ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

Ingegneria Biomedica, Elettrica e dei Sistemi
(curriculum Ricerca Operativa)

Ciclo XXXII

Settore Concorsuale: 01/A6 - RICERCA OPERATIVA

Settore Scientifico Disciplinare: MAT/09 - RICERCA OPERATIVA

Models and algorithms for decomposition problems

Presentata da:
Paolo PARONUZZI

Coordinatore Dottorato:
Prof. Daniele VIGO

Supervisore:
Prof. Enrico MALAGUTI

Esame finale anno 2020

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

Abstract

Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione
"Guglielmo Marconi" (DEI)

Ingegneria Biomedica, Elettrica e dei Sistemi
(curriculum Ricerca Operativa)

**Models and algorithms
for decomposition problems**

by Paolo PARONUZZI

This thesis deals with the decomposition both as a solution method and as a problem itself. A decomposition approach can be very effective for mathematical problems that present a specific structure in which the associated matrix of coefficients is very sparse and, in particular, it is *diagonalizable* in blocks. But, this kind of structure may not be immediately evident from the most natural formulation of the problem. For this reason, its coefficient matrix may be preprocessed by solving a structure detection problem in order to understand if a decomposition method can successfully be applied. So, this thesis deals with the k-Vertex Cut problem, that is the problem of finding the minimum subset of nodes whose removal disconnects a graph into at least k components, and it models relevant applications in matrix decomposition for solving systems of equations by parallel computing. The capacitated k-Vertex Separator problem, instead, asks to find a subset of vertices of minimum cardinality the deletion of which disconnects a given graph in at most k shores and the size of each shore must not be larger than a given capacity value u. Also this problem is of great importance for matrix decomposition algorithms. Indeed, it can be viewed as the problem of assigning the *rows* of a given matrix A to k disjoint *blocks*, each one of size at most u. If A is the constraint matrix of a (mixed) integer problem, once this problem is solved and the matrix is consequently decomposed, then a decomposition technique might be applied to the original reformulated problem.

This thesis also addresses the Chance-Constrained Mathematical Program that represents a significant example in which decomposition techniques can be successfully applied. This is a class of stochastic optimization problems in which the feasible region depends on the realization of a random variable and the solution must optimize a given objective function while belonging to the feasible region with a probability that must be above a given value. In this thesis, a decomposition approach for this problem is introduced: we define a master problem containing the chance constraint and one sub-problem for each possible realization of the random variable. Then, we devise a Branch-and-Cut algorithm where we add cutting planes to the master problem as outer approximation point cuts.

The thesis continues by introducing the Fractional Knapsack Problem with Penalties, a variant of the knapsack problem in which items can be split at the expense of a penalty depending on the fractional quantity, and it concludes with a real-world application in which an optimization problem belonging to the family of the Lot Sizing problems is formulated.

Acknowledgements

First of all, I thank my supervisor Prof. Enrico Malaguti who supported me during all the three years of my Ph.D. He gave me the opportunity to explore several subjects of the Operations Research, he allowed me to know and to work together with other experienced researchers coming from different part of the world and he has always been willing to devote his time to clarify any kind of doubt.

Secondly, I am grateful to Dr. Giacomo Nannicini who agreed to be my tutor for the five months I spent in New York as a trainee at the IBM T.J. Watson research center. He made my experience very productive and formative.

I would also like to thank the co-authors of my first two scientific publications: Prof. Fabio Furini, Prof. Michele Monaci, Prof. Ivana Ljubic and Prof. Ulrich Pferschy. In particular, I take this opportunity to thank Fabio and Michele who also found the time to give me important advices and recommendations for my current and future academic career.

Thanks also to all the other professors of the group of Operations Research of the Department of Electrical, Electronic and Information Engineering "Guglielmo Marconi" (DEI) of the University of Bologna: Prof. Valentina Cacchiani, Prof. Silvano Martello, Prof. Paolo Toth, Prof. Daniele Vigo; and thanks to all my current and previous colleagues: Luca Accorsi, Dr. Carlos Emilio Contreras Bolton, Cristiano Fabbri, Naga Venkata Chaitanya Gudapati, Federico Naldini, Carlos Rodrigo Rey Barra and Dr. Dimitri Thomopulos.

Finally, a special thanks to my family and especially to my girlfriend Sara who immediately supported me in deciding to leave my previous job to take this new challenging and gratifying path.

Contents

List of Figures

List of Tables

Dedicated to Sara and Capo

Chapter 1

Introduction

Nowadays, decomposition techniques are widely known for being a very effective approach to solve mathematical programming problems whose formulation is too large to guarantee an exact resolution if it is attacked as a whole single problem. When the coefficient matrix of the mathematical programming problem is very sparse and, in particular, it is (almost) *diagonalizable* in blocks, then the original problem can be decomposed in smaller sub-problems that, hopefully, are easier to solve. In addition to be smaller in size, these sub-problems may also present a specific structure for which efficient *ad-hoc* algorithms exist.

Also within the field of Mathematical Programming there are many cases in which the use of decomposition techniques brings important advantages when solving (mixed) integer problems whose natural formulation results intractable due to a large number of variables or constraints. The two most known decomposition techniques in the field of Mathematical Programming are the Dantzig-Wolfe decomposition [1] and the Benders' decomposition [2].

The first method is usually applied to formulations in which the constraint matrix present a block structure with linking constraints; if these constraints were removed, then each block would represent a smaller sub-problem that would be solvable independently. The method works by defining a master problem where only the linking constraints are kept, while the other "blocks" of constraints are implicitly treated by generating their extreme points and rays, by means of a Column Generation algorithm.

Benders' decomposition can be seen as the dual of the Dantzig-Wolfe decomposition and it is effectively applied to those formulations in which there is a set of "complicating variables" the fixing of which significantly simplifies the resolution of the problem. In this case, the master problem only contains these variables. Once the master problem is solved, the feasibility (and the optimality) of its solution is checked through the dual of the sub-problem. If the solution turns out infeasible (or not optimal), then a "Benders cut" is generated and added to the master problem, and the procedure repeats.

However, there may exist cases in which it is not immediately clear if the problem formulation allows a decomposition approach. Chapter 2 of this thesis addresses the k-Vertex Cut problem, that is the problem of finding the minimum subset of nodes whose removal disconnects a graph into k components, and it models relevant applications in matrix decomposition for solving systems of equations by parallel computing. Indeed, given a system of equations with a coefficient matrix A, one can build the *intersection graph* associated to A by defining a vertex for each variable and an edge between a pair of vertices if and only if there exists a row in A where both the associated variables have a nonzero coefficient. When the system is solved by decomposition, it is divided into smaller subsystems that are solved separately. The

solutions of the subsystems have to be merged in a consistent way to obtain a solution of the whole system. In other words, if the same variable appears in multiple subsystems, it must take the same value in all of them. The effort for performing this task increases with the number of variables that appear in more than one subsystem. If one wants to partition the equations into k subsystems, the problem of minimizing the number of common variables can be formulated as a k-Vertex Cut problem.

In Sections 2.3 and 2.4, we introduce two new integer linear programming formulations for the k-Vertex Cut problem, along with families of strengthening valid inequalities. Both models involve an exponential number of constraints for which polytime separation procedures are provided together with the respective Branch-and-Cut algorithms.

In the first formulation, defined as *Representative Formulation*, one representative vertex is chosen for each of the k mutually disconnected vertex subsets. This way, it is enough to impose non-connectivity among the representatives to obtain pairwise disconnected subsets. This condition is ensured by *Path constraints* imposing that at least one vertex of each path between a pair of representatives is in the vertex cut (thus disconnecting the two representatives). This family of constraints is exponential in size and, in section 2.5, we describe a polytime separation procedure based on the solution of a Shortest Path problem.

In the second formulation, so-called *Natural Formulation*, we derive the model from the perspective of a two-stage Stackelberg game in which a leader deletes the vertices in the first stage, and in the second stage a follower builds a cycle-free subgraph in the remaining graph. The solution of the leader is feasible, if and only if the number of connected components in the subgraph corresponding to the the follower's optimal response is at least k. The leader wants to find a feasible solution where the set of deleted vertices (i.e., the k-vertex cut) has minimum size. In section 2.4.2, we present a linearization of this bilevel model obtained by reformulating the follower's subproblem in a way such that the set of its feasible solutions does not depend on the leader. Then, we derive a single-level reformulation with an exponential number of constraints. The separation procedure generating these constraints (section 2.5) aims to detect a maximum-weighted cycle free subgraph that violates the corresponding inequality, and it runs in polynomial time both for fractional and integer solutions.

Since both the described formulations make use of the same variables to decide which vertices are removed from the graph, we also design a hybrid model in which valid inequalities of both formulations are combined. Computational study (section 2.6) demonstrates that this hybrid model significantly outperforms the state-of-the-art exact methods from the literature.

Also in the case of Mixed Integer Programming (MIP) problems, it may be necessary to perform a preprocessing on the constraint matrix in order to detect and extrapolate the specific structure needed to approach the problem by means of a decomposition technique. So, in Chapter 3 we introduce the capacitated k-Vertex Separator problem, that is an other problem belonging to the same family of the k-Vertex Cut problem. Given a graph, this problem asks to find a subset of vertices of minimum cardinality the deletion of which disconnects the graph in at most k shores and the size of each shore is not bigger than a given capacity value u. Also this problem is of great importance for matrix decomposition algorithms. Indeed, it can be viewed as the problem of assigning the *rows* of a given matrix A to k disjoint *blocks*. The objective is to remove a minimum number of rows from A and to assign the remaining rows to the blocks so that each block contains at most u rows. The problems

are equivalent by defining a graph with a vertex for each row, and an edge for each pair of vertices if there is at least a column in A with nonzero entries in the corresponding rows. If A is the constraint matrix of a (mixed) integer problem, once the k-Vertex Separator problem is solved and the matrix is consequently decomposed, then a decomposition technique like those described in the first paragraph might be applied.

Both the k-Vertex Cut problem and the capacitated k-Vertex Separator problem belong to the family of Critical Node Detection Problems that, generally, ask for finding a subset of vertices, deletion of which minimizes or maximizes a predefined connectivity measure on the remaining network. They find important applications not only in matrix decomposition problems, but also in the analysis and protection of communication or social networks against possible viral attacks in which contest the vertices identified by the optimal solution can be seen as the most vital or critical vertices of a graph, with respect to connectivity.

In section 3.3, we define a new bilevel interpretation of the k-Vertex Separator problem, and we model it as a two-player Stackelberg game, in which the leader interdicts the vertices (i.e., decides on the *separator*), and the follower determines the maximum connected component on the resulting graph. In addition, the leader has to make sure that the connected components can be packed in at most k shores each of the size at most u. This approach allows us to develop a computational framework based on an Integer Programming formulation in the natural space of the variables. Thanks to this bilevel interpretation, we define three different families of strengthening inequalities: *Component cuts* (3.3.1), *Benders cuts* (3.3.2) and *Degree cuts* (3.3.3). In section 3.3.4, we study the dominance relationship between these three families of inequalities, and, in section 3.6, we present their separation routines running in polynomial time.

After an extensive computational analysis on the the relative computational performance of each family of inequalities and their computational interaction when separated in a Branch-and-Cut fashion (section 3.7), we determine the best configuration of a newly developed Branch-and-Cut algorithm. This exact method is competitive against the state-of-the-art algorithms for the k-Vertex Separator problem, and is able to improve the best known results for several difficult classes of instances.

One of the most significant example in which decomposition techniques can be successfully applied is represented by the Chance-Constrained Mathematical Program (CCP), that we address in Chapter 4. CCP is a class of stochastic optimization problems in which the feasible region depends on the realization of a random variable and the solution must optimize a given objective function while belonging to the feasible region with a probability that must be above a given value, defined by the decision maker. Under specific assumptions, this problem can be modeled as a Mixed Integer Not Linear Problem (MINLP).

In the case of the CCP, the applicability of a decomposition method in the approach of the problem is quite natural and intuitive due to the problem structure that usually presents a set of variables that represents the decision to be taken in the first stage of a considered time window and other sets of variables that, instead, play a role only in the following stages and only if the possible related realization actually takes place. For this reason, in the problem formulation the former set of variables is shared by all the existing constraints, while the latter sets of variables only appear in some specific subsets of constraints. The result is a model in which the coefficient matrix has a blocks structure and its *intersection graph* is a graph in which the removal of the vertices associated with the first stage variables (i.e., the

assignment of these vertices to the *separator*) leads to a partition having a shore for each different scenario. Hence, the application of a decomposition method is very appealing and promising.

In Section 4.2, we propose a decomposition approach for the MINLP reformulation of CCP whereby we define a single master problem and one sub-problem for each possible realization of the random variable. We devise a Branch-and-Cut algorithm where we generate cutting planes as outer approximation point cuts, when possible. This approach generalizes the one proposed by Lodi et al. [3] that only applies to the case in which the scenario variables are continuous. Given an infeasible solution of the master problem, we distinguish between three different cases: a first case in which the method proposed by Lodi et al. can be applied to generate a valid cutting plane, after the integrality constraint on the scenario variables has been relaxed; a second case that we address by mean of a novel procedure that returns a valid cut, when it exists; and a last case in which some kind of spatial branching is required in order to discard the infeasible solution proposed by the master problem.

In Chapter 5, we consider integer problems where variables can potentially take fractional values, but this occurrence is penalized in the objective function. This general situation has relevant examples in scheduling (preemption), routing (split delivery), cutting and telecommunications, just to mention a few. However, the general case in which variables integrality can be relaxed at cost of introducing a general penalty was not discussed before. As a case study, we consider the possibly simplest combinatorial optimization problem, namely the classical Knapsack Problem. We introduce the Fractional Knapsack Problem with Penalties (FKPP), a variant of the knapsack problem in which items can be split at the expense of a penalty depending on the fractional quantity.In addition to the possible applications mentioned above, FKPP also arises as the subproblem to be solved in Dantzig-Wolfe decomposition approaches for packing problems where item fragmentation is allowed, as the one considered in [4].

In Section 5.1, we introduce relevant properties of this problem and we show that, when the functions describing the penalties are concave, then an optimal solution exists in which at most one item is taken at a fractional level.

Then, we present alternative mathematical models (Section 5.2) and we discuss the relation between them. The first general model has a linear number of variables and constraints, but a non-linear objective function; the second model restricts the weight contribution of each variable to integers and it is equivalent to a Multiple-Choice Knapsack problem (MCKP).

In Section 5.3, we introduce a Fully Polynomial Time Approximation Scheme (FPTAS) for the approximate solution of the general problem, and, in Section 5.4, we devise a new dynamic programming approach that computes the optimal solution when the previously described condition on the penalty functions holds. Our dynamic programming algorithm has an improved computational complexity with respect to the procedure proposed in [4].

In Section 5.5, we also present three simple heuristic algorithms that provide approximate solutions for the general FKPP.

We conclude the chapter by computationally testing the proposed models and algorithms on a large set of instances derived from benchmarks from the literature (Section 5.6). The experiments show that we are able to effectively solve medium-sized FKPP instances and they highlight the improvement in term of computing time that we obtain by means of our improved dynamic programming algorithm.

Finally, in Chapter 6, motivated by a real-world application, we consider an optimization problem belonging to the family of the Capacitated Lot Sizing Problems (CLSP). Given the amount of an independent demand that varies over time, the lot sizing problem aims to determine in which period the production must be performed in order to satisfy the demand, while minimizing the total required cost. Typically, the problem involves variable production costs, that increase as the produced quantity does, fixed set up costs, that occur any time a machine starts the production of an item in some period, and holding costs, that occur when a product is stored in the inventory from a period to the next one.

The case study we deal with includes several products (multi-item), whose production must be scheduled on machines belonging to different plants (multi-plant), and the demands come from different customers. Then, each plant has an associated internal inventory, with finite capacity, as well as an uncapacitated external inventory, that can be used at the expense of an extra cost. Furthermore, each demand can remain partially unsatisfied by paying a penalty for each undelivered product. So, in addition to the traditional production and inventory costs, our problem also takes into account the transportation costs from the plants to the customers, the external inventory costs and the out of stock costs. Moreover, the problem is further complicated by the minimum lot size condition that imposes a minimum quantity to be produced when an order is assigned to some machine.

In Section 6.2, we give a formal definition of the problem at hand and introduce a mathematical model based on a Mixed Integer Linear Program (MILP).

In Section 6.3, we present a fast heuristic algorithm based on an iterative constructive method. This procedure considers one demand at a time and determines the best policy, according to a greedy strategy. More precisely, the algorithm decides which is the line and time period in which production must take place, the amount of items to be produced, and updates the production schedule accordingly.

Since the constructive heuristic algorithm may produce solutions in which the use of some lines is not fully optimized, in Section 6.4 we also propose a metaheuristic approach based on the ruin-and-recreate paradigm (see, Schrimpf et al. [5]). The idea of a ruin-and-recreate algorithm is to determine improving solutions by (i) destroying a considerable part of a feasible solution (*ruin* phase), and (ii) applying a rebuilding procedure (*recreate* phase) to complete the solution. In our case, the initial feasible solution is produced by the constructive heuristic algorithm and we use three different procedures to destroy and repair the solution.

Finally, in Section 6.5 we give the outcome of computational experiments for the proposed approaches on two real-world test-cases and on a large set of realistic instances that are derived from the real ones. The results show that, in all cases, the approximations obtained in short computing time are very close to those that can be achieved by running a state-of-the-art commercial solver on the mathematical model for a very long computing time.

Chapter 2

On Integer and Bilevel Formulations for the *k*-Vertex Cut Problem

[1]

In the analysis of networks, their correct functioning frequently depends on a small number of important vertices whose malfunctioning can significantly degrade the performance of the whole network. Depending on the crucial properties that need to be maintained (or achieved) in the network, different vertices may be considered as important. So, for example, if the major concern of a decision maker is the way how information is diffused in the network, we might be interested in finding the key-player vertices or the most influential vertices in the network (see [7]). Similarly, if the decision maker wants to protect the network against malicious attacks that may affect or destroy connectivity, we are talking about the detection of critical vertices of a network. Although there may be some vertices that remain critical no matter which connectivity measure is considered, very often the importance of a vertex changes with the definition of the connectivity measure (see, e.g. [8], [9]).

The family of Critical Node Detection Problems asks for finding a subset of vertices, deletion of which minimizes or maximizes a predefined connectivity measure on the remaining network (see, e.g., [9] for a recent survey). Related to CNDPs is the family of problems in which we are searching for a subset of vertices of minimum weight, deletion of which changes the predefined connectivity measure of the remaining network by a certain value, specified by the decision maker in advance. In this chapter we study the *k*-Vertex Cut Problem, which belongs to the latter family of problems, and which is defined as follows.

Definition 0 (*k*-Vertex-Cut) *A vertex cut is a set of vertices whose removal disconnects the graph into several connected components. If the number of connected components is at least k, this set is called a k-vertex cut. Given a graph $G = (V, E)$, a positive weight w_u for each vertex $u \in V$, and an integer $k \geq 2$, the k-vertex cut problem is to find a k-vertex cut of minimum weight.*

Besides applications in the analysis of networks, the *k*-vertex cut problem also models relevant applications in matrix decomposition for solving systems of equations by parallel computing [10]. Given a system of equations with the coefficient matrix A, the *intersection graph* associated to A has one vertex for each column and an edge between a pair of vertices if and only if there exists a row in A where both

[1] The results of this chapter appears in: F. Furini, I. Ljubić, E. Malaguti, and P. Paronuzzi, "On integer and bilevel formulations for the k-vertex cut problem", *Mathematical Programming Computation*, 1-32, 2019. [6]

variables have a nonzero coefficient. When the system is solved by decomposition, it is divided into smaller subsystems that are solved separately. The solutions of the subsystems have to be merged in a consistent way to obtain a solution of the whole system (i.e., if the same variable appears in multiple subsystems, it must take the same value in all of them). The effort for performing this task increases with the number of variables that appear in more than one subsystem. If one wants to partition the equations into k subsystems, the problem of minimizing the number of common variables can be formulated as a k-Vertex Cut problem. Figure 2.1 illustrates an example of a graph with 10 vertices, all with the same weight, along with an optimal solution for the 3-vertex-cut problem: a vertex-cut is of size 3 (given in black), and removal of these vertices results in 3 connected components in the remaining graph.

