literature, the *RAIL* set was originally introduced in the year of 1994 and comprise some very-large scale instances which arise from train-scheduling problems for the Italian railway company, *Ferrovie dello Stato Italiane S.p.A*. At least initially, the two most successful heuristic algorithms for SCP, and specially for the RAIL instances, were the lagrangian-relaxation based heuristics reported by CAPRARA *et al.* [8] in 1996 and by CERIA *et al.* [9] in 1998. However, in subsequent years, LS procedures for SCP were investigated and have shown improvements both in terms of running time as well as solution quality, when compared to these lagrangian approaches. In this respect, it is worth mentioning the *3-flip local-search* of YAGIURA *et al.* [10] and to the *row-weighting based local-search* originally proposed by GAO *et al.* [34] for the general case, and later adapted to the unit-cost case by GAO *et al.* [11]. In fact, compared to the results reported in 9 and 8, these two LS algorithms further improved solutions for a couple of test sets, including the RAIL one.

In the present work, we propose a novel local-search algorithm for SCP with exponentially many constraints. Our LS is based on the row-weighting scheme described in 34 and 11. Additionally, we propose three adaptations to them. One allows us to handle the exceedingly large number of constraints; Another one is an alternative criteria for breaking ties in the choice of NM; Finally, the remaining one comprises the removal of variables from the solution as they become redundant.

6.1 Our proposed local search for SCP

We now describe our proposed LS algorithm in detail. Suppose that a feasible solution y is known. The goal of LS is to iteratively modify y until another feasible solution y' such that $C(y') < C(y)$ is found. Naturally, since $C(y') < C(y)$ must hold, then at least one variable must be removed from y to obtain such y'. On the other hand, note that if y is minimal, then the removal of any variable renders some constraints uncovered. Therefore, in order to ensure feasibility once again, at least one variable must be inserted to cover these constraints.

At any LS iteration, let $y \in \{0,1\}^n$ be the current solution and let $C^y = \sum_{j \in \mathbf{N}} c_j y_j$ be its corresponding objective value. Additionally, let $y^* \in \{0,1\}^n$ denote the best feasible solution at our disposal and let $C^* = \sum_{j \in \mathbf{N}} c_j y^* j$ be its corresponding objective value. LS begins with a feasible solution. Whenever y is feasible, LS initializes a NM by picking a neighboring infeasible solution obtained by removing columns from y. Moving to an infeasible neighbor is enforced so that one is taken to a different region of the feasible region, favoring diversification. This kind of NM move is carried out as described next. First, if y is not minimal, then its redundant columns are removed. Once y is minimal, exactly one column is removed from it and the resulting solution, let us keep calling it y, is infeasible. In contrary, if y is infeasible, then each NM consists of both removing and inserting columns to y while maintaining $C^y < C(y^*)$. If at some iteration y happens to be feasible, since $C^y < C(y^*)$ holds, then a better feasible solution has been found and y^* is updated. Procceding with LS from this point, a NM is applied to $y = y^*$, thus making the resulting, updated, y infeasible. Accordingly, one then attempts to obtain a feasible out of it, as previously described. Finally, LS is eventually stopped, when a predefined *stopping criterion* is reached. Some possibilities for such criteria are: specifying an overall time limit for LS runs; imposing a time limit to improving the best feasible solution y^* currently in hand; imposing a limit of iterations without any improvement to y^*; some combination of the previous criteria.

Given an infeasible $y \in \{0,1\}^n$, the the following definition of neighborhood is suggested in 34 and 11:

$$N(y) = \{y' \in \{0,1\}^n : C(y') < C^* \text{ and } \sum_{j=1}^{n} \max\{0, y_j - y'_j\} = 1\}$$

This means that at each iteration, if y is infeasible, then LS picks a neighboring solution $y' \in N(y)$ such that: $C(y') < C^*$ holds; there is a unique j^- such that $y_{j^-} = 1$ and $y'_{j^-} = 0$; and there are $j_1^+, \ldots, j_k^+$ such that $y_{j_\ell^+} = 0$ and $y'_{j_\ell^+} = 1$, for each $\ell \in \{1, \ldots, k\}$. In conclusion, each NM consists of removing exactly one column and inserting exactly k columns to the current solution. Note that in the

case of unit-cost SCP, since $C(y') < C(y^*)$ must hold, then $k = 1$, meaning that exactly one variable is inserted at each NM. Although each NM amounts to choosing one column for removal and one or more columns for insertion, as suggested in 34 and 11, in practice each NM is performed as a series of individual decisions. First, a column j^- is chosen for removal. This is followed by consecutive choices of columns for insertion. More specifically, while $C^y < C(y^*)$ holds, one column j^+ is chosen for insertion. The process stops when $C^* + c_{j^+} \geq C^*$ holds, thus concluding the NM. Each individual choice is made according to *score functions* which are based on the *row-weighting scheme* described in 34. For each $i \in \mathbf{M}$, a value $w_i \in \mathbb{Z}_{\geq 0}$ is associated with constraint i and is called its *weight*. At each LS iteration, the weights are updated so as to push the search to yet unexplored areas of the feasible region and diversify it. At this point, we introduce our first refinement to the LS described in [34] and [11]. In order to deal with the exceedingly large number of constraints, our LS considers a set $\mathcal{A} \subseteq \mathbf{M}$ of *active* constraints and a set $\mathcal{I} \subseteq \mathbf{M}$ of *inactive* constraints such that $\mathcal{A} \cup \mathcal{I} = \mathbf{M}$. The active constraints are handled *explicitly*, whereas the inactive constraints are handled *implicitly*. For each active constraint $i \in \mathcal{A}$, the weight w_i is *active*, meaning that w_i is non-zero, updated at each iteration and takes effective part in NM choice one makes. In contrary, for each inactive constraint $i \in \mathcal{I}$, the weight w_i is *inactive*, meaning that w_i is set to zero, not updated and takes no part in the NM choice. In practice, only the active constraints and weights are stored in memory and used for computation. This adaptation ensures that considerably less computational effort is required to perform each LS iteration and enables the procedure to be computationally viable.

For the sake of notational convenience, at any LS iteration:

- For each $i \in \mathcal{A}$, let $\gamma_i = \sum_{j \in J_i} y_j$ denote the number of times that constraint i is covered by y.
- For each $j \in \mathbf{N}$ and $k \in \{0, 1, 2\}$, let $\theta_j^k = \{i \in I_j \cap \mathcal{A} : \gamma_i = k\}$ be the subset of active constraints in I_j which are covered exactly k times by y.
- For each $j \in \mathbf{N}$ and $k \in \{0, 1, 2\}$, let $W_j^k = \sum_{i \in \theta_j^k} w_i$.
- Let $\theta^0 = \{i \in \mathcal{A} : \gamma_i = 0\}$ be the set of active constraints which are not covered by y.

Note that y is feasible if it covers all constraints in $\mathbf{M}$. In particular, y must cover all of the active constraints in $\mathcal{A}$. Therefore, with the goal of producing a feasible solution, LS prioritizes NMs aimed at covering all constraints that are currently active. If at some iteration $\theta^0 = \emptyset$ holds, then y covers $\mathcal{A}$. At this point, a *separation procedure* is applied in order to identify the set $\mathcal{V} \subseteq \mathcal{I}$ of inactive

constraints violated by y. If $\mathcal{V} = \emptyset$, then y is feasible. Accordingly, the best solution currently available is updated. In contrary, if $\mathcal{V} \neq \emptyset$, then y is infeasible. In that case, the violated constraints are activated.

Given an initial set of active and inactive constraints, w_i is initially set to 1 for each $i \in \mathcal{A}$ and set to 0 for each $i \in \mathcal{I}$. Additionally, whenever a constraint is activated, its weight is initialized with value 1. According to the row-weighting scheme, at each iteration, w_i is incremented by 1 for each $i \in \mathcal{A}$ such that $\gamma_i = 0$.

Although each NM amounts to choosing one column for removal and one or more columns for insertion, each choice is made one at a time, starting with the choice of a column j^- for removal, followed by consecutive choices of a column j^+ for insertion for as long as $C^y < C(y^*)$ holds. If $C^y + c_{j^+} \geq C(y^*)$, then j^+ is not inserted and the NM is concluded. Each choice is based on *score functions*. In turn, the score functions depend on the current weight values w.

Since LS aims to explore a diverse set of candidate solutions, some mechanisms are incorporated to LS to diversify the search [11, 34]. The first mechanism is the so-called *tabu list*, which prevents the search from revisiting regions already explored. More specifically, when a column is inserted into y, it is placed in the tabu list and deemed tabu. Tabu columns are not allowed to be removed from y for a fixed number of iterations. We remark that the implementation of the tabu list reported in 34 for the non unit-cost case is slightly different from the one reported in 11 for the unit-cost case. Recall that in the non unit-cost case, $k \geq 1$ variables are inserted at each NM. In that case, in accordance with 34, variables remain in the tabu list for exactly one iteration. On the other hand, in the unit-cost case, exactly one column is inserted at each NM. As reported in 11, the tabu list is implemented as a first-in first-out (FIFO) queue of fixed size ℓ. Furthermore, whenever a column is inserted to the current solution, it is pushed to the tabu list in the back of the queue, and the column on the first position is removed from the queue. Therefore a variable remains in the tabu list for ℓ consecutive iterations. To allow our LS to encompass both cases, we propose the following implementation for the tabu list. For each $j \in \mathbf{N}$, let $\tau_j \in \mathbb{Z}_{\geq 0}$ denote the *timestamp* of column j. At any iteration, τ_j corresponds to the last iteration t such that j was inserted or removed from the current solution. Moreover, we let ℓ denote the *length* of the tabu list. At each iteration $t \geq 0$, column j is tabu if $t - \tau_j \leq \ell$. In this way, if column j is inserted at iteration t, then it is considered tabu for the subsequent ℓ iterations $t+1, \ldots, t+\ell$. To replicate the implementations in 34 and in 11, respectively, we let $\ell = 1$ for the non unit-cost case, whereas $\ell = k$ for the unit-cost case. Note that this implementation allows for $\mathcal{O}(1)$ complexity when checking whether a column is tabu. The second diversification mechanism is to consider only a subset of columns as *candidates* for insertion or removal at each NM. For each $j \in \mathbf{N}$, let $\varphi_j \in \{\text{false}, \text{true}\}$ denote

whether j is a candidate. A column j is a candidate if $\varphi_j = \text{true}$ holds. Whenever j is removed from the solution, φ_j is set to false and $\varphi_{j'}$ is set to true for each $j' \in N_j$. Additionally, whenever j is inserted to the solution, $\varphi_{j'}$ is set to true for each $j' \in N_j$. Thus a column that has been removed can only be inserted back to the solution if the state of one of its neighboring columns has been altered. Finally, the third mechanism is based on the timestamps. As suggested in 34, ties in the score function are broken by choosing the column with smaller timestamp. This strategy prioritizes columns whose state has remained unaltered for longer, hence favoring diversification.

For each $j \in \mathbf{N}$ and each $k \in \{0, 1, 2\}$, the k-th order *score* of j is defined as:

$$\sigma_j^k = \frac{W_j^k}{c_j} = \frac{1}{c_j} \sum_{i \in \theta_j^k} w_i$$

To perform each neighborhood move, the variable j^- chosen to be removed from the current solution is given by:

$$j^- = \arg\min\{\sigma_j^1 : y_j = 1 \text{ and } \varphi_j = \text{true and } j \text{ is not tabu}\}$$

In order to incorporate randomness into the row-weighting scheme, GAO *et al.* [34] select columns for insertion in the following way: first, a constraint i is randomly sampled from the set of active currently uncovered constraints θ^0. Then the variable j^+ chosen to be inserted to the current solution is given by:

$$i \leftarrow \text{uniformly sample from } \theta^0$$

$$j^+ = \arg\max\{\sigma_j^0 : j \in J_i \text{ and } y_j = 0 \text{ and } \varphi_j = \text{true}\}$$

Furthermore, we propose an adaptation to the procedure of 34, which was motivated by the following observation. Along the LS procedure, with the insertion of variables into the current solution, it is possible that a column j is such that $y_j = 1$ and $\theta_j^1 = \emptyset$. In that case, j is redundant, since there is no constraint uniquely covered by j. The procedure described in 34 delays the removal of redundant columns to the moment y becomes feasible. Though rare, during our experiments the following situation was observed. At some iteration, y is infeasible and has a redundant column j. Additionally, there is j' which, if inserted to y, would lead to a feasible solution better than y^*. Nevertheless, the outright insertion of j' is not allowed because $C^y + c_{j'} \geq C^*$. However, if column j were removed from y, then $C^y + c_{j'} - c_j < C^*$ would hold and a better feasible solution thus results. To address this shortcoming, we propose to keep removing columns until a non redundant one is chosen. Note

that j is redundant if and only if $\theta_j^1 = \emptyset$. In that case, note that $W_j^1 = 0$ and hence $\sigma_j^1 = 0$. Additionally, j is not redundant if and only if $\theta_j^1 \neq \emptyset$. Since $w_i \geq 1$ for each $i \in \mathcal{A}$, then $W_j^1 > 0$ and $\sigma_j^1 > 0$ as long as $I_j \cap \mathcal{A} \neq \emptyset$. Assuming that $I_j \cap \mathcal{A} \neq \emptyset$ for every $j \in \mathbf{N}$, we conclude that $\sigma_j^1 > 0$ for every non redundant column j. Moreover, if y is infeasible, then there is some j such that $\sigma_j^0 > 0$. Hence the choice of j^+ prevents the insertion of a column that will cover no yet uncovered constraint. We conclude that the only way this situation can happen is if a column j is rendered redundant after the insertion of some $j' \in N_j$. Therefore, if y contains at least one redundant column, then j^- is redundant. An exception is made for columns which are tabu. For the sake of diversification, tabu variables are not removed even if they are redundant. By repeatedly picking the column which minimizes σ^1 until a non redundant one is chosen, the non-tabu redundant columns are guaranteed to be removed before any non-redundant one. This procedure stops after the removal of the first non redundant column.

Intuitively, each weight w_i can be interpreted as a measure of priority that is placed on constraint i. Note that if $y_j = 0$, then θ_j^0 denotes the set of active constraints which are currently not covered by y and that would become covered with the insertion of j. On the other hand, note that if $y_j = 1$, then θ_j^1 denotes the set of active constraints which are currently covered by y and that would become uncovered with the removal of j. Following this interpretation, θ_j^0 is the sum of the priorities of the constraints that would become covered with the insertion of j. Accordingly, θ_j^0 measures the priority for the insertion of j, and σ_j^0 is its ratio of priority to cost. Implementing a sort of a greedy manner, the most favorable column for insertion seems to be the one with highest priority to cost ratio. In turn, θ_j^0 is the sum of priorities of constraints which would become uncovered with the removal of j. Accordingly, θ_j^0 measures the contribution of j to the current solution, and σ_j^1 is its ratio of contribution to cost. Once again, in a greedy manner, the most favorable column for removal seems to be the one with lowest contribution to cost ratio. Whenever an active constraint is uncovered by the current solution, its weight is incremented by 1. In this way, note that constraints that have been uncovered many times have their priority increased. Consequently, the score function will prioritize their coverage in subsequent iterations.

Since the weights w are positive integers, it is likely that ties in the score value will happen. It was suggested in 34 and in 11 to break them by picking the variable with smaller timestamp. However, we favored the consideration of a second level of greediness in our attempts to break ties. More specifically, we suggest that:

- When choosing a column for insertion, if $\sigma_j^0 = \sigma_{j'}^0$, pick the one with greater value of σ^1.

- When choosing a column for removal, if $\sigma_j^1 = \sigma_{j'}^1$, pick the one with smaller value of σ^2.

If there is a tie in both the first-level and the second-level scores, then the variable with smaller timestamp is preferred.

Note that if $y_j = 0$, then θ_j^1 denotes the set of active constraints which are covered exactly once by y and that would become covered twice with the insertion of j. On the other hand, note that if $y_j = 1$, then θ_j^2 denotes the set of active constraints which are currently covered twice by y and that would become covered once with the removal of j. First, suppose that columns j and j' are such that $y_j = y_{j'} = 0$ and $\sigma_j^0 = \sigma_{j'}^0$. If j is inserted to the solution instead of j', then its cost ratio contribution after the insertion will correspond to the current value of σ_j^1. Reciprocally, if j' is inserted to the solution instead of j, then its cost ratio contribution after the insertion will correspond to the current value of $\sigma_{j'}^1$. Hence the second level of greediness prioritizes the insertion of the column that will lead to a higher contribution to cost ratio. Second, suppose that columns j and j' are such that $y_j = y_{j'} = 1$ and $\sigma_j^1 = \sigma_{j'}^1$. If j' is chosen for removal instead of j, then cost ratio contribution of j after removal will correspond to the current value of σ_j^2. Reciprocally, if j is chosen for removal instead of j', then the cost ratio contribution of j' after removal will correspond to the current value of $\sigma_{j'}^2$. Hence the second level of greediness prioritizes the removal of the column that would have a smaller cost ratio contribution if not removed. Moreover, recall that given a feasible solution, a column j is redundant if each constraint in I_j is covered at least twice, and that redundant columns can be pruned to produce a feasible solution with smaller objective value. At least intuitively, our proposed method for breaking ties also prioritizes the choice which seems more likely to lead to pruning when a feasible solution is found.

As a final remark, note that values such as C^y, γ, θ, W need not be recomputed at every iteration. These values need only be initialized at the start of LS and be updated with each insertion, removal, activation or weight increment. More specifically, if column j is inserted to or removed from the solution, these values are updated for the active constraints in I_j and for the columns in N_j. The insertion and removal procedures are described in Algorithms 12 and 13, respectively. Similarly, whenever a constraint i is activated, values associated with i and with the columns in J_i are updated. The procedure for activating a constraint is described in Algorithm 14. The pruning procedure is described in Algorithm 15. Finally, the local-search algorithm for the set cover problem with exponentially many constraints is described in Algorithm 16.

6.2 An iterative local-search procedure

According to our computational experiments, which are reported in Chapter 7, our proposed LS is effective in producing high-quality solutions within reasonable amount of time. Nevertheless, as shown by our experiments, LS does occasionally fail to produce an optimal solution. In this section we propose an iterative procedure to possibly further improve the best solution found by LS.

A common idea employed by some of the most effective SCP heuristics consists of heuristically reducing the instance size and solving this reduced instance to obtain a new and hopefully better solution. The reduced instance contains only a subset from the original rows and columns. By iteratively considering a different reduced instance, the procedure explores a diverse set of feasible solutions. For example, this idea has been employed by CAPRARA *et al.* [8], YAGIURA *et al.* [10], LAN *et al.* [107] and by MARCHIORI and STEENBEEK [108].

Consider an SCP instance (A, c) with row set $\mathbf{M}$ and column set $\mathbf{N}$. Given a subset of rows $\mathbf{M}' \subset \mathbf{M}$ and a subset of columns $\mathbf{N}' \subset \mathbf{N}$, a reduced instance (A', c') with row set corresponding to $\mathbf{M}'$ and column set corresponding to $\mathbf{N}'$ is obtained in the following way: let $A' = [a_{ij}]_{i \in \mathbf{M}', j \in \mathbf{N}'}$, meaning that A' is the submatrix of A containing those elements a_{ij} such that $i \in \mathbf{M}'$ and $j \in \mathbf{N}'$; additionally, let $c' = [c_j]_{j \in \mathbf{N}'}$. Since (A, c) is an SCP, then (A', c') is also an SCP, and with a smaller amount of rows and columns.

In the SCP literature [8, 10, 107, 108], a reduced instance is most usually defined by selecting a subset of columns $\mathbf{N}_0 \subset \mathbf{N}$ to have their values fixed to 0 and another subset $\mathbf{N}_1 \subset \mathbf{N}$ to have their values fixed to 1. Accordingly, in the reduced instance, a column with value fixed to 0 cannot be part of any feasible solution, whereas a column with value fixed to 1 must be in every feasible solution. Since these columns have their values fixed, there is no decision associated with them and they need not be part of the reduced instance. Hence $\mathbf{N}' = \mathbf{N} \setminus (\mathbf{N}_0 \cup \mathbf{N}_1)$. Additionally, note that $\bigcup_{j \in \mathbf{N}_1} I_j$ is the set of constraints covered by the columns in $\mathbf{N}_1$. Since columns in $\mathbf{N}_1$ have their values fixed to 1, these constraints are guaranteed to be covered by any feasible solution to the reduced instance and therefore need not be included as constraints in the reduced instance. Hence $\mathbf{M}' = \mathbf{M} \setminus \bigcup_{j \in \mathbf{N}_1} I_j$.

We now describe our proposed refinement procedure, which we call *Iterative Local Search*. At any point during Iterative LS, we let y^* denote the best solution obtained so far and let $\mathcal{C}^* = \{j \in \mathbf{N} : y^*_j = 1\}$ be the cover induced by it. Initially y^* corresponds to the best solution found during LS, and it is updated whenever a feasible solution with better objective value is found. Our Iterative LS performs a sequence of rounds until a stopping criteria is met. At each round, a reduced instance is defined and our LS procedure is employed as to produce a feasible solution to the

reduced instance. More specifically, LS starts from y^*, the best solution found so far. Note that by considering a different reduced instance at each round, LS effectively starts from different initial conditions and follows a different path in feasible space, thus possibly exploring yet unvisited solutions. This happens since for each reduced instance, the set of columns with fixed value and the constraints covered by them do not effectively take part in the problem. Consequently, the weights and scores are different at each start, and LS ends up following a different path. This ensures that our Iterative LS explores a wider area of the feasible region and favors diversification. Additionally, each application of LS begins from a slightly different starting point and therefore might move away from a local optimum.

Our column fixing heuristic is based on the procedures of CAPRARA *et al.* [8] and of YAGIURA *et al.* [10]. At each round, the value $\rho \in [0, 1]$ controls the instance percentage reduction. More specifically, given $\rho \in [0, 1]$, a total of $k_\rho = \lceil \rho \cdot |\mathcal{C}^*| \rceil$ columns are chosen to be fixed to 1. Let λ^* denote the best vector of lagrangian multipliers found by NDRC. Moreover, for each column $j \in \mathbf{N}$, let $c_j^{\lambda^*} = c_j - \sum_{i \in I_j} \lambda_i^*$ denote its corresponding lagrangian cost. Following the suggestion of YAGIURA *et al.* [10], the set of columns to be fixed to 1 are chosen in the following way. Let $\overline{c} = \max\{c_j^{\lambda^*} : j \in \mathcal{C}^*\}$ be the maximum lagrangian cost from the columns in $\mathcal{C}^*$. For each $j \in \mathcal{C}^*$, let $p'_j = \overline{c} - c_j^{\lambda^*}$. Additionally, let $P = \sum_{j \in \mathcal{C}^*} p'_j$. Then $\mathbf{N}_1$ is determined by taking a random sample of size k_ρ from $\mathcal{C}^*$ without replacement, with each column $j \in \mathcal{C}^*$ having probability $p_j = \frac{p'_j}{P}$. Note that columns with smaller lagrangian cost are more likely to be fixed to 1. When large-scale instances with up to thousands or even millions of columns are considered, it is convenient to select a subset of columns to be fixed to 0. Note that this has no effect on the amount of constraints in the reduced subproblem, but it might help the search in finding a better solution. However, since the focus of our experiments were specially on small and medium sized SCP and MCDSP instances, we do not employ column fixing to 0, meaning that $\mathbf{N}_0$ is empty at each round. For alternative choices of $\mathbf{N}_0$, the reader is referred to [8, 10]. Our Iterative LS takes two parameters $\underline{\rho}, \overline{\rho} \in [0, 1]$, denoting the minimum and maximum instance percentage reduction, respectively. Naturally, $\underline{\rho} \leq \overline{\rho}$ must hold. Moreover, given a time limit parameter $\overline{T}$ for Iterative LS and a time limit parameter $\bar{t}$ for each LS round, the total number of rounds is estimated by $R = \left\lceil \frac{\overline{T}}{\bar{t}} \right\rceil$. Accordingly, an increment $\rho^+ = \frac{\overline{\rho} - \underline{\rho}}{R-1}$ is computed. At the first round, the percentage reduction ρ is set to $\underline{\rho}$. Then, after each round, it is incremented by ρ^+. By increasing the percentage reduction, we ensure that the problem is further reduced at each round. A similar approach is adopted by CAPRARA *et al.* [8]. The procedure for defining a reduced SCP instance is described in Algorithm 17. Finally, our proposed Iterative Local Search algorithm is described in Algorithm 18.

Algorithm 12 Insert column j into an SCP solution

function INSERT COLUMN(j)

$y_j \leftarrow 1$

$C^y \leftarrow C^y + c_j$

$\tau_j \leftarrow t$

for $i \in I_j \cap \mathcal{A}$ **do**

if $\gamma_i = 0$ **then**

$\theta^0 \leftarrow \theta^0 \setminus \{i\}$

for $j' \in J_i$ **do**

$\theta^0_{j'} \leftarrow \theta^0_{j'} \setminus \{i\}$

$W^0_{j'} \leftarrow W^0_{j'} - w_i$

$\theta^1_{j'} \leftarrow \theta^1_{j'} \cup \{i\}$

$W^1_{j'} \leftarrow W^1_{j'} + w_i$

else if $\gamma_i = 1$ **then**

for $j' \in J_i$ **do**

$\theta^1_{j'} \leftarrow \theta^1_{j'} \setminus \{i\}$

$W^1_{j'} \leftarrow W^1_{j'} - w_i$

$\theta^2_{j'} \leftarrow \theta^2_{j'} \cup \{i\}$

$W^2_{j'} \leftarrow W^2_{j'} + w_i$

else if $\gamma_i = 2$ **then**

for $j' \in J_i$ **do**

$\theta^2_{j'} \leftarrow \theta^1_{j'} \setminus \{i\}$

$W^2_{j'} \leftarrow W^1_{j'} - w_i$

$\gamma_i \leftarrow \gamma_i + 1$

Algorithm 13 Remove column j from an SCP solution

function REMOVE COLUMN(j)

$y_j \leftarrow 0$
$C^y \leftarrow C^y - c_j$
$\tau_j \leftarrow t$
for $i \in I_j \cap \mathcal{A}$ **do**
  if $\gamma_i = 1$ **then**
    $\theta^0 \leftarrow \theta^0 \cup \{i\}$
    for $j' \in J_i$ **do**
      $\theta^0_{j'} \leftarrow \theta^0_{j'} \cup \{i\}$
      $W^0_{j'} \leftarrow W^0_{j'} + w_i$
      $\theta^1_{j'} \leftarrow \theta^1_{j'} \setminus \{i\}$
      $W^1_{j'} \leftarrow W^1_{j'} - w_i$
  else if $\gamma_i = 2$ **then**
    for $j' \in J_i$ **do**
      $\theta^1_{j'} \leftarrow \theta^1_{j'} \cup \{i\}$
      $W^1_{j'} \leftarrow W^1_{j'} + w_i$
      $\theta^2_{j'} \leftarrow \theta^2_{j'} \setminus \{i\}$
      $W^2_{j'} \leftarrow W^2_{j'} - w_i$
  else if $\gamma_i = 3$ **then**
    for $j' \in J_i$ **do**
      $\theta^2_{j'} \leftarrow \theta^2_{j'} \cup \{i\}$
      $W^2_{j'} \leftarrow W^2_{j'} + w_i$
  $\gamma_i \leftarrow \gamma_i - 1$

Algorithm 14 Activate a constraint during the SCP local-search

function ACTIVATE CONSTRAINT(i)

$\mathcal{A} \leftarrow \mathcal{A} \cup \{i\}$
$\mathcal{I} \leftarrow \mathcal{A} \setminus \{i\}$
$\theta^0 \leftarrow \theta^0 \cup \{i\}$
$w_i \leftarrow 0$
$\gamma_i \leftarrow 0$
for $j \in J_i$ **do**
  $\theta^0_j \leftarrow \theta^0_j \cup \{i\}$
  $W^0_j \leftarrow W^0_j + 1$

Algorithm 15 Prune an SCP solution

```
function PRUNE SOLUTION(y)
    while θ^0 = ∅ do
        j^- = arg min{σ_j^1 : y_j = 1 and φ_j = true}
        REMOVE COLUMN(j^-)
        V ← SEPARATION PROCEDURE(y)
        if V = ∅ then
            y* ← y
            C* ← C(y*)
        else
            for i ∈ V do
                ACTIVATE CONSTRAINT(i)
```

Algorithm 16 Local search for SCP with exponentially many constraints

```
function LOCAL SEARCH
    while elapsed time < time limit do
        PRUNE SOLUTION(y)
        removal column is redundant ← true
        while removal column is redundant do
            j^- ← arg min{σ_j^1 : y_j = 1 and φ_j = true and j is not tabu}
            removal column is redundant ← (θ^1_{j^-} == ∅)
            REMOVE COLUMN(j^-)
        while θ^0 ≠ ∅ do
            i ← UNIFORMLY SAMPLE(θ^0)
            j^+ ← arg max{σ_j^0 : j ∈ J_i and y_j = 0 and φ_j = true}
            if C^y + c_{j^+} < C* then
                INSERT COLUMN(j^+)
                for i ∈ θ^0 do
                    w_i ← w_i + 1
                    for j ∈ J_i do
                        W_j^0 ← W_j^0 + 1
            else
                break
        t ← t + 1
```

Algorithm 17 Defining a reduced SCP instance

function REDUCE INSTANCE($\rho, \mathcal{C}^*$)

$k_\rho = \lceil \rho \cdot |\mathcal{C}^*| \rceil$

$\overline{c} = \max\{c_j^{\lambda^*} : j \in \mathcal{C}^*\}$

$P \leftarrow 0$

for $j \in \mathcal{C}^*$ **do**

$p'_j \leftarrow \overline{c} - c_j^{\lambda^*}$

$P \leftarrow P + p'_j$

for $j \in \mathcal{C}^*$ **do**

$p_j \leftarrow \frac{p'_j}{P}$

$\mathbf{N}_1 \leftarrow$ sample k_ρ elements from $\mathcal{C}^*$ without replacement with probabilities $\{p_j : j \in \mathcal{C}^*\}$

$\mathbf{N}' \leftarrow \mathbf{N} \setminus \mathbf{N}_1$

$\mathbf{M}' \leftarrow \mathbf{M} \setminus \bigcup_{j \in \mathbf{N}_1} I_j$

$A' \leftarrow [a_{ij}]_{i \in \mathbf{M}', j \in \mathbf{N}'}$

$c' \leftarrow [c_j]_{j \in \mathbf{N}'}$.

return A', c'

Algorithm 18 Iterative Local search

function LOCAL SEARCH

$\rho \leftarrow \underline{\rho}$

$R \leftarrow \lceil \frac{\overline{T}}{\overline{t}} \rceil$

$\rho^+ \leftarrow \frac{\overline{\rho} - \underline{\rho}}{R-1}$

while elapsed time $<$ time limit **do**

$A', c' \leftarrow$ REDUCE INSTANCE($\rho, \mathcal{C}^*$)

$y^* \leftarrow$ LOCAL SEARCH(y^*, A', c')

$\rho \leftarrow \rho + \rho^+$

Chapter 7

Results

Recall that in Section 4.3 we proposed a strengthened lagrangian formulation for SCP. Furthermore, we propose a non-delayed relax-and cut (NDRC) and a local-search (LS) algorithm for SCP with exponentially many constraints, which are described in Chapters 5 and 6, respectively.

In this chapter, we perform a series of computational experiments to assess our contributions and compare them with other approaches reported in the literature. In Section 7.2 the two lagrangian relaxation formulations for SCP are compared. In Section 7.3, both our NDRC and LS are applied to solve the min connected dominating set problem. To do so, we consider the SCP formulation (SC-CDS) of MCDSP described in Section 3.3. Additionally, we compare our results with state-of-the-art MCDSP algorithms reported in the literature. These experiments are performed on benchmark instances for SCP as well as MCDSP. They are described in greater detail in Section 7.1.

Both NDRC and LS were implemented in the *Julia* programming language [109]. The computational experiments were executed on a Dell Inspiron 5490 notebook equipped with an Intel Core i7-10510U CPU with 1.80 GHz and running Windows 11 as operating system.

7.1 Instances

We begin by describing in greater detail the instances considered in our computational experiments. We consider benchmark instances which are standard in the SCP and MCDSP literature.

For the case of SCP in its most usual form, i.e, with a polynomial amount of constraints, we consider instances from "Beasley's OR Library" [1]. More specifically, we consider the sets **4**, **5**, **6**, **E**, **F**, **G** and **H** of SCP instances. These instances have number of rows ranging from 200 to 1000, number of columns ranging from 1000 to 10,000 and columns costs ranging from 1 to 100. These instances are described in

greater detail in Table 7.1. In Table (7.1), each row corresponds to an SCP instance. Column "Instance" corresponds to its name. Each instance defines a binary incidence matrix. Column $|\mathbf{M}|$ corresponds to the number of rows. Column $|\mathbf{N}|$ corresponds to the number of columns. Column "Density" corresponds to the matrix density, defined as the percentage of non-zero entries in the matrix. Finally, column "Costs" corresponds to the range of column costs, from the smallest up to the largest.

For domination problems in graphs, described in Section 2.2, we consider some graphs from the literature. More specifically, the set of graphs proposed by SIMONETTI *et al.* [2], referred to as "Type I", and the unit-disk graphs proposed by JOVANOVIC and TUBA [3], referred to as "Type II". Both Type I and Type II instances have become benchmark instances for MCDSP and were later considered in many other works[95, 110–115]. Type I instances are generated in the following way: given the number of vertices $|V|$ and the graph density $d \in [0, 1]$, a connected graph G with $|V|$ vertices and density d is constructed. To ensure G is connected, first a random permutation $p = (v_1, \ldots, v_{|V|})$ is uniformly sampled and the edge (v_k, v_{k+1}) is inserted to G, for each $k \leq |V| - 1$. Then pairs of distinct vertices are uniformly sampled and inserted as an edge until the desired density is met. In turn, Type II instances are randomly generated unit-disk graph instances and their generation is described in JOVANOVIC and TUBA [3]. Type I and Type II instances are described in greater detail in Tables 7.2 and 7.3. In Tables 7.2 and 7.3, each row corresponds to a graph instance. Column "Instance" corresponds to its name. Column "$|V|$" corresponds to the number of vertices. Column "$|E|$" corresponds to the number of edges. Column "Density" corresponds to the graph density. In the case of MCDSP, all column costs are assumed to equal 1, without loss of generality.

Instance	$\|\mathbf{M}\|$	$\|\mathbf{N}\|$	Density	Costs
scp41	200	1000	2	[1, 100]
scp42	200	1000	2	[1, 100]
scp43	200	1000	2	[1, 100]
scp44	200	1000	2	[1, 100]
scp45	200	1000	2	[1, 100]
scp46	200	1000	2	[1, 100]
scp47	200	1000	2	[1, 100]
scp48	200	1000	2	[1, 100]
scp49	200	1000	2	[1, 100]
scp51	200	2000	2	[1, 100]
scp52	200	2000	2	[1, 100]
scp53	200	2000	2	[1, 100]
scp54	200	2000	2	[1, 100]
scp55	200	2000	2	[1, 100]
scp56	200	2000	2	[1, 100]
scp57	200	2000	2	[1, 100]
scp58	200	2000	2	[1, 100]
scp59	200	2000	2	[1, 100]
scp61	200	1000	5	[1, 100]
scp62	200	1000	5	[1, 100]
scp63	200	1000	5	[1, 100]
scp64	200	1000	5	[1, 100]
scp65	200	1000	5	[1, 100]

Instance	$\|\mathbf{M}\|$	$\|\mathbf{N}\|$	Density	Costs
scpnre1	500	5000	10	[1, 100]
scpnre2	500	5000	10	[1, 100]
scpnre3	500	5000	10	[1, 100]
scpnre4	500	5000	10	[1, 100]
scpnre5	500	5000	10	[1, 100]
scpnrf1	500	5000	20	[1, 100]
scpnrf2	500	5000	20	[1, 100]
scpnrf3	500	5000	20	[1, 100]
scpnrf4	500	5000	20	[1, 100]
scpnrf5	500	5000	20	[1, 100]
scpnrg1	1000	10000	2	[1, 100]
scpnrg2	1000	10000	2	[1, 100]
scpnrg3	1000	10000	2	[1, 100]
scpnrg4	1000	10000	2	[1, 100]
scpnrg5	1000	10000	2	[1, 100]
scpnrh1	1000	10000	5	[1, 100]
scpnrh2	1000	10000	5	[1, 100]
scpnrh3	1000	10000	5	[1, 100]
scpnrh4	1000	10000	5	[1, 100]
scpnrh5	1000	10000	5	[1, 100]

Table 7.1: Details of SCP instances from Beasley [1], including number of rows and columns, matrix density and cost range.

Instance	|V|	|E|	Density
v30_d10	30	44	10
v30_d20	30	87	20
v30_d30	30	131	30
v30_d50	30	218	50
v30_d70	30	305	70
v50_d05	50	61	5
v50_d10	50	123	10
v50_d20	50	245	20
v50_d30	50	368	30
v50_d50	50	613	50
v50_d70	50	858	70
v70_d05	70	121	5
v70_d10	70	242	10
v70_d20	70	483	20
v70_d30	70	725	30
v70_d50	70	1208	50
v70_d70	70	1691	70
v100_d05	100	248	5
v100_d10	100	495	10
v100_d20	100	990	20
v100_d30	100	1485	30
v100_d50	100	2475	50
v100_d70	100	3465	70

Instance	|V|	|E|	Density
v120_d05	120	357	5
v120_d10	120	714	10
v120_d20	120	1428	20
v120_d30	120	2142	30
v120_d50	120	3570	50
v120_d70	120	4998	70
v150_d05	150	559	5
v150_d10	150	1118	10
v150_d20	150	2235	20
v150_d30	150	3353	30
v150_d50	150	5588	50
v150_d70	150	7823	70
v200_d05	200	995	5
v200_d10	200	1990	10
v200_d20	200	3980	20
v200_d30	200	5970	30
v200_d50	200	9950	50
v200_d70	200	13930	70

Table 7.2: Details of the graph instances of SIMONETTI *et al.* [2], including the number of vertices $|V|$, the number of edges $|E|$ and graph density.

Instance	|V|	|E|	Density
400_80_60	80	563	18
400_80_70	80	647	20
400_80_80	80	735	23
400_80_90	80	262	8
400_80_100	80	329	10
400_80_110	80	400	13
400_80_120	80	474	15
600_100_80	100	435	9
600_100_90	100	500	10
600_100_100	100	575	12
600_100_110	100	335	7
600_100_120	100	368	7
700_200_70	200	1305	7
700_200_80	200	1501	8
700_200_90	200	1730	9
700_200_100	200	756	4
700_200_110	200	921	5
700_200_120	200	1113	6
1000_200_100	200	756	4
1000_200_110	200	871	4
1000_200_120	200	997	5
1000_200_130	200	1127	6
1000_200_140	200	1269	6
1000_200_150	200	1400	7
1000_200_160	200	1541	8

Instance	|V|	|E|	Density
1500_250_130	250	903	3
1500_250_140	250	1011	3
1500_250_150	250	1119	4
1500_250_160	250	1246	4
2000_300_200	300	1577	4
2000_300_210	300	1710	4
2000_300_220	300	1849	4
2000_300_230	300	1990	4
2500_350_200	350	1461	2
2500_350_210	350	1555	3
2500_350_220	350	1668	3
2500_350_230	350	1787	3
3000_400_210	400	1522	2
3000_400_220	400	1621	2
3000_400_230	400	1750	2
3000_400_240	400	1880	2

Table 7.3: Details of the unit-disk graph instances of JOVANOVIC and TUBA [3], including the number of vertices $|V|$, the number of edges $|E|$ and graph density.

7.2 Comparing the lagrangian formulations

Recall that in Chapter 4 two alternative approaches were described for formulating the lagrangian dual problem for SCP. The first one is standard in the SCP literature and is here referred to as the *standard lagrangian* (StandLag). It is based on the lagrangian subproblem (SCLagSp) and its corresponding lagrangian dual problem (SCLagDual). The second one, originally proposed in this work, is called the *strengthened lagrangian* (StrongLag). It is based on the strengthened lagrangian subproblem (SCLagSp+) and its corresponding lagrangian dual problem (SCLagDual+). We perform a series of experiments to compare the two lagrangian formulations and evaluate the contribution of our proposed strengthened formulation in comparison with the standard one. To do so, we consider benchmark instances and for each formulation, we perform two independent rounds of experiments, each one consisting of solving one of the two lagrangian dual problems with the subgradient method (SM). In the first round, we solve the standard lagrangian dual (SCLagDual) and in the second one the strengthened lagrangian dual (SCLagDual+). Note that the performance of SM is affected by a variety of factors. In what follows, we discuss the most relevant ones and describe some of the choices made here in order to lead to a fair comparison between the two lagrangian formulations.

First of all, note that the performance of SM is affected by the knowledge of a primal bound $\overline{Z}$. In practical cases, this is initially unknown at the start of SM and a heuristic algorithm is employed to generate primal solutions and update $\overline{Z}$ during the algorithm. The value of $\overline{Z}$ affects the step sizes and therefore contributes to the convergence of SM, thus having an effect on the reported dual bound and the total execution time to convergence. To mitigate the dependency of SM on the quality of $\overline{Z}$ and lead to a fairer comparison between the two lagrangian formulations, for this set of experiments we initiate the value of $\overline{Z}$ with the optimal or best known primal bound for each instance. In this way, the SM algorithm begins with the tightest possible upper bound for both choices of lagrangian formulation.