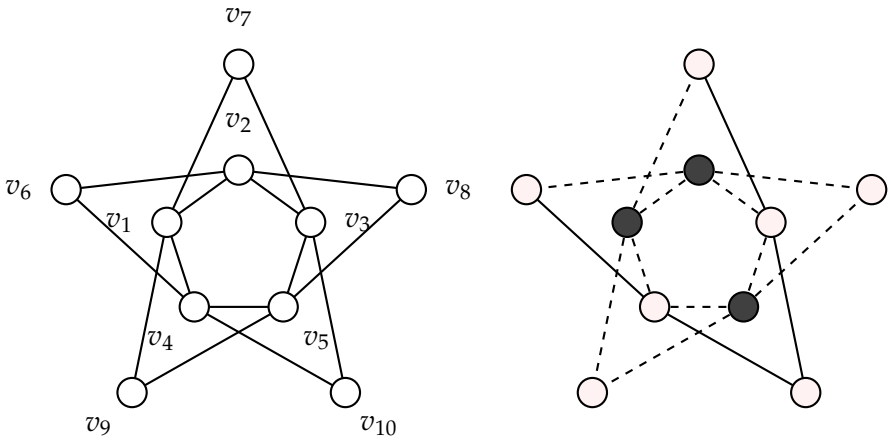


FIGURE 2.1: A graph with 10 vertices of equal weight and an optimal 3-vertex cuts (on the right) represented by the black vertices $\{v_1, v_2, v_5\}$.

By the equivalence with the vertex k-multiclique problem on the complement graph, it has been shown that for any fixed $k \geq 3$, even with unitary weights, the problem is NP-hard [11]. On the other hand, for $k = 2$, the problem can be solved in polynomial time: For uniform vertex weights, the problem is equivalent to calculating the vertex-connectivity of the graph; For the more general case of non-uniform weights, the problem boils down to calculating $O(n^2)$ maximum flows, see [12].

Our Contribution. In this chapter, we study exact solution approaches to the k-Vertex-Cut problem. We first provide two new Integer Linear Programming (ILP) formulations, along with some families of strengthening valid inequalities. Both models involve an exponential number of constraints for which we provide separation procedures and implement branch-and-cut algorithms. The first formulation, to which we refer to as *Representative Formulation*, asks to choose one representative for each of the k mutually disconnected subsets of the remaining graph. In the second, so-called *Natural Formulation*, we derive the model from the perspective of a two-stage Stackelberg game in which a leader deletes the vertices in the first stage, and in the second stage a follower builds connected components in the remaining graph. In our computational study, we implement these models, compare them with the state-of-the-art approach from [11] and report results of a Hybrid approach in which

the Representative and Natural formulations are combined, to provide the new best performing method for the k-vertex cut problem.

The chapter is organized as follows: in the remainder of this section, we introduce the notation, we provide a detailed literature overview, and we recall a compact formulation for the problem that was introduced in [11], [13]. In Section 2.2, we derive theoretical properties that allow us to fix some vertices in the optimal solution. The Representative Formulation, along with valid inequalities is given in Section 2.3, and the bilevel modeling approach is shown in Section 2.4. Separation procedures for both models are provided in Section 2.5. Finally, a detailed computational study is provided in Section 2.6 and conclusions are drawn in Section 2.7.

Notation. Let K denote the set of integers $\{1, ..., k\}$. Given a simple undirected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, for an edge $uv \in E$, we say that u and v are *neighbours*. The complement of graph $G = (V, E)$ is a graph $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{uv : uv \notin E\}$. Let $N(u) = \{v \in V | uv \in E\}$ denote the *neighborhood* of u and $\overline{N}(u) = V \setminus (N(u) \cup \{u\})$ denote the *anti-neighborhood* of u. A subset of vertices $W \subset V$ is a *clique of* G, if any two vertices of W are neighbours. A subset of vertices $W \subset V$ is a *stable set* if it is a clique in $\overline{G}$; the cardinality of the largest stable set of G, called the *stability* number of G, is denoted as $\alpha(G)$.

We indicate by $deg_G(v)$ the number of edges incident on v in graph G.

Given a subset of edges $E' \subseteq E$ of G, we say that E' is *spanning* if for every vertex v of G there is at least an edge in E' incident with v. We denote by *component* of a graph G a connected subgraph, while a generic *subset* of vertices of G can induce several components. This distinction is relevant because the removal of a k-vertex cut from a graph G can disconnect G in *more* than k components, and we may need to refer instead to exactly k subgraphs, induced by k subsets of vertices.

We will use the observation that a k-vertex cut V_0 is a set of vertices such that $V \setminus V_0$ can be partitioned into k non-empty subsets $V_1, ..., V_k$ that are pairwise disconnected, i.e., there is no edge between two subsets V_i and V_j for all $i \neq j \in \{1, \dots, k\}$. A necessary and sufficient condition for G to have a k-vertex cut is given in the following

Observation 1 *A graph $G = (V, E)$ admits a k-vertex cut if and only if $\alpha(G) \geq k$.*

Without loss of generality we will assume the condition of Proposition 1 to be satisfied (otherwise, the input instance can be discarded as infeasible). If q is the number of (connected) components of G, we will also assume that $q < k$, otherwise the problem can be trivially solved (empty vertex cut).

Literature review. The k-Vertex Cut problem is polynomially solvable for $k = 2$ [12], and it is NP-hard for $k \geq 3$, when k is part of the input [14]. Only very recently, in [11] the authors show that even for a fixed value of k, the problem remains NP-hard for $k \geq 3$. In addition, the first study on exact methods for the vertex k-cut problem is given in [11]. The authors provide a compact integer programming formulation and a formulation with an exponential number of variables, for which a branch-and-price algorithm is implemented and tested on benchmark instances with up to 200 vertices.

A well studied problem in combinatorial optimization is a closely related problem of finding the minimum-weight *edge k-cut*. The problem consists of finding a subset of edges (instead of vertices) of minimum weight, whose removal separates the graph in at least k connected components. Mainly complexity results are known

about this problem: in [15], the author exploits submodularity property to obtain a poly-time lower bound for the problem. For a fixed value of k, the problem reduces to $O(n^{k^2})$ minimum cut problems [16]. Better running times for a fixed value of k are given in [17]. Very recently in [18] an FPT algorithm is given in which the value of k is used as a parameter and which improves the 2-approximation results from e.g., [19].

Another well-studied problem variant is the *multiway cut problem* (sometimes also called the *multiterminal cut problem*), in which a set of terminal vertices T is given and one has to find a minimum-weight subset of edges that separates each terminal from all others. For this problem, complexity is studied in [20] where the authors show that for $|T| \geq 3$ the problem is already NP-hard, and that for a fixed size of T, the problem is solvable in polynomial time on planar graphs. A polyhedral study is given in [21]. There also exists the vertex-counterpart of the multiway cut problem, called the *multi-terminal vertex k-cut problem*, in which one searches for the minimum-weight subset of vertices to remove from a graph, so that every pair of terminals is disconnected (here $k = |T|$). Clearly, a vertex multiway cut exists only if the terminals form an independent set. For this problem, the authors of [22], [23] give an approximation preserving reduction from the vertex cover problem, and provide a 2-approximation algorithm. In [24] the W[1]-hardness of this problem is shown. A path-based integer programming formulation along with some valid inequalities is given in [25]. In addition, a polyhedral analysis is also performed and an efficient branch-and-cut algorithm is developed.

Finally, there also exist problem variants in which cardinality bounds on each component/vertex set are imposed. In the *k-separator problem* the goal is to find a vertex cut whose removal results in a disconnected graph such that the maximum size of each connected component is bounded by k. A bound on the number of components may also be imposed. This problem is introduced in [13] and motivated by matrix decomposition. The authors propose a model (with binary variables indicating the assignment of vertices to the partitions), which is solved by a tailored branch-and-cut algorithm. The complexity of this problem is studied in [26], where also an approximation algorithm is given, along with a integer programming formulations and a polyhedral study. Recently, the authors of [27] present an exponential size integer programming formulation which they solve by branch-and-price, and perform an extensive computational study, in particular on graphs coming from matrix decomposition. The proposed approach consistently solves instances with a large bound on the number of components, and thus complements previous exact approaches that work better/only for smaller number of components. A closely related problem is the one where the cardinality constraints are imposed not on the size of the connected components but on vertex sets. More precisely, the problem consists in finding a subset of vertices to remove from G so that the remaining graph can be partitioned into two sets of cardinality at most k with no edge being incident to both sets. Observe that each set may contain several connected components. This problem is NP-hard even for planar graphs [28] or maximum degree 3 graphs [29]. A first polyhedral study on this problem is done in [30] from which a branch-and-cut algorithm is derived [10].

2.1. Compact Formulation

In this section, we recall the compact formulation, which has been introduced in [11] (for the case where $w_u = 1$ for all $v \in V$). The formulation exploits the fact that

a k-vertex cut V_0 is a set of vertices such that $V \setminus V_0$ can be partitioned into k non-empty subsets $V_1, ..., V_k$ that are pairwise disconnected. This formulation is similar to the one introduced in [13] for the the k-separator problem, in particular, it uses the same variables: for each vertex $v \in V$ and each integer $i \in K$, a binary variable y_v^i is defined, such that

$$y_v^i = \begin{cases} 1 & \text{if vertex } v \text{ belongs to subset } i \\ 0 & \text{otherwise} \end{cases} \qquad i \in K, v \in V.$$

The vertices that remain unassigned to any of the subsets V_k (i.e., for which $y_v^i = 0$, for all $i \in K$), are the ones defining the k-vertex cut. This is why instead of minimizing the weight of the k-vertex cut, one can equivalently maximize the sum of the weights of vertices out of the vertex cut (i.e., the weight of vertices in the union $\cup_{i \in K} V_i$).
This compact ILP formulation (denoted as $COMP$) reads as follows:

$$(COMP) \qquad \min \sum_{v \in V} w_v - \sum_{i \in K} \sum_{v \in V} w_v y_v^i \tag{2.1}$$

$$\sum_{i \in K} y_v^i \leq 1 \qquad v \in V \tag{2.2}$$

$$y_u^i + \sum_{j \in K \setminus \{i\}} y_v^j \leq 1 \qquad i \neq j \in K, uv \in E \tag{2.3}$$

$$\sum_{v \in V} y_v^i \geq 1 \qquad i \in K \tag{2.4}$$

$$y_v^i \in \{0, 1\} \qquad i \in K, v \in V. \tag{2.5}$$

Constraints (2.2) impose that each vertex belongs to at most one of the subsets V_i, $i \in K$. Constraints (2.3) ensure that the subsets are pairwise disconnected, i.e., whenever there is an edge between a pair of vertices u and v, these two vertices are not permitted to belong to two different subsets V_i and V_j, $i, j \in K, i \neq j$. Finally, constraints (2.4) avoid having empty subsets in a feasible solution.

The model $COMP$ has some serious drawbacks. First the number of variables increases linearly with the value of k, and the LP relaxation bound of this model is always equal to zero (we can obtain an optimal LP-solution by setting $y_v^i = 1/k$, for all $v \in V$, $i \in K$, see [11]). Second, the model suffers from symmetries, as the variables can be permuted by obtaining an equivalent solution. This is why an alternative modeling approach has been considered in [11]. A model with an exponential number of variables has been proposed, in which each column represents one of the subsets V_i, $i \in K$, and the corresponding branch-and-price algorithm has been implemented. In what follows, we derive two alternative ways to model the problem after having presented some preprocessing techniques.

2.2. Preprocessing

In this section we discuss necessary conditions under which a vertex must belong to any optimal k-vertex cut and, *de facto*, the size of the input graph can be reduced.

Assume that a vertex $u \in V$ is not in a k-vertex cut, so that all the vertices in its neighbourhood $N(u)$ either belong to the same subset as u, or are in the k-vertex

cut. Therefore, the size of the anti-neighbourhood of u gives an upper bound on the number of disconnected non-empty components that can be obtained.

Proposition 1 *In any feasible solution to the k-vertex cut problem, if for a vertex $u \in G$ we have $k \geq |\overline{N}(u)| + 2$, then vertex u must belong to any optimal k-vertex cut.*

Proof. Observe that $|\overline{N}(u)|$ is a straight-forward upper bound on the number of components in the anti-neighborhood of u, assuming that the anti-neighborhood defines a stable set, i.e., $\alpha(\overline{N}(u)) = |\overline{N}(u)|$. Vertex u, if not in the k-vertex cut, makes at most a single component along with the vertices in its neighborhood, which leads to at most $k-1$ components, and hence, such a solution would be infeasible.□

We can strengthen this upper bound by analyzing the connected components in the graph induced by the anti-neighborhood of $u \in V$. Let n_C be the number of connected components $(C_1, \ldots, C_{n_C})$ in the subgraph $G[\overline{N}(u)]$ induced by $\overline{N}(u)$. Let

$$m(C_i) = \max_{S \subset V(C_i)} \{ \text{number of connected components of } G[V(C_i) \setminus S] \}$$

(where $V(C_i)$ is the vertex set of the component C_i). Therefore we have:

Proposition 2 *Consider $u \in V$ and let $(C_1, \ldots, C_{n_C})$ be connected components in $G[\overline{N}(u)]$. If we have*

$$k \geq \sum_{i=1}^{n_C} m(C_i) + 2,$$

then vertex u must belong to the k-vertex cut.

Proof. Same reasoning as for Proposition 1.□

The following proposition allows us to compute the exact values of $m(C)$ for each of the n_C components:

Proposition 3 *The maximum number of components that can be obtained by deleting some vertices from a connected component C of G is equal to the stability number of C, that is, $m(C) = \alpha(C)$.*

Proof. If C contains a stable set of cardinality $\alpha(C)$, we have $\alpha(C)$ non-empty components composed by the vertices of the stable set, so $m(C) \geq \alpha(C)$. Viceversa, if C can be decomposed in $m(C)$ non-empty components, each of these components contains vertices that are not adjacent to any vertex of the other components. By picking a vertex per component, we define a stable set of cardinality $m(C)$, so $m(C) \leq \alpha(C)$, i.e., $m(C) = \alpha(C)$.□

See the computational Section 2.6, for further implementation details concerning the preprocessing and its effectiveness in reducing the size of input graphs.

2.3. Representative Formulation

We now propose a novel, alternative formulation for the k-Vertex Cut Problem which is based on the idea of identifying a vertex that is the *representative* of each subset V_i, $i \in K$. This way, it is enough to impose non-connectivity among the representatives

to obtain pairwise disconnected subsets. Connected components that are disconnected from any representative can be feasibly assigned to any subset.

The non-connectivity of the representatives can be obtained via an exponential number of path inequalities, similarly to what was done by [25], [31] for the multi-terminal vertex k-cut problem, where each representative is denoted as terminal and it is fixed as an input. We consider two sets of binary variables associated with the vertices, denoting whether a vertex is a representative, and whether a vertex is in the k-vertex cut, respectively. We have

$$z_v = \begin{cases} 1 & \text{if vertex } v \text{ is the representative of a subset} \\ 0 & \text{otherwise} \end{cases} \qquad v \in V,$$

$$x_v = \begin{cases} 1 & \text{if vertex } v \text{ is in the } k\text{-vertex cut} \\ 0 & \text{otherwise} \end{cases} \qquad v \in V,$$

and the corresponding *Representative Formulation* reads as follows:

$$(REP) \quad \min \sum_{v \in V} w_v x_v \tag{2.6}$$

$$\sum_{v \in V} z_v = k \qquad v \in V \tag{2.7}$$

$$z_u + z_v \leq 1 \qquad uv \in E \tag{2.8}$$

$$\sum_{t \in V(P) \setminus \{u,v\}} x_t \geq z_u + z_v - 1 \qquad u, v \in V, P \in \Pi_{uv}, uv \notin E \tag{2.9}$$

$$x_v, z_v \in \{0,1\} \qquad v \in V. \tag{2.10}$$

In this model, P denotes a simple path in G, $V(P)$ are the vertices connected by P, and Π_{uv} is the set of all simple paths between vertices u and v. The objective function (2.6) minimizes the weight of the vertices in the k-vertex cut. Constraint (2.7) ensures that exactly k representative vertices are selected, and constraints (2.8) impose the set of representative vertices to be a stable set. *Path constraints* (2.9), in exponential number, impose that at least one vertex of each path $P \in \Pi_{uv}$ between a pair of representative u and v is in the vertex cut (thus disconnecting the two representatives). Note that condition $uv \notin E$ in (2.9) serves to remove redundant inequalities for which the right-hand-side is equal to zero due to (2.8).

Proposition 4 *For $k \leq n/2$, the LP relaxation bound of the formulation (2.6)-(2.10) is equal to zero.*

Proof. It can be checked that for $k \leq n/2$, setting $z_v = k/n$ results in a feasible solution in which $x_v = 0$, for all $v \in V$. □

Strengthening Inequalities. Constraints (2.9) can be lifted by observing that, each time a path in Π_{uv} includes a representative vertex, an additional vertex of the path must be in the vertex-cut:

$$\sum_{t \in V(P) \setminus \{u,v\}} x_t \geq z_u + z_v + \sum_{t \in V(P) \setminus \{u,v\}} z_t - 1,$$
$$u, v \in V, P \in \Pi_{uv}, uv \notin E. \tag{2.11}$$

Other families of constraints in polynomial number can be considered in order to strengthen the linear relaxation of the representative model.

Given a vertex u and its neighbourhood $N(u)$, if u is not in a k-vertex cut, then together with (some of) its neighbors it belongs to the same connected component, and hence, at most one of the vertices from $N(u) \cup \{u\}$ can be chosen as representative. Alternatively, if u is in the k-vertex cut, at most $\deg_G(u) = |N(u)|$ vertices can be representatives, which can be expressed by the following *neighborhood constraints*:

$$z_u + \sum_{v \in N(u)} z_v \leq 1 + (\deg_G(u) - 1)x_u \qquad u \in V, \tag{2.12}$$

paired with the additional condition that a vertex u cannot be a representative and be in the vertex cut at the same time:

$$x_u + z_u \leq 1 \qquad u \in V. \tag{2.13}$$

Note that an integer solution violating (2.13) cannot be optimal, so these constraints are not necessary for the correctness of formulation REP. Indeed, consider a solution where for a vertex u we have $x_u = z_u = 1$: by (2.9) any path from u to another representative vertex w must be disconnected, so u cannot be the (only) vertex disconnecting a path from w to a third representative vertex v. As a consequence, we can set $x_u = 0$ and reduce the cost of the solution while keeping feasibility.

2.4. Bilevel Approach

We now provide a bilevel point-of-view to the problem, which will allow us to derive a valid ILP formulation in the natural space of the x_v, $v \in V$, variables only.

We can see the k-Vertex Cut problem as a sequential two-player Stackelberg game in which there are two players: a leader and a follower. In the first step, the leader "interdicts" the follower by deleting some vertices from the graph, and in the following step, the follower looks for the largest cycle-free subgraph problem in the remaining graph. The solution of the leader is feasible, if and only if the number of connected components in the subgraph corresponding to the the follower's optimal response is at least k. The leader wants to find a feasible solution where the set of deleted vertices (i.e., the k-vertex cut) has minimum weight.

In the following, we first provide a bilevel integer programming formulation (BILP), which follows the description of the two sequential steps described above. We start by describing a graph property that allows us to model the follower's subproblem as an ILP. It is well known that a graph G is connected if and only if it contains a spanning tree, i.e., the number of edges in its spanning cycle-free subgraph is $|V| - 1$. If G contains multiple connected components, this property can be generalized as follows:

Observation 2 *A graph G has at least k connected components if and only if any cycle-free subgraph of G contains at most $|V| - k$ edges.*

Clearly, a graph G contains at least k connected components if and only if any *maximum* cycle-free subgraph (with respect to the number of edges) contains at most $|V| - k$ edges. By exploiting this property, the k-vertex cut problem can be seen as a Stackelberg game in which the leader searches the smallest subset of vertices V_0 to delete from G, and the follower maximizes the size of the cycle-free subgraph on the remaining graph.

Observation 3 *The solution $V_0 \subset V$ of the leader is feasible if and only if the value of the optimal follower's response (i.e., the maximum number of edges of a cycle-free subgraph in the remaining graph) is at most $|V| - |V_0| - k$.*

2.4.1 A Bilevel Integer Programming Formulation

The leader decisions are encoded by the same x variables used for the Representative Formulation, where x_v is one if vertex v is "interdicted" (e.g., vertex v is in the k-vertex cut), and zero otherwise. To model the decisions of the follower, we use additional binary variables associated with the edges of G:

$$e_{uv} = \begin{cases} 1 & \text{if edge } uv \text{ is selected to be in the cycle-free subgraph} \\ 0 & \text{otherwise} \end{cases} \qquad uv \in E,$$

The BILP formulation of the k-vertex cut problem reads as follows:

$$(BILP) \quad \min \sum_{v \in V} w_v x_v \tag{2.14}$$

$$\Phi(x) \leq n - k - \sum_{v \in V} x_v \tag{2.15}$$

$$x_v \in \{0,1\} \qquad v \in V. \tag{2.16}$$

Constraint (2.15) ensures Observation 3, i.e., it guarantees the feasibility of the solution x of the leader. Thereby, $\Phi(x)$ is the solution value of the follower subproblem, in which the follower searches for cycle-free subgraph on the remaining graph having the largest number of edges. For a solution x^* of the leader, which represents an incidence vector of a set V_0 of interdicted vertices, the follower's subproblem is:

$$\Phi(x^*) = \max \sum_{uv \in E} e_{uv} \tag{2.17}$$

$$e(S) \leq |S| - 1 \qquad S \subseteq V, |S| \geq 3 \tag{2.18}$$

$$e_{uv} \leq \begin{cases} 1 - x^*_v \\ 1 - x^*_u \end{cases} \qquad uv \in E \tag{2.19}$$

$$e_{uv} \in \{0,1\} \qquad uv \in E, \tag{2.20}$$

where $e(S) = \sum_{uv \in E; u,v \in S} e_{uv}$. In this model, the subtour elimination constraints (2.18) ensure that solution of the follower contains no cycles, where constraints (2.19) guarantee that the follower cannot use the edges that are adjacent to interdicted (deleted) vertices.

It is straightforward to see that any optimal solution of the follower spans the subgraph $G^* = G[V \setminus V_0]$ (except for the vertices with a completely interdicted neighborhood). Indeed, assume that there is a vertex which is not isolated in G^* but has a degree of zero in an optimal follower solution; then adding a random edge adjacent to this vertex improves the value of the follower solution without creating any cycle, fact that leads to a contraction. Hence, the only vertices not spanned by an optimal follower solution are the isolated vertices in the interdicted graph G^*.

The BILP formulation (2.14)-(2.16) is non-continuous and non-linear, hence it cannot be plugged into a general purpose solver. Instead, we propose a linearization of the BILP model that results in a new formulation to which we refer as *Natural Formulation*, since it lays in the space of the natural x_v, $v \in V$, variables.

2.4.2 Single-Level Reformulation

In the following, we propose a linearization of the BILP model (2.14)-(2.16), by reformulating the follower's subproblem in such a way that the set of its feasible solutions does not depend on the leader. We then derive a single-level reformulation with an exponential number of constraints, associated to extreme points of the follower's polytope. This idea, which resembles the Benders decomposition approach for mixed ILPs, is often applied to (network) interdiction problems [32], [33]. The major challenge of this approach is in finding the tightest possible way to reformulate the follower's subproblem, since this reformulation directly affects the quality of the LP relaxation bounds of the associated single-level model. It is known that a tight reformulation is possible in some special cases. For example, if the leader interdicts vertices (edges), and the follower's subproblem admits a hereditary property[2] for its vertex (resp., edge) induced subgraphs, a tight single-level reformulation is possible (see [8], [33]). However, there is no clear rule on how to derive a tight reformulation in general.

In our setting, the leader interdicts *vertices*, but the follower's subproblem is hereditary with respect to *edge-induced* subgraphs, so that the results from [33] cannot be directly applied. Instead, we have the following result:

Proposition 5 *The follower subproblem can be equivalently restated as*

$$\Phi(x^*) = \max \sum_{uv \in E} e_{uv} \cdot \left(1 - x^*_u - x^*_v\right) \tag{2.21}$$

$$e(S) \leq |S| - 1 \qquad S \subseteq V, |S| \geq 3 \tag{2.22}$$

$$e_{uv} \in \{0, 1\} \qquad uv \in E. \tag{2.23}$$

Proof. Any optimal solution e^* of (2.17) -(2.20) corresponds to a maximum cycle-free subgraph in the *interdicted* graph G^*. Instead, notice that in (2.21) -(2.23) the follower solves the *maximum weighted cycle free subgraph* problem on the *original* graph G, with edge weights $w_{uv} := 1 - x^*_u - x^*_v$. However, the weights of an edge uv in E are positive if and only if this edge is not adjacent to any interdicted vertex in V^*. Otherwise, the weight of an edge is zero or -1 (if both its end points are interdicted). Hence, any optimal solution in G^* can be mapped to an optimal solution on G (with the same weight). On the contrary, there always exists an optimal solution on G of the problem (2.21)-(2.23) with positive edge weights only, which corresponds to an optimal solution on G^*.□

Observe that the space of feasible solutions of the redefined follower subproblem does not depend on the leader anymore; the only dependence to the solution of the leader is through the objective function. Hence, we can enumerate all feasible solutions of the follower and restate the whole problem as a single-level formulation. This formulation has an exponential number of constraints, one for each extreme point of the follower polytope.

[2] A hereditary property is a property of a graph which also holds for its induced subgraphs.

Let $\mathcal{T}$ denote the set of all cycle-free subgraphs of G corresponding to extreme points of the polytope defined as the convex hull of all points satisfying constraints (2.22) and (2.23). The non-linear constraint (2.15) from the BILP formulation can now be replaced by the following exponential family of inequalities:

$$\sum_{uv \in E(T)} \big(1 - x_u - x_v\big) \leq n - \sum_{v \in V} x_v - k \qquad T \in \mathcal{T}. \tag{2.24}$$

Since every vertex v is counted $\deg_T(v)$ many times in the above constraints (2.24), they can also be restated as:

$$\sum_{v \in V} \big(\deg_T(v) - 1\big) x_v \geq k - n + |E(T)| \qquad T \in \mathcal{T}. \tag{2.25}$$

The following result shows that constraints (2.25) do not have to be imposed for any extreme point from $\mathcal{T}$, it is namely sufficient to concentrate on spanning subgraphs from $\mathcal{T}$ only. Let $\mathcal{T}_G$ denote the *subset* of extreme points from $\mathcal{T}$ being *spanning subgraphs* in G. The following result holds:

Proposition 6 *The following single-level formulation, denoted as Natural Formulation, is a valid model for the k-vertex cut problem:*

$$(NAT) \qquad \min \sum_{v \in V} w_v x_v \tag{2.26}$$

$$\sum_{v \in V} \big(\deg_T(v) - 1\big) x_v \geq k - n + |E(T)| \qquad T \in \mathcal{T}_G \tag{2.27}$$

$$x_v \in \{0, 1\} \qquad v \in V. \tag{2.28}$$

Proof. It is sufficient to show that any inequality associated to a subgraph $T \in \mathcal{T} \setminus \mathcal{T}_G$ can be replaced by an inequality associated to some $T' \in \mathcal{T}_G$. Let us assume for a moment that $|T| = n - 2$ and let $v \notin V(T)$. To create T', given an integer solution x^* that violates the constraint (2.25), we choose to connect v with some $u \in V(T)$ such that $x^*_v + x^*_u \leq 1$ (this is always possible, unless v and all its neighbours are interdicted). By setting $T' = T \cup \{uv\}$ we obtain a spanning subgraph inequality of type (2.27) with the same violation as for the inequality (2.25). For $|T| < n - 2$, this "growing" of the subgraph T can be subsequently repeated until all vertices of G are spanned by T', without changing the violation of the inequality. Finally, in case an interdicted vertex v has an interdicted neighbourhood (however, this cannot happen in an optimal solution, because removing the interdicted vertex from the vertex-cut would improve the leader solution) we need to add the extra constraints:

$$x_u + \sum_{v \in N(u)} x_v \leq \deg_G(u) \qquad u \in V. \tag{2.29}$$

□

Coefficient lifting. For any $T \in \mathcal{T}_G$, the coefficients next to x_v variables are all non-negative, and hence inequalities (2.27) can be lifted to:

$$\sum_{v \in V} \left(\min \left\{ \gamma, \deg_T(v) - 1 \right\} \right) x_v \geq \gamma \qquad T \in \mathcal{T}_G, \tag{2.30}$$

where $\gamma = k - n + |E(T)|$.

Figures 2.2-2.3 illustrate a cycle-free subgraph $T \in \mathcal{T} \setminus \mathcal{T}_G$ and a spanning cycle-free subgraph $T' \in \mathcal{T}_G$, along with the associated inequalities. Both inequalities are able to cut off the infeasible solution of Figure 2.2.

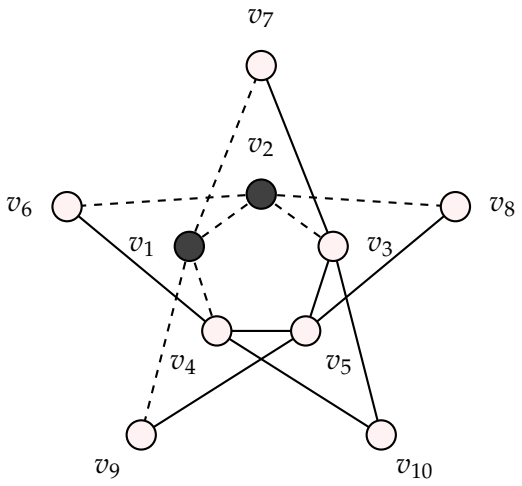


FIGURE 2.2: Infeasible solution for $k = 3$, with the black vertices $\{v_1, v_2\}$ in the vertex cut (the remaining vertices form one connected component).

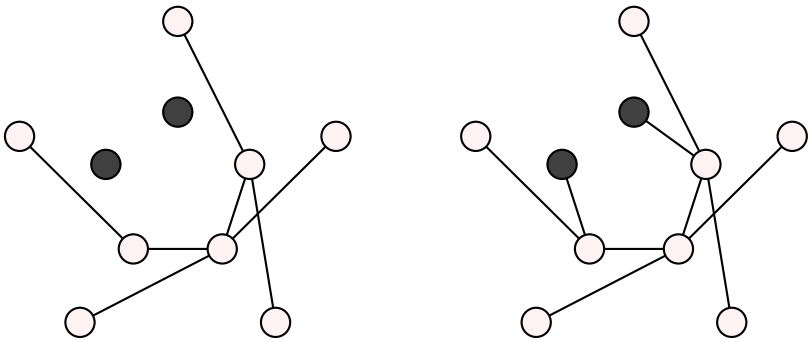
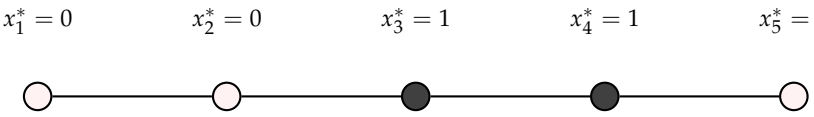


FIGURE 2.3: A cycle-free subgraph $T \in \mathcal{T} \setminus \mathcal{T}_G$ and the associated inequality (2.25): $-x_1 - x_2 + 2x_3 + x_4 + 3x_5 \geq 0$ (left part). A spanning cycle-free subgraph $T \in \mathcal{T}_G$ and the associated inequality (2.27): $3x_3 + 2x_4 + 3x_5 \geq 2$; downlifted according to (2.30) to $x_3 + x_4 + x_5 \geq 1$ (right part).

Finally, given the fact that imposing the inequalities (2.25) associated to spanning subgraphs from $\mathcal{T}_G$ guarantees a valid formulation, a natural question arises: would it be sufficient to impose these inequalities only for spanning trees of G? The following result provides a negative answer to this question:

Proposition 7 *Inequalities* (2.25) *derived from spanning trees only are not sufficient to ensure a valid formulation for the k-vertex cut problem.*

Proof. To prove this result, we provide an instance in which an infeasible solution x^* is not cut off by spanning tree inequalities. Consider a graph composed by a path of 5 vertices, $k = 3$, and the solution x^* depicted in the figure below, where the black vertices represent interdicted ones ($x_3^* = x_4^* = 1$, the remaining values are zero). The solution x^* separates G into only 2 components, hence it is infeasible.



There is a single spanning tree in G, and the associated cut, which is $x_2 + x_3 + x_4 \geq 2$, does not cut off the infeasible point x^*. □

The following propositions characterize the strength of the LP relaxation of the Representative and Natural formulations.

Proposition 8 *If $k \leq n/2$, the bound for the k-Vertex Cut problem provided by the optimal solution value of the LP relaxation of formulation* (2.26)-(2.28) *strictly dominates the corresponding bound provided by the formulation* (2.6)-(2.10).

Proof. We first show that any feasible solution x^* of the LP relaxation of (2.26) -(2.28) can be mapped into a feasible solution of the LP relaxation of (2.6)-(2.10) with the same objective function value. The two objective functions are the same, thus we only have to determine z^* satisfying all the constraints of formulation (2.6)-(2.10). By exploiting Proposition 4, z_u^* can be fixed to n/k, for each $u \in V$. It is straightforward to check that all the constraints of formulation (2.6)-(2.10) are satisfied by (x^*, z^*).

To prove the strictness of the relation, we show that the value of the optimal solution of the LP relaxation of (2.26) -(2.28) is strictly larger then 0 for any graph G which is not yet disconnected in at least k components, while by Proposition 4 those of (2.6)-(2.10) is always 0. Indeed, any solution of value 0 for (2.26)-(2.28) must have $x_u = 0 \ \forall u \in V$. Consider a graph G with q connected components. Any acyclic subgraph of G has at most $n - q$ edges, let t be a subgraph with exactly $n - q$ edges (so it is spanning). By plugging $x_u = 0 \ \forall u \in V$ in (2.24) (which are equivalent to (2.27)) for t, we get $n - q \leq n - k$ which is violated if $k > q$, so the solution of cost 0 is infeasible. □

2.5. Separation Algorithms

In this section, we address separation procedures for the valid inequalities introduced in Sections 2.3 and 2.4.2.

Separation of constraints (2.9)**.** Given a (fractional) solution $x^*, z^* \in [0,1]^V$ to the LP relaxation of model *REP*, separation of constraints (2.9) asks for finding a pair of vertices u, v such that there is a path $P^* \in \Pi_{uv}$ with

$$z_u^* + z_v^* > \sum_{t \in V(P^*) \setminus \{u,v\}} x_t^* + 1. \tag{2.31}$$

For each pair of vertices, we can search for such a path in polynomial time by solving a shortest path problem from u to v on $G(V, E)$, where we define the length of each edge $(i, j) \in E$ as

$$l_{ij} = \frac{x_i^* + x_j^*}{2} \tag{2.32}$$

(note that the constant term $\frac{x_u^* + x_v^*}{2}$ has to be removed from the length of each path).

Concerning the computation of shortest paths, for fractional solutions, one can solve the All Pairs Shortest Path problem through the Floyd Warshall algorithm. In the case of integer solutions, finding a shortest path between a vertex u and all other vertices can be done by performing a simple breath-first search (BFS) procedure in the support graph G^* in which vertices v such that $x_v^* = 1$ are removed. The BFS tree guarantees that each vertex v at layer ℓ in that tree has the shortest distance ℓ from the source u. If the vertices are not connected, the distance is ∞. Hence, separation of integer solutions can be done in $O(|V||E|)$ time.

Observation 4 *Separation of constraints* (2.9) *can be performed in polynomial time.*

Separation of constraints (2.11)**.** Constraints (2.11), that are the lifted version of (2.9), can be still carried on by solving a shortest path problem from u to v on $G(V, E)$, where we define the length of each edge $(i, j) \in E$ as

$$l_{ij} = \frac{x_i^* + x_j^*}{2} - \frac{z_i^* + z_j^*}{2}, \tag{2.33}$$

(still the constant term $\frac{x_u^* + x_v^*}{2}$ has to be removed from the length of each path). Since edges can have negative weight, we solve heuristically this problem in two steps (where the first step can be skipped):

- first we heuristically solve a *longest path* problem with lengths as defined in (2.33) with opposite sign. We implemented a greedy procedure that obtains such *long path* starting from the edge with the largest weight and then it builds a path by adding the edge with the largest and positive weight that is adjacent to the current path, without closing a cycle;
- second, if in the previous step no violated cut is found, we compute the shortest paths P^* with the nonnegative lengths as defined in (2.32), and then check the value of the z_w^* variables for $w \in V(P^*) \setminus \{u, v\}$. This way, we have a separation procedure that is exact for (2.9) and heuristic for (2.11).

Separation of constraints (2.27)**.** Let x^* be the current solution to the LP relaxation of model NAT. We define edge-weights as

$$w_{uv}^* = 1 - x_u^* - x_v^*, \quad uv \in E$$

and search for the maximum-weighted cycle-free subgraph in G. Let W^* denote the weight of the obtained subgraph; if $W^* > n - k - \sum_{v \in V} x_v^*$, we have detected a violated inequality.

For fractional points x^* the maximum-weighted cycle-free subgraph can be detected in $O(|E| \log |V|)$ by running an adaptation of Kruskal's algorithm for minimum-spanning trees. Edges are sorted in a non-increasing order according to their weight,

and then Kruskal's algorithm is applied, i.e., each edge in this ordering is selected to be included in the subgraph being constructed, provided it does not close a cycle. The algorithm stops as soon as an edge with negative weight is encountered in the ordering.

Separation of integer points x^* can be performed in $O(|E|)$ time. In this case, all edge weights are equal to 1, 0 or -1. Following the result of Proposition 5, it is sufficient to consider the graph defined by edges with weight equal to one, which corresponds to $G^* = G[V \setminus V_0]$ where V_0 are interdicted vertices encoded by x^*. Hence, it is sufficient to run any graph traversal algorithm on G^* (like, e.g., BFS) to find connected components in G^*.

Observation 5 *Separation of constraints* (2.27) *can be performed in polynomial time.*

Since there are several alternative subgraphs describing connected components, to avoid shallow cuts, when separating integer points we shuffle the set of edges of G^* before each separation call. This procedure guarantees to find a cut of type (2.25), where the associated subgraph T is not necessarily spanning all vertices from V.

However, it is (always) possible to construct a Spanning Subgraph cut starting from an infeasible integer solution x^* and a cut associated with a (nonspanning) acyclic subgraph $T \in \mathcal{T}$ violated by this solution. To do so, we scan first isolated vertices in the interdicted graph G^* and we assign them an interdicted neighbor, then for all still non-spanned interdicted vertices we assign them to one of their neighbors in $V \setminus V_0$. In this way we do not change the weight of the obtained $T \in \mathcal{T}$ since these edges have 0 weight. This repairing step requires $O(|E|)$ steps, so that the total separation time remains $O(|E|)$.

2.6. Computational results

The goal of our computational experiments is to test the performance of the proposed formulations, i.e., the Representative Formulation *REP* (Section 2.3) and the Natural Formulation *NAT* (Section 2.4.2). Both formulations, having an exponential number of constraints, are solved within a branch-and-cut framework. We have proposed several variants and valid inequalities for each formulation, and thus a second goal of this section is to identify their best configuration. In addition, we propose and test a *Hybrid Formulation* obtained by combining elements of the two formulations.