Also note that the overall SM performance is affected by the frequency at which the lagrangian heuristic and separation procedures are called. Since our goal is to compare formulations based on their strength as dual problems, we additionally disable the application of the heuristic algorithm and separation procedures, so that the reported execution times correspond only to SM iterations. Consequently, each SM iteration comprises only the solution of the lagrangian subproblem and a subgradient step on the lagrangian multipliers. In this way, different performances of SM for the two different formulations are mostly due to the dual strength of each formulation and their effect on the convergence of SM. Since there is no application

of the heuristic and the separation procedure, we only consider instances with a polynomial amount of constraints and set all of them as active from the beginning. In this way, since there is no dynamic dualization of constraints, the algorithm is a standard SM, and thus not NDRC.

It follows from the theory of linear programming that an optimal solution to (SCLagDual) and an optimal solution to (SCLagSp+) lead to the same dual bound, which in turn is equal to the dual bound $\underline{\text{LP}}$ given by solving the linear programming relaxation (LP) of (SCP-IP). However, since SM is not guaranteed to provide an optimal dual solution, the reported dual bound might in fact be less than $\underline{\text{LP}}$. Hence, for the sake of better analyzing the results, we also present the LP dual bound for each instance. The closer the lagrangian dual bound is to the LP dual bound, the better.

Also note that the stopping criteria of SM influences the total number of subgradient iterations and total running time. Recall from Chapter 5 that SM keeps running as long as the step parameter α is greater than a given tolerance $\tilde{\alpha}$ and that α is reduced after $\tilde{\eta}$ consecutive SM iterations without an improvement to the dual bound. Some preliminary experiments have shown that when one of the lagrangian formulations leads to a greater dual bound, it might end up performing more SM iterations and taking longer time than the other, since the dual bound keeps improving and thus the stopping criteria takes longer to be met. This might lead to a false impression that one of the formulations has a slower convergence than the other, when, in fact, one of them is further improving the dual bound. For a fairer comparison, for this set of experiments we provide a target dual bound to SM and stop it as soon as that target bound is met. More specifically, for each instance the target dual bound is set to $\underline{\text{LP}} - 0.05$, where $\underline{\text{LP}}$ denotes the LP dual bound. In this way, we compare how each lagrangian formulation affects SM in finding the same bound.

Finally, note that the performance of SM is influenced by the initial vector of lagrangian multipliers. The method for determining these initial values is described in Section 5.4. In our proposed algorithm, these values are deterministic. However, note that a "lucky" guess of initial multipliers will speed up the convergence of SM, whereas an "unlucky" guess will slow it down. For a fairer comparison between the formulations, and to also take into account the robustness of each one to the initial value of lagrangian multipliers, for this set of experiments the initial multipliers are randomly perturbed and a series of 10 rounds are executed for each instance and for each formulation. For each round, the initial lagrangian multipliers are perturbed in the following way: each initial value is multiplied by p, where p is independently and uniformly sampled from the real interval $[0.75, 1.25]$, representing a 25% perturbation. For the k-th round, the random seed is set to k, so that each

round is run with a different random seed and therefore a different perturbation. And since for each instance and each formulation there are a total of 10 rounds of SM with possibly different results, we report the average lower bound, number of SM and running time from the 10 rounds.

We consider two classes of SCP instances. First, standard SCP non-unit cost instances from Beasley's OR Library [1]. Second, minimum total dominating set problem (MTDSP) instances on Type I and Type II graphs. Note that MTDSP instances are unit-cost, meaning all costs are equal to 1.

The results are presented in Table 7.4 for Beasley' instances, in Table 7.5 for MTDSP on Type I graphs and in Table 7.6 for MTDSP on Type II graphs. In Tables 7.4, 7.5 and 7.6, each row corresponds to an instance. Column "LP" denotes the dual bound given the linear programming relaxation of SCP. Column "Standard Lagrangian" corresponds to the solution of each instance considering the standard lagrangian dual (SCLagDual). Column "Strengthened Lagrangian" corresponds to the solution of each instance considering the strengthened lagrangian dual (SCLagDual+). Under columns "Standard Lagrangian" and "Strengthened Lagrangian", column "LB" corresponds to the average dual bound given by solving the lagrangian dual with SM, column "Iter" corresponds to the average number of SM iterations and column "Time" corresponds to the average CPU time for the execution of SM. For each instance, the average is taken from 10 runs with different random seeds. Finally, column "% Diff" corresponds to the percentage difference between the the results in column "Standard Lagrangian" and in "Strengthened Lagrangian". For each row and given one choice of measure("Iter" or "Time"), let x be the value measured for "Standard Lagrangian" and let y be the value measured for "Strengthened Lagrangian". Then the value in column "% Diff" is equal to $\frac{y-x}{x}$. Hence a positive (negative) value means that "Strengthened Lagrangian" presented an increase (decrease) in that measure when compared with "Standard Lagrangian". Additionally, another row is appended to the end of each of Tables 7.4, 7.5 and 7.6. This row, denoted by "Better", reports for each formulation, the amount of instances for which this formulation lead to a better result than the other one. By better, it is understood fewer subgradient iterations in the case of column "Iter" or shorter amount of time in the case of column "Time". Additionally, a row is included at the bottom of the tables, denoted by "Better". For columns "Standard Lagrangian" and "Strengthened Lagrangian", the entry in this row corresponds to the number of instancesmfor which that lagrangian formulation performed better than the other, considering the values from columns "LB", "Iter" or "Time".

For this set of experiments, we consider the following parameters for SM: the step coefficient α is initialized with value 1 and is halved after 100 consecutive iterations without an improvement to the best dual bound for Beasley's instances, and 50

iterations for Type I and II instances. SM stops as soon as α is smaller than 10^{-4}.

We now begin analyzing the results. We evaluate the strength of each formulation based specially on the following factors: number of SM iterations until convergence; and total SM execution time. Assuming that SM is successful in finding the target dual bound, doing so within fewer iterations and/or in a shorter amount of time is considered better.

First, note from columns "LB" that both StandLag and StrongLag were successful in finding near-optimal lagrangian multipliers. In fact, for all instances, both StandLag and StrongLag reported a dual bound greater than or equal to the target dual bound of $\underline{\text{LP}} - 0.05$. Since both lagrangian formulations were successful in meeting the target bound, and thus in converging to near-optimal lagrangian multipliers, we evaluate the strength of each formulation based on the number of subgradient iterations and total running time to convergence.

First, note from column "% Diff" that, when compared to StandLag, our proposed StrongLag consistently reduced the number of SM iterations required for meeting the target dual bound. In fact, it did so for 36 out of 43 of Beasley's instances, 33 out of 41 Type I instances and 40 out of 41 Type II instances. For example, StrongLag lead to percentage reductions in SM iterations ranging from 11% to 26%, 6% to 45%, 12% to 52% and 41% to 56% for sets **E**, **F**, **G** and **H**, respectively. Some examples of the most significant reductions include instances such as: *scpnrh5*, from 8791 SM iterations with StandLag to 3843 with StrongLag, amounting to a 56% reduction; *v200_d70*, from 2443 iterations to 468, a 81% reduction; *2000_300_210*, from 2346 to 1093, a 53% reduction; *3000_400_230*, from 2236 to 1219, a 45% reduction. On the other hand, StandLag required less SM iterations than StrongLag less frequently. It did so for 7 out of 43 of Beasley's instances, 8 out of 41 Type I instances and 1 out of 41 Type II instances. Whereas StrongLag lead to reductions as large as 81%, the increases were usually less significant. The most significant increases occurred for instances: *scp57*, from 1680 iterations with StandLag to 2231 with StrongLag, amounting to a 33% increase; *v30_d30*, from 62 to 110 with StrongLag, a 76% increase; *400_80_100*, from 98 to 129, a 31% increase. However, note that *v30_d30* and *400_80_100* required relatively very few iterations and their execution time is very short, less than a millisecond. Apart from these three instances, all increases were limited to 13% or less. For example: *scp58*, from 6794 to 7697, a 13% increase; *v70_d30*, from 427 to 476, a 12% increase. It is worth noting that each one of the three sets of instances is generated in a different way, such that the instances from each set have a particular structure that is different from the other sets. One could expect each particular structure to possibly favor one or the other formulation. However, note that StrongLag was consistent in reducing the amount of SM iterations on all three sets. Moreover, we bring attention to the fact that as

the instances sizes increase, our results suggest that StrongLag is more consistent than StandLag. For example, StrongLag lead to a reduction in SM iterations for: all 10 instances from sets **E** and **F**, with 500 rows and 5000 columns; all 10 instances from sets G and H, with 1000 rows and 10000 columns; for 23 out of 24 Type I instances with $|V| \geq 100$; and for 40 out of 41 Type II instances. On the other hand, increases were only observed: for 7 out of 23 instances from sets **4**, **5** and **6**, with 200 rows and 1000, 2000 and 1000 columns, respectively; for 7 out of 17 Type I instances with $|V| < 100$ and 1 out of 24 with $|V| \geq 100$; for 1 out of 41 Type II instances.

Aside from consistently reducing the amount of SM iterations to convergence, also note from column "% Diff" that, when compared to StandLag, our proposed StrongLag reduced the running time to meet the target dual bound for over half of the instances. In fact, it did so for 19 out of 43 of Beasley's instances, 26 out of 41 Type I instances and 27 out of 41 Type II instances. For example, StrongLag lead to percentage reductions in running time ranging from 45% to 57%, 49% to 70% and 57% to 69% for sets **E**, **F** and **H**, respectively. Some examples of the most significant reductions include instances such as: *scpnrf3*, from 4.41 seconds with StandLag to 1.30 with StrongLag, amounting to a 70% reduction; *scpnrh5*, from 7.41 seconds to 2.44, a 67% reduction; *v200_d30*, from 47.3 milliseconds to 14.0, a 70% reduction; *v200_d70*, from 132.6 milliseconds to 13.7, a 90% reduction; *2000_300_200*, from 32.1 milliseconds to 16.5, a 49% reduction. On the other hand, StandLag required less time than StrongLag in slightly less than half of the instances. It did so for 24 out of 43 of Beasley's instances, 15 out of 41 Type I instances and 14 out of 41 Type II instances. We note, however, that most of the observed increases occurred for smaller instances, whereas StrongLag more consistently reduced running time for relatively larger instances. Out of the 24 increases observed on Beasley's instances, 23 of them occurred on sets **4**, **5** or **6**, with 500 rows and 1000, 2000 and 1000 columns, respectively. In a similar way: out of the 15 increases observed for Type I instances, 10 of them occurred on instances with $|V| \leq 70$; out of the 14 increases observed for Type II instances, 8 and 11 of them occurred on instances with $|V| \leq 70$ and $|V| \leq 200$, respectively. On the other hand, StrongLag lead to a reduction in running time for: all 10 instances from sets **E** and **F**, with 500 rows and 5000 columns; 9 out of 10 instances from sets **G** and **H**, with 1000 rows and 10000 columns; for 18 out of 24 Type I instances with $|V| \geq 100$; for 23 out of 29 Type II instances with $|V| \geq 200$.

With a decrease in the amount of SM iterations, one should also expect a decrease in the overall running time. However, the results of our experiments show that even when StrongLag is successful in reducing the number of SM iterations to find the target dual bound, it might still take longer time than StandLag. We

now discuss this issue in greater detail. Recall that the standard lagrangian requires the solution of (SCLagSp), which is done in $\mathcal{O}(n)$ time, whereas the strengthened lagrangian requires the solution of (SCLagSp+), which is done in $\mathcal{O}(n \log \overline{K})$, for some $\overline{K} \leq n$. For each instance, given a known primal bound $\overline{Z}$, one can compute an appropriate value of $\overline{K}$, denoting the maximum possible cardinality of an optimal SCP solution. A detailed description of the computation of $\overline{K}$ is given in Section 4.3. So there is a trade-off in reducing the amount of SM iterations and taking longer to solve the lagrangian subproblem at each SM iteration. Besides computing the solution of the lagrangian subproblem, each SM iteration also includes: computing the subgradient vector, which takes $\mathcal{O}(q)$ time, where $q = \sum_{i \in \mathbf{M}} \sum_{j \in \mathbf{N}} a_{ij}$ is the number of non-zero elements in the incidence matrix; the subgradient step on lagrangian multipliers, which takes $\mathcal{O}(m)$ time; the update of lagrangian costs, which takes $\mathcal{O}(q)$. Note that a reduction in running time depends on the values of q and $\overline{K}$. Additionally, it depends on number of subgradient iterations until convergence, which is not known in advance. For these reasons, one cannot predict whether StrongLag will effectively lead to a reduction in overall running time. However, a few observations can be made. First, note that a higher density leads to a higher value of q, which in turn increases the complexity of each subgradient iteration. Moreover, a higher density tends to lead to a smaller value of $\overline{K}$, since on average each column covers more rows, and therefore a cover usually requires less columns. In turn, a smaller value of $\overline{K}$ reduces the complexity of solving the strengthened lagrangian subproblem. We conclude that as instance density increases, the price to be paid for solving the strengthened lagrangian subproblem is expected to decrease. Additionally, the majority of time is spent on computing the subgradient and updating the lagrangian costs, which take $\mathcal{O}(q)$ time, whereas solving the strengthened lagrangian subproblem takes $\mathcal{O}(n \log \overline{K})$ time. As discussed before, our proposed StrongLag was consistent in significantly reducing the number of SM iterations for convergence to the target dual bound when compared to StandLag. Considering that each SM iteration takes $\mathcal{O}(q)$ time, which dominates the $\mathcal{O}(n \log \overline{K})$ complexity of solving the strengthened lagrangian subproblem, a reduction in the number of SM iterations seems advantageous, despite the worse complexity for solving the lagrangian subproblem. All in all, as the instances sizes increase, and specially for higher density ones, our results show that StrongLag outperforms StandLag, consistently reducing the number of SM iterations and the total running time for convergence to near-optimal lagrangian multipliers.

Instance	LP	Standard Lagrangian			Strengthened Lagrangian			% Diff	
		LB	Iter	Time	LB	Iter	Time	Iter	Time
scp41	429	429	167	1.1 ms	429	96	1.7 ms	-43	58
scp42	512	512	146	1.0 ms	512	116	2.0 ms	-20	95
scp43	516	516	82	602.4 µs	516	49	943.3 µs	-41	57
scp44	494	494	210	1.4 ms	494	156	2.6 ms	-26	93
scp45	512	512	74	517.6 µs	512	47	957.2 µs	-35	85
scp46	557.25	557.25	6679	40.7 ms	557.25	7319	89.7 ms	10	121
scp47	430	430	254	1.7 ms	430	151	2.5 ms	-41	46
scp48	488.66	488.62	2341	14.1 ms	488.62	2490	32.5 ms	6	131
scp49	638.53	638.48	6115	41.4 ms	638.49	4310	59.4 ms	-30	44
scp51	251.22	251.18	2180	23.2 ms	251.18	2315	51.9 ms	6	123
scp52	299.76	299.71	3439	43.2 ms	299.71	3139	76.8 ms	-9	78
scp53	226	226	46	511.1 µs	226	29	1.1 ms	-36	118
scp54	240.50	240.50	3533	30.2 ms	240.50	3724	68.7 ms	5	127
scp55	211	211	82	840.1 µs	211	53	2.0 ms	-35	136
scp56	212.50	213	91	959.7 µs	213	77	2.6 ms	-15	173
scp57	291.77	291.73	1680	15.2 ms	291.73	2231	50.1 ms	33	229
scp58	287	287.00	6794	105.7 ms	287.00	7697	177.0 ms	13	67
scp59	279	279	322	2.9 ms	279	247	7.1 ms	-23	140
scp61	133.13	133.09	2644	30.6 ms	133.09	2374	40.6 ms	-10	32
scp62	140.45	140.41	3092	36.7 ms	140.41	2961	49.2 ms	-4	34
scp63	140.13	140.09	3421	38.4 ms	140.09	3331	55.0 ms	-3	44
scp64	129	128.95	2666	20.1 ms	128.95	2789	38.8 ms	5	93
scp65	153.35	153.30	3232	37.2 ms	153.30	2974	51.9 ms	-8	39
scpnre1	21.37	21.33	3706	1.55 s	21.33	2979	732.5 ms	-20	-53
scpnre2	22.36	22.31	3723	1.62 s	22.31	2751	689.0 ms	-26	-57
scpnre3	20.48	20.44	3394	1.47 s	20.44	3033	802.9 ms	-11	-45
scpnre4	21.35	21.31	3423	1.59 s	21.31	2977	695.9 ms	-13	-56
scpnre5	21.32	21.27	3561	1.29 s	21.28	2975	688.1 ms	-16	-47
scpnrf1	8.79	8.75	3628	3.19 s	8.75	3404	1.61 s	-6	-49
scpnrf2	9.99	9.95	3743	3.27 s	9.95	2794	1.38 s	-25	-58
scpnrf3	9.49	9.44	5453	4.41 s	9.44	3018	1.30 s	-45	-70
scpnrf4	8.47	8.42	4188	3.83 s	8.42	3265	1.55 s	-22	-59
scpnrf5	7.83	7.79	4408	3.88 s	7.79	3301	1.64 s	-25	-58
scpnrg1	159.88	159.82	7650	2.22 s	159.84	4559	1.44 s	-40	-35
scpnrg2	142.07	142.02	4232	1.07 s	142.02	3728	1.16 s	-12	8
scpnrg3	148.26	148.22	5127	1.34 s	148.22	4007	1.28 s	-22	-5
scpnrg4	148.94	148.88	8590	2.20 s	148.90	4163	1.31 s	-52	-40
scpnrg5	148.23	148.18	6424	1.69 s	148.18	4333	1.36 s	-33	-19
scpnrh1	48.12	48.07	8438	7.68 s	48.08	3834	2.47 s	-55	-68
scpnrh2	48.63	48.57	8414	8.17 s	48.59	3839	2.55 s	-54	-69
scpnrh3	45.19	45.15	7671	7.01 s	45.15	3877	2.61 s	-49	-63
scpnrh4	44.04	43.99	6052	5.27 s	43.99	3571	2.24 s	-41	-57
scpnrh5	42.37	42.31	8791	7.41 s	42.32	3843	2.44 s	-56	-67
Better			7/43	24/43		36/43	19/43		

Table 7.4: Comparison of the performance of the subgradient algorithm for solving the standard and the strengthened lagrangian dual problems for benchmark SCP instances.

Instance	LP	Standard Lagrangian			Strengthened Lagrangian			% Diff	
		LB	Iter	Time	LB	Iter	Time	Iter	Time
v30_d10	12	12	23	35.5 µs	12	21	49.2 µs	-6	38
v30_d20	5.87	6	12	24.8 µs	6	7	25.4 µs	-40	3
v30_d30	3.31	4	62	88.9 µs	4	110	171.2 µs	76	93
v30_d50	2.01	1.97	328	560.6 µs	1.96	300	513.5 µs	-9	-8
v30_d70	1.43	1.39	296	521.8 µs	1.38	306	523.2 µs	3	1
v50_d05	21	21	13	30.8 µs	21	12	44.7 µs	-12	45
v50_d10	10.23	11	204	327.6 µs	11	165	368.7 µs	-19	13
v50_d20	5.21	5.16	464	889.8 µs	5.17	490	1.1 ms	6	19
v50_d30	3.46	3.42	438	1.1 ms	3.42	487	1.2 ms	11	3
v50_d50	2.05	3	391	1.5 ms	3	395	1.2 ms	1	-18
v50_d70	1.46	1.41	306	1.4 ms	1.42	276	833.2 µs	-10	-39
v70_d05	21.75	21.70	400	724.6 µs	21.70	390	1.1 ms	-2	53
v70_d10	11.05	11.00	541	1.3 ms	11.00	526	1.6 ms	-3	19
v70_d20	5.29	5.24	478	1.6 ms	5.24	450	1.5 ms	-6	-1
v70_d30	3.37	3.33	427	1.9 ms	3.33	476	1.8 ms	12	-8
v70_d50	2.08	3	301	1.9 ms	3	308	1.5 ms	2	-23
v70_d70	1.45	1.40	417	3.1 ms	1.40	332	1.9 ms	-21	-39
v100_d05	20.43	20.38	744	2.3 ms	20.38	722	3.1 ms	-3	36
v100_d10	10.86	10.81	597	2.3 ms	10.81	568	2.5 ms	-5	7
v100_d20	5.35	5.30	714	4.2 ms	5.30	649	3.4 ms	-9	-19
v100_d30	3.43	3.38	577	5.0 ms	3.38	599	3.7 ms	4	-26
v100_d50	2.12	2.07	571	7.1 ms	2.07	565	4.6 ms	-1	-35
v100_d70	1.46	1.41	660	10.2 ms	1.41	479	4.7 ms	-27	-54
v120_d05	22.35	22.30	683	2.6 ms	22.30	634	3.3 ms	-7	26
v120_d10	10.51	10.46	850	4.1 ms	10.46	633	3.4 ms	-25	-17
v120_d20	5.29	5.24	654	5.1 ms	5.24	618	4.0 ms	-6	-21
v120_d30	3.43	3.38	678	7.5 ms	3.38	615	5.0 ms	-9	-33
v120_d50	2.02	1.97	627	10.9 ms	1.97	544	6.3 ms	-13	-43
v120_d70	1.44	1.39	873	19.2 ms	1.39	413	6.5 ms	-53	-66
v150_d05	21.48	21.43	769	3.7 ms	21.43	678	4.2 ms	-12	13
v150_d10	10.77	10.72	736	5.8 ms	10.72	702	5.1 ms	-5	-12
v150_d20	5.20	5.15	713	8.2 ms	5.15	693	6.1 ms	-3	-25
v150_d30	3.48	3.43	748	12.0 ms	3.43	731	8.5 ms	-2	-30
v150_d50	2.00	1.95	1128	28.3 ms	1.95	611	10.1 ms	-46	-64
v150_d70	1.45	1.40	1376	45.6 ms	1.40	414	8.5 ms	-70	-81
v200_d05	22.39	22.34	812	6.0 ms	22.34	777	6.9 ms	-4	16
v200_d10	10.56	10.51	1252	13.8 ms	10.51	843	8.8 ms	-33	-36
v200_d20	5.02	4.97	968	18.8 ms	4.97	857	12.4 ms	-12	-34
v200_d30	3.37	3.32	1802	47.3 ms	3.32	765	14.0 ms	-58	-70
v200_d50	2.02	1.97	1601	69.4 ms	1.97	609	16.5 ms	-62	-76
v200_d70	1.45	1.39	2443	132.6 ms	1.40	468	13.7 ms	-81	-90
Better			8/41	15/41		33/41	26/41		

Table 7.5: Comparison of the performance of the subgradient algorithm for solving the standard and the strengthened lagrangian dual problems for min total dominating set instances on Type I graphs.