Finally, we assess the computational performance of our best branch-and-cut algorithm by comparison with the Compact Formulation *COMP* (Section 2.1), and with a state-of-the-art branch-and-price algorithm proposed in [11], and based on a formulation with exponentially many variables.

The source code of our branch-and-cut algorithm can be downloaded at `https://github.com/paoloparonuzzi/k-Vertex-Cut-Problem/`. The software was given the DOI (Digital Object Identifier) `https://doi.org/10.5281/zenodo.3333560`.

2.6.1 Experimental Setting

Benchmark instances. In our experiments we have two sets of instances, which are the ones considered in the computational experiments of [11]. All instances have weights $w_v = 1$ for all $v \in V$. The first set includes all the classical Vertex Coloring instances [34] having up to 200 vertices, and all the 10th DIMACS instances [35] having up to 300 vertices (instances with $\alpha(G) \geq 5$). The features of the 59 selected

instances are reported in the first part of Table 2.1 where, after the instance name, we show the number of vertices (n) and edges (m), the stability number ($\alpha(G)$), and the optimal solution value of the k-Vertex Cut problem (size of the optimal vertex cut) for values of $k \in \{5, 10, 15, 20\}$, when it can be found by one of the methods discussed in this section or in [11]. Missing entries correspond to infeasible problems ($\alpha(G) < k$), while unknown optimal values are indicated by a "-" (these are the instances which are not solved within timelimit). Trivially solved instances are indicated by a "·" (these are the instances which, before or after preprocessing, have q connected components, with $q \geq k$). The second set of 59 instances, whose features are given in Table 2.2, were proposed in [10]. This set is a collection of intersection graphs of the coefficient matrices of linear equations systems, arising from various applications. When solving the k-Vertex Cut problem for a given value of k, we remove from our analysis all infeasible and trivial instances.

All the instances are preprocessed off-line by checking the condition of Proposition 3. In particular, for each vertex the stability number of its anti-neighborhood is computed and, when the condition of the proposition is met, the vertex is removed. Although this asks for solving an NP-hard Maximum Stable Set problem, the associated computing time in negligible for the size of graphs we consider. As long as at least a vertex is removed from the graph, the procedure is iteratively repeated. In our testbed, graph reductions are achieved only for a limited subset of instances, namely, 20, 11, 17 and 16 instances for $k = 5, 10, 15$ and 20, respectively. While for many instances only one or two vertices are removed, in some cases many vertices are removed, with up to 113 vertices out of 125. In 6 cases the resulting instance is solved (i.e., it is disconnected in q components, with $q \geq k$). These instances are marked as trivial in Tables 2.1 and 2.2. Preprocessing is applied before instances are tackled by any of the solution algorithms here described, in other words, all methods receive the same input (preprocessed) graph.

Detailed results for the preprocessing are reported in the Appendix.

Computational environment. All the experiments, including the runs of the branch-and-price algorithm from [11], are performed on a computer with an i7-6900K processor clocked at 3.20 GHz and 64 GB RAM under GNU/Linux Ubuntu 16.04. We use `CPLEX 12.7.1` and the `Concert Technology` framework to implement our branch-and-cut algorithms. The Compact Formulation is solved with the `CPLEX` MIP solver. `CPLEX` is run in single-threaded mode and all `CPLEX` parameters are set to their default values. A time limit of one hour is set for each tested instance.

2.6.2 Results for Representative, Natural and Hybrid Formulations

We tested several different configurations of the Representative Formulation (e.g., changing the separation strategy, removing strengthening constraints, etc.), and we report detailed computational results for the following two configurations:

- we denote by REP_p the formulation (2.6) - (2.8), (2.10) - (2.13). Constraints (2.11) are separated by only applying the second step of the procedure described in Section 2.5, that is, by computing shortest paths on a graph with positive edge weights;
- we denote by REP_{lp-p} the same formulation, where (2.11) are separated by applying both steps of the procedure described in Section 2.5, that is, by heuristically computing a long path in a graph with positive and negative edge weights.

				Optimal Values								Optimal Values			
	n	m	$\alpha(G)$	$k=5$	$k=10$	$k=15$	$k=20$		n	m	$\alpha(G)$	$k=5$	$k=10$	$k=15$	$k=20$
1-FullIns_3	30	100	14	7	11			miles750	128	2113	12	20	75		
1-FullIns_4	93	593	45	9	13	18	22	mug100_1	100	166	33	5	10	15	20
1-Insertions_4	67	232	32	7	12	16	22	mug100_25	100	166	33	5	10	15	20
2-FullIns_3	52	201	25	8	13	17	23	mug88_1	88	146	29	4	9	15	20
2-Insertions_3	37	72	18	6	10	16		mug88_25	88	146	29	4	9	14	19
2-Insertions_4	149	541	74	7	11	17	22	mulsol.i.2	188	3885	90	·	·	·	18
3-FullIns_3	80	346	37	9	14	17	25	mulsol.i.3	184	3916	86	·	·	18	19
3-Insertions_3	56	110	27	6	11	16	21	mulsol.i.4	185	3946	86	·	·	18	19
4-FullIns_3	114	541	55	9	15	18	23	mulsol.i.5	186	3973	88	·	·	18	19
4-Insertions_3	79	156	39	6	11	16	21	myciel3	11	20	5	·			
5-FullIns_3	154	792	72	9	15	19	22	myciel4	23	71	11	7	12		
adjnoun	112	425	53	2	6	11	16	myciel5	47	236	23	8	13	18	23
anna	138	493	80	1	1	2	2	myciel6	95	755	47	9	14	19	24
celegansneural	297	2148	110	1	1	2	6	myciel7	191	2360	95	10	15	20	25
chesapeake	39	170	17	7	12	17		polbooks	105	441	43	8	15	19	25
david	87	406	36	·	·	4	9	queen10_10	100	1470	10	-	90		
dolphins	62	159	28	2	7	13	19	queen11_11	121	1980	11	-	-		
DSJC125.1	125	736	34	-	-	-	-	queen12_12	144	2596	12	-	-		
DSJC125.5	125	3891	10	-	115			queen13_13	169	3328	13	-	-		
football	115	613	21	-	-	-	71	queen14_14	196	4186	14	-	-		
games120	120	638	22	-	-	-	67	queen5_5	25	160	5	20			
huck	74	301	27	1	3	6	9	queen6_6	36	290	6	28			
jazz	198	2742	40	4	12	25	-	queen7_7	49	476	7	38			
jean	80	254	38	1	1	2	4	queen8_12	96	1368	8	-			
karate	34	78	20	2	4	6	11	queen8_8	64	728	8	48			
lesmis	77	254	35	1	2	3	5	queen9_9	81	1056	9	59			
miles1000	128	3216	8	53				r125.1	125	209	49	·	·	1	5
miles1500	128	5198	5	115				r125.1c	125	7501	7	116			
miles250	128	387	44	·	·	4	11	r125.5	125	3838	5	91			
miles500	128	1170	18	7	-	-									

TABLE 2.1: Instance features (Coloring and DIMACS)

				Optimal Values								Optimal Values			
	n	m	$\alpha(G)$	$k=5$	$k=10$	$k=15$	$k=20$		n	m	$\alpha(G)$	$k=5$	$k=10$	$k=15$	$k=20$
arc130	130	7763	6	83				L120.fidap022	120	4307	5	87			
ash219	85	219	29	7	16	26	34	L120.fidap025	120	2787	5	·			
ash331	104	331	30	8	21	-	-	L120.fidapm02	120	4626	5	91			
ash85	85	616	14	22	-			L120.rbs480a	120	3273	6	76			
bcspwr01	39	118	13	7	16			L120.wm2	120	3387	23	3	8	13	41
bcspwr02	49	177	16	7	16	24		L125.ash608	125	390	37	8	-	-	-
bcspwr03	118	576	32	10	23	35	46	L125.bcsstk05	125	2701	9	41			
bfw62a	62	639	8	22				L125.can__161	125	1257	15	-	-	-	
can__144	144	1656	12	-	-			L125.can__187	125	1022	20	-	-	-	102
can61	61	866	6	39				L125.dwt__162	125	943	16	-	-	-	
can62	62	210	18	7	17	27		L125.dwt__193	125	2982	8	56			
can73	73	652	13	28	-			L125.fs_183_1	125	3392	9	16			
can96	96	912	10	-	-			L125.gre__185	125	1177	19	27	-	-	
curtis54	54	337	9	16				L125.lop163	125	1218	17	-	-	-	
dwt___59	59	256	15	10	25	41		L125.west0167	125	444	39	5	11	17	24
dwt66	66	255	13	15	-			L125.will199	125	386	45	5	13	20	27
dwt72	72	170	24	7	16	26	36	L80.cavity01	80	1201	31	10	10	20	31
dwt87	87	726	16	11	29	54		L80.fidap025	80	1201	5	·			
gre__115	115	576	33	12	24	-	-	L80.steam2	80	1272	6	48			
ibm32	32	179	8	16				L80.wm1	80	1786	15	15	36	49	
impcol_b	59	329	20	5	13	23	38	L80.wm2	80	1848	11	4	48		
L100.cavity01	100	1844	36	10	19	21	32	L80.wm3	80	1739	13	4	12		
L100.fidap025	100	2031	5	·				lund_a	147	2837	10	-	-		
L100.fidapm02	100	3090	5	80				pores_1	30	179	6	20			
L100.rbs480a	100	2550	5	66				rw136	136	641	39	7	-	-	-
L100.steam2	100	1766	6	56				steam3	80	712	7	32			
L100.wm1	100	2956	17	15	28	48		west0067	67	411	12	20	-		
L100.wm2	100	3039	12	4	41			west0132	132	560	39	5	12	21	29
L100.wm3	100	2934	15	4	12	53		will57	57	304	10	7	22		
L120.cavity01	120	2972	36	10	21	23	32								

TABLE 2.2: Instance features (Intersection graphs)

Different frequencies and tolerances for the separation procedure were tested for all configurations. According to our extensive preliminary computational experiments, the best choice is to stop the cut separation when the absolute violation is smaller than 0.5 (*violation tolerance*). We call the separation procedure for all integer points and for fractional points every 100 nodes of the branching tree.

Inequalities (2.8), that are expressed for each edge in $E(G)$, can be strengthened to clique inequalities. However (as confirmed by our preliminary computational experiments) modern MIP solvers are very effective in the automatic separation of clique inequalities, and hence we keep edge constraints in our formulation.

Similarly, we tested several different configurations of the Natural Formulation, and we report detailed computational results for the following two:

- we denote by NAT the formulation (2.25), (2.26) and (2.28), where (2.25) are lifted to (2.30) when spanning;
- we denote by NAT_s the previous formulation where the family of constraints (2.25) are made spanning for all integer solutions, and then lifted to (2.30).

We tested different frequencies and tolerances of the separation procedure and the best choice for the violation tolerance is also in this case 0.5. We call the separation procedure for all integer points and for fractional points at all the nodes of the branching tree.

The Representative and the Natural Formulations use the same natural variables x_v, $v \in V$, to describe which vertices are in the k-vertex cut, and implement alternative sets of contraints to impose the required number of nonempty disconnected components. Although the Natural Formulation showed more effective than the Representative Formulation (see results in the following), there are some instances on which the latter has a better performance. In addition, in our preliminary computational experiments we observed that, thanks to the presence of a stable set constraints (2.8), the Representative Formulation is much faster in detecting infeasible instances (i.e., those with $\alpha(G) < k$). Infeasible instances were removed from our testbed, however, we expect the Representative Formulation to be fast in detecting infeasibilities also at the nodes on the branch-and-cut tree. Therefore, it makes sense trying to obtain a more effective formulation by integrating the two into a *Hybrid* model.

In order to explore the direction of embedding into the Natural Formulation the advantages of the Representative one (i.e., solving some specific instance and fast detection of infeasibilities after branching), we designed the following Hybrid configuration:

- we denote by HYB_s Formulation NAT_s with additional constraints (2.7), (2.8), (2.10), (2.12) and (2.13).

Aggregated results for the first set of instances (Vertex Coloring and DIMACS) are reported in Table 2.3, where the first column gives the considered value of k. Then the table reports, for each configuration of the Representative, Natural and Hybrid Formulations described above, the number of instances solved to optimality; the average computing time in seconds (for the subset of instances solved to optimality by all configurations), the average number of explored nodes in the branching tree (for the subset of instances solved to optimality by all configurations); the average percentage gap of the LP relaxation computed as $100 \cdot ((UB - LP)/UB)$, where

UB is the optimal or best known solution value and LP is the optimal value of the LP relaxation; the average time to solve the LP relaxation. Violation tolerance is set to 0.1 when solving LPs. The last three rows of the table report the averages over all values of k.

The configurations reported in Table 2.3 have improving performance. When moving from REP_p to REP_{lp-p}, the number of instances solved to optimality is increased for all value of k, except $k = 5$. The improved results are explained by comparing the values of the LP gap of REP_p and REP_{lp-p}: the table clearly shows that separating inequalities (2.11) by applying both steps of the procedure described in Section 2.5 allows to close much more LP gap. Using Natural Formulations (NAT and NAT_s) for all values of k the number of instances solved to optimality is increased, and the number of nodes explored by the branch-and-cut algorithm is reduced by 3 orders of magnitude. This can be attributed to the significantly smaller LP relaxation gaps of Natural Formulations, when compared to those obtained using Representative Formulations. Comparing formulations NAT and NAT_s, the latter has a slightly better performance, and can solve 2 more instances on the whole set. Finally, the table shows that the best computational performances is provided by HYB which is able to solve 132 instances (out of 169). The number of explored nodes by the branch-and-cut algorithm is one third of that of NAT_s. As anticipated, this is as a result of the introduction of the constraints from the Representative Formulation, which allow to fast detect infeasible nodes in the branching tree. Summarizing from Table 2.3 we can conclude that HYB is the best formulation proposed in this chapter. We now compare its performances with the state-of-the-art algorithm present in the literature for the k-Vertex Cut problem.

2.6.3 Comparison with state-of-the-art solution methods

In this section we compare the results of our best formulation (HYB_s) with the solution of the *Compact Formulation* (denoted as $COMP$) solved by means of the general purpose `CPLEX` MIP solver, and with a state-of-the-art branch-and-price algorithm proposed in [11] (denoted as BP).

When solving the *Compact Formulation*, as suggested in [11], the formulation is enhanced by a preprocessing phase in which a subset of variables is removed so as to reduce the symmetry of the formulation and to improve the quality of the associated LP relaxation. In this preprocessing, we search for $k-1$ vertex-disjoint cliques $C_1, \ldots, C_i, \ldots, C_{k-1}$ of the graph G, and remove the following variables

$$y_v^h, \quad i = 1, \ldots, k-1, \quad v \in C_i, \quad h = i+1, \ldots, k. \tag{2.34}$$

Indeed, two vertices u, v of a clique cannot be in two different subsets V_i and V_j. Then for all solutions we can reorder the sets $V_1, ..., V_k$ to ensure that each vertex of a clique C_i must be in one set V_j $j \leq i$ or in the vertex cut. Thus we can remove the variables (2.34) to reduce the symmetry.

The comparison, whose results are reported in Table 2.4, is performed on the whole set of instances including Vertex Coloring, DIMACS and Intersection graphs described in Section 2.6.1. The table has the same structure of the previous one, and reports the number of instances solved to optimality, the average computing time in seconds and the average number of explored nodes (for solved instances). The table clearly shows that HYB is the best performing method on average, being able to solve 202 out of the 304 tested instances. $COMP$ and BP can both solve 168 instances. On the subset of instances that are solved by all the three methods, the computing

TABLE 2.3: Performance comparison for different configurations of the Representative, Natural and Hybrid Formulations on the first set of instances (Vertex Coloring and DIMACS).

k		REP_p	REP_{lp-p}	NAT	NAT_s	HYB_s
	Opt. (out of 51)	29	27	33	34	35
	Avg Time	148.70	105.57	7.40	3.79	1.07
5	Avg Nodes	61524	24174	70	73	29
	LP Avg Gap	89.55	67.15	22.96	22.76	22.85
	LP Avg Time	0.01	0.17	0.24	0.21	0.32
	Opt. (out of 41)	20	23	29	30	32
	Avg Time	201.66	319.21	2.11	1.52	2.43
10	Avg Nodes	41683	32568	6	7	5
	LP Avg Gap	72.27	46.34	13.88	13.94	14.00
	LP Avg Time	0.05	1.32	0.37	0.33	0.54
	Opt. (out of 38)	22	24	33	32	33
	Avg Time	96.75	52.17	316.91	226.47	3.57
15	Avg Nodes	48078	10923	39	35	12
	LP Avg Gap	65.99	48.75	16.91	16.96	16.94
	LP Avg Time	0.06	138.57	0.18	0.17	0.33
	Opt. (out of 36)	18	22	31	32	32
	Avg Time	141.32	351.13	190.94	41.70	3.66
20	Avg Nodes	47735	25595	58	48	11
	LP Avg Gap	58.65	38.37	17.12	17.11	17.12
	LP Avg Time	0.07	1.93	0.24	0.24	0.49
	Total Opt. (out of 166)	89	96	126	128	132
	Total Avg Time	146.75	194.04	121.10	66.20	2.55
	Total Avg Nodes	50656	23169	45	43	15
	Total Avg LP Gap	73.19	51.69	18.11	18.07	18.11
	Total Avg LP Time	0.04	34.98	0.25	0.24	0.41

TABLE 2.4: Performance comparison between the Hybrid Formulation and the state-of-the-art methods on the complete instance set (Vertex Coloring, DIMACS and Intersection graphs).

k		$COMP$	BP	HYB_s
	Opt. (out of 107)	92	60	71
5	Avg Time	31.84	59.93	84.78
	Avg Nodes	10768	30	106
	Opt. (out of 80)	37	43	51
10	Avg Time	105.64	52.19	1.39
	Avg Nodes	67123	7	26
	Opt. (out of 65)	29	36	46
15	Avg Time	219.33	23.38	2.81
	Avg Nodes	41750	19	25
	Opt. (out of 52)	19	29	38
20	Avg Time	196.06	169.52	0.39
	Avg Nodes	58673	16	6
	Total Opt. (out of 304)	177	168	206
	Total Avg Time	98.66	61.78	43.66
	Total Avg Nodes	33040	22	64

time of BP is approximately 2/3 the computing time of $COMP$, while the computing time of HYB is approximately halved with respect to the computing time of $COMP$. An important information is given by the average number of nodes explored in the branch-and-cut tree, in particular $COMP$ explores ≈33,000, BP ≈22 and HYB ≈64 nodes, respectively. By analyzing these figures, it clearly emerges that $COMP$ explores many more nodes than the other two methods. This fact is due to the poor quality of the LP relaxation bound provided by the Compact Formulation. BP and HYB explore fewer nodes, and the reason is the quality of the LP bounds provided by these formulations. BP is the algorithm which explores the smallest number of nodes on average. By analyzing the results for each value of k separately, the table shows that $COMP$ provides the best computational performances for $k = 5$ but then, as far as $k \geq 10$, HYB always guarantees the best computational performances on this set of instances, being able to solve 49 out of 80 instances, 46 out of 65 and 38 out of 52, for $k = 10$, $k = 15$ and $k = 20$, respectively. Also the BP algorithm shows a better performance than $COMP$ as soon as $k \geq 10$.