Instance	LP	Standard Lagrangian			Strengthened Lagrangian			% Diff	
		LB	Iter	Time	LB	Iter	Time	Iter	Time
400_80_60	14.28	14.23	616	1.6 ms	14.23	561	2.0 ms	-9	29
400_80_70	11.10	12	486	1.4 ms	12	392	1.4 ms	-19	0
400_80_80	9.13	9.09	687	2.1 ms	9.09	655	2.4 ms	-5	17
400_80_90	7.90	7.86	689	2.2 ms	7.86	591	2.3 ms	-14	6
400_80_100	6.37	7	98	399.7 µs	7	129	522.8 µs	31	31
400_80_110	5.50	6	60	245.1 µs	6	58	274.2 µs	-3	12
400_80_120	4.96	4.92	555	2.3 ms	4.92	481	2.2 ms	-13	-4
600_100_80	16.44	16.40	676	2.2 ms	16.40	561	2.6 ms	-17	17
600_100_90	15.15	15.11	708	2.3 ms	15.11	621	2.8 ms	-12	20
600_100_100	12.28	12.24	774	2.7 ms	12.24	599	2.7 ms	-23	-1
600_100_110	10.94	10.90	754	2.9 ms	10.90	671	3.2 ms	-11	10
600_100_120	9.63	9.59	790	3.3 ms	9.59	670	3.3 ms	-15	0
700_200_70	29.84	29.80	1046	7.0 ms	29.80	898	8.5 ms	-14	22
700_200_80	24.12	24.06	2149	16.1 ms	24.08	1188	10.9 ms	-45	-33
700_200_90	20.14	20.10	1068	8.8 ms	20.10	883	8.4 ms	-17	-6
700_200_100	17.30	17.26	1314	10.1 ms	17.26	936	9.0 ms	-29	-11
700_200_110	15.11	15.06	1917	16.9 ms	15.07	910	8.9 ms	-53	-48
700_200_120	12.82	12.77	1295	16.3 ms	12.78	925	9.8 ms	-29	-40
1000_200_100	29.84	29.80	1046	7.0 ms	29.80	898	8.0 ms	-14	15
1000_200_110	25.48	25.43	1938	13.8 ms	25.44	1043	9.5 ms	-46	-31
1000_200_120	22.54	22.50	1170	8.7 ms	22.50	823	7.6 ms	-30	-12
1000_200_130	19.88	19.83	1843	14.8 ms	19.84	977	9.3 ms	-47	-37
1000_200_140	17.67	17.63	982	8.4 ms	17.63	883	8.7 ms	-10	3
1000_200_150	16.36	16.31	1007	8.9 ms	16.31	800	7.6 ms	-21	-15
1000_200_160	14.44	14.39	1445	12.1 ms	14.40	842	8.6 ms	-42	-29
1500_250_130	38.24	38.19	1860	15.3 ms	38.19	1105	12.1 ms	-41	-21
1500_250_140	33.59	33.54	1961	17.1 ms	33.55	1007	10.9 ms	-49	-36
1500_250_150	30.32	30.26	2164	20.3 ms	30.28	1355	15.1 ms	-37	-25
1500_250_160	27.28	27.22	2220	21.9 ms	27.23	1093	14.5 ms	-51	-34
2000_300_200	30.98	30.91	2283	32.1 ms	30.93	1217	16.5 ms	-47	-49
2000_300_210	28.60	28.54	2346	28.0 ms	28.55	1093	15.4 ms	-53	-45
2000_300_220	26.60	26.51	2234	26.6 ms	26.55	1770	25.0 ms	-21	-6
2000_300_230	24.92	24.83	2248	29.7 ms	24.86	2168	31.7 ms	-4	7
2500_350_200	46.24	46.19	1988	24.4 ms	46.20	1475	23.3 ms	-26	-4
2500_350_210	43.29	43.22	2195	29.7 ms	43.23	1763	38.5 ms	-20	30
2500_350_220	40.10	40.04	2259	34.2 ms	40.06	1736	27.4 ms	-23	-20
2500_350_230	37.40	37.33	2237	28.4 ms	37.36	1558	25.5 ms	-30	-10
3000_400_210	58.92	58.88	1719	23.6 ms	58.88	1299	23.7 ms	-24	1
3000_400_220	54.26	54.20	1944	26.1 ms	54.21	1448	25.6 ms	-26	-2
3000_400_230	50.64	50.58	2236	31.1 ms	50.60	1219	22.1 ms	-45	-29
3000_400_240	46.95	46.89	2451	36.0 ms	46.90	1926	34.0 ms	-21	-6
Better			1/41	14/41		40/41	27/41		

Table 7.6: Comparison of the performance of the subgradient algorithm for solving the standard and the strengthened lagrangian dual problems for min total dominating set instances on Type II graphs.

Subgradient norms and reducing zig-zag

The SM algorithm is known to show a *zig-zag* pattern. This kind of behavior is investigated in detail by GUTA [116]. By zig-zag, it is understood that if the sequence of lagrangian multipliers visited at SM iterations were plotted in $\mathbb{R}^m_{\geq 0}$, it would resemble a zig-zag pattern, with two consecutive points λ^t and λ^{t+1} differing by a straight line parallel which is parallel to g^t, the subgradient vector at iteration t. This kind of behavior is investigated in detail by GUTA [116]. As a consequence, due to the stark difference between lagrangian multipliers in two consecutive iterations, one also observes significant variations in the dual bound. More specifically, at one iteration, the lagrangian multipliers might be "good" and lead to a tight dual bound, and at the next one they might be far from optimal and lead to a poor dual bound. In this way, the subgradient step g^t taken at each SM iteration "overshoots" in a certain direction. This endorses the importance of an appropriate choice of step size for the subgradient steps. Recall from Chapter 5 that the step size at an iteration t is given by $\alpha_t \frac{\overline{f} - f(\lambda^t)}{||g^t||^2}$. According to CAPRARA *et al.* [8], the steepest ascent direction is given by finding the minimum norm convex combination of active subgradients. Determining such direction requires solving a quadratic programming problem and is therefore impractical to do every SM iteration. Considering our proposed strengthened lagrangian formulation, note that for certain values of lagrangian multipliers, the constraints $\overline{Z} \leq \sum_{j \in \mathbf{N}} c_j y_j$ and $\sum_{j \in \mathbf{N}} c_j y_j \leq \overline{Z}$ ensure that the solution to (SCLagSp+) leads to a subgradient vector with a significantly smaller norm than the solution to (SCLagSp). In this way, by considering a stricter lagrangian subproblem, our strengthened lagrangian formulation naturally reduces the subgradient norm and therefore can be considered as a heuristic for finding a steeper ascent direction. Our experiments show that StrongLag consistently reduces the zig-zagging effect in comparison with StandLag. This is illustrated in Figures 7.1, 7.2, 7.3 and 7.4. Each figure corresponds to an SCP instance. Like in the experiments of Table 7.4, for each instance, we independently solve the lagrangian dual problem with SM. Additionally, when doing so, at each SM iteration we record the current dual bound and subgradient norm. In order to compare the two lagrangian formulations, we present the results(either the dual bound or subgradient norm) corresponding to both formulations on a scatter plot. By superposing the points corresponding to StandLag and StrongLag on the same plot, we compare the zig-zag behavior presented by each one. In these figures, the horizontal axis corresponds to the SM iteration and the vertical axis corresponds to either the dual bound or the subgradient norm. The points corresponding to StandLag are plotted as red circles, while those corresponding to StrongLag are plotted as blue "x" marks.

Figures 7.1 and 7.3 comprise instances for which StrongLag lead to a time in-

crease compared to StandardLag. In contrary, Figures 7.2 and 7.4 comprise instances for which StrongLag lead to a reduction. In general, all these figures show that StrongLag significantly reduces the subgradient norm in comparison with StandLag, regarless of the effect in running time. Note from Figure 7.1 and 7.3 that when StandLag shows little zig-zag behaviour, there is less room for StrongLag to improve the convergence. On the other hand, note from Figures 7.2 and 7.4 that StrongLag significantly reduces the zig-zagging effect compared to StandLag. In those cases, this results in significant time reductions.

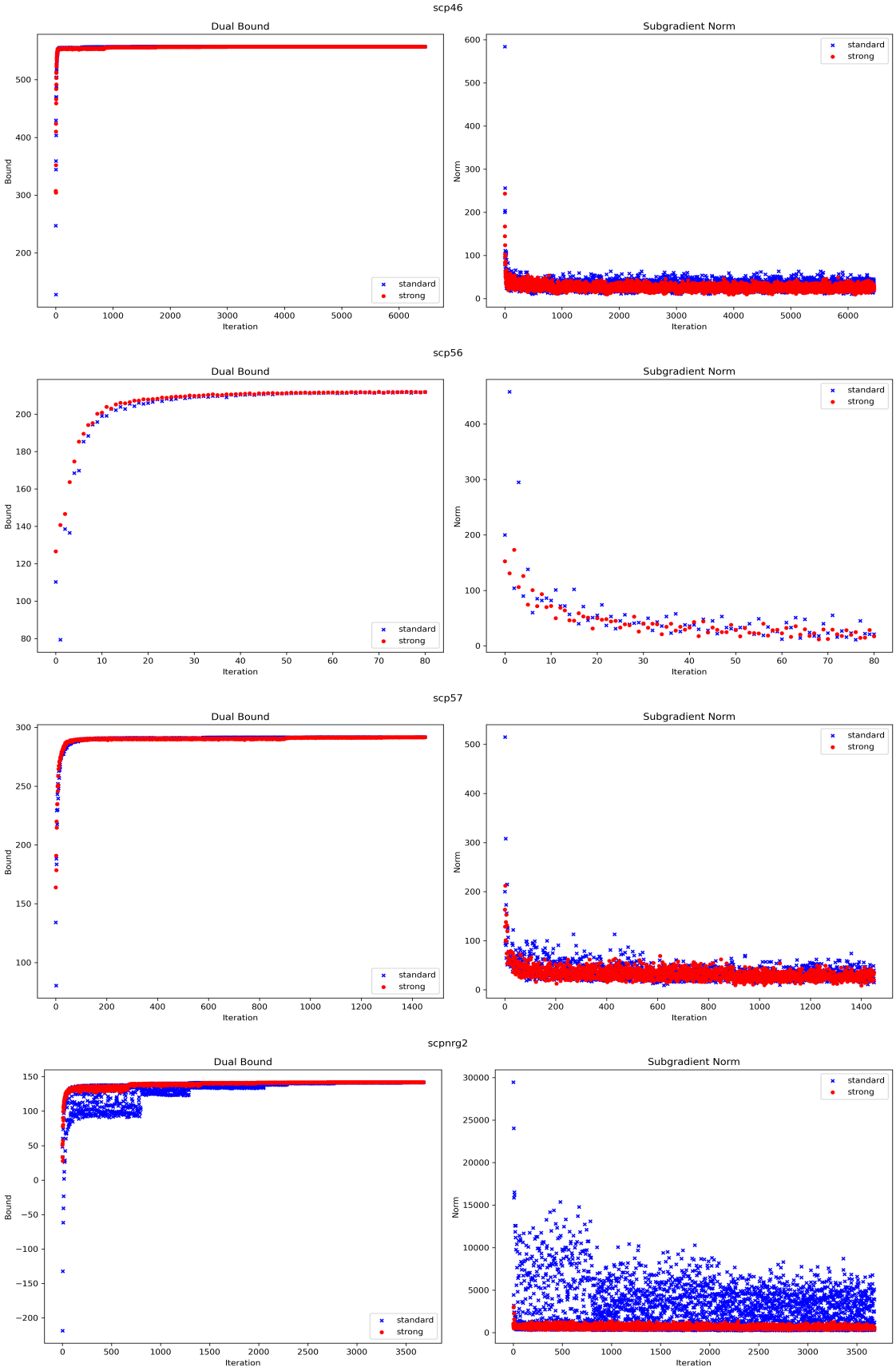

Figure 7.1: Comparison of dual bounds (left) and subgradient norms (right) between the standard (blue) and strengthened (red) lagrangian formulation for selected Beasley's instances.

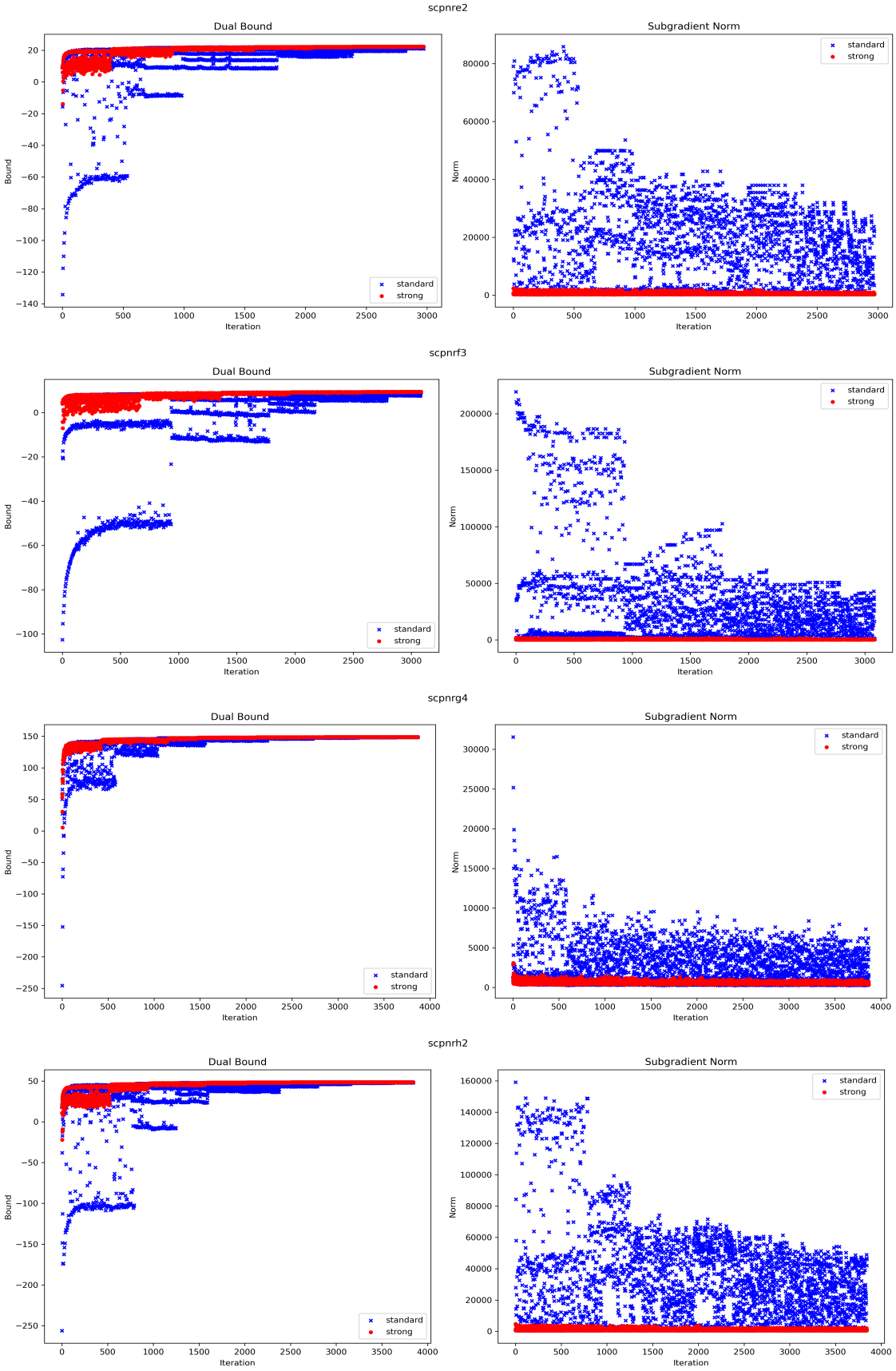

Figure 7.2: Comparison of dual bounds (left) and subgradient norms (right) between the standard (blue) and strengthened (red) lagrangian formulation for selected Beasley's instances.

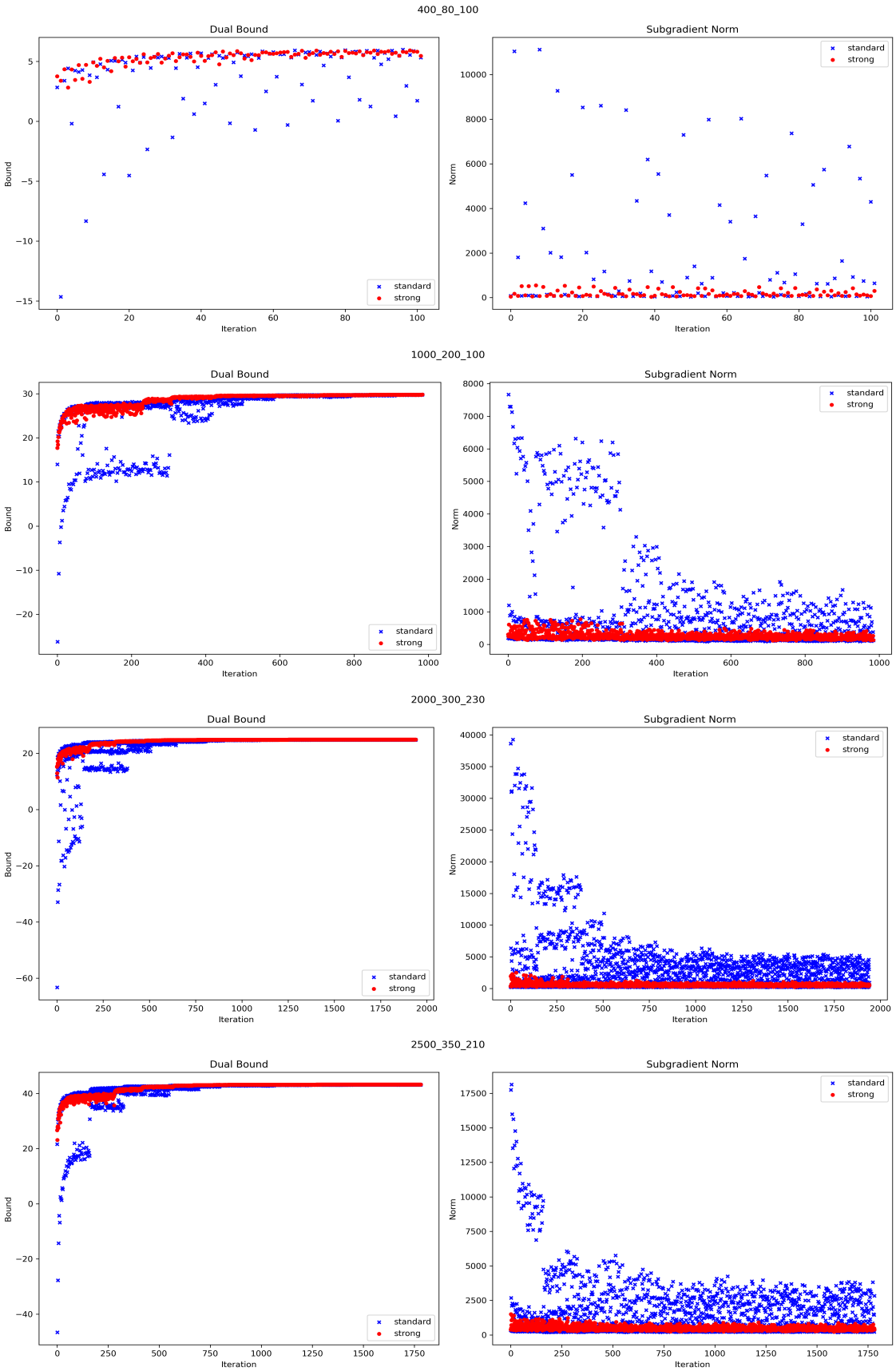

Figure 7.3: Comparison of dual bounds (left) and subgradient norms (right) between the standard (blue) and strengthened (red) lagrangian formulation for selected Type II instances.