A graphical representation of the relative performance of the three compared approaches is given by the performance profiles of Figures 2.4 and 2.5, for unweighed and weighted (see Section 2.6.3) instances respectively. Following the guidelines suggested by [36], the performance profiles are defined as follows. Let m be any solution method and i denote an instance of the problem. In addition let $t_{i,m}$ be the time required by method m to solve instance i. We define the *performance ratio* for pair (i, m) as

$$r_{i,m} = \frac{t_{i,m}}{\min_{m \in M}\{t_{i,m}\}}$$

TABLE 2.5: Performance comparison between the Hybrid Formulation and the state-of-the-art methods on the complete instance set with weights (Vertex Coloring, DIMACS and Intersection graphs).

k		$COMP$	BP	HYB_s
	Opt. (out of 107)	92	60	71
5	Avg Time	35.99	67.55	210.67
	Avg Nodes	11350	77	217
	Opt. (out of 80)	37	43	51
10	Avg Time	69.61	174.96	2.30
	Avg Nodes	22872	21	26
	Opt. (out of 65)	29	37	47
15	Avg Time	343.26	36.61	21.76
	Avg Nodes	109726	180	86
	Opt. (out of 52)	19	30	39
20	Avg Time	559.17	300.40	1.15
	Avg Nodes	180529	31	15
	Total Opt. (out of 304)	177	170	208
	Total Avg Time	151.21	112.23	106.13
	Total Avg Nodes	48594	77	127

where M is the set of the considered methods. Then, for each method $m \in M$, we define:

$$\rho_m(\tau) = \frac{|\{i \in I : r_{i,m} \leq \tau\}|}{|I|}$$

where I is the set of the instances. Intuitively, $r_{i,m}$ denotes the worsening (with respect to computing time) incurred when solving instance i using method m instead of the best possible one, whereas $\rho_m(\tau)$ gives the percentage of instances for which the computing time of method m was not larger than τ times the time of the best performing method. For each value of τ in the horizontal axis, the vertical axis reports the percentage of the instances for which the corresponding algorithm spends no more than τ times the computing time of the fastest algorithm. The curves originates from a point denoting the percentage of instances for which the corresponding algorithm is the fastest, and at the right end of the chart, they show the percentage of instances solved within time limit. The best performance algorithm is graphically represented by the curve in the upper part of the Figures. The horizontal axis is represented in logarithmic scale. The figures clearly show that the relative performance of the 3 algorithms depends on the value k considered.

For $k = 5$, Figure 2.4 shows that HYB and $COMP$ are the fastest method in ≈40% of the instances. HYB can solve ≈65% of the instances, while $COMP$ can solve ≈85%, and the corresponding curve dominates those of HYB in most of the chart. BP is the fastest method in ≈5% and can solve ≈55% of the instances. For $k = 5$, the best option appears to solve the problem by means of the $COMP$ formulation. As soon as the value of k increases, the performance of the three solution methods changes. For $k = 10$, the figure shows that HYB is the fastest method in ≈50% and it can solve ≈60% of the instances. It dominates the other two methods on the whole chart; BP is the fastest method in ≈20% and can solve ≈50% of the instances, while $COMP$ is the fastest method in ≈10% and can solve ≈40% of the instances.

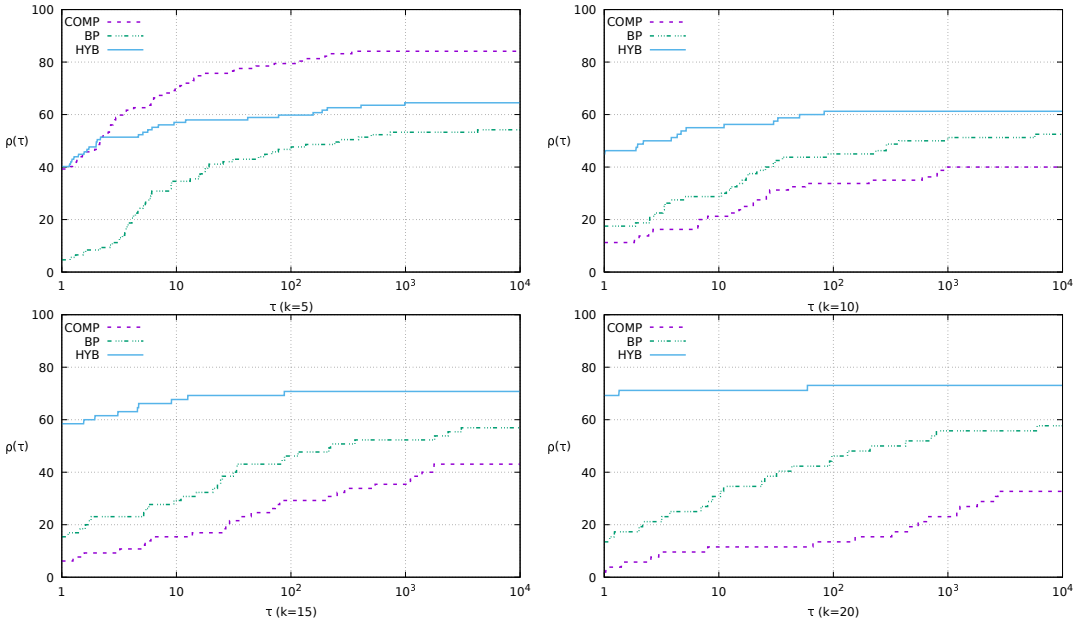


FIGURE 2.4: Performance profile of exact methods for the k-Vertex Cut problem.

The primacy of HYB increases with increasing k: for $k = 15$, the figure shows that HYB is the fastest method in ≈60% and it is able to solve ≈70% of the instances. It dominates the other two methods on the whole chart; BP is the fastest method in ≈15% and can solve ≈60% of the instances, while $COMP$ is the fastest method in ≈5% and can solve ≈40% of the instances. For $k = 20$, the figure shows that shows that HYB is the fastest method in ≈70% and it is able to solve ≈75% of the instances. It dominates the other two methods on the whole chart; BP is the fastest method in ≈15% and can solve ≈55% of the instances, while $COMP$ is the fastest method in less than 5% and can solve ≈30% of the instances.

Summarizing for $k = 5$ the best method on average is $COMP$ which is able to solve the largest percentage of the instances, even if HYB remains the fastest in almost half of them. For all the other values of k, i.e., $k \in \{10, 15, 20\}$, the best computational performance is provided by HYB which is always able to solve the largest percentage of the instances and it is always the fastest methods in more that 50% of them. As far as the comparison between $COMP$ and BP is concerned, the results we obtain are in line with the results presented in [11], i.e., BP is dominated by $COMP$ when $k = 5$, while an opposite behavior is experienced for larger values of k.

Weighted case

In the previous sections we focused the computational analysis on the case where vertices have the same weight (without loss of generality, equal to 1), but all the described formulations, as well as the BP algorithm can also tackle the *weighted case*, that is, the case in which each vertex $v \in V$ has an integer weight w_v. According to our computational experiments the best among the formulations proposed in this chapter for the weighted case is still HYB. Hence, in this section we report on the performance of HYB, $COMP$ and BP on the complete set of instances including Vertex Coloring, DIMACS and Intersection graphs, where a random integer weight with uniform distribution in $\{1, \ldots, 10\}$ is generated for each vertex $v \in V$. As reported in

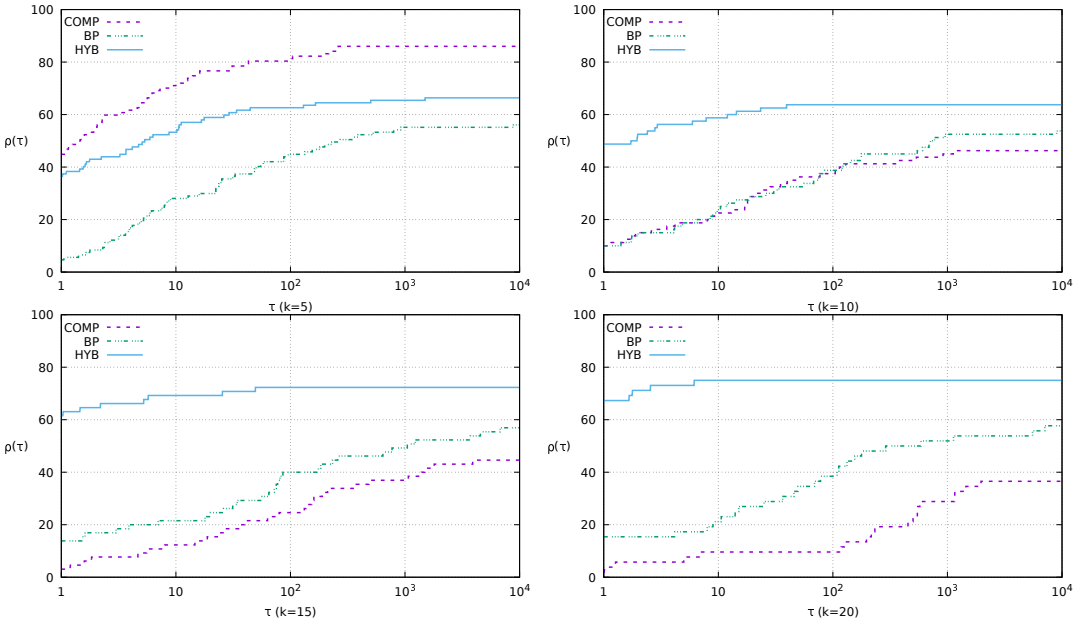


FIGURE 2.5: Performance profile of exact methods for the k-Vertex Cut problem with weights.

Table 2.5, the results in terms of number of solved instances are very similar to those obtained in the unweighted case, confirming the superior performance of HYB, with 208 out of 304 instances solved to optimality, followed by $COMP$ and BP with 177 and 170 solved instances, respectively. The distribution of optimal solution among the separate values of k shows that $COMP$ provides the best computational performances for $k = 5$ but then, as far as $k \geq 10$, HYB is always the best method, and BP always performs better than $COMP$. Despite the (almost identical) number of solved instances by each algorithm, the weighted instances appear more challenging for what concerns computing times and number of Branch-and-Bound nodes: $COMP$ requires approximately 50% more nodes and seconds while both BP and HYB approximately double the number of Branch-and-Bound nodes and the computing time.

Performance profiles for the weighted case are reported in Figure 2.5, and are very close to the profiles obtained in the unweighted case. For $k = 5$, the curve corresponding to $COMP$ dominates that of HYB, and the best option appears to solve the problem by means of the $COMP$ formulation. The performance of BP is the worst. As soon as $k = 10$, the performance of HYB becomes the best. The primacy of HYB increases with increasing k and it largely dominates the other solution methods. Further details on the experiments for the weighted case are reported in the Appendix.

2.7. Conclusions

We have considered a prototype problem in the family of Critical Node Detection Problems, that is, the problem of removing a (minimum weight) set of vertices from a graph so as to disconnect the resulting graph in several components. The so-called k-vertex cut problem has relevant applications not only in network analysis, but also in matrix decomposition for solving systems of equations by parallel computing.

We have described two new integer linear programming formulations, both involving an exponential number of constraints for which we provided separation procedures and implemented branch-and-cut algorithms.

Both formulations use a *natural* set of variables to identify the removed vertices (the k-vertex cut). The first considers additional variables to denote which vertex is *representative* of each component of the disconnected graph, while in the second formulation, the model is derived from the perspective of a two-phase Stackelberg game in which a leader deletes the vertices in the first phase, and in the second phase a follower builds connected components in the remaining graph.

Extensive computational experiments on a set of benchmark instances allowed us to identify the strengths and weaknesses of the two formulations, that in the end we combined in a hybrid one. The experiments also showed that the hybrid formulation significantly outperforms a state-of-the-art branch-and-price method recently proposed for the problem.

The presented idea of looking into the k-vertex cut problem from the perspective of a two-players Stackelberg game can be used in a more general setting for solving Critical Node/Edge Detection Problems. Derivation of new formulations in the natural space of decision variables for this large family of problems will be subject of future research.

Appendix

In Table 2.6 we report, for each considered value of k, the number of vertices removed by preprocessing in the instances where it has some effects. In the first two columns of the table, we report the number of vertices (n) and the number of edges (m) of the graph. In Table 2.7 we report, for each instance (Vertex Coloring and DIMACS) and for each considered value of k, the computational times required by $COMP$, BP and HYB to find the optimal value ("tl" means that the time limit is reached). In tables 2.8, 2.9 and 2.10, the same information is reported for Intersection graph instances, Vertex Coloring and DIMACS instances in the weighted case and Intersection graph instances in the weighted case, respectively. In table 2.11, we report the optimal values for each weighted instance (Vertex Coloring and DIMACS) and for each considered value of k (instances that are infeasible and/or trivial for all values of k are not reported). In table 2.12, the same information for Intersection graph instances in the weighted case is reported.

TABLE 2.6: Number of vertices removed by preprocessing

	n	m	$k = 5$	$k = 10$	$k = 15$	$k = 20$
2-FullIns_3	52	201				6
2-Insertions_3	37	72			1	
chesapeake	39	170	1	2	5	
david	87	406	1	1	1	1
DSJC125.5	125	3891		113		
football	115	613				3
huck	74	301			1	1
karate	34	78		1	2	7
miles1500	128	5198	108			
mulsol.i.2	188	3885				3
mulsol.i.3	184	3916			3	3
mulsol.i.4	185	3946			3	3
mulsol.i.5	186	3973			3	3
myciel3	11	20	6			
myciel4	23	71		7		
myciel5	47	236			3	8
r125.1c	125	7501	97			
r125.5	125	3838	8			
bcspwr02	49	177			11	
can61	61	866	11			
dwt___59	59	256			31	
dwt87	87	726			13	
impcol_b	59	329				34
L100.cavity01	100	1844				2
L100.fidap025	100	2031	66			
L100.fidapm02	100	3090	57			
L100.rbs480a	100	2550	64			
L100.wm1	100	2956		10	37	
L100.wm3	100	2934			50	
L120.cavity01	120	2972				2
L120.fidap022	120	4307	80			
L120.fidap025	120	2787	80			
L120.fidapm02	120	4626	50			
L120.rbs480a	120	3273	34			
L120.wm2	120	3387				23
L125.can__161	125	1257			32	
L125.can__187	125	1022				73
L125.dwt__162	125	943			5	
L125.dwt__193	125	2982	4			
L125.fs_183_1	125	3392	1			
L80.cavity01	80	1201				8
L80.fidap025	80	1201	52			
L80.steam2	80	1272	4			
L80.wm1	80	1786	2	15	47	
L80.wm2	80	1848		29		
lund_a	147	2837		21		
pores_1	30	179	4			
west0067	67	411		3		
will57	57	304		16		

	COMP				*BP*				HYB_s			
	$k=5$	$k=10$	$k=15$	$k=20$	$k=5$	$k=10$	$k=15$	$k=20$	$k=5$	$k=10$	$k=15$	$k=20$
1-FullIns_3	0.38	2.10			0.66	2.25			0.15	0.14		
1-FullIns_4	48.53	tl	tl	tl	tl	tl	tl	tl	13.51	16.40	21.42	5.78
1-Insertions_4	16.74	tl	2717.20	tl	174.89	48.54	56.18	55.01	2.96	2.80	2.46	4.99
2-FullIns_3	5.70	1329.02	217.61	560.95	51.79	34.25	23.20	20.38	8.95	1.34	0.74	0.47
2-Insertions_3	0.93	51.14	334.47		1.90	21.97	20.11		0.45	0.25	0.24	
2-Insertions_4	247.02	tl	tl	tl	tl	tl	tl	tl	198.26	163.67	674.54	1159.77
3-FullIns_3	25.32	tl	tl	tl	tl	tl	70.38	579.20	43.78	39.64	2.89	23.00
3-Insertions_3	8.91	tl	tl	tl	13.35	24.88	24.64	24.35	0.83	1.73	2.59	2.76
4-FullIns_3	60.07	tl	tl	tl	tl	tl	tl	tl	2520.50	829.77	159.96	113.36
4-Insertions_3	36.61	tl	tl	tl	29.07	34.23	34.50	34.33	2.20	13.72	19.13	15.83
5-FullIns_3	311.26	tl	tl	tl	tl	tl	tl	tl	2968.00	2161.91	247.28	72.32
adjnoun	1.35	82.85	854.49	tl	2200.58	866.14	tl	1777.17	0.01	2.98	10.71	18.32
anna	0.31	0.26	0.73	1.55	42.60	57.32	23.51	60.09	0.00	0.00	0.01	0.01
celegansneural	3.96	16.08	88.06	tl	tl	tl	tl	tl	0.02	0.02	0.05	11.17
chesapeake	0.39	6.74	6.59		0.62	5.70	11.53		0.17	0.54	0.10	
david	·	·	12.28	753.15	·	·	78.22	27.56	·	·	0.89	1.16
dolphins	0.15	8.22	165.39	tl	5.03	15.57	26.81	26.44	0.00	0.52	0.79	2.56
DSJC125.1	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl
DSJC125.5	tl	0.27			tl	0.01			tl	0.00		
football	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl
games120	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl
huck	0.06	5.96	7.60	5.31	3.89	9.95	53.86	10.64	0.00	0.01	0.03	0.08
jazz	32.79	1817.51	tl	tl	66.46	tl	82.90	tl	0.66	269.73	tl	tl
jean	0.06	0.20	0.36	4.56	2.64	2.89	3.62	7.15	0.00	0.00	0.00	0.01
karate	0.03	0.08	0.06	0.03	0.09	0.18	0.11	0.08	0.00	0.00	0.00	0.01
lesmis	0.14	0.57	0.45	6.47	2.41	3.72	2.14	8.53	0.00	0.00	0.00	0.02
miles1000	1307.71				tl				tl			
miles1500	0.12				0.06				0.01			
miles250	·	·	tl	tl	·	·	35.09	36.08	·	·	1.69	9.65
miles500	116.00	tl	tl		31.14	tl	tl		11.88	tl	tl	
miles750	359.62	tl			tl	tl			tl	tl		
mug100_1	269.56	tl	tl	tl	22.89	27.89	28.30	29.29	1.22	8.34	17.58	23.91
mug100_25	397.28	tl	tl	tl	22.56	23.90	29.64	35.81	1.17	8.65	5.71	17.77
mug88_1	62.20	tl	tl	tl	22.23	27.61	33.55	36.35	0.87	2.37	5.83	11.45
mug88_25	16.36	tl	tl	tl	18.83	27.29	28.67	30.66	0.14	2.15	5.28	0.96
mulsol.i.2	·	·	·	382.63	·	·	·	tl	·	·	·	0.14
mulsol.i.3	·	·	230.54	594.89	·	·	tl	tl	·	·	0.13	0.33
mulsol.i.4	·	·	197.86	186.83	·	·	tl	tl	·	·	37.05	0.34
mulsol.i.5	·	·	176.49	559.76	·	·	tl	tl	·	·	0.16	0.44
myciel4	0.29	0.21			1.58	0.05			0.10	0.01		
myciel5	3.80	732.31	222.53	625.69	18.71	26.63	90.83	22.24	0.60	0.86	0.41	0.24
myciel6	43.87	tl	tl	tl	tl	tl	tl	tl	6.26	18.63	16.34	19.99
myciel7	940.63	tl	tl	tl	tl	tl	tl	tl	580.86	1733.41	1416.07	865.15
polbooks	78.21	tl	tl	tl	67.08	388.26	50.71	38.09	411.21	394.90	237.70	2259.55
queen10_10	tl	1.06			tl	5.76			tl	0.16		
queen11_11	tl	tl			tl	tl			tl	tl		
queen12_12	tl	tl			tl	tl			tl	tl		
queen13_13	tl	tl			tl	tl			tl	tl		
queen14_14	tl	tl			tl	tl			tl	tl		
queen5_5	0.01				0.09				0.02			
queen6_6	1.38				4.40				2.29			
queen7_7	25.63				1338.27				tl			
queen8_12	tl				tl				tl			
queen8_8	411.59				tl				tl			
queen9_9	3078.70				tl				tl			
r125.1	·	·	203.76	tl	·	·	30.70	35.53	·	·	0.00	0.17
r125.1c	0.34				0.79				0.13			
r125.5	364.18				tl				tl			

TABLE 2.7: Computational times (Coloring and DIMACS)

	COMP				*BP*				HYB_s			
	$k=5$	$k=10$	$k=15$	$k=20$	$k=5$	$k=10$	$k=15$	$k=20$	$k=5$	$k=10$	$k=15$	$k=20$
arc130	0.10				tl				40.89			
ash219	57.19	tl	tl	tl	26.01	43.36	435.94	263.85	132.71	tl	tl	tl
ash331	190.01	tl	tl	tl	24.67	534.41	tl	tl	1923.06	tl	tl	tl
ash85	219.87	tl			tl	tl			tl	tl		
bcspwr01	1.04	9.50			1.89	11.62			0.37	3.57		
bcspwr02	4.55	88.12	0.58		9.39	25.13	2.77		1.54	13.42	0.11	
bcspwr03	175.02	tl	tl	tl	230.01	44.96	474.05	230.46	tl	tl	tl	tl
bfw62a	2.33				180.84				tl			
can__144	tl	tl			tl	tl			tl	tl		
can61	0.97				4.68				2.11			
can62	11.61	tl	tl		17.87	25.25	24.89		13.49	2099.11	2179.20	
can73	160.08	tl			tl	tl			tl	tl		
can96	tl	tl			tl	tl			tl	tl		
curtis54	9.67				39.62				1812.30			
dwt___59	17.02	966.14	0.54		20.11	23.88	0.67		20.88	tl	0.02	
dwt66	566.29	tl			tl	tl			tl	tl		
dwt72	59.40	tl	tl	tl	23.85	28.15	163.40	436.94	6.41	tl	tl	tl
dwt87	44.31	tl	485.65		32.90	1994.16	28.33		20.34	tl	16.66	
gre__115	998.31	tl	tl	tl	2158.27	3139.10	tl	tl	tl	tl	tl	tl
ibm32	0.67				1.85				8.07			
impcol_b	0.35	4.17	9.60	0.08	0.91	5.17	13.91	0.07	0.17	2.07	29.60	0.01
L100.cavity01	0.86	5.22	5.48	5.78	tl	tl	tl	tl	0.40	170.85	3.49	7.77
L100.fidapm02	0.60				15.62				0.23			
L100.rbs480a	0.03				22.14				0.14			
L100.steam2	290.48				tl				tl			
L100.wm1	3.74	6.80	0.51		tl	tl	0.58		21.04	3.71	0.99	
L100.wm2	0.10	7.49			0.41	tl			0.14	225.32		
L100.wm3	0.11	2.88	0.44		0.39	tl	0.35		0.14	11.09	3.17	
L120.cavity01	1.03	9.89	6.56	7.43	tl	tl	tl	tl	0.68	501.24	10.24	7.18
L120.fidap022	0.05				0.19				7.80			
L120.fidapm02	0.31				tl				64.38			
L120.rbs480a	19.99				335.74				tl			
L120.wm2	0.16	0.50	0.69	12.04	1.07	1.88	tl	tl	0.06	0.95	3.16	4.71
L125.ash608	843.17	tl	tl	tl	26.46	tl	tl	tl	tl	tl	tl	tl
L125.bcsstk05	159.35				tl				tl			
L125.can__161	tl	tl	tl		tl	tl	tl		tl	tl	tl	
L125.can__187	tl	tl	tl	tl	tl	tl	tl	38.72	tl	tl	tl	34.61
L125.dwt__162	tl	tl	tl		tl	tl	tl		tl	tl	tl	
L125.dwt__193	1271.68				tl				tl			
L125.fs_183_1	2.23				tl				2195.21			
L125.gre__185	3244.87	tl	tl		tl	tl	tl		tl	tl	tl	
L125.lop163	tl	tl	tl		tl	tl	tl		tl	tl	tl	
L125.west0167	38.97	tl	tl	tl	23.80	30.37	34.33	39.01	2.74	157.35	431.59	tl
L125.will199	16.82	tl	tl	tl	21.25	35.26	47.00	41.12	4.60	tl	tl	tl
L80.cavity01	0.53	0.22	4.81	2.75	tl	tl	tl	1538.51	0.39	0.09	1.50	1.94
L80.steam2	29.73				tl				tl			
L80.wm1	0.82	3.27	0.26		tl	tl	0.15		5.71	6.38	0.01	
L80.wm2	0.05	0.66			0.27	14.19			0.06	2.86		
L80.wm3	0.04	0.16			0.23	37.53			0.08	0.75		
lund_a	tl	tl			tl	tl			tl	tl		
pores_1	0.24				0.65				0.18			
rw136	157.57	tl	tl	tl	37.21	tl	tl	tl	75.83	tl	tl	tl
steam3	186.29				tl				tl			
west0067	144.60	tl			tl	688.55			tl	tl		
west0132	24.78	tl	tl	tl	40.00	34.71	39.58	169.35	12.83	385.95	tl	tl
will57	1.12	0.05			2.78	0.17			0.82	0.11		

TABLE 2.8: Computational times (Intersection graphs)

	COMP				*BP*				HYB_s			
	$k=5$	$k=10$	$k=15$	$k=20$	$k=5$	$k=10$	$k=15$	$k=20$	$k=5$	$k=10$	$k=15$	$k=20$
1-FullIns_3	0.41	1.28			1.33	2.95			0.18	0.16		
1-FullIns_4	11.21	438.59	tl	tl	tl	tl	tl	tl	5.62	15.44	10.81	8.58
1-Insertions_4	4.41	190.63	tl	tl	tl	913.62	2283.63	796.23	23.58	10.79	12.99	7.31
2-FullIns_3	1.64	147.15	52.52	72.91	343.85	72.01	26.37	21.16	1.50	1.29	1.32	0.58
2-Insertions_3	0.41	22.00	16.84		5.24	24.21	20.21		0.06	0.17	0.11	
2-Insertions_4	331.64	tl	tl	tl	tl	tl	tl	tl	tl	2679.20	2090.26	566.87
3-FullIns_3	19.10	263.12	432.30	2564.03	tl	105.84	tl	281.41	31.64	7.53	2.70	5.67
3-Insertions_3	1.86	241.05	tl	tl	16.18	33.77	31.82	23.50	0.32	0.45	0.44	0.34
4-FullIns_3	69.45	tl	tl	tl	tl	tl	tl	tl	50.74	1064.56	41.78	99.47
4-Insertions_3	4.20	842.10	tl	tl	24.89	70.35	67.74	30.89	0.26	0.92	2.16	2.05
5-FullIns_3	29.44	tl	tl	tl	tl	tl	tl	tl	108.07	420.77	166.03	126.83
adjnoun	5.22	77.60	340.18	tl	tl	570.49	145.39	389.75	0.05	0.74	1.79	15.56
anna	0.43	0.77	4.84	14.50	88.39	89.78	67.74	70.59	0.01	0.01	0.01	0.01
celegansneural	9.87	22.50	1559.04	tl	tl	tl	tl	tl	0.05	0.02	0.40	599.76
chesapeake	0.31	2.88	3.52		0.42	2.77	8.24		0.05	0.31	0.24	
david	·	·	5.46	2931.76	·	·	25.65	26.84	·	·	0.04	2.53
dolphins	0.31	6.18	247.26	1042.79	5.28	36.77	48.72	26.80	0.01	0.30	1.87	1.91
DSJC125.1	2477.17	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl
DSJC125.5	tl	0.25			tl	0.02			tl	0.00		
football	1062.01	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl
games120	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl
huck	0.41	7.17	6.34	25.56	9.17	19.09	23.11	14.61	0.01	0.02	0.03	0.11
jazz	67.51	2980.98	tl	tl	32.00	tl	73.38	tl	30.33	334.29	tl	tl
jean	0.11	0.18	14.85	5.50	7.96	7.11	12.29	5.88	0.01	0.00	0.01	0.01
karate	0.12	0.17	0.08	0.05	0.16	0.26	0.18	0.09	0.01	0.01	0.01	0.01
lesmis	0.15	0.40	3.75	3.93	1.55	6.90	10.46	5.78	0.00	0.01	0.01	0.02
miles1000	1197.96				tl				tl			
miles1500	0.14				0.05				0.01			
miles250	·	·	tl	tl	·	·	227.79	240.53	·	·	0.05	1.55
miles500	204.84	tl	tl		28.15	tl	tl		313.19	tl	tl	
miles750	458.78	tl			tl	tl			tl	tl		
mug100_1	15.42	tl	tl	tl	10.84	109.05	30.32	29.65	0.06	0.18	0.13	0.38
mug100_25	18.26	tl	tl	tl	24.71	31.93	51.74	34.30	0.07	0.18	0.20	0.75
mug88_1	94.87	tl	tl	tl	24.21	55.01	72.20	34.92	0.44	0.81	0.96	61.92
mug88_25	17.73	tl	tl	tl	23.95	43.42	69.66	31.22	0.18	0.36	0.81	7.50
mulsol.i.2	·	·	·	393.14	·	·	·	tl	·	·	·	0.20
mulsol.i.3	·	·	305.87	758.72	·	·	tl	tl	·	·	0.17	0.66
mulsol.i.4	·	·	153.20	325.31	·	·	tl	tl	·	·	0.11	0.65
mulsol.i.5	·	·	299.17	338.89	·	·	tl	tl	·	·	0.28	0.59
myciel4	0.27	0.13			6.79	0.12			0.14	0.00		
myciel5	1.76	40.91	66.09	45.47	153.19	66.42	53.73	23.64	1.80	2.18	0.83	0.21
myciel6	49.44	tl	tl	tl	tl	tl	tl	tl	31.89	63.06	47.48	51.60
myciel7	477.80	tl	tl	tl	tl	tl	tl	tl	tl	tl	tl	1556.57
polbooks	23.36	tl	tl	tl	35.81	445.88	130.27	40.37	20.19	256.03	80.52	103.30
queen10_10	tl	0.89			tl	84.83			tl	0.49		
queen11_11	tl	tl			tl	tl			tl	tl		
queen12_12	tl	tl			tl	tl			tl	tl		
queen13_13	tl	tl			tl	tl			tl	tl		
queen14_14	tl	tl			tl	tl			tl	tl		
queen5_5	0.09				0.09				0.01			
queen6_6	1.69				6.51				1.87			
queen7_7	30.27				761.89				tl			
queen8_12	tl				tl				tl			
queen8_8	181.76				tl				tl			
queen9_9	1051.71				tl				tl			
r125.1	·	·	tl	tl	·	·	37.11	55.21	·	·	0.01	0.01
r125.1c	0.46				0.95				0.12			
r125.5	574.91				tl				tl			

TABLE 2.9: Computational times for instances with weights (Coloring and DIMACS)

	COMP				*BP*				HYB_s			
	$k=5$	$k=10$	$k=15$	$k=20$	$k=5$	$k=10$	$k=15$	$k=20$	$k=5$	$k=10$	$k=15$	$k=20$
arc130	0.13				tl				65.34			
ash219	112.44	tl	tl	tl	185.48	185.63	604.08	1205.06	251.06	2673.94	tl	tl
ash331	320.54	tl	tl	tl	25.31	tl	tl	tl	664.89	tl	tl	tl
ash85	762.38	tl			tl	tl			tl	tl		
bcspwr01	0.68	5.98			7.36	16.43			0.33	1.68		
bcspwr02	2.07	87.13	2.47		33.68	36.41	3.96		1.42	20.78	0.07	
bcspwr03	407.38	tl	tl	tl	85.71	666.73	671.30	1796.12	tl	tl	tl	tl
bfw62a	2.20				437.56				3293.22			
can__144	tl	tl			tl	tl			tl	tl		
can61	0.98				7.86				8.54			
can62	9.83	tl	3279.77		70.55	56.08	51.94		2.14	156.57	272.13	
can73	216.41	tl			tl	tl			tl	tl		
can96	tl	tl			tl	tl			tl	tl		
curtis54	7.69				18.52				128.46			
dwt___59	18.52	2904.33	0.80		67.36	58.75	2.32		195.93	tl	0.03	
dwt66	30.47	tl			159.22	tl			1032.61	tl		
dwt72	40.69	tl	tl	tl	62.20	39.54	40.08	79.69	1.32	77.44	1982.92	tl
dwt87	49.22	tl	2778.26		1165.64	tl	223.66		2193.55	tl	147.73	
gre__115	115.53	tl	tl	tl	23.57	215.78	tl	tl	240.00	tl	tl	tl
ibm32	0.68				1.81				3.46			
impcol_b	0.25	3.08	40.15	0.07	3.45	4.82	7.22	0.08	0.11	1.18	7.47	0.01
L100.cavity01	0.95	6.56	2.05	5.70	tl	tl	tl	tl	0.55	51.38	1.71	5.61
L100.fidapm02	0.04				15.56				0.13			
L100.rbs480a	0.03				8.15				0.53			
L100.steam2	939.05				tl				tl			
L100.wm1	3.76	7.95	1.10		tl	tl	0.60		23.76	6.87	1.32	
L100.wm2	0.13	8.98			0.51	tl			0.21	353.92		
L100.wm3	0.13	4.97	0.42		0.41	2710.89	0.35		0.23	9.81	0.09	
L120.cavity01	0.65	8.16	6.66	6.02	tl	tl	tl	tl	0.41	191.10	9.72	10.02
L120.fidap022	0.05				0.28				8.27			
L120.fidapm02	0.36				tl				10.47			
L120.rbs480a	32.84				811.22				tl			
L120.wm2	0.13	0.57	1.31	580.75	0.89	1.97	tl	tl	0.04	0.30	7.51	4.92
L125.ash608	577.32	tl	tl	tl	102.70	tl	tl	tl	609.68	tl	tl	tl
L125.bcsstk05	392.61				tl				tl			
L125.can__161	tl	tl	tl		tl	tl	tl		tl	tl	tl	
L125.can__187	tl	tl	tl	tl	tl	tl	tl	23.10	tl	tl	tl	0.13
L125.dwt__162	tl	tl	tl		tl	tl	tl		tl	tl	tl	
L125.dwt__193	516.22				tl				tl			
L125.fs_183_1	2.40				139.12				312.08			
L125.gre__185	tl	tl	tl		tl	tl	tl		tl	tl	tl	
L125.lop163	tl	tl	tl		tl	tl	tl		tl	tl	tl	
L125.west0167	21.70	tl	tl	tl	23.82	42.86	45.71	39.75	3.89	10.32	15.21	244.72
L125.will199	6.46	tl	tl	tl	15.36	76.29	86.93	38.87	9.34	224.00	2214.50	tl
L80.cavity01	0.40	0.33	2.75	3.50	tl	7.43	tl	3127.89	0.36	0.22	1.73	2.77
L80.steam2	32.78				tl				tl			
L80.wm1	2.07	3.64	0.48		tl	tl	0.14		9.79	21.64	0.02	
L80.wm2	0.06	1.56			0.28	7.60			0.25	3.75		
L80.wm3	0.06	0.55			0.25	5.74			0.22	0.95		
lund_a	tl	tl			tl	tl			tl	tl		
pores_1	0.30				0.55				0.24			
rw136	230.13	tl	tl	tl	324.92	tl	tl	tl	2459.85	tl	tl	tl
steam3	78.03				249.55				tl			
west0067	107.00	tl			tl	49.39			tl	tl		
west0132	7.29	tl	tl	tl	18.66	62.73	56.18	39.44	3.19	44.36	tl	tl
will57	0.79	0.02			10.09	0.17			1.23	0.24		

TABLE 2.10: Computational times for instances with weights (Intersection graphs)

	Optimal Values					Optimal Values			
	$k = 5$	$k = 10$	$k = 15$	$k = 20$		$k = 5$	$k = 10$	$k = 15$	$k = 20$
1-FullIns_3	35	53			miles500	42	-	-	
1-FullIns_4	35	66	90	122	miles750	120	-		
1-Insertions_4	40	68	100	125	mug100_1	10	27	46	69
2-FullIns_3	42	71	92	125	mug100_25	11	30	52	77
2-Insertions_3	18	50	73		mug88_1	20	43	68	99
2-Insertions_4	42	69	99	124	mug88_25	14	38	63	93
3-FullIns_3	33	53	76	106	mulsol.i.2	·	·	·	96
3-Insertions_3	22	47	72	95	mulsol.i.3	·	·	96	98
4-FullIns_3	40	81	98	127	mulsol.i.4	·	·	96	98
4-Insertions_3	17	43	68	94	mulsol.i.5	·	·	96	98
5-FullIns_3	35	72	95	113	myciel4	38	68		
adjnoun	11	29	51	81	myciel5	47	77	105	129
anna	7	7	9	15	myciel6	57	87	115	138
celegansneural	5	5	15	37	myciel7	67	-	-	148
chesapeake	28	60	92		polbooks	34	79	103	136
david	·	·	17	50	queen10_10	-	486		
dolphins	10	30	66	89	queen11_11	-	-		
DSJC125.1	106	-	-	-	queen12_12	-	-		
DSJC125.5	-	645			queen13_13	-	-		
football	101	-	-	-	queen14_14	-	-		
games120	-	-	-	-	queen5_5	103			
huck	7	17	33	54	queen6_6	149			
jazz	23	70	133	-	queen7_7	199			
jean	2	4	14	19	queen8_12	339			
karate	11	23	34	61	queen8_8	239			
lesmis	4	6	13	21	queen9_9	296			
miles1000	297				r125.1	·	·	2	9
miles1500	626				r125.1c	648			
miles250	·	·	7	30	r125.5	505			

TABLE 2.11: Optimal values of the instances with weights, instances that are infeasible and/or trivial for all values of k are not reported (Coloring and DIMACS).

	Optimal Values					Optimal Values			
	$k = 5$	$k = 10$	$k = 15$	$k = 20$		$k = 5$	$k = 10$	$k = 15$	$k = 20$
arc130	442				L120.cavity01	49	100	115	168
ash219	36	78	120	164	L120.fidap022	486			
ash331	39	-	-	-	L120.fidapm02	509			
ash85	117	-			L120.rbs480a	433			
bcspwr01	28	70			L120.wm2	7	28	67	239
bcspwr02	38	87	133		L125.ash608	37	-	-	-
bcspwr03	53	113	168	235	L125.bcsstk05	218			
bfw62a	114				L125.can__161	-	-	-	
can__144	-	-			L125.can__187	-	-	-	541
can61	207				L125.dwt__162	-	-	-	
can62	31	78	130		L125.dwt__193	291			
can73	144	-			L125.fs_183_1	71			
can96	-	-			L125.gre__185	-	-	-	
curtis54	74				L125.lop163	-	-	-	
dwt___59	59	141	226		L125.west0167	19	46	73	109
dwt66	54	-			L125.will199	21	60	92	127
dwt72	26	64	105	169	L80.cavity01	43	49	92	154
dwt87	66	-	313		L80.steam2	257			
gre__115	44	108	-	-	L80.wm1	88	218	281	
ibm32	80				L80.wm2	24	264		
impcol_b	22	58	109	202	L80.wm3	23	74		
L100.cavity01	49	91	100	162	lund_a	-	-		
L100.fidapm02	443				pores_1	99			
L100.rbs480a	370				rw136	40	-	-	-
L100.steam2	303				steam3	145			
L100.wm1	78	169	274		west0067	97	188		
L100.wm2	20	237			west0132	21	57	97	132
L100.wm3	20	76	303		will57	33	113		

TABLE 2.12: Optimal values of the instances with weights, instances that are infeasible and/or trivial for all values of k are not reported (Intersection graphs).