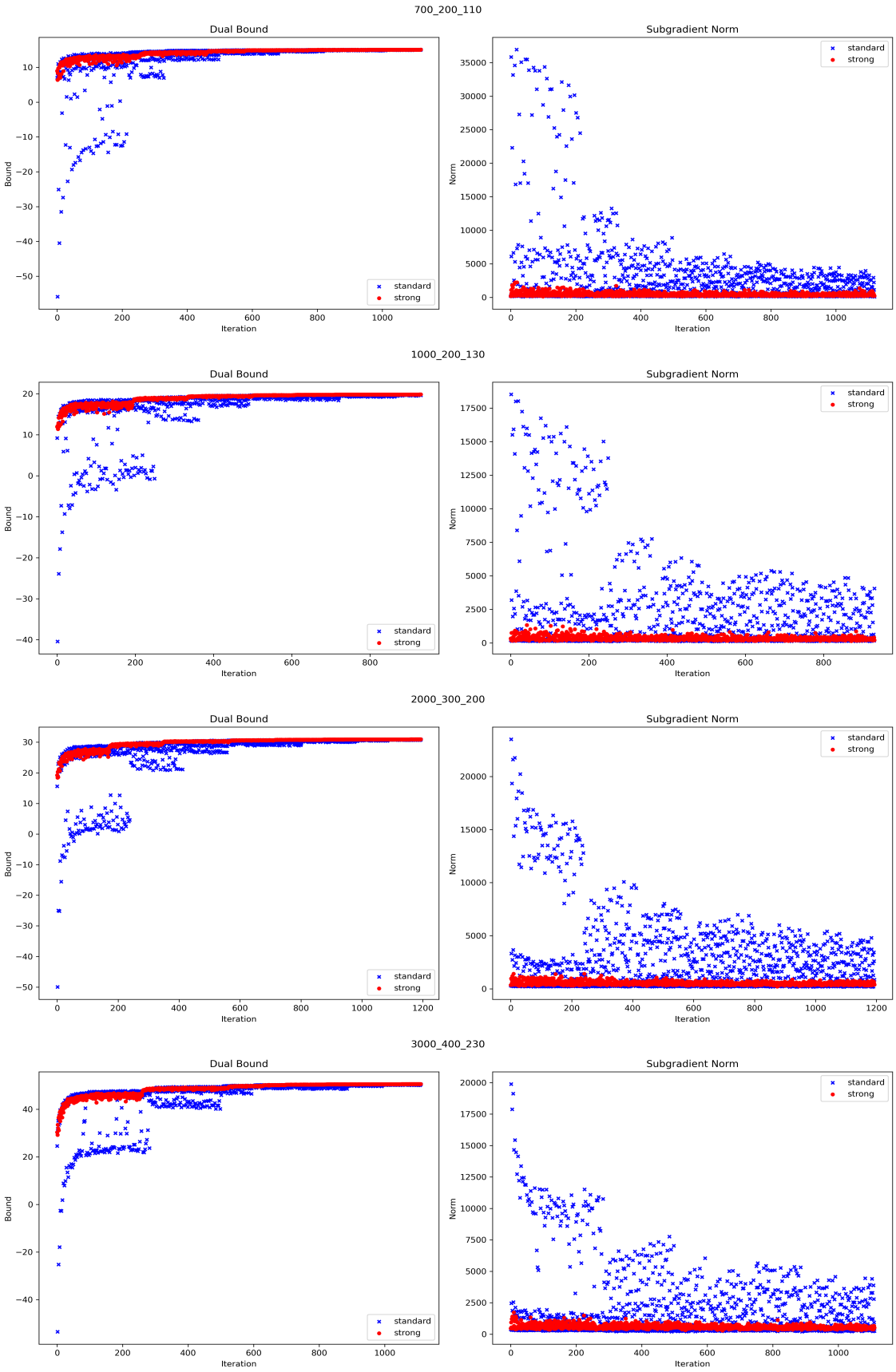

Figure 7.4: Comparison of dual bounds (left) and subgradient norms (right) between the standard (blue) and strengthened (red) lagrangian formulation for selected Type II instances.

Lagrangian branch-and-bound

In order to further assess the contribution of our strengthened lagrangian formulation, we implement a lagrangian relaxation based branch-and-bound procedure. This means that at each node in the enumeration tree, a corresponding lagrangian dual problem is formulated and solved with the subgradient method. The branch-and-bound procedure, the enumeration tree and SM methods are programmed in the Julia Programming Language [109].

Similarly to the previous experiments, for each instance we perform two independent runs, one considering the standard lagrangian formulation and another considering the strengthened one. Therefore, each instance is solved exactly with the lagrangian BB method. Each choice of lagrangian formulation influences the BB method since the formulation of the lagrangian dual problem at each enumeration node is different, corresponding to (SCLagDual) in the case of StandLag and to (SCLagDual+). Since solving each instance exactly likely requires longer time, for each instance and each formulation a single round is performed. Additionally, each instance is also solved by linear programming based BB with the *CPLEX* commercial solver. Again, to lead to a fair comparison between the two lagrangian formulations based on their dual strength, we disable the application of the lagrangian heuristic during SM, and the optimal solution is given as initial solution to the BB algorithm. In this way, we mitigate the dependence of both SM and BB on the primal bound $\overline{Z}$ and the reported times correspond only to SM iterations. Accordingly, some CPLEX parameters are set to approximate this. More specifically: the number of threads is set to 1; parameter *CPXPARAM_MIP_Strategy_HeuristicEffort* is set to 0, to disable the heuristic; parameter *CPXPARAM_Emphasis_MIP*, thus prioritizing the improvement of the best dual bound; and parameters *CPXPARAM_LPMethod*, *CPXPARAM_MIP_SubMIP_SubAlg* and *CPXPARAM_MIP_SubMIP_StartAlg* are set to 2, denoting the choice of the dual simplex method for solving the LP relaxation at the root and enumeration nodes. Additionally, CPLEX is also given the optimal solution as initial solution and the application of the heuristic is disabled. By doing so, we believe that any comparison between our lagrangian based BB with CPLEX is reasonably fair.

The results are presented in Table 7.7. Column "Standard Lagrangian" corresponds to the solution by BB of each instance considering the standard lagrangian dual (SCLagDual). Column "Strengthened Lagrangian" corresponds to the solution by BB of each instance considering the strengthened lagrangian dual (SCLagDual+). Column "CPLEX" corresponds to the solution by LP based BB of each instance using CPLEX. Under these columns, column "LB" corresponds to the best dual bound given by BB, column "Nodes" corresponds to the number of enumeration nodes ex-

plored until proven optimality and column "Time" corresponds to the elapsed CPU time. Finally, column "% Diff" corresponds to the percentage difference between the the results in column "Standard Lagrangian" and in "Strengthened Lagrangian". Additionally, a row is included at the bottom of the table, denoted by "Better". For each column "Standard Lagrangian" and "Strengthened Lagrangian", the entry in this row corresponds to the number of instances for which that lagrangian formulation performed better than the other, considering the values from columns "Nodes" or "Time".

We now begin comparing the results. Note that for instances for which both the standard and the strengthened lagrangian did not prove optimality, exploring more nodes is considered better, as it suggests that it would probably be faster in converging to the optimal solution if given enough time. On the other hand, when both formulations prove optimality, doing so by exploring less nodes and in shorter time is better.

Note from Table 7.7 that for exactly eleven instances, both StandLag and StrongLag proved the optimality of the given initial solution in the root node, with no need to enumerate. This happens for instances for which the LP dual bound is tight enough to prove optimality. On the other hand, note that for instances *v200_d05* and *v200_d20*, StandLag reaches the time limit without proving optimality, while StrongLag does so in 1401, and 1751 seconds, respectively. Besides, *v200_d10* was the only instance for which all three algorithms reach the time limit of 3600 seconds without proving optimality. Note that for this instance, StrongLag explored 412713 nodes and reported a dual bound of 14.21, whereas StandLag explored 391907 nodes and reported a dual bound of 13.61. The 5% increase in the number of nodes explored suggest that, considering the average among all enumeration tree nodes, StrongLag leads to a faster convergence of SM in comparison with StandLag. Moreover, considering that the optimal solution has objective value of 16, note that StandLag reports a 14.9% optimality gap, whereas for StrongLag this gap is 11.1%. Therefore StrongLag led to a 25% reduction in the optimality gap compared to StandLag. This suggests that not only is StrongLag decreasing the SM elapsed time, but also leading to tighter bounds on each node.

On the remaining 29 instances, note that: StrongLag required the exploration of less nodes and less time than StandLag in all of them.

We also note that for exactly nine instances, StrongLag explored less nodes than CPLEX to prove optimality. One example is instance *v200_d20*. Note that StrongLag required the enumeration of 434664 nodes to prove optimality, whereas CPLEX required 582375. Similarly, for instance *v200_d30*, StrongLag explored 6618 nodes, while CPLEX explored 7395. In this case, the convergence was in fact faster with StrongLag, requiring 35.88 seconds compared to 39.16 of CPLEX.

Together, the results from Tables 7.4, 7.5, 7.6 and 7.7 suggest that our proposed strengthened lagrangian formulation is effective in improving the performance of the SM algorithm in solving SCP. Our results show that the strengthened formulation is successful in improving the efficiency and robustness of SM, specially for larger instances or for solving SCP exactly. Moreover, at least for small scale instances, by considering the strengthened formulation, our lagrangian based BB algorithm shows performance which is comparable to that of a commercial LP solver.

Instance	CPLEX			Standard Lagrangian			Strengthened Lagrangian			Percentage Difference	
	LB	Nodes	Time	LB	Nodes	Time	LB	Nodes	Time	Nodes	Time
v30_d10	12	0	0.02	12	0	<0.01	12	0	<0.01	0	15
v30_d20	6	0	0.02	6	0	<0.01	6	0	<0.01	0	-5
v30_d30	4	0	0	4	0	<0.01	4	0	<0.01	0	50
v30_d50	3	0	0.02	3	0	<0.01	3	0	<0.01	0	-2
v30_d70	2	0	0	2	0	<0.01	2	0	<0.01	0	-5
v50_d05	21	0	0	21	0	<0.01	21	0	<0.01	0	0
v50_d10	11	0	0	11	0	<0.01	11	0	<0.01	0	-3
v50_d20	7	15	0.16	7	112	0.06	7	32	0.03	-71	-48
v50_d30	5	0	0.23	5	50	0.05	5	22	0.02	-56	-54
v50_d50	3	0	0.02	3	0	<0.01	3	0	<0.01	0	-2
v50_d70	2	0	0	2	0	<0.01	2	0	<0.01	0	8
v70_d05	23	0	0.02	23	28	0.02	23	30	0.02	7	9
v70_d10	13	24	0.11	13	296	0.24	13	74	0.10	-75	-57
v70_d20	7	18	0.12	7	142	0.18	7	54	0.08	-62	-55
v70_d30	5	3	0.41	5	88	0.14	5	28	0.05	-68	-68
v70_d50	3	0	0	3	0	<0.01	3	0	<0.01	0	-14
v70_d70	2	0	0.02	2	0	<0.01	2	0	<0.01	0	4
v100_d05	23	46	0.28	23	1338	1.78	23	294	0.56	-78	-68
v100_d10	13	122	0.42	13	464	0.79	13	140	0.33	-70	-59
v100_d20	8	1058	1.78	8	4716	6.78	8	822	1.50	-83	-78
v100_d30	6	3138	3.83	6	6864	10.88	6	1586	2.89	-77	-73
v100_d50	4	167	0.89	4	1294	2.48	4	332	0.69	-74	-72
v100_d70	3	9	0.16	3	182	0.85	3	118	0.39	-35	-54
v120_d05	25	136	0.64	25	3738	6.43	25	962	2.57	-74	-60
v120_d10	13	238	0.95	13	1570	3.03	13	176	0.53	-89	-82
v120_d20	8	2343	4.73	8	15520	33.21	8	2108	4.98	-86	-85
v120_d30	6	4722	6.95	6	7982	18.34	6	1760	4.21	-78	-77
v120_d50	4	407	1.99	4	4226	10.55	4	942	2.51	-78	-76
v120_d70	3	10	0.26	3	218	1.33	3	144	0.64	-34	-52
v150_d05	24	53	0.73	24	380	1.24	24	158	0.81	-58	-35
v150_d10	14	7387	15.44	14	70488	180.89	14	7080	25.35	-90	-86
v150_d20	8	2309	7.73	8	26000	72.65	8	2672	8.92	-90	-88
v150_d30	6	7097	15.94	6	11970	40.52	6	2730	9.32	-77	-77
v150_d50	4	620	5.39	4	6270	24.04	4	1160	4.26	-81	-82
v150_d70	3	11	0.49	3	278	2.45	3	196	1.36	-29	-44
v200_d05	27	131490	395.57	25.57	512047	3600.00	27	227360	1401.37	-56	-61
v200_d10	14.33	505644	3600.00	13.61	391907	3600.00	14.21	412713	3600.00	5[1]	0
v200_d20	9	582375	1631.19	7.57	632131	3600.00	9	434664	1751.51	-31	-51
v200_d30	6	7395	39.16	6	26552	172.78	6	6618	35.88	-75	-79
v200_d50	4	793	12.88	4	12308	81.26	4	2484	13.36	-80	-84
v200_d70	3	12	1.44	3	380	5.89	3	268	3.42	-29	-42
Better					1/41	5/41		29/41	34/41		

Table 7.7: Comparison of the performance of the lagrangian relaxation based branch-and-bound algorithm considering the standard and the strengthened lagrangian formulations for benchmark SCP instances.

[1] Recall that for instances for which both the standard and the strengthened lagrangian did not prove optimality, exploring more nodes is considered better, as it suggests that it would probably be faster in converging to the optimal solution if given enough time. On the other hand, when both formulations prove optimality, doing so by exploring less nodes and in shorter time is better.

7.3 Results for connected dominating sets

Recall that in Section 3.3 the min connected dominating set problem was formulated as SCP with exponentially many constraints. Accordingly, MCDSP can be solved by our proposed non-delayed relax and cut (NDRC), local-search (LS) and Iterative Local Search (Iterative LS) algorithms described in Sections 5.2, 6.1 and 6.2, respectively. In this section, in order to evaluate the performance of our proposed algorithms and assess their effectiveness in producing high-quality primal solutions, we employ them to solve benchmark instances and compare the results with some state-of-the-art MCDSP algorithms reported in the literature.

We consider the SCP formulation of MCDSP described in (SC-CDS). For each instance, we first apply NDRC. By frequently applying the Lagrangian heuristic during NDRC, the algorithm provides a primal and a dual bound. The best primal solution found during NDRC is then given as initial solution to LS for possible further improvement. Finally, the best solution found during LS is then given as initial solution to Iterative LS. Additionally, the vector of lagrangian multipliers found during NDRC that lead to the best dual bound is given to Iterative LS for the problem reduction heuristic. Recall that the NDRC procedure begins with a given initial set of active constraints and applies a separation procedure in order to identify violated constraints and activate them. In the case of MCDSP, we consider the initial set of active constraints containing a total domination constraint $\sum_{u \in N(v)} y_u \geq 1$ for each vertex $v \in V$, which equates to exactly $|V|$ initial active constraints in total. In accordance with the formulation presented in (SC-CDS), the connectivity of the solution is guaranteed by further imposing a vertex-cut constraint $\sum_{v \in S^V} y_v \geq 1$ for every vertex-cut $S^V \in \mathcal{S}^V(G)$. For this set of experiments, we let all the vertex-cut constraints to be initially inactive, and we employ the separation procedure of violated vertex-cuts described in Section 3.4. In this way, during the NDRC algorithm, each time the lagrangian heuristic is called, a tentative solution is produced and the separation procedure is applied one or more times until this solution does not violate any vertex-cut constraint. The constraints separated during the greedy heuristic are then activated in the NDRC and associated with an active lagrangian multiplier. The frequency at which the lagrangian heuristic is called during NDRC is given by the following rule. Recall that NDRC keeps track of the best dual bound found so far. For this set of experiments, the lagrangian greedy heuristic is called at every iteration in which the best dual bound is improved. After NDRC ends, we apply our proposed LS algorithm to the best primal solution found during NDRC. Recall that LS also takes a given initial set of active constraints and possibly activates others during its execution, in a manner similar to NDRC. Let $\mathcal{A}$ denote the set of active constraints at the end of NDRC. We then initialize the set of LS active constraints

as $\mathcal{A}$. Similarly, the separation procedure from Section 3.4 is applied every time that the current LS solution satisfies all of the currently active constraints. The vertex-cuts violated by this solution are then activated and associated with an active weight. At termination, LS returns the best primal solution found during its execution. Finally, we employ our proposed Iterative LS to the best solution found during LS. In a similar manner, at the beginning of Iterative LS, the set of active constraints corresponds to the set of active constraints at the end of LS.

Note that both LS and Iterative LS are randomized algorithms. Following the standard approach from the literature, for each instance we perform 10 rounds, each one with a different random seed, hence leading to a possibly different solution. More specifically, for the k-th round, the random seed is set to k. Each round comprises the application of NDRC, LS and Iterative LS, specifically in that order. Since each round is performed with a different random seed, it possibly leads to a different solution and, consequently, different bounds. For that reason, for each instance we report the minimum, average and maximum primal bound found during the 10 rounds. Additionally, we report the amount of times that the minimum primal bound was found, i.e, out of the 10 rounds, the amount of rounds in which the minimum bound from those rounds was found. Finally, these values are reported for each one of the NDRC, LS and Iterative LS algorithms. Note that since LS takes the best solution from NDRC, then the primal bound reported by LS is necessarily less than or equal to the the one reported by NDRC. And the same is true for Iterative LS and LS.

Parameter tuning

Note that each of NDRC, LS and Iterative LS require a few parameters. Each combination of parameters might lead to a different performance of the algorithms, specially when the solution found by one algorithm is given as input to the other. To determine an appropriate combination of parameters for NDRC and LS, we employ a *grid-search* procedure. Our grid-search procedure is generally described in the following way. Let $k \geq 1$ be the number of parameters to be tuned. For each $i \in \{1, \ldots, k\}$, we consider a finite set A_k of possible values for this parameter. A grid is defined by the cartesian product of all these finite sets, i.e, $A_1 \times A_2 \times \cdots \times A_k$. Hence, each element in this grid corresponds to a particular combination of parameter values. For each element in this grid, we evaluate its merit by solving a set of MCDSP instances with the parameter values of NDRC and LS corresponding to that element. More specifically, given a set of instances, for each grid element and for each instance we perform 10 rounds of NDRC followed by LS, with each round having a different random seed. We then store the primal bounds reported by LS. For each instance, we consider the average and minimum bound of the 10 rounds.

In this way, for each grid element, its merit corresponds to the average, among the set of instances, of the average primal bound for each instance. Equivalently, if the optimal primal bound or some other value, such as the best known bound from the literature, is taken as a target T, the merit can be offset by this target value. Therefore, for each round, we record the difference between the primal bound and the target value, i.e, $\overline{Z} - T$. Note that, as long as the target value is less than or equal to any primal bound, then both merit definitions are equivalent, meaning that the ordering according to merit remains the same, regardless of the particular choice of definition. However, the second definition allows for a more intuitive interpretation, since in that case the merit denotes the average deviation from the target bound. For a minimization problem, the grid element leading to the smallest merit is deemed the best.

In particular, we apply grid-search to determine an appropriate combination of NDRC and LS parameters. We do not include the tuning of Iterative LS during the grid-search for two reasons: first, because it increases the required execution time; secondly, at least for the instances considered here, some preliminary experiments have shown that the performance of Iterative LS was very similar even for different combinations of parameters. The one parameter that seems to affect Iterative LS the most is the time limit, which is discussed in detail later. In particular, for NDRC, we are interested in the variation of two parameters. Recall that our NDRC reduces the step size coefficient α by a factor of $\check{\alpha}$ after $\tilde{\eta}$ iterations without an improvement to the best dual bound. We are insterested in tuning the values of $\check{\alpha}$ and $\tilde{\eta}$. As for LS, we are interested in the variation of one parameter, the tabu list size. So we let: $A_1 = \{1.05, 1.1, 1.5, 2\}$ be the set of possible values for the step coefficient divider $\check{\alpha}$; $A_2 = \{50, 75, 100, 125\}$ be the set of possible values for $\tilde{\eta}$, the number of NDRC iterations without improvement; and $A_3 = \{0, 1, 2\}$ be the set of possible values for the tabu list size. Our preliminary experiments have shown that for all of the Type I and nearly all of the Type II instances, the combination of NDRC and LS was successful in finding the best known primal bound reported in the literature for every choice of parameter combination from this grid. The only exceptions were the Type II instances *2500_350_230*, *3000_400_210*, *3000_400_230* and *3000_400_240*. In special, for these four instances, the combination of parameters was determinant in matching or not the best known primal bound. In Table 7.8 we present the results of our grid search only for these four instances. Each row corresponds to a grid element, i.e., to one particular combination of NDRC and LS parameters. Column "$\tilde{\eta}$" denotes the number of NDRC iterations without an improvement. Column "$\check{\alpha}$" denotes the step coefficient divider. Column "$|T|$" denotes the tabu list size. Columns "350_230", "400_210", "400_230" and "400_240" correspond to the results on instances *2500_350_230*, *3000_400_210*, *3000_*-

400_230 and *3000_400_240*, respectively. Under each one of these columns, "Min" and "Avg" denote the minimum and average primal bounds among the 10 rounds for that instance. Finally, column "Merit" denotes the merit of each grid element, where the best known primal bound is considered as target or offset. Hence the merit denotes the average deviation from the best known bound. Additionally, when the value in "Min" matches the best known primal bound, the value is displayed in bold. Moreover, a row is included at the bottom of the table, denoted by "BKS". For each column corresponding to an instance, the entry in this row denotes the number of grid-elements for which the BKS of that was found.