Chapter 3

The bilevel combinatorial structure of the *k*-Vertex Separator problem

[1]

Given a simple undirected graph $G = (V, E)$, where V is the set of its vertices and E is the set of its edges, we are interested in studying the problem of disconnecting G by removing a subset of vertices. Formally:

Definition 0 (vertex separator) *A vertex subset $S \subset V$ is called a vertex separator if the removal of S disconnects the graph G.*

In [38], [39], the *Vertex Separator problem* (VSP) has been addressed, formally: Given an integer $u \in \mathbb{N}$, and a cost $c_v \in \mathbb{N}$ associated with each vertex $v \in V$, the VSP asks for a partition of V into three disjoint nonempty subsets V_1, V_2, S, where V_1 and V_2 are the *shores* of the *separator* S, such that $v \in V_1$ and $w \in V_2$ implies $(v, w) \notin E$, the size of each shore is bounded by u, and the function $\sum_{v \in S} c_v$ is minimized. The VSP is an $\mathcal{NP}$-hard problem (see [39]) and it has several applications for different connectivity problems (we refer the interested reader to [40]–[42], and to [39] for a survey of such applications), one of the most important ones is related to the efficient solution of linear systems [43], [44]. In this chapter, we study the following problem closely related to the VSP:

Definition 0 (capacitated k-Vertex Separator problem (*k*-VSP)) *Given a graph $G = (V, E)$ and two integer values $k, u \in \mathbb{N}, k \geq 2$, the capacitated k-Vertex Separator problem (k-VSP) asks for a partition of V into $k+1$ disjoint subsets $V = \{V_1, V_2, \ldots, V_k\} \cup S$, where V_i $(i = 1, \ldots, k)$ are the shores of the separator S, such that $v \in V_i$ and $w \in V_j$ with $i, j = \{1, \ldots, k\}, j > i$ implies $(v, w) \notin E$, the size of each shore is bounded by u, and the cardinality of S is minimized.*

Note that in our definition of the k-VSP we allow empty shores. With this assumption, the problem is equivalent to the *Matrix Decomposition problem* studied in [13]. Indeed, it can be viewed as the problem of assigning the *rows* of a matrix A to k disjoint *blocks*. The objective is to remove a minimum number of rows from A and to assign the remaining rows to the blocks so that: (i) each row is assigned to at most one block, (ii) each block contains at most u rows, and (iii) no two rows in different blocks have a common nonzero entry in a column. The problems are equivalent by defining a row of A for each vertex of G and by defining a column of A for each edge

[1] The results of this chapter appears in: F. Furini, I. Ljubić, E. Malaguti, and P. Paronuzzi, "Casting light on the hidden bilevel combinatorial structure of the k-Vertex Separator problem", *Technical Report OR-19-6*, http://or.dei.unibo.it/technical-reports, 2019. [37]

of G, with nonzero entries at the rows corresponding to the endpoints of the edge. Conversely, given A, we define a vertex in G for each row, and an edge for each pair of vertices if there is at least a column in A with nonzero entries in the corresponding rows. The problem is $\mathcal{NP}$-hard as discussed in [13].

There exist several other relevant applications related to the k-VSP. Detecting a critical infrastructure in communication (or social) networks is important for understanding the vulnerability of the networks and the location of points that need to be protected in case of viral attacks. The goal is to find a smallest subset of vertices to protect (i.e., "vaccinate" or "interdict") so that the damage caused by the spread of the virus through the network is limited. Thereby, the network defender does not know which vertex of the network will be attacked, but assumes that the virus will spread instantaneously from the infected vertex to its neighbors, as long as the neighbor is not protected. Hence, for diffusing information through a social network, or spreading a (computer) virus through a (communication) network, the network elements need to be connected. In this context, the vertices identified by k-VSP (optimal) solutions can be seen as the "most vital" or "critical" vertices of a graph, with respect to connectivity. Therefore, solving the k-VSP determines a smallest subset of vertices that need to be vaccinated, to ensure that the damage caused by a sudden outbreak of a virus, measured as the size of the largest connected component of the network from which vaccinated vertices are removed, is then kept below the given parameter u.

In Figure 3.1, we give an example graph of 10 vertices and 13 edges and we provide an optimal solution for the k-VSP with $k = u = 3$. The vertex separator is composed by the grey vertices, i.e., the set $S = \{v_8, v_9\}$. After the removal of S, the graph $G[V \setminus S]$ is disconnected. More precisely, the graph is partitioned into 3 pairwise disconnected subsets of vertices (the shores) and the separator, namely: $V_1 = \{v_1, v_2, v_7\}$ (the first shore), $V_2 = \{v_3, v_4\}$ (the second shore), $V_3 = \{v_5, v_6, v_{10}\}$ (the third shore) and $S = \{v_8, v_9\}$ (the separator).

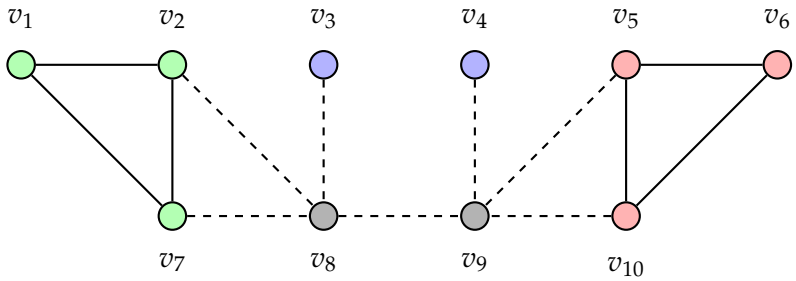


FIGURE 3.1: An example graph G for the k-VSP, with 10 vertices and 13 edges. The vertices of an optimal k-VSP solution with $k = 3$ and $u = 3$ are shown in grey, i.e., the separator $S = \{v_8, v_9\}$. Dashed lines represent the edges which are incident to the removed vertices and they do not appear in $G[V \setminus \{v_8, v_9\}]$.

Notation. Let K denote the set of integers $\{1, ..., k\}$. Given a simple undirected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, for an edge $wv \in E$, we say that w and v are *neighbours*. The complement of graph $G = (V, E)$ is a graph $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{wv : wv \notin E\}$. Let $N(w) = \{v \in V | wv \in E\}$ denote the *neighborhood* of w. A subset of vertices $W \subset V$ is a *clique of* G, if any two vertices of W are neighbours. A *clique cover* of G is a partition of V such that each element of the partition is a

clique. We indicate by $deg_G(v)$ the number of edges incident to v in graph G. Given a subset of edges $E' \subseteq E$ of G, we say that E' is *spanning* G if for every vertex v of G there is at least an edge in E' incident with v. Given a non-adjacent pair of distinct vertices $w, v \in V$, a set $F \subset V$ is called *v-w-separator* if and only if removing F from G disconnects w from v. Given $W \subset V$, a subgraph $G[W] = (W, E[W])$ induced by W contains all vertices of W and all edges $E[W] \subset E$ whose both ends belong to W.

Literature review. In this section, we provide a literature review of the exact algorithms present in the literature for the k-VSP and its closely related problems. To the best of our knowledge, the first exact algorithm for the k-VSP, addressed as *matrix decomposition problem*, has been proposed in [13]. An *integer programming* (IP) formulation is proposed and a Branch-and-Cut algorithm, based on polyhedral investigations, has been designed. The main motivation of the study was to verify whether the constraint matrix of a linear or integer program can be decomposed into the so-called *bordered block diagonal form* (see also [45] for further details).

Recently, an alternative exact algorithm for the k-VSP has been proposed in [27]. In this paper, the k-VSP has been called the *Capacitated Hypergraph Vertex Separator problem* and a branch-and-price algorithm has been designed based on specialized algorithms to solve the pricing problems. In addition, a branching scheme tailored for the problem is proposed and enhanced by a number of speed-up techniques. It is worth mentioning that, even though in [27] the problem has been defined on hypergraphs, an equivalent problem defined on simple graphs is obtained by replacing each hyperedge with a *clique*. We compare the computational performance of this branch-and-price algorithm with our newly developed Branch-and-Cut algorithm in Section 3.7.

In case $k = \infty$, our problem is closely related to the interdiction problem in which at most B vertices need to be removed from the graph, so that the size of the largest connected component in the remaining graph is minimized. This problem is strongly $\mathcal{NP}$-hard, but can be solved in polynomial time on trees and series-parallel graphs (see [46]). In [47], the authors propose an extended MIP formulation and a family of valid inequalities to solve this problem. Besides the fact that in our setting $k < \infty$, another major difference is in the type of objective function: instead of dealing with a min-max objective function, we are minimizing the budget, while making sure the largest connected component in the remaining graph will contain no more than u vertices.

Another problem related to the k-VSP is the *k-Vertex Cut problem*, which is obtained by setting $u = \infty$ and forbidding the empty shores in our problem definition. Formally: A *vertex cut* is a set of vertices whose removal disconnects the graph into several connected components. If the number of connected components is *at least* k, this set is called a *k-vertex cut*. Given a graph G, a positive weight c_v for each vertex $v \in V$, and an integer $k \geq 2$, the *k-Vertex Cut problem* (k-VCP) is to find a k-vertex cut of minimum weight. The k-VCP has been object of research in the recent years and we address the interested reader to, e.g., [48] where an exact branch-and-price algorithm has been proposed. Recently, we proposed in [6] a Branch-and-Cut algorithm for the k-VCP, exploiting a bilevel point-of-view of the problem, which allowed us to derive a valid IP formulation in the natural space of the variables and to beat state-of-the-art results from [48].

Our Contribution. This chapter studies a canonical IP formulation for solving the k-VSP in which several new families of valid inequalities are derived by exploiting

a "bilevel" point of view. The problem is seen as a two-player Stackelberg game in which a leader interdicts the network by removing some of its vertices, and a follower determines the maximum connected component in the remaining graph (we refer the interested reader to e.g., [49]–[52], for other relevant problems related to Stackelberg games). In addition, the leader has to make sure, the connected components can be packed in at most k shores, each of the size at most u. We first provide a basic canonical formulation, and show how to use the value function reformulation of the follower's optimization problem to derive new sets of valid inequalities. The value function reformulation has been convexified in two different manners: the first one adds penalties for the violation of some constraints in the objective function, the second one is Benders reformulaton derived from an extended formulation. Theoretical analysis reveals that Benders cuts are dominated by the first family of cuts. We show that the new inequalities can be separated at integer points in polynomial time, and explain details of an efficient Branch-and-Cut implementation. Computational study which is performed on a large set of publicly available benchmark instances shows that our new exact method is competitive against the state-of-the-art Branch-and-Price procedure given in [27]. Moreover, we are able to improve the best known results for several difficult classes of instances and to provide optimal solution values for 61 previously unsolved instances from the literature.

The chapter is structured as follows. In Section 3.1, we present a compact Integer Programming formulation for the k-VSP. In Section 3.2, we develop our new formulation in the natural space of the variables obtained through a bilevel interpretation of the problem. In this section we present several families of valid inequalities whose separation procedures are presented in Section 3.6. In Section 3.7, we discuss extensive computational results comparing a newly developed Branch-and-Cut algorithm with the state-of-the-art algorithms for the k-VSP. Finally, in Section 3.8, we present the conclusions of our work and some future lines of research.

3.1. A Compact Integer Programming Formulation

A first IP formulation for the k-VSP was introduced in [13], where for each vertex $v \in V$ and each integer $i \in K$, a binary variable ξ_v^i is defined, such that

$$\xi_v^i = \begin{cases} 1 & \text{if vertex } v \text{ belongs to the shore } V_i \\ 0 & \text{otherwise} \end{cases} \qquad i \in K, v \in V.$$

The vertices that remain unassigned to any of the shores (i.e., for which $\xi_v^i = 0$, for all $i \in K$), are the ones defining the separator S. This is why instead of minimizing the cardinality $|S|$ of the separator, one can equivalently maximize the number of

vertices out of S (i.e., the vertices in $\cup_{i\in K} V_i$), thus obtaining the following IP formulation

$$\max \quad \sum_{i\in K}\sum_{v\in V} \xi_v^i \tag{3.1}$$

$$\sum_{i\in K} \xi_v^i \le 1 \qquad v \in V, \tag{3.2}$$

$$\xi_w^i + \sum_{j\in K\setminus\{i\}} \xi_v^j \le 1 \qquad i \ne j \in K,\ wv \in E, \tag{3.3}$$

$$\sum_{v\in V} \xi_v^i \le u \qquad i \in K, \tag{3.4}$$

$$\xi_v^i \in \{0,1\} \qquad i \in K,\ v \in V. \tag{3.5}$$

The objective function (3.1) maximizes the number of vertices assigned to the shores of the separator. Constraints (3.2) impose that each vertex is assigned to at most one shore and (3.3) imposes that the shores induce pairwise disconnected subgraphs. Constraints (3.4) impose that the capacity of each shore is not exceeded, i.e., the number of vertices assigned to each shore is not larger than the capacity u. This formulation is known to suffer from symmetries, given that any permutation of indices $\{1,\dots,K\}$ results in the same feasible (LP-)solution.

In [27], this model is reformulated by defining a clique cover $\mathcal{Q}$ of G and introducing a binary variable ψ_Q^i for each integer $i \in K$ and each clique $Q \in \mathcal{Q}$ such that

$$\psi_Q^i = \begin{cases} 1 & \text{if some vertex } v \in Q \text{ belongs to the shore } i \\ 0 & \text{otherwise} \end{cases} \qquad i \in K, Q \in \mathcal{Q}.$$

Constraints (3.2) and (2.3) are replaced by

$$\sum_{i\in K} \psi_Q^i \le 1 \qquad Q \in \mathcal{Q}, \tag{3.6}$$

$$\xi_v^i - \psi_Q^i \le 0 \qquad i \in K,\ Q \in \mathcal{Q},\ \sqsubseteq \in \mathcal{Q}. \tag{3.7}$$

In [13], formulation (3.1)–(3.5) was strengthened by several valid inequalities and solved by a tailored Branch-and-Cut algorithm. Among the inequalities introduced in [13], the so called *block-invariant inequalities* are invariant under a permutation of the indices of the shores $i \in K$ (called blocks in [13]). These inequalities can be expressed for aggregated variables defined as

$$z_v = \sum_{i\in K} \xi_v^i, \quad v \in V,$$

which define if a vertex v is assigned to a shore ($z_v = 1$) or it is removed from G ($z_v = 0$), i.e., it is in the separator S. In the next section, we present an IP formulation based on the complement of these variables, and recall some of the block-invariant inequalities that we exploit to strengthen our formulation.

3.2. A canonical IP formulation

We now study an IP model that exploits the complement of the aggregate variables z_v, $v \in V$, introduced in the previous section. Our goal is to provide a "thin" formulation that lives in the natural space of decision variables, namely those constituting

the objective function. This will allow us to tackle more challenging (and potentially denser) instances, using only a linear number of decision variables. To this end, a binary variable x_v is defined such that

$$x_v = \begin{cases} 1 & \text{if vertex } v \text{ belongs to the separator } S \\ 0 & \text{otherwise} \end{cases} \qquad v \in V.$$

Given an arbitrary set $W \subset V$, let

$$\sigma(W) = \begin{cases} \text{\# bins of size } u \text{ needed to pack conn. components of } G[W], & \text{if possible} \\ \infty, & \text{otherwise} \end{cases}$$

The k-VSP can now be modeled as follows:

$$\min \quad \sum_{v \in V} x_v \tag{3.8}$$

$$\sum_{v \in W} x_v \geq 1 \qquad W \subseteq V : \sigma(W) \geq k \tag{3.9}$$

$$x_v \in \{0,1\} \qquad v \in V, \tag{3.10}$$

where the objective function (3.8) minimizes the number of deleted vertices, i.e., the vertices in the separator S. Constraints (3.9), denoted as *Bin Packing constraints* in the following, guarantee that any vertex separator S (encoded by x) which does not allow to "pack" the connected components of $G[V \setminus S]$ into k shores of size u, has to be discarded. These constraints have been proposed in [13]. The authors show that their separation corresponds to solving an instance of the Bin Packing problem, in which items correspond to connected components C of $G[W]$ of weight $|C|$, and bins have capacity u. If more than k bins are used by an optimal solution, the connected components induced by W cannot be "packed" into the k shores of capacity u, so at least one vertex in W must belong to the separator. As a special case, if there exists a component C such that $|C| > u$ (which can be determined in polynomial time), the packing is infeasible, and hence, at least one vertex from W must belong to the separator. It is not difficult to see that in the latter case, Bin Packing constraints are dominated by

$$\sum_{v \in C} x_v \geq 1, \quad C \subseteq W : C \text{ connected and } |C| = u + 1. \tag{3.11}$$

Besides its sparsity, another major advantage of this model compared to the formulation from Section 3.1 is that we get rid of the symmetries (i.e., the degeneracy caused by index permutations). This comes at a cost of having an $\mathcal{NP}$-hard procedure to check feasibility of any integer point of the Branch-and-Cut tree.

To (partially) overcome this difficulty, in the remainder of this section we propose new valid inequalities in the space of x variables, that can be used to enhance this basic model, and whose separation can be performed in polynomial time. To derive these inequalities, we approach the problem from a bilevel perspective.

3.3. A bilevel interpretation of the problem

Bilevel optimization has recently attracted a lot of attention of the research community, not only because of its relevance for the real-world applications but also because of the recent advancements in the development of off-the-shelf MILP solvers.

The latter ones are the major driving force for the methods of computational optimization to be pushed to the next frontiers [53]–[57]. Our problem can be seen in the context of Defender-Attacker games, which are typically solved using the tools and methods of bilevel optimization (see, e.g., [33], [58], [59]). In this section we exploit some modeling ideas borrowed from the bilevel optimization to improve the modeling power and understanding of the k-VSP.

For $k = \infty$, the k-VSP can be viewed as a sequential two-player Stackelberg game in which there are two players: a *leader* (i.e., Defender) and a *follower* (i.e., Attacker). In the first step, the leader "interdicts" the follower by deleting (i.e., protecting, vaccinating) some vertices from the graph. In the following step, the follower determines the *maximum connected component* in the remaining graph. Hence, from the perspective of the leader, the problem is to find the smallest subset of vertices to delete from G, so that the size of the optimal follower solution (i.e., the number of vertices in the maximum connected component) is at most u. For $k < \infty$, we are interested in finding at most k shores, hence the leader solution must additionally satisfy the Bin Packing constraints (3.9).

Independently on the value of k, using the value function reformulation for the follower, we can impose the following condition:

$$\Phi(x) \leq u \tag{3.12}$$

where $\Phi(x)$ denotes the optimal solution value of the follower subproblem for a given vector x. In general, the value function $\Phi(x)$ does not need to be convex. Hence, one possible way to deal with the problem and to derive a single-level problem reformulation is to try to *convexify* the value function. In the following we discuss two possible ways to convexify this function and derive valid inequalities.

3.3.1 Convexification by penalization

Given a *binary* realization of the leader variables x^*, the value $\Phi(x^*)$ can be calculated in $O(|E|)$ time by simply removing the vertices v such that $x^*_v = 1$ and searching for a largest connected component in the resulting graph. Nevertheless, as our next goal is to use the value function reformulation to derive valid linear constraints in the x space, in the following we are providing a sparse IP formulation for finding $\Phi(x^*)$. In this follower's subproblem, for each vertex $v \in V$, a binary variable y_v is defined such that

$$y_v = \begin{cases} 1 & \text{if vertex } v \text{ belongs to a maximum connected component} \\ 0 & \text{otherwise} \end{cases} \qquad v \in V,$$

recalling that maximum connected component has to be determined in the interdicted graph. The follower IP formulation reads

$$\Phi(x^*) = \max \quad \sum_{v \in V} y_v \tag{3.13}$$

$$y_v \leq 1 - x^*_v \qquad v \in V \tag{3.14}$$

$$\sum_{v \in F} y_v \geq y_w + y_v - 1 \qquad F \in \mathcal{F}_{wv}, wv \notin E \tag{3.15}$$

$$y_v \in \{0, 1\} \qquad v \in V. \tag{3.16}$$

The objective function (3.13) maximizes the number of the selected vertices. Constraints (3.14) ensure that the interdicted vertices cannot be selected. Constraints

(3.15) impose that the optimal follower solutions correspond to connected components (in the interdicted graph). These constraints are defined with respect to the collection $\mathcal{F}_{wv}$ of all the (minimal) *w-v-separators* for each pair of vertices $wv \notin E$. More precisely, constraints (3.15) impose that if a pair of vertices w and v is selected, at least one vertex in each $F \in \mathcal{F}_{wv}$ must be selected too (see, e.g., [60] for further details).