First of all, note that none of the parameter combinations with a tabu list size of zero had a worse merit than any of the combinations for which the tabu list size was 1 or 2. This endorses the contribution of the tabu strategy in incorporating diversification into LS.

Note from Table 7.8 that when all grid elements are considered, the combination NDRC with LS was successful in finding the BKS for every instance. On the other hand, note that on this particular grid, for no single parameter combination the BKS was found for every instance. This highlights the importance of our proposed Iterative LS procedure. In fact, as shall be presented later, our Iterative LS is able to find the BKS for all of Type I and II instances with a single combination of parameters.

Based on the results of the grid-search procedure, the parameter values for the next set of experiments are set as follows. For NDRC, we set $\tilde{\eta} = 50$, $\check{\alpha} = 1.05$ and $\tilde{\alpha} = 10^{-4}$. This means that α is initialized with value 2 and is divided by 1.05 after 50 consecutive iterations to the best dual bound, and it terminates as soon as α is less than 10^{-4}. For LS, we set $|T| = 2$, meaning the tabu list has size two. Additionally, the LS time limit is set to 60 seconds. Finally, for Iterative LS, we set the minimum and maximum problem reduction to 5% and 20%, respectively. This means that during Iterative LS, the problem reduction factor is initially 5%, and is increased during its execution up to 20%. Moreover, for each round, if we t_0 denote the total time elapsed during NDRC, then the time limit for Iterative LS is set to $443 - t_0 - 60$ seconds. In this way, each round runs for up to 443 seconds, and the time limit for Iterative LS is discounted by the time elapsed during NDRC and LS. A more detailed explanation of this particular choice is given further on.

$\tilde{\eta}$	$\check{\alpha}$	$\lvert T\rvert$	*350_230*		*400_210*		*400_230*		*400_240*		Merit
			Min	Avg	Min	Avg	Min	Avg	Min	Avg	
50	1.05	2	49	49.2	75	75.1	**64**	65.3	61	61.4	1.250
50	2.00	2	49	49.3	75	75.2	65	65.3	61	61.4	1.300
125	1.10	1	49	49.6	75	75.0	65	65.3	**60**	61.3	1.300
100	1.10	2	49	49.3	75	75.1	**64**	65.2	61	61.7	1.325
125	1.50	2	49	49.6	**74**	74.9	**64**	65.3	61	61.5	1.325
100	1.10	1	49	49.4	**74**	74.8	65	65.9	61	61.4	1.375
125	1.50	1	49	49.5	75	75.1	65	65.4	61	61.5	1.375
125	1.05	1	49	49.4	75	75.1	65	65.6	61	61.5	1.400
100	2.00	1	**48**	49.2	75	75.1	65	65.8	61	61.5	1.400
75	2.00	2	49	49.4	**74**	75.0	**64**	65.6	61	61.6	1.400
75	1.05	2	**48**	49.2	75	75.3	65	65.6	61	61.5	1.400
50	1.15	2	49	49.4	75	75.0	65	65.8	61	61.4	1.400
50	1.10	2	**48**	49.5	75	75.2	65	65.6	61	61.3	1.400
125	1.10	2	49	49.4	75	75.4	65	65.5	61	61.3	1.400
100	1.05	1	49	49.3	**74**	74.9	65	65.9	61	61.5	1.400
100	1.50	1	49	49.5	**74**	75.1	65	65.6	61	61.5	1.425
50	1.50	1	49	49.4	**74**	75.0	65	65.8	61	61.5	1.425
50	1.50	2	49	49.3	75	75.2	65	65.5	61	61.7	1.425
75	1.10	2	49	49.5	**74**	75.2	65	65.2	61	61.8	1.425
75	1.50	1	**48**	49.6	**74**	75.0	65	65.7	61	61.4	1.425
75	2.00	1	49	49.6	**74**	75.1	65	65.6	61	61.5	1.450
100	2.00	2	**48**	49.4	75	75.1	**64**	65.6	61	61.7	1.450
75	1.05	1	49	49.3	75	75.3	65	65.5	61	61.7	1.450
50	1.05	1	49	49.5	**74**	75.2	65	65.5	61	61.6	1.450
100	1.05	2	**48**	49.4	75	75.2	**64**	65.5	61	61.8	1.475
125	2.00	1	49	49.7	75	75.1	65	65.6	61	61.6	1.500
100	1.50	2	49	49.5	**74**	75.2	65	65.6	61	61.7	1.500
50	2.00	1	49	49.7	**74**	75.2	65	65.6	61	61.6	1.525
50	1.10	1	49	49.3	75	75.3	65	65.8	61	61.8	1.550
75	1.10	1	**48**	49.8	75	75.1	65	65.8	61	61.6	1.575
125	1.05	2	49	49.5	**74**	75.5	65	65.8	61	61.6	1.600
75	1.50	2	49	49.6	75	75.2	65	65.8	61	61.8	1.600
125	2.00	2	49	49.5	75	75.6	65	65.7	61	61.6	1.600
75	1.05	0	49	50.0	75	75.6	65	65.9	61	62.2	1.925
100	1.05	0	49	50.4	**74**	75.3	65	66.1	61	62.0	1.950
100	1.50	0	49	50.4	75	75.5	66	66.0	61	61.9	1.950
50	1.50	0	49	50.2	75	75.8	65	66.0	61	61.9	1.975
125	1.05	0	49	50.3	75	75.4	66	66.2	61	62.0	1.975
100	1.10	0	49	50.5	75	75.6	65	66.0	61	62.0	2.025
50	1.05	0	49	50.2	**74**	75.6	65	66.0	61	62.3	2.025
125	1.10	0	49	50.4	**74**	75.6	65	66.0	62	62.4	2.100
75	1.50	0	50	50.5	75	75.8	65	66.1	61	62.0	2.100
75	1.10	0	50	50.4	75	75.7	66	66.3	62	62.0	2.100
125	1.50	0	50	50.4	75	75.7	66	66.5	61	61.8	2.100
50	2.00	0	50	50.4	**74**	75.6	66	66.4	62	62.3	2.175
50	1.10	0	50	50.5	75	76.0	66	66.2	62	62.2	2.225
BKS			7/46		17/46		6/46		1/46		

Table 7.8: Results of the grid-search algorithm for tuning NDRC and LS parameters.

Results

The results for Type I and Type II graphs are presented in Tables 7.9 and 7.10, respectively. Each row corresponds to an MCDSP instance. In Table 7.9, column "OPT" corresponds to the optimal objective value, whereas in Table 7.10 column "BKS" corresponds to the best primal bound known for each instance, taken from the literature. Column "LP" corresponds to the dual bound given by the LP relaxation of the initial SCP problem containing only the total domination constraints. At the end of NDRC, the set of active constraints contains the total domination constraints, as well as active vertex-cut constraints. Together, these constraints define a stronger SCP and we solve its LP relaxation to obtain the value reported in column "LP+". Ideally, the dual bound reported by NDRC should match this value. However, since NDRC is not guaranteed to converge to an optimal dual solution, the reported NDRC dual bound dual can be smaller than it. A smaller difference is considered better, since it shows that NDRC was succesful in converging to near-optimal lagrangian multipliers. Columns "NDRC" corresponds to the results of the NDRC algorithm. Under this column, "LB" corresponds to the dual bound given by NDRC, "UB" corresponds to the best primal bound given by NDRC and "Time" corresponds to the average CPU time out of the 10 rounds of NDRC. Recall that since NDRC is deterministic, both the primal and dual bounds are the same on all rounds, except for very small numerical error. Columns "Local Search" and "Iterative LS" corresponds to the results of the LS and Iterative LS algorithms, respectively. Under each one of these columns, "Min", "Avg" and "Max" correspond to the minimum, average and maximum primal bound found on 10 rounds. Additionally, column "% Min" corresponds to the frequency, out of the 10 rounds, that algorithm found the minimum primal bound from those rounds. For this set of experiments, we consider the LS stopping criteria to be a time limit of 60 seconds. This means that every LS round takes 60 seconds. However, at each round, LS only takes some time $t \leq 60$ to find the best solution of that round, which corresponds to the reported bound. So we record the amount of time, in seconds, that LS took to find the best solution in each round. Column "Time" corresponds to the average amount of time each LS round took to find the best solution of that round. The same reasoning is applied to Iterative LS, which was set to a time limit of 400 seconds. We note that when NDRC finds the optimal or best known solution, then its subsequent LS and Iterative LS rounds will return the same primal bound. In those cases, the results reported for LS and Iterative LS are omitted. Moreover, for any instance, when the same primal bound is found in all 10 rounds, we only report the minimum bound and omit the average and maximum, since all three are the same. Additionally, a row is included at the bottom of each table. In the case of Table 7.9, this row

is denoted by "Optimal", whereas for Table 7.10 this column is denoted by "BKS". For each column corresponding to an algorithm, the entry in row "Optimal" denotes the number of instances for which that algorithm found the optimal solution. Likewise, for Table 7.10, the entry in row "BKS" denotes the number of instances for which that algorithm found the best known solution taken from the literature, since optimality has not been proven.

We begin by analyzing the results from Tables 7.9 and 7.10. First, we place our attention to columns "LP", "LP+" and "LB". The LP dual bound reported in column "LP" is obtained by considering the SCP formulation containing only the total dominating set constraints and by solving its LP relaxation. In turn, the lagrangian dual reported in column "LB" obtained by NDRC is affected by the activation of vertex-cut constraints during NDRC, which hopefully should increase the dual bound. Column "LP+" is taken as a benchmark for "LB". According to the results reported in Tables 7.9 and 7.10, the activation of vertex-cut constraints does lead to increases in "LB" when compared to "LP". The majority of them are slight, but in a few cases they are more significant. In fact, for a few instances the dual bound increased to another integer level. Some examples include: *v50_d05*, from 21.0 to 24.42; *v70_d05*, from 21.75 to 22.94; *2000_300_200*, from 30.98 to 31.09; and *3000_400_210*, from 58.93 to 59.21. These increases are evidence that the activation of vertex-cuts during NDRC is contributing to the algorithm by bringing new and relevant information to the problem, and by consequently strengthening the lagrangian dual problem, as endorsed by the increase in the dual bound. Moreover, NDRC was successful in finding a near-optimal dual bound in reasonable time for all instances, getting close to the value reported in column "LP+".

We now consider the primal bound given by NDRC. Type I graphs have been solved to optimality and reported by GENDRON *et al.* [95]. On the other hand, Type II instances have not been solved to optimality, so for the sake of comparison we consider the BKS reported in the literature. For easier instances, usually meaning graphs with relatively fewer vertices or higher density, NDRC consistently reported the optimal primal bound, without the need of combination with LS for further improvement. On the other hand, for harder instances, specially for Type II graphs, NDRC did not match the BKS. This suggests that, as a primal heursitic, NDRC still requires improvement. As a line of future work, we consider incorporating the idea of a *core problem*, as suggested by CAPRARA *et al.* [8]. This consists of working on a reduced SCP instance obtained by heuristically choosing a small subset of columns. The core problem is solved iteratively by the subgradient method and the subset of columns composing the core problem is frequently updated. As reported by the authors, this approach is very successful in reducing the overall time to convergence to a near-optimal dual solution, as well as in finding primal solutions of good or

optimal quality.

We analyze the performance of LS and Iterative LS. Note that LS found the optimal solution for every Type I instance with a 10/10 frequency, meaning that it found an optimal solution in every round. Additionally, note that for 39 out of 41 Type II instances, LS matched the BKS bound in at least one round. The only two exceptions were *3000_400_210* and *3000_400_240*. More specifically, for instance *3000_400_210*, LS found a solution with objective value of 75 on seven rounds, and an average bound of 75.3, whereas a BKS of 74 is known. However, note that for this same instance, Iterative LS found a solution with objective value of 74 on eight rounds, and the average bound was 74.2. In a similar manner, for instance *3000_400_240*, LS found a solution with objective value of 61 on four rounds, and an average bound of 61.6, whereas a BKS of 60 is known. In turn, Iterative LS found a solution with objective value of 60 on three rounds, and an average bound of 60.7. This is supporting evidence of the effectiveness of our proposed Iterative LS as a primal heuristic. Not only was it successful in improving a sub-optimal solution to match the BKS, it did so very frequently. Besides the two instances for which LS did not match the BKS, there are exactly 14 instances for which it was able to match the BKS bound, but not on every run. For five of these instances, Iterative LS found the BKS with a frequency of 10/10. And for the other nine, it led to an improvement in the frequency in comparison with LS, including some significant ones. For example: *2500_350_210*, from 2/10 rounds for LS to 8/10 for Iterative LS; *2500_350_220*, from 2 to 7; *2500_350_230*, from 2 to 8; *3000_400_220*, from 2 to 9. Moreover, for nine of these fourteen instances, Iterative LS reduced the maximum upper bound compared to LS. One example is instance *3000_400_220*. In this case, LS found the BKS of 70 on exactly two rounds. On the other eight rounds, the reported bound was either 71 or 72. In contrary, Iterative LS reported the BKS on nine rounds, and reported 71 for the remaining round. Additionally, note that of the 41 Type II instances: for exactly one, the maximum deviation from the BKS was 2; exactly eleven, it was one; exactly twenty nine, the BKS was found on all ten rounds.

All in all, our results show that the combination of LS and Iterative LS is a very effective and robust method for generating high-quality MCDSP primal solutions. In fact, for all 41 of Type I instances and all 41 of Type II instances, this combination was successful in matching the BKS from the literature. Moreover, considering all instances, the maximum deviation from the BKS was at most 2.

Instance	OPT	LP	LP+	NDRC			Local Search		
				LB	UB	Time	Min UB	% Min UB	Time
v30_d10	15	12	12	12	15	0.07	-	-	-
v30_d20	7	5.87	5.92	7	7	0.02	-	-	-
v30_d30	4	3.31	3.31	4	4	0.00	-	-	-
v30_d50	3	2.01	2.01	3	3	0.00	-	-	-
v30_d70	2	1.43	1.43	2	2	0	-	-	-
v50_d05	31	21	24.75	24.75	31	0.27	-	-	-
v50_d10	12	10.23	10.32	10.32	12	0.12	12	10/10	<0.01
v50_d20	7	5.21	5.21	5.21	7	0.06	-	-	-
v50_d30	5	3.46	3.46	3.46	5	0.06	-	-	-
v50_d50	3	2.05	2.05	3	3	0.00	-	-	-
v50_d70	2	1.46	1.46	2	2	0	-	-	-
v70_d05	27	21.75	22.96	22.94	27	0.34	27	10/10	<0.01
v70_d10	13	11.05	11.07	11.07	13	0.13	-	-	-
v70_d20	7	5.29	5.29	5.29	7	0.10	-	-	-
v70_d30	5	3.37	3.37	3.37	5	0.08	-	-	-
v70_d50	3	2.08	2.08	3	3	0.00	-	-	-
v70_d70	2	1.45	1.45	2	2	0.03	-	-	-
v100_d05	24	20.43	20.63	20.62	24	0.44	24	10/10	<0.01
v100_d10	13	10.86	10.86	10.86	13	0.16	13	10/10	<0.01
v100_d20	8	5.35	5.35	5.35	8	0.15	-	-	-
v100_d30	6	3.43	3.43	3.43	6	0.16	-	-	-
v100_d50	4	2.12	2.12	2.12	4	0.18	-	-	-
v100_d70	3	1.46	1.46	1.47	3	0.22	-	-	-
v120_d05	25	22.35	22.37	22.37	25	0.30	25	10/10	<0.01
v120_d10	13	10.51	10.51	10.51	13	0.21	13	10/10	<0.01
v120_d20	8	5.29	5.29	5.28	8	0.18	-	-	-
v120_d30	6	3.43	3.43	3.43	6	0.19	-	-	-
v120_d50	4	2.02	2.02	2.02	4	0.25	-	-	-
v120_d70	3	1.44	1.44	1.44	3	0.31	-	-	-
v150_d05	26	21.48	21.55	21.55	26	0.42	26	10/10	<0.01
v150_d10	15	10.77	10.77	10.77	15	0.23	14	10/10	<0.01
v150_d20	9	5.20	5.20	5.20	9	0.27	-	-	-
v150_d30	6	3.48	3.48	3.48	6	0.29	-	-	-
v150_d50	4	2.00	2.00	2.00	4	0.38	-	-	-
v150_d70	3	1.45	1.45	1.45	3	0.49	-	-	<0.01
v200_d05	27	22.39	22.39	22.39	28	0.44	27	10/10	<0.01
v200_d10	16	10.56	10.56	10.56	16	0.31	16	10/10	<0.01
v200_d20	9	5.02	5.02	5.02	9	0.38	-	-	-
v200_d30	7	3.37	3.37	3.37	7	0.45	-	-	-
v200_d50	4	2.02	2.02	2.02	4	0.61	-	-	-
v200_d70	3	1.45	1.45	1.45	3	0.83	-	-	-
Optimal					31/41		41/41		

Table 7.9: Results of NDRC and LS for solving MCDSP on Type I graphs.

Instance	BKS	LP	LP$^+$	NDRC			Local Search					Iterative Local Search				
				LB	UB	Time	Min UB	Avg UB	Max UB	% Min UB	Time	Min UB	Avg UB	Max UB	% Min UB	Time
400_80_60	18	14.28	14.56	14.56	18	0.57	18	-	-	10/10	<0.01	-	-	-	-	-
400_80_70	14	11.11	11.17	11.16	14	0.39	14	-	-	10/10	<0.01	-	-	-	-	-
400_80_80	12	9.14	9.17	9.16	12	0.31	12	-	-	10/10	<0.01	-	-	-	-	-
400_80_90	10	7.90	8.11	8.08	10	0.37	10	-	-	10/10	<0.01	-	-	-	-	-
400_80_100	8	6.37	6.47	6.47	8	0.33	8	-	-	10/10	<0.01	-	-	-	-	-
400_80_110	7	5.50	5.60	5.58	7	0.26	7	-	-	10/10	<0.01	-	-	-	-	-
400_80_120	6	4.97	5.02	6	6	0.23	6	-	-	10/10	<0.01	-	-	-	-	-
600_100_80	21	16.45	16.48	16.48	21	1.09	21	-	-	10/10	<0.01	-	-	-	-	-
600_100_90	19	15.16	15.26	15.22	20	0.80	19	-	-	10/10	<0.01	-	-	-	-	-
600_100_100	16	12.29	12.35	12.33	17	0.76	16	-	-	10/10	<0.01	-	-	-	-	-
600_100_110	14	10.94	10.97	10.96	15	0.73	14	-	-	10/10	0.1	-	-	-	-	-
600_100_120	13	9.64	9.67	9.66	13	0.61	13	-	-	10/10	<0.01	-	-	-	-	-
700_200_70	38	29.85	30.04	30.02	42	3.65	38	38.2	39	8/10	11.0	38	38.1	39	9/10	5.7
700_200_80	32	24.13	24.14	24.13	35	2.81	32	-	-	10/10	1.2	-	-	-	-	-
700_200_90	26	20.14	20.19	20.16	30	2.52	26	-	-	10/10	0.2	-	-	-	-	-
700_200_100	22	17.31	17.38	17.36	24	2.11	22	-	-	10/10	1.1	-	-	-	-	-
700_200_110	20	15.12	15.12	15.12	22	1.81	20	-	-	10/10	0.2	-	-	-	-	-
700_200_120	17	12.82	12.86	12.83	19	1.99	17	-	-	10/10	0.2	-	-	-	-	-
1000_200_100	38	29.85	30.04	30.02	42	3.58	38	38.2	39	8/10	10.6	38	38.1	39	9/10	65.6
1000_200_110	34	25.49	25.52	25.51	37	3.07	34	-	-	10/10	0.9	-	-	-	-	-
1000_200_120	29	22.55	22.56	22.55	32	2.67	29	29.1	30	9/10	13.1	29	-	-	10/10	6.4
1000_200_130	26	19.89	19.92	19.91	29	2.72	26	-	-	10/10	0.3	-	-	-	-	-
1000_200_140	23	17.68	17.70	17.69	25	2.10	23	-	-	10/10	0.5	-	-	-	-	-
1000_200_150	21	16.36	16.41	16.40	23	2.24	21	-	-	10/10	2.0	-	-	-	-	-
1000_200_160	19	14.45	14.49	14.47	21	2.01	19	-	-	10/10	3.6	-	-	-	-	-
1500_250_130	49	38.24	38.35	38.31	54	5.52	49	49.2	50	8/10	6.8	49	-	-	10/10	16.5
1500_250_140	43	33.60	33.67	33.63	48	4.45	43	43.9	44	1/10	5.6	43	43.4	44	6/10	112.5
1500_250_150	40	30.33	30.41	30.38	44	4.58	40	40.4	41	6/10	5.2	40	40.2	41	8/10	45.2
1500_250_160	36	27.28	27.34	27.33	41	4.26	36	-	-	10/10	2.1	-	-	-	-	0
2000_300_200	41	30.98	31.13	31.10	47	5.70	42	-	-	10/10	4.4	41	41.1	42	9/10	48.0
2000_300_210	38	28.60	28.69	28.67	43	5.34	38	38.5	39	5/10	2.4	38	-	-	10/10	21.7
2000_300_220	35	26.61	26.67	26.65	40	4.77	35	-	-	10/10	9.7	-	-	-	-	-
2000_300_230	33	24.93	25.01	24.95	38	5.07	33	33.3	34	7/10	14.8	33	-	-	10/10	29.6
2500_350_200	60	46.25	46.38	46.33	67	8.37	60	60.5	61	5/10	11.9	60	-	-	10/10	81.6
2500_350_210	54	43.29	43.47	43.43	63	8.16	54	55.3	56	2/10	6.4	54	54.3	56	8/10	31.5
2500_350_220	51	40.11	40.21	40.15	59	7.87	51	52.1	53	2/10	14.1	51	51.3	52	7/10	101.2
2500_350_230	48	37.41	37.55	37.47	57	7.39	48	49.2	50	2/10	12.7	48	48.2	49	8/10	68.2
3000_400_210	74	58.93	59.33	59.25	84	10.16	75	75.3	76	7/10	14.7	74	74.2	75	8/10	163.0
3000_400_220	70	54.26	54.53	54.45	78	10.73	70	71	72	2/10	11.8	70	70.1	71	9/10	109.6
3000_400_230	64	50.65	50.85	50.77	72	9.68	64	65.3	66	1/10	9.4	64	64.9	65	1/10	30.9
3000_400_240	60	46.95	47.04	46.98	69	8.80	61	61.6	62	4/10	7.7	60	60.7	61	3/10	96.6
BKS					9/41		38/41					41/41				

Table 7.10: Results of NDRC and LS for solving MCDSP on Type II graphs.

Comparsion with other MCDSP algorithms

To further assess the performance of our proposed algorithm LS as a primal heuristic for SCP, and specifically for MCDSP, we compare our results with state-of-the-art MCDSP algorithms. Namely, we consider the VDNS of WANG *et al.* [114], RNS-TS of WU *et al.* [111], the RVNS of BOUAMAMA *et al.* [115] and the NuCDS of [113]. Since these algorithms are randomized, following the standard in the literature, for each instance and each algorithm, a total of 10 rounds are performed and the minimum and average bounds are reported. The same is done for our proposed LS and Iterative LS algorithms.

We remark that, in reality, we have only run experiments for the NDRC, LS and Iterative LS algorithms, whereas the other results correspond to what has been reported in their original papers. In particular, we mention that running times are obtained on different computational environments with distinct processing power. The times corresponding to VDNS are as reported in LI *et al.* [112]. The times corresponding to RNS-TS are reported in [111]. The times corresponding to RVNS are reported in [115]. Running times are not reported for NuCDS. However, in their experiments, the authors set the time limit of NuCDS to 1000 seconds. In order to account for the different computational environments and processing capabilities, and to lead to a fairer comparison of execution times, the reported times are normalized. More specifically, for each CPU, we consider its so-called *mark* from the *PassMark Single Thread Performance* list, which is available on [117]. For a given CPU, its mark is a measure of its single-threaded processing capability, with a higher mark denoting a better capability. The CPU used for running NDRC, LS and Iterative LS is taken as reference, and the times reported in other CPUs are normalized by multiplying it by a constant factor which depends on the marks of both CPUs. More specifically, let M_0 be the mark of the reference CPU. Consider any other CPU with mark M. Then the factor for normalizing times from this CPU to the reference CPU is given by $\frac{M}{M_0}$. In this way, a reported time t is normalized to $t \cdot \frac{M}{M_0}$. The specific computational settings at which each algorithm was executed are presented in Table 7.11. Additionally, we present their corresponding single-threaded CPU marks and the normalization factors.

Algorithm	Reference	CPU	Frequency	Mark	Factor
VDNS	WANG *et al.* [114]	Intel i3	2.1	1727	0.756
RNS-TS	WU *et al.* [111]	Intel i3	3.0	1727	0.756
RVNS	BOUAMAMA *et al.* [115]	Intel Xeon 5670	2.93	1400	0.613
NuCDS	LI *et al.* [113]	Intel Xeon E7-4830	2.0	1012	0.443
Ours		Intel Core i7-10510U	1.8	2283	1

Table 7.11: Different computational enviroments on which each algorithm was executed, their single-threaded marks and normalization factors to our CPU.

In [113], the time limit for NuCDS is set to 1000 seconds. From the algorithms from the literature considered here for comparison, NuCDS shows the best overall performance. For this reason, to allow for a fair compasion between our approach with NuCDS, we consider this time limit, but normalized. Considering the CPU marks presented in Table 7.11, this is equivalent to 443 seconds on our CPU. For this reason, our experiments are set to comply with this normalized time limit. For each round, first NDRC is run, and its running time t_0 is recorded. Then LS is run with time limit of 60 seconds. Finally, Iterative LS is run with time limit of $443 - t_0 - 60$.

The results are presented in Table 7.12. Each row corresponds to an instance. For each of the aforementioned algorithms, there is a column with the results obtained by each algorithm. Additionally, for each algorithm, columns "Min", "Avg" correspond to the minimum and average primal bound found during 10 rounds of that algorithm. Column "Time" corresponds to the average amount of time to find the best solution in each round. Since the results reported for LS are in fact a combination of NDRC with LS, the times reported for LS in Table 7.12 correspond to the total NDRC running time in addition to the time taken for LS to find the best solution in each round. In a similar manner, the results reported for Iterative LS comprise a sum of NDRC, LS and Iterative LS running times. Additionally, a row is included at the bottom of the table, denoted by "Better or equal". For each column corresponding to an algorithm and for columns "Min" and "Avg", the entry in row "Better or equal" denotes the number of instances for which the value reported by that algorithm was strictly better than all other algorithms or equal to the best value reported.

We now analyze the results. Compared to these algorithms, our proposed LS and Iterative LS perform very well. In fact, considering the reported minimum bounds, note that: our Iterative LS ranks first, finding a better or equal minimum bound on 41/41 instances; NuCDS ranks second, with 40/41; our LS ranks third, with 38/41; RVNS ranks fourth, with 37/41; RSN-TS ranks fifth, with 30/41. In turn, when the average bounds are considered, note that: both our Iterative LS and NuCDS tie in first, finding a better or equal average bound on 36/41 instances; RSN-TS ranks

third, with 29/41; RVNS ranks fourth, with 27/41; our LS ranks fifth, with 24/41. Note that, despite having the highest average deviation from the BKS, our LS is also the fastest among these methods, with reported average times varying up to 25 seconds. Although running times are not reported for NuCDS, considering that it was run with a time limit of 443 (normalized) seconds, and by considering the average times reported for RVNS and RSN-TS, note that our LS performs similarly to NuCDS and better than RVNS and RSN-TS in drastically shorter amount of time, as far as the minimum bound is concerned. Whereas LS is effective in producing high-quality solutions in a short amount of time, our results show that by combining it with Iterative LS, solutions are frequently further improved and the deviation from the BKS is reduced. In fact, Iterative LS is the only one to find the BKS for every Type II instance. Additionally, when the average bounds are considered, it ties in first place with NuCDS, having reported the better average bound on 36/41 instances. This means that there were only five instances for which the average bound reported by Iterative LS was greater than the best average from the other algorithms. Not only does Iterative LS perform better than all other algorithms when minimum and average bounds are considered, its reported times are also better or at least comparable with the others. Note that Iterative LS, RVNS and RSN-TS have reported average times varying up to 155, 106 and 214 seconds, respectively. Moreover, note that the average running times reported for Iterative LS are significantly lower than the time limit of 443 (normalized) seconds set for NuCDS.

In conclusion, our proposed LS performs nearly as well as NuCDS and significantly better than RVNS and RSN-TS in producing high-quality solutions, but does so in significantly less time. However, LS shows an increased deviation from the BKS. On the other hand, given that LS is a fast and effective heuristic procedure, it can be successfully combined with Iterative LS, leading to a more robust method. In fact, our results show that the combination of LS and Iterative LS outperformed all other MCDSP algorithms in producing high-quality primal solutions, being the only algorithm to find the BKS for every Type II instance. Additionally, its superior effectiveness as a primal heuristic does not come at the price of higher execution time, since Iterative LS times are on par with the other methods. All in all, in comparison with state-of-the-art MCDSP algorithms such as NuCDS, RVNS and RSN-TS, our results suggest that Iterative LS is the most effective algorithm for producing high-quality MCDSP solutions in reasonable amount of time.

Instance	BKS	Our Iterative LS			Our LS			NuCDS		RVNS			RSN-TS		
		Min	Avg	Time	Min	Avg	Time	Min	Avg	Min	Avg	Time	Min	Avg	Time
400_80_60	18	18	18	0.6	18	18	0.574	18	18	18	18	0.0	18	18	<0.01
400_80_70	14	14	14	0.4	14	14	0.398	14	14	14	14	0.18	14	14	<0.01
400_80_80	12	12	12	0.3	12	12	0.314	12	12	12	12	0.0	12	12	<0.01
400_80_90	10	10	10	0.4	10	10	0.373	10	10	10	10	0.0	10	10	<0.01
400_80_100	8	8	8	0.3	8	8	0.335	8	8	8	8	0.24	8	8	<0.01
400_80_110	7	7	7	0.3	7	7	0.263	7	7	7	7	0.0	7	7	1.512
400_80_120	6	6	6	0.2	6	6	0.236	6	6	6	6	0.24	6	6	0.756
600_100_80	21	21	21	1.1	21	21	1.092	21	21	21	21	0.06	21	21	<0.01
600_100_90	19	19	19	0.8	19	19	0.812	19	19	19	19	0.0	19	19	0.756
600_100_100	16	16	16	0.8	16	16	0.776	16	16	16	16	0.12	16	16	0.756
600_100_110	14	14	14	0.9	14	14	0.867	14	14	14	14	2.14	15	15	<0.01
600_100_120	13	13	13	0.6	13	13	0.609	13	13	13	13	0.06	13	13	<0.01
700_200_70	38	38	38.1	15.2	38	38.2	14.670	38	38.1	38	38.1	31.2	39	39	12.85
700_200_80	32	32	32	4.0	32	32	4.011	32	32	32	32	17.7	32	32	11.34
700_200_90	26	26	26	2.8	26	26	2.761	26	26	26	26	4.84	26	26	3.780
700_200_100	22	22	22	3.2	22	22	3.239	22	22	22	22	15.9	22	22	46.11
700_200_110	20	20	20	2.0	20	20	1.988	20	20	20	20	0.79	20	20	2.268
700_200_120	17	17	17	2.2	17	17	2.162	17	17	17	17	2.08	17	17	8.316
1000_200_100	38	38	38.1	20.7	38	38.2	14.145	38	**38**	38	38.2	14.8	38	38.4	93.74
1000_200_110	34	34	34	4.0	34	34	3.987	34	34	34	34	1.28	34	34	6.804
1000_200_120	29	29	29	16.4	29	29.1	15.806	29	29	29	29	29.4	29	29	65.01
1000_200_130	26	26	26	3.0	26	26	3.025	26	26	26	26	4.22	26	26	9.072
1000_200_140	23	23	23	2.5	23	23	2.521	23	23	23	23	1.34	23	23	3.780
1000_200_150	21	21	21	4.2	21	21	4.223	21	21	21	21	6.13	21	21	6.804
1000_200_160	19	19	19	5.6	19	19	5.607	19	19	19	19	17.0	19	19	53.67
1500_250_130	49	49	49	15.7	49	49.2	12.362	49	49	49	49.1	28.1	49	49	83.91
1500_250_140	43	43	**43.4**	66.3	43	43.9	10.060	43	43.7	43	43.9	5.63	44	44	14.36
1500_250_150	40	40	40.2	18.9	40	40.4	9.825	40	40	40	40.7	25.6	40	40	68.79
1500_250_160	36	36	36	6.4	36	36	6.397	36	36	36	36	2.69	36	36	23.43
2000_300_200	41	41	**41.1**	53.3	42	42	10.057	41	41.5	42	42.1	12.9	42	42	102.8
2000_300_210	38	38	38	18.6	38	38.5	7.701	38	38	38	38	24.2	38	38.3	76.35
2000_300_220	35	35	35	14.5	35	35	14.466	35	35	35	35.1	23.6	35	35	101.3
2000_300_230	33	33	33	28.7	33	33.3	19.853	33	33	33	33	35.7	33	33	214.7
2500_350_200	60	60	60	61.1	60	60.5	20.299	60	60	60	60.7	66.4	61	61.1	90.72
2500_350_210	54	54	**54.3**	36.7	54	55.3	14.603	54	54.9	55	56.1	29.1	56	56	89.96
2500_350_220	51	51	51.3	82.7	51	52.1	22.003	51	**51.1**	51	52.7	66.2	52	52	92.98
2500_350_230	48	48	**48.2**	67.8	48	49.2	20.078	49	49.1	48	49.5	46.0	50	50	100.5
3000_400_210	74	74	74.2	155.2	75	75.3	24.876	74	**74**	74	75.5	106.4	75	75.3	194.2
3000_400_220	70	70	**70.1**	99.3	70	71	22.523	70	**70**	70	70.8	52.4	71	71	97.52
3000_400_230	64	64	**64.9**	31.4	64	65.3	19.086	64	**64.8**	65	65.9	54.9	65	65.6	112.6
3000_400_240	60	**60**	**60.7**	84.2	61	61.6	16.543	60	60.9	61	61.7	90.1	61	61.5	195.0
Better or equal		41/41	36/41		38/41	24/41		40/41	36/41	37/41	27/41		30/41	29/41	

Table 7.12: Comparison of our proposed algorithms with state-of-the-art MCDSP algorithms on Type II graphs.

Chapter 8

Conclusions and Future Work

The set cover problem is a fundamental integer optimization problem. Due to its relevance in a wide range of applications, as well as the observed practical difficulty in finding good or optimal SCP solutions specially for medium and large-scale instances, many heuristic approaches have been investigated for SCP. After decades of research, many ideas were investigated, and some very effective algorithms have been developed. Furthermore, mechanisms were incorporated to handle very large-scale instances efficiently. Consequently, state-of-the-art SCP heuristics are able to produce solutions of very good quality in reasonable amount of time, even for large-scale instances with up to millions of columns. Considering the advanced state of SCP algorithms, we consider that by formulating an optimization problem in the form of SCP can be advantageous since it could benefit from the developments made for SCP. For example, the original integer programming formulation of the min connected dominating set problem was not in the form of an SCP. On the other hand, BUCHANAN *et al.* [25] demonstrated a characterization of connected dominating sets that allowed MCDSP to be formulated as SCP. In a similar manner, in this work, we present characterizations of connected vertex-covers and connected edge-dominating sets, presented in Section 3.1. Our characterizations also allow the min connected vertex-cover and min edge-dominating set problems to be formulated as SCP. With this motivation, we investigated the use of state-of-the-art SCP algorithms to solve these combinatorial optimizations problems. However, considering that these problems are formulated as SCP with exponentially many constraints, adaptations to the usual SCP algorithms were necessary in order to make this approach viable.

Considering that the lagrangian heuristic of CAPRARA *et al.* [8] is one of the best performing SCP heuristics, in Section 5.4 we propose a non-delayed relax-and-cut algorithm to solve the lagrangian dual of SCP with exponentially many constraints. Our NDRC allows the dynamic dualization of constraints, constituting a lagrangian analogous of a cutting plane algorithm. One common issue of the sub-

gradient method is the zig-zagging phenomena, which affects the overall convergence of SM and consequently its running time. In an attempt to reduce zig-zagging, in Section 4.3 we propose a strengthened formulation for the lagrangian dual of SCP. The results presented in Section 7.2 show that the strengthened lagrangian dual was successful in attenuating the zig-zag behaviour and consequently reducing the number of subgradient iterations and total running time for convergence to near-optimal lagrangian multipliers. On the other hand, the increase in time complexity for solving the strengthened lagrangian subproblem led to an increase in running time in some cases. The increases were more frequent on smaller-scale instances, but our results show that as the instance size and/or its density increase, our proposed strengthened formulation is advantegeous, and in practice leads to significant performance gains. The current strengthened formulation leads to a lagrangian subproblem whose solution requires partial sorting, leading to a $\mathcal{O}(n \log K)$ worst-case complexity. One line of future work is to investigate alternatives approaches for formulating a stronger lagrangian subproblem with possibly lower complexity.

Furthermore, since our focus is in effectively producing good quality primal solutions, we propose a local-search algorithm for SCP with exponentially many constraints, which is described in Section 6.1. Additionally, we propose an Iterative Local Search algorithm, which successively reduces the problem size and employs our proposed local-search to try to improve the best known solution. In Section 7.3 we employ our proposed NDRC, LS and Iterative LS algorithms to solve benchmark instances of MCDSP. Comparison with state-of-the-art MCDSP algorithms reported in the literature suggest that our approach is the most effective in finding solutions of good quality and is performs better than the other algorithms reported in the literature. By combining our NDRC, LS and Iterative LS methods, we were able to find the best known MCDSP solution for all of the 41 Type II graphs. In comparison with state-of-the-art MCDSP algorithms, our method is the first one to achieve this.

Our results show that our algorithms perform well for small and medium sized SCP and MCDSP instances. However, no special treatment was made for handling large-scale ones. Inspired by the works of CAPRARA *et al.* [8], YAGIURA *et al.* [10] and LI *et al.* [113], we aim to incorporate into our algorithms techniques for handling large-scale instances more efficiently. In that matter, we believe that a promising line of future work is to incorporate the idea of a core problem into the algorithms, with the goal of improving its efficiency as well as its effectiveness as a primal heuristic. This idea was originally suggested for SCP by CAPRARA *et al.* [8] and consists of successively defining a reduced SCP, called the core problem, by choosing a small subset of columns at a time. By working with a small portion of the columns at each time, the subgradient method converges significantly faster, and the lagrangian heuristic is also more effective. This idea was successfully incorporated in the local-

search procedures of YAGIURA *et al.* [10] and GAO *et al.* [34], with the addition of heuristically fixing columns to have value 1, hence covering some SCP constraints and reducing the number of rows. By iteratively updating the core problem and applying local-search, each round of local-search considers a different SCP problem. Besides the reduction in complexity and time obtained by reducing the problem, this also favors diversification, and the results reported by the authors show the method is among the most effective SCP heuristics. We believe that incorporating this idea into our NDRC and LS will not only be helpful in improving the effectiveness of our algorithms in producing good primal solutions, but also lead to a significant reduction in runnning time. This is a fundamental step in making both NDRC and LS more efficient and becoming better suited for solving domination and cover problems on massive graphs.

References

[1] BEASLEY, J. E. "OR-Library: distributing test problems by electronic mail", *Journal of the operational research society*, v. 41, n. 11, pp. 1069–1072, 1990.

[2] SIMONETTI, L., SALLES DA CUNHA, A., LUCENA, A. "The minimum connected dominating set problem: Formulation, valid inequalities and a branch-and-cut algorithm". In: *International conference on network optimization*, pp. 162–169. Springer, 2011.

[3] JOVANOVIC, R., TUBA, M. "Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem", *Computer Science and Information Systems*, v. 10, n. 1, pp. 133–149, 2013.

[4] KARP, R. M. *Reducibility among combinatorial problems*. Springer, 2010.

[5] KORTE, B. H., VYGEN, J., KORTE, B., et al. *Combinatorial optimization*, v. 1. Springer, 2011.

[6] GAREY, M. R. "Computers and intractability: A guide to the theory of np-completeness, freeman", *Fundamental*, 1997.

[7] CHVATAL, V. "A greedy heuristic for the set-covering problem", *Mathematics of operations research*, v. 4, n. 3, pp. 233–235, 1979.

[8] CAPRARA, A., FISCHETTI, M., TOTH, P. "A Heuristic Algorithm for Very-Large Scale Set Covering Problems". In: *Operations Research Proceedings 1995: Selected Papers of the Symposium on Operations Research (SOR'95), Passau, September 13–September 15, 1995*, pp. 48–53. Springer, 1996.

[9] CERIA, S., NOBILI, P., SASSANO, A. "A Lagrangian-based heuristic for large-scale set covering problems", *Mathematical Programming*, v. 81, pp. 215–228, 1998.

[10] YAGIURA, M., KISHIDA, M., IBARAKI, T. "A 3-flip neighborhood local search for the set covering problem", *European Journal of Operational Research*, v. 172, n. 2, pp. 472–499, 2006.

[11] GAO, C., YAO, X., WEISE, T., et al. "An efficient local search heuristic with row weighting for the unicost set covering problem", *European Journal of Operational Research*, v. 246, n. 3, pp. 750–761, 2015.

[12] WEDELIN, D. "An algorithm for large scale 0–1 integer programming with application to airline crew scheduling", *Annals of operations research*, v. 57, n. 1, pp. 283–301, 1995.

[13] CAPRARA, A., TOTH, P., VIGO, D., et al. "Modeling and solving the crew rostering problem", *Operations research*, v. 46, n. 6, pp. 820–830, 1998.

[14] RASMUSSEN, M. S., JUSTESEN, T., DOHN, A., et al. "The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies", *European journal of operational research*, v. 219, n. 3, pp. 598–610, 2012.

[15] FOSTER, B. A., RYAN, D. M. "An integer programming approach to the vehicle scheduling problem", *Journal of the Operational Research Society*, v. 27, pp. 367–384, 1976.

[16] SALVESON, M. E. "The assembly-line balancing problem", *Transactions of the American Society of Mechanical Engineers*, v. 77, n. 6, pp. 939–947, 1955.

[17] BAUTISTA, J., PEREIRA, J. "Modeling the problem of locating collection areas for urban waste management. An application to the metropolitan area of Barcelona", *Omega*, v. 34, n. 6, pp. 617–629, 2006.

[18] DI RICERCA OPERATIVA, A. I. "Ferrovie dello Stato SpA", *Metodi di Ottimizzazione per la Costruzione dei Turni-FARO, Bando di Concorso*, 1994.

[19] PEZZELLA, F., FAGGIOLI, E. "Solving large set covering problems for crew scheduling", *Top*, v. 5, n. 1, pp. 41–59, 1997.

[20] CAPRARA, A., FISCHETTI, M., TOTH, P., et al. "Algorithms for railway crew management", *Mathematical programming*, v. 79, pp. 125–141, 1997.

[21] CAPRARA, A., FISCHETTI, M., GUIDA, P. L., et al. "Solution of large-scale railway crew planning problems: The Italian experience". In: *Computer-Aided Transit Scheduling: Proceedings, Cambridge, MA, USA, August 1997*, pp. 1–18. Springer, 1999.

[22] FUJIE, T. "The maximum-leaf spanning tree problem: Formulations and facets", *Networks: An International Journal*, v. 43, n. 4, pp. 212–223, 2004.

[23] FUJITO, T. "How to trim an MST: A 2-approximation algorithm for minimum cost tree cover". In: *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I 33*, pp. 431–442. Springer, 2006.

[24] LUCENA, A., MACULAN, N., SIMONETTI, L. "Reformulations and solution algorithms for the maximum leaf spanning tree problem", *Computational Management Science*, v. 7, n. 3, pp. 289–311, 2010.

[25] BUCHANAN, A., SUNG, J. S., BUTENKO, S., et al. "An integer programming approach for fault-tolerant connected dominating sets", *INFORMS Journal on Computing*, v. 27, n. 1, pp. 178–188, 2015.

[26] CHEN, L., HUANG, X., ZHANG, Z. "A simpler PTAS for connected k-path vertex cover in homogeneous wireless sensor network", *Journal of Combinatorial Optimization*, v. 36, n. 1, pp. 35–43, 2018.

[27] SLATER, P. J. "On locating a facility to service areas within a network", *Operations Research*, v. 29, n. 3, pp. 523–531, 1981.

[28] MINIEKA, E. "The optimal location of a path or tree in a tree network", *Networks*, v. 15, n. 3, pp. 309–321, 1985.

[29] RICHEY, M. B. "Optimal location of a path or tree on a network with cycles", *Networks*, v. 20, n. 4, pp. 391–407, 1990.

[30] HAKIMI, S. L., SCHMEICHEL, E. F., LABBÉ, M. "On locating path-or tree-shaped facilities on networks", *Networks*, v. 23, n. 6, pp. 543–555, 1993.

[31] SLATER, P. J. "Locating central paths in a graph", *Transportation Science*, v. 16, n. 1, pp. 1–18, 1982.

[32] KLOSE, A. "A Lagrangean relax-and-cut approach for the two-stage capacitated facility location problem", *European journal of operational research*, v. 126, n. 2, pp. 408–421, 2000.

[33] DA CUNHA, A. S., LUCENA, A., MACULAN, N., et al. "A relax-and-cut algorithm for the prize-collecting Steiner problem in graphs", *Discrete Applied Mathematics*, v. 157, n. 6, pp. 1198–1217, 2009.

[34] GAO, C., WEISE, T., LI, J. "A weighting-based local search heuristic algorithm for the set covering problem". In: *2014 IEEE Congress on Evolutionary Computation (CEC)*, pp. 826–831. IEEE, 2014.

[35] WOLSEY, L. A. *Integer programming*. John Wiley & Sons, 2020.

[36] MEINDL, B., TEMPL, M. "Analysis of commercial and free and open source solvers for linear optimization problems", *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS*, v. 20, pp. 64, 2012.

[37] MITTEN, L. "Branch-and-bound methods: General formulation and properties", *Operations Research*, v. 18, n. 1, pp. 24–34, 1970.

[38] CHRISTOFIDES, N., KORMAN, S. "Note—A computational survey of methods for the set covering problem", *Management Science*, v. 21, n. 5, pp. 591–599, 1975.

[39] LEMKE, C. E., SALKIN, H. M., SPIELBERG, K. "Set covering by single-branch enumeration with linear-programming subproblems", *Operations Research*, v. 19, n. 4, pp. 998–1022, 1971.

[40] PIERCE, J. F. "Application of combinatorial programming to a class of all-zero-one integer programming problems", *Management Science*, v. 15, n. 3, pp. 191–209, 1968.

[41] PIERCE, J. F., LASKY, J. S. "Improved combinatorial programming algorithms for a class of all-zero-one integer programming problems", *Management Science*, v. 19, n. 5, pp. 528–543, 1973.

[42] FISHER, M. L. "The Lagrangian relaxation method for solving integer programming problems", *Management science*, v. 27, n. 1, pp. 1–18, 1981.

[43] BALAS, E., HO, A. "Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study", *Math. Programming Study*, v. 12, pp. 37–60, 1980.

[44] BEASLEY, J. E. "An algorithm for set covering problem", *European Journal of Operational Research*, v. 31, n. 1, pp. 85–93, 1987.

[45] BALAS, E., CARRERA, M. C. "A dynamic subgradient-based branch-and-bound procedure for set covering", *Operations research*, v. 44, n. 6, pp. 875–890, 1996.

[46] BEASLEY, J. E., JÖRNSTEN, K. "Enhancing an algorithm for set covering problems", *European Journal of Operational Research*, v. 58, n. 2, pp. 293–300, 1992.

[47] NOBILI, P., SASSANO, A. "A Separation Routine for the Set Covering Polytope." In: Balas, E., Cornuéjols, G., Kannan, R. (Eds.), *IPCO*, pp. 201–219. Carnegie Mellon University, 1992. Available at: <`http://dblp.uni-trier.de/db/conf/ipco/ipco1992.html#NobiliS92`>.

[48] CAPRARA, A., TOTH, P., FISCHETTI, M. "Algorithms for the set covering problem", *Annals of Operations Research*, v. 98, n. 1-4, pp. 353–371, 2000.

[49] NEMHAUSER, G. L., SAVELSBERGH, M. W., SIGISMONDI, G. C. "MINTO, a mixed INTeger optimizer", *Operations Research Letters*, v. 15, n. 1, pp. 47–58, 1994.

[50] FEO, T. A., RESENDE, M. G. "A probabilistic heuristic for a computationally difficult set covering problem", *Operations research letters*, v. 8, n. 2, pp. 67–71, 1989.

[51] CAPRARA, A., FISCHETTI, M., TOTH, P. "A heuristic method for the set covering problem", *Operations research*, v. 47, n. 5, pp. 730–743, 1999.

[52] BAUTISTA, J., PEREIRA, J. "A GRASP algorithm to solve the unicost set covering problem", *Computers & Operations Research*, v. 34, n. 10, pp. 3162–3173, 2007.

[53] PESSOA, L. S., RESENDE, M. G., RIBEIRO, C. C. "A hybrid Lagrangean heuristic with GRASP and path-relinking for set k-covering", *Computers & Operations Research*, v. 40, n. 12, pp. 3132–3146, 2013.

[54] MUSLIU, N. "Local search algorithm for unicost set covering problem". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pp. 302–311. Springer, 2006.

[55] NAJI-AZIMI, Z., TOTH, P., GALLI, L. "An electromagnetism metaheuristic for the unicost set covering problem", *European Journal of Operational Research*, v. 205, n. 2, pp. 290–300, 2010.

[56] BEASLEY, J. E., CHU, P. C. "A genetic algorithm for the set covering problem", *European journal of operational research*, v. 94, n. 2, pp. 392–404, 1996.

[57] BRUSCO, M. J., JACOBS, L. W., THOMPSON, G. M. "A morphing procedure to supplement a simulated annealing heuristic for cost-andcoverage-correlated set-covering problems", *Annals of Operations Research*, v. 86, n. 0, pp. 611–627, 1999.

[58] JACOBS, L. W., BRUSCO, M. J. "Note: A local-search heuristic for large set-covering problems", *Naval Research Logistics (NRL)*, v. 42, n. 7, pp. 1129–1140, 1995.

[59] KRITTER, J., BRÉVILLIERS, M., LEPAGNOT, J., et al. "On the optimal placement of cameras for surveillance and the underlying set cover problem", *Applied Soft Computing*, v. 74, pp. 133–153, 2019.

[60] BRÉVILLIERS, M., LEPAGNOT, J., IDOUMGHAR, L. "GECCO 2021 Competition on the Optimal Camera Placement Problem (OCP) and the Unicost Set Covering Problem (USCP)". 2021.

[61] SLATER, P. J. "LP-duality, complementarity, and generality of graphical subset parameters". In: *Domination in graphs*, Routledge, pp. 1–30, 2017.

[62] ARKIN, E. M., HALLDÓRSSON, M. M., HASSIN, R. "Approximating the tree and tour covers of a graph", *Information Processing Letters*, v. 47, n. 6, pp. 275–282, 1993.

[63] HAYNES, T. W., HEDETNIEMI, S., SLATER, P. *Fundamentals of domination in graphs*. CRC press, 2013.

[64] HAYNES, T. *Domination in graphs: volume 2: advanced topics*. Routledge, 2017.

[65] COCKAYNE, E. J., HEDETNIEMI, S. T. "Towards a theory of domination in graphs", *Networks*, v. 7, n. 3, pp. 247–261, 1977.

[66] COCKAYNE, E. "Domination of undirected graphs—a survey". In: *Theory and Applications of Graphs: Proceedings, Michigan May 11–15, 1976*, pp. 141–147. Springer, 1978.

[67] CHANG, G. J. "Algorithmic aspects of domination in graphs", *Handbook of Combinatorial Optimization: Volume1–3*, pp. 1811–1877, 1998.

[68] FOMIN, F. V., KRATSCH, D., WOEGINGER, G. J. "Exact (exponential) algorithms for the dominating set problem". In: *Graph-Theoretic Concepts in Computer Science: 30th International Workshop, WG 2004, Bad Honnef, Germany, June 21-23, 2004. Revised Papers 30*, pp. 245–256. Springer, 2005.

[69] VAN ROOIJ, J. M., BODLAENDER, H. L. "Exact algorithms for dominating set", *Discrete Applied Mathematics*, v. 159, n. 17, pp. 2147–2164, 2011.

[70] WANG, Y., CAI, S., CHEN, J., et al. "A fast local search algorithm for minimum weight dominating set problem on massive graphs." In: *IJCAI*, pp. 1514–1522, 2018.

[71] ROMANIA, Q. S. "Ant colony optimization applied to minimum weight dominating set problem". In: *Proceedings of the 12th WSEAS International Conference on Automatic Control, Modelling & Simulation, Catania, Italy*, pp. 29–31, 2010.

[72] ABED, S. A., RAIS, H. M., WATADA, J., et al. "A hybrid local search algorithm for minimum dominating set problems", *Engineering Applications of Artificial Intelligence*, v. 114, pp. 105053, 2022.

[73] COCKAYNE, E. J., DAWES, R., HEDETNIEMI, S. T. "Total domination in graphs", *Networks*, v. 10, n. 3, pp. 211–219, 1980.

[74] HENNING, M. A., YEO, A. *Total domination in graphs*. Springer, 2013.

[75] ÁLVAREZ-MIRANDA, E., SINNL, M. "Exact and heuristic algorithms for the weighted total domination problem", *Computers & Operations Research*, v. 127, pp. 105157, 2021.

[76] HU, S., LIU, H., WANG, Y., et al. "Towards efficient local search for the minimum total dominating set problem", *Applied Intelligence*, v. 51, n. 12, pp. 8753–8767, 2021.

[77] YUAN, F., LI, C., GAO, X., et al. "A novel hybrid algorithm for minimum total dominating set problem", *Mathematics*, v. 7, n. 3, pp. 222, 2019.

[78] CHALUPA, D. "An order-based algorithm for minimum dominating set with application in graph mining", *Information Sciences*, v. 426, pp. 101–116, 2018.

[79] DALIRI KHOMAMI, M. M., REZVANIAN, A., BAGHERPOUR, N., et al. "Minimum positive influence dominating set and its application in influence maximization: a learning automata approach", *Applied Intelligence*, v. 48, pp. 570–593, 2018.

[80] KARBASI, A. H., ATANI, R. E. "Application of dominating sets in wireless sensor networks", *Int. J. Secur. Its Appl*, v. 7, pp. 185–202, 2013.

[81] LI, Y., THAI, M. T., WANG, F., et al. "On greedy construction of connected dominating sets in wireless networks", *Wireless Communications and Mobile Computing*, v. 5, n. 8, pp. 927–932, 2005.

[82] DAI, F., WU, J. "On constructing k-connected k-dominating set in wireless networks". In: *19th IEEE International Parallel and Distributed Processing Symposium*, pp. 10–pp. IEEE, 2005.

[83] JEREMY, B., MIN, D., ANDREW, T., et al. "Connected dominating set in sensor networks and manets", *Handbook of Combinatorial, Springer, US*, 2004.

[84] AOUN, B., BOUTABA, R., IRAQI, Y., et al. "Gateway placement optimization in wireless mesh networks with QoS constraints", *IEEE Journal on Selected Areas in Communications*, v. 24, n. 11, pp. 2127–2136, 2006.

[85] YU, J., WANG, N., WANG, G., et al. "Connected dominating sets in wireless ad hoc and sensor networks–A comprehensive survey", *Computer Communications*, v. 36, n. 2, pp. 121–134, 2013.

[86] SAFAR, M., TAHA, M., HABIB, S. "Modeling the communication problem in wireless sensor networks as a vertex cover". In: *2007 IEEE/ACS International Conference on Computer Systems and Applications*, pp. 592–598. IEEE, 2007.

[87] DAGDEVIREN, Z. A. "Weighted connected vertex cover based energy-efficient link monitoring for wireless sensor networks towards secure internet of things", *IEEE Access*, v. 9, pp. 10107–10119, 2021.

[88] DAGDEVIREN, Z. A. "A metaheuristic algorithm for vertex cover based link monitoring and backbone formation in wireless ad hoc networks", *Expert Systems with Applications*, v. 213, pp. 118919, 2023.

[89] CHIN, W.-P., NTAFOS, S. "Optimum watchman routes". In: *Proceedings of the second annual symposium on Computational geometry*, pp. 24–33, 1986.

[90] MÖLLE, D., RICHTER, S., ROSSMANITH, P. "Enumerate and expand: Improved algorithms for connected vertex cover and tree cover", *Theory of Computing Systems*, v. 43, pp. 234–253, 2008.

[91] WANG, Y., BUCHANAN, A., BUTENKO, S. "On imposing connectivity constraints in integer programs", *Mathematical Programming*, v. 166, pp. 241–271, 2017.

[92] REHFELDT, D., FRANZ, H., KOCH, T. "Optimal connected subgraphs: Integer programming formulations and polyhedra", *Networks*, v. 80, n. 3, pp. 314–332, 2022.

[93] LAPORTE, G. "Generalized subtour elimination constraints and connectivity constraints", *Journal of the Operational Research Society*, v. 37, n. 5, pp. 509–514, 1986.

[94] GOEMANS, M. X. "The Steiner tree polytope and related polyhedra", *Mathematical programming*, v. 63, n. 1, pp. 157–182, 1994.

[95] GENDRON, B., LUCENA, A., DA CUNHA, A. S., et al. "Benders decomposition, branch-and-cut, and hybrid algorithms for the minimum connected dominating set problem", *INFORMS Journal on Computing*, v. 26, n. 4, pp. 645–657, 2014.

[96] NGUYEN, V. H. "Approximation algorithms for metric tree cover and generalized tour and tree covers", *RAIRO-Operations Research*, v. 41, n. 3, pp. 305–315, 2007.

[97] LUCENA, A. "Non delayed relax-and-cut algorithms", *Annals of Operations Research*, v. 140, pp. 375–410, 2005.

[98] POLYAK, B. T. "Subgradient methods: a survey of Soviet research", *Nonsmooth optimization*, v. 3, pp. 5–29, 1978.

[99] BEASLEY, J. E. "Lagrangian relaxation. Chapter 6 in Modern Heuristic Techniques for Combinatorial Problems, CR Reeves". 1993.

[100] BALAS, E., CHRISTOFIDES, N. "A restricted Lagrangean approach to the traveling salesman problem", *Mathematical Programming*, v. 21, n. 1, pp. 19–46, 1981.

[101] GAVISH, B. "Augmented Lagrangean based algorithms for centralized network design", *IEEE Transactions on Communications*, v. 33, n. 12, pp. 1247–1257, 1985.

[102] LUCENA, A. "Steiner problem in graphs: Lagrangean relaxation and cutting planes", *Coal Bulletin*, v. 21, n. 2, pp. 2–8, 1992.

[103] LUCENA, A. "Tight bounds for the Steiner problem in graphs", *Proceedings of NETFLOW93*, pp. 147–154, 1993.

[104] ESCUDERO, L. F., GUIGNARD, M., MALIK, K. "A Lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships", *Annals of Operations Research*, v. 50, pp. 219–237, 1994.

[105] CAVALCANTE, V. F., DE SOUZA, C. C., LUCENA, A. "A Relax-and-Cut algorithm for the set partitioning problem", *Computers & operations research*, v. 35, n. 6, pp. 1963–1981, 2008.

[106] KLIEWER, G., TIMAJEV, L. "Relax-and-cut for capacitated network design". In: *European Symposium on Algorithms*, pp. 47–58. Springer, 2005.

[107] LAN, G., DEPUY, G. W., WHITEHOUSE, G. E. "An effective and simple heuristic for the set covering problem", *European journal of operational research*, v. 176, n. 3, pp. 1387–1403, 2007.

[108] MARCHIORI, E., STEENBEEK, A. G. "An iterated heuristic algorithm for the set covering problem", 1998.

[109] BEZANSON, J., EDELMAN, A., KARPINSKI, S., et al. "Julia: A fresh approach to numerical computing", *SIAM review*, v. 59, n. 1, pp. 65–98, 2017. Available at: <https://doi.org/10.1137/141000671>.

[110] LI, R., HU, S., GAO, J., et al. "GRASP for connected dominating set problems", *Neural Computing and Applications*, v. 28, pp. 1059–1067, 2017.

[111] WU, X., LÜ, Z., GALINIER, P. "Restricted swap-based neighborhood search for the minimum connected dominating set problem", *Networks*, v. 69, n. 2, pp. 222–236, 2017.

[112] LI, R., HU, S., LIU, H., et al. "Multi-start local search algorithm for the minimum connected dominating set problems", *Mathematics*, v. 7, n. 12, pp. 1173, 2019.

[113] LI, B., ZHANG, X., CAI, S., et al. "Nucds: An efficient local search algorithm for minimum connected dominating set". In: *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pp. 1503–1510, 2021.

[114] WANG, L., ZHOU, T., WU, X., et al. "Variable-depth neighborhood search algorithm for the minimum-connected dominating-set problem", *Sci. Sin. Inf. Sci*, v. 46, pp. 445–460, 2016.

[115] BOUAMAMA, S., BLUM, C., FAGES, J.-G. "An algorithm based on ant colony optimization for the minimum connected dominating set problem", *Applied Soft Computing*, v. 80, pp. 672–686, 2019.

[116] GUTA, B. *Subgradient optimization methods in integer programming with an application to a radiation therapy problem.* Ph.D. Thesis, Technische Universität Kaiserslautern, 2003.

[117] "PassMark Single Thread Performance Benchmarks". `https://www.cpubenchmark.net/singleThread.html`. Accessed: 2024-11-29.