We aim at finding a reformulation of the follower's subproblem whose feasible space does not depend on x^*, with an adapted objective function, so that for any choice of x^*, the two problems provide the same optimal solution. In our setting, we apply *convexification by penalization*, as it is done in e.g. [32], [33], [52]. The major goal is to remove interdiction constraints $y_v \leq 1 - x_v^*$ from the follower's subproblem, and to introduce penalty terms in the objective function instead, so that the existence of the optimal follower solution satisfying $y_v x_v^* = 0$ is guaranteed. The reformulation can be obtained as stated in the following observation.

Observation 6 *The follower subproblem can be restated as*

$$\Phi(x^*) = \max \left\{ \sum_{v \in V} y_v - \sum_{v \in V} M_v \, x_v^* \, y_v : \ (3.15), \ (3.16) \right\} \tag{3.17}$$

where M_v are sufficiently large values that guarantee that $y_v = 0$ whenever $x_v^* = 1$.

With the above observation and a proper choice of multipliers M_v, $v \in V$, the value function $\Phi(x)$ becomes a piece-wise convex function defined as

$$\Phi(x) = \max_{y^* \in \mathcal{C}} \sum_{v \in V} y_v^* - \sum_{v \in V} M_v \, y_v^* \, x_v,$$

where $\mathcal{C}$ denotes all extreme points of the polytope of the (unrestricted) follower subproblem. In our setting, $\mathcal{C}$ corresponds to the connected subgraphs of G.

Hence, constraint (3.12) can now be replaced by the following family of cuts:

$$\sum_{v \in C} (1 - M_v x_v) \leq u, \qquad C \in \mathcal{C} \tag{3.18}$$

The new constraints (3.18) have been obtained by replacing in (3.12) the expression of $\Phi(x)$ by the objective function of (3.17). They can be equivalently restated as

$$\sum_{v \in C} M_v x_v \geq |V(C)| - u \qquad C \in \mathcal{C}, \tag{3.19}$$

imposing that, for each connected subgraph C of G, the sum of the M_v coefficients associated with the interdicted vertices is greater or equal than the cardinality of C minus the capacity u.

A straightforward down-lifting of the coefficients gives

$$\sum_{v \in V} \min\{|V(C)| - u, M_v\} \, x_v \geq |V(C)| - u \qquad C \in \mathcal{C}. \tag{3.20}$$

In order to obtain a tight formulation, the values of M_v should be as small as possible.

Given a tree T with $|V(T)| > u$, let $\text{comp}_T(v)$ denote the cardinality of the vertex set of a largest connected component obtained after removing v. Let $C \in \mathcal{C}$ be a tree T, then:

$$M_v = |V(C)| - \text{comp}_T(v).$$

Indeed, the value of M_v in this case indicates the deterioration of the optimal objective value ($|V(C)|$) by interdicting vertex v. In the general case, i.e., when C is a generic connected subgraph from $\mathcal{C}$ with $|V(C)| > u$, the above mentioned constraints remain valid, and can be imposed for *any spanning tree* T of C. Hence, we have the following result:

Proposition 9 *Let C be a connected subgraph of G with $|V(C)| > u$ and let $\mathcal{T}(C)$ be the set of all spanning trees of C, then the following inequalities*

$$\sum_{v \in V(C)} (|V(C)| - \text{comp}_T(v)) x_v \geq |V(C)| - u \qquad T \in \mathcal{T}(C), \tag{3.21}$$

which can be downlifted as in (3.20), are valid for the k-VSP. These cuts will be referred to as Component Cuts in the remainder of this chapter.

3.3.2 Convexification by dualization

Based on the Proposition 2 from [47], we can calculate $\Phi(x)$ using the following extended LP formulation:

$$\begin{aligned}
\Phi(x) \quad = \min \quad & \lambda && && \text{(3.22a)}\\
& \lambda \geq \sum_{v \in V} \sigma_{vl} && l \in V && \text{(3.22b)}\\
& \sigma_{wl} - \sigma_{vl} \geq -x_w - x_v && vw \in A, l \in V && \text{(3.22c)}\\
& \sigma_{ll} \geq 1 && l \in V && \text{(3.22d)}\\
& \sigma_{vw} \geq 0 && v, w \in V, && \text{(3.22e)}
\end{aligned}$$

where for each edge $vw \in E$ we define two arcs $vw, wv \in A$. The variables σ_{vw} are defined for any two vertices $v, w \in V$, and there exists an optimal solution of this LP in which $\sigma_{vw} = 1$ if and only if v and w belong to the same connected component of the interdicted graph. Hence, by calculating $\sum_{v \in V} \sigma_{vl}$, we are counting the number of vertices which are in the same component as the vertex $l \in V$. Constraints (3.22b) impose that λ is bounded from below by the size of the largest component. Constraints (3.22c) guarantee that if two neighboring vertices v and w are not interdicted, and v is in the same component as l, then w must be in the same component as well.

Since this LP formulation results in a convex way of describing the value function, we obtain an *extended formulation* of the k-VSP as follows:

$$\min \left\{ \sum_{v \in V} x_v : \quad \sum_{v \in V} \sigma_{vl} \leq u, \;\; l \in V, \quad \text{(3.22c)} - \text{(3.22d)} \;\; \text{(3.9)}, \;\; \text{(3.10)} \right\}.$$

To the best of our knowledge, this extended formulation is new and has not been considered in the previous literature. As our major motivation is to study the IP models in the natural space of x variables, our next goal is to project out σ variables from this model. This can be done in a Benders fashion, resulting in the following family of *Benders feasibility cuts*:

$$\sum_{l \in V} \left(\tilde{\gamma}_l - \sum_{vw \in A} \tilde{\beta}^l_{vw} (x_v + x_w) \right) \leq u \tag{3.23}$$

where $(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$, associated to constraints (3.22b)-(3.22d), represents an optimal solution of the following LP:

$$\Phi(x) = \max \sum_{l \in V} \left(\gamma_l - \sum_{vw \in A} \beta^l_{vw} (x_v + x_w) \right) \tag{3.24a}$$

$$\sum_{wv \in \delta^-(v)} \beta^l_{wv} - \sum_{vw \in \delta^+(v)} \beta^l_{wv} \leq \begin{cases} \alpha_l, & v \neq l \\ \alpha_l - \gamma_l, & v = l \end{cases} \qquad v, l \in V \tag{3.24b}$$

$$\sum_{l \in V} \alpha_l = 1 \tag{3.24c}$$

$$(\alpha, \beta, \gamma) \geq 0 \tag{3.24d}$$

We notice that, in an interdicted graph with binary values of x, this dual represents a single-commodity flow formulation imposed for each "root" $l \in V$. For a connected component C, a vertex l is chosen as a root, and α_l units of flow are sent from l to every other vertex $v \in V(C)$. Thereby, the value γ_l contains the total amount of flow sent from l plus α_l (which is exactly the size of C, assuming $\alpha_l = 1$). The flow is sent along a spanning tree T of C, and each value β^l_{vw} counts the total amount of flow carried along the arc (v, w) of that tree. Since we are looking for a distribution of the values of α_l among the vertices of G, and we penalize each arc (v, w) whose end vertices are interdicted (cf. the second term in the objective function), an optimal solution is obtained by choosing a largest component in the interdicted graph, randomly picking one of its vertices l as a root and setting $\alpha_l = 1$. Hence, instead of detecting Benders cuts using a black-box LP formulation, based on the above arguments we can use a combinatorial procedure to detect the following family of valid inequalities:

Proposition 10 *Let C be a connected subgraph of G with $|V(C)| > u$ and let T be a spanning tree of C, and assume that one unit of flow is sent from a chosen root $l \in V(C)$ to all other $v \in V(C), v \neq l$ along the edges of T. Let a^l_v be the sum of flows sent into the vertex v and out of v. Then the following inequalities*

$$\sum_{v \in V(C)} a^l_v x_v \geq |V(C)| - u \qquad T \in \mathcal{T}(C), \quad l \in V(C) \tag{3.25}$$

which can be down-lifted as in (3.20), are valid for the k-VSP. These cuts will be referred to as Benders Cuts in the remainder of this chapter.

Proof. We first observe that inequalities (3.23) can be rewritten as

$$\sum_{l \in V} \sum_{vw \in A} \tilde{\beta}^l_{vw} (x_v + x_w) \geq \sum_{l \in V} \tilde{\gamma}_l - u. \tag{3.26}$$

Following the discussion from above, we then choose a root $l \in V(C)$, calculate the coefficients $\tilde{\beta}^l$ and set $a^l_v = \sum_{vw \in \delta^+(v) \cup \delta^-(v)} \tilde{\beta}^l_{vw}$. Recall that the value of $\tilde{\gamma}^l$ is $|V(C)|$ for the chosen l, and that all $\tilde{\beta}^{l'}$ and $\tilde{\gamma}^{l'}$ are zero for $l' \neq l$. □

3.3.3 Another bilevel point-of-view

Consider a connected subgraph C of G, with $|V(C)| > u$. If we denote by $x(C) \subset V(C)$ the vertices of C that are interdicted, then the minimum number q of components into which C has to fall apart, so that each resulting component contains no

more than u vertices is given as:

$$q = \left\lceil \frac{|V(C)| - |x(C)|}{u} \right\rceil.$$

Hence, from an alternative bilevel perspective, we could see this as a Stackleberg game: the leader interdicts some vertices, and, for each connected subgraph C, such that $|C| > u$, the follower calculates the number of connected components in the interdicted graph. If the number of components is smaller than q, the solution of the leader is infeasible. Let $\Psi_C(x)$ be the number of connected components of subgraph C in the interdicted graph. The latter condition can be imposed as the following constraint:

$$\Psi_C(x) \geq q \quad C \subseteq W : C \text{ connected and } |C| > u.$$

In chapter 2 we showed that the condition for a generic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to be partitioned into at least q nonempty components by interdicting vertices can be expressed by the following exponential family of inequalities, where $\mathcal{S}$ denotes the set of all cycle-free spanning subgraphs of $\mathcal{G}$ and $n = |\mathcal{V}|$

$$\sum_{uv \in \mathcal{E}(S)} (1 - x_u - x_v) \leq n - \sum_{v \in \mathcal{V}} x_v - q \qquad S \in \mathcal{S}. \tag{3.27}$$

Hence, we can apply the previous result to any component $C \in \mathcal{C}$, such that $|V(C)| > u$. We restrict ourselves to spanning trees $T \in \mathcal{T}(C)$:

$$\sum_{uv \in E(T)} (1 - x_u - x_v) \leq |V(T)| - \sum_{v \in V(T)} x_v - \left\lceil \frac{|V(T)| - \sum_{v \in V(T)} x_v}{u} \right\rceil \quad T \in \mathcal{T}(C). \tag{3.28}$$

After removing the rounding and using $|E(T)| = |V(T)| - 1$, we obtain the following result:

Proposition 11 *Let C be a connected subgraph of G with $|V(C)| > u$, and let $\mathcal{T}(C)$ be the set of all spanning trees of C, then the following inequalities*

$$\sum_{v \in V(T)} [u(\deg_T(v) - 1) + 1] \, x_v \geq |V(T)| - u \qquad T \in \mathcal{T}(C), \tag{3.29}$$

which can be down-lifted as in (3.20), are valid for the k-VSP. These cuts will be referred to as Degree Cuts in the remainder of this chapter.

We notice that according to Proposition 6, constraints (3.27) should be imposed for each acyclic spanning subgraph of C. In the context of the current chapter however it is correct to consider only spanning trees of C, because, if there is a non-connected acyclic spanning subgraph (i.e., a forest) violating the constraint, we would add the corresponding constraint for each tree of the forest as well. Furthermore, for (3.29), the right hand side is always positive, and all coefficients next to the vertices are non-negative so that the constraints can be lifted as in (3.20).

3.3.4 Comparison between Component, Degree and Benders cuts

By comparing Component Cuts (3.21) and Degree Cuts (3.29), we observe that the latter are obtained by selecting a tree T spanning C and by setting

$$M_v = u(\deg_T(v) - 1) + 1, \qquad v \in V(T).$$

Despite the fact that both families of cuts are associated with trees, where each vertex v in the selected tree appears with a coefficient M_v, the values of these coefficients differ in the two cases. An example given in Figure 3.2a) for $u = 2$ shows that, when these cuts are imposed for the same $C \in \mathcal{C}$, the two cuts do not dominate each other. For the given example, the Component cut and, respectively, the Degree cut are given as:

$$x_1 + 3x_2 + 4x_3 + 3x_4 + x_5 + x_6 + x_7 \geq 5$$
$$x_1 + 5x_2 + 3x_3 + 5x_4 + x_5 + x_6 + x_7 \geq 5$$

For this example, the Benders cuts (with the root any non-leaf vertex, e.g., $l = v_2$) are dominated by Component and Degree cuts, and read:

$$x_1 + 5x_2 + 5x_3 + 5x_4 + x_5 + x_6 + x_7 \geq 5$$

However, another example depicted in Figure 3.2b) for $u = 3$ shows that, when Benders cuts and Degree cuts are imposed for the same $C \in \mathcal{C}$, the two cuts do not dominate each other. The Benders cut (with the root $l = v_3$) and, respectively, the Degree cut are:

$$x_1 + 3x_2 + 5x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \geq 5$$
$$x_1 + 4x_2 + 5x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \geq 5$$

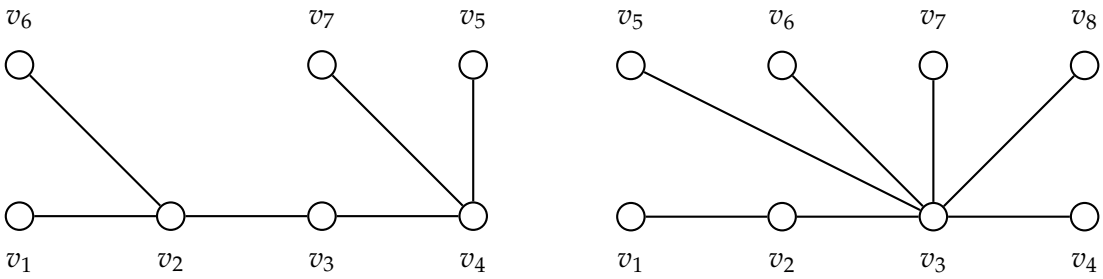


FIGURE 3.2: Two examples demonstrating relationships between studied inequalities. a) $u = 2$, b) $u = 3$.

We obtained this stronger result:

Proposition 12 For a given connected subgraph C and its spanning tree $T \in \mathcal{T}(C)$, Benders cuts (3.25) are dominated by the Component cuts (3.21).

Proof. We prove this result by showing that, for each vertex $v \in V(T)$, the coefficient a_v in the Benders cut is not smaller than the coefficient M_v in the Component cut. Let $|V(T)| = n$. When v is a leaf, it is trivial. If v is the root l, $a_v = n - 1$, whereas $M_v \leq n - 1$. If v is neither the root l nor a leaf, by removing v we partition T in $deg(v)$ components: a component containing l, and $deg(v) - 1$ components not containing l. Let these components include κ vertices in total. We have

$a_v = 2\kappa + 1$. If the largest component of T after removing v is the one including l, $M_v = n - (n - (\kappa + 1)) = \kappa + 1$, and hence, $a_v > M_v$. If the largest component of T after removing v does not include l, let $q \leq \kappa$ be its cardinality; we have $M_v = n - q$. Having $M_v > a_v$ would imply $n > 2\kappa + 1 + q$. But this would imply that the component including l has cardinality $n - \kappa - 1 > 2\kappa + 1 + q - \kappa - 1 = \kappa + q$ which is a contradiction since we assumed the largest component has cardinality q. □

3.4. Cover inequalities

These inequalities exploit the concept of connectivity. Let $W \subseteq V$ be a subset such that the induced subgraph $G[W] = (W, E(W))$ is r-vertex-connected (i.e., at least r vertices have to be removed to disconnect W). Then the following inequality, which can be derived from the corresponding block-invariant inequality of [13], is valid

$$\sum_{v \in W} x_v \geq \min(|W| - u, r) \qquad W \subseteq V,\ |W| > u. \tag{3.30}$$

3.5. Additional valid inequalities

Additional sets of valid inequalities, derived from the corresponding block-invariant inequalities of [13], can be exploited in order to strengthen the formulations of the previous sections.

- The *star* inequalities

$$\sum_{w \in N(v)} x_w \geq (\deg(v) + 1 - u)(1 - x_v) \qquad v \in V,\ \deg(v) \geq u \tag{3.31}$$

  impose that for each vertex $v \in V$ having a degree larger than or equal to the capacity, if vertex v is not interdicted than at least $\deg(v) + 1 - v$ of its adjacent vertices have to be interdicted. The number of these constraints is of the order of the cardinality of the vertex set. After some rewriting, it is not difficult to see that star inequalities are a special case of down-lifted version of component cuts (3.21), imposed for the star centered at v.

- In addition, we can impose precedence conditions between vertices, by observing that when the neighbourhood of a vertex w is included in the neighbourhood of a vertex v, then there is no reason to interdict w before v is interdicted. Indeed, any feasible solution where $x_w = 1$ and $x_v = 0$ can be transformed to a feasible solution of the same cost where $x_w = 0$ and $x_v = 1$. Precedence conditions are then stated as

$$x_w \leq x_v \qquad v, w \in V,\ N(w) \setminus \{v\} \subset N(v) \setminus \{w\}. \tag{3.32}$$

  Instead, when two vertices share the same neighbourhood, interdicting the first or the second does not make any difference. In this case we can get rid of potential symmetries with constraints (3.33) by imposing that the lowest index vertex can be interdicted only after the largest index vertex is interdicted. These additional precedence conditions are stated as

$$x_w \leq x_v \qquad v, w \in V,\ w < v,\ N(w) \setminus \{v\} = N(v) \setminus \{w\}. \tag{3.33}$$

3.6. Separation routines

This section describes separation strategies for the presented inequalities. All inequalities we propose (with the exception of cover inequalities) are given for a specific tree associated with a connected component. Trees are constructed during the detection of connected components, which can be performed by DFS or BFS. By construction, trees obtained through BFS define inequalities where one vertex (having large degree) receives a large coefficient, and the other vertices receive a small coefficient. After down-lifting, these constraints tend to be computationally more effective than the corresponding constraints where the initial trees are detected by DFS.

3.6.1 Separation of Degree Inequalities

Integer case. Given an integer solution x^*, if we denote by $V(x^*) \subset V$ the subset of interdicted vertices, the interdicted graph is the subgraph of G induced by $V \setminus V(x^*)$. The separation problem reduces to finding a connected subgraph C of the interdicted graph such that $|V(C)| > u$, yielding a violation $|V(C)| - u$ of constraints (3.29). The most violated inequality is obtained by choosing the (element-wise) maximal subgraph C. Our inequalities (3.29) are however not associated with connected subgraphs, but with subtrees contained in the subgraphs, whose number can be exponential in the size of each subgraph.

So, during the separation procedure, a tree for each connected subgraph of the interdicted graph that exceeds the capacity is built by means of a BFS, where the edges are processed in a random order. As soon as the size of the tree under construction is larger than u, and for each edge later on added to the tree, an inequality is defined and included in the formulation. So, at the end of the procedure, for each connected subgraph C (of the interdicted graph) that exceeds the capacity u, $|V(C)| - u$ inequalities of type (3.31) are added to the model.

Fractional case. Given a fractional solution x^*, after rewriting (3.29), we can see that checking whether a violated constraint exists is equivalent to finding a subtree T that maximizes the following function

$$\sum_{vw \in E(T)} (1 - x^*_v - x^*_w) - \sum_{v \in V(T)} (1 - x^*_v)(1 - \frac{1}{u}). \tag{3.34}$$

If the obtained value is positive, a violated cut (3.29) is added to the model. The above separation problem can be seen as the Prize-Collecting Steiner Tree problem (PCSTP)

$$\max \left\{ \sum_{v \in V} w_v \, y_v - \sum_{vw \in E} w_{vw} \, z_{vw} : \ (z, y) \text{ is a subtree of } G \right\} \tag{3.35}$$

with $w_{vw} = x^*_v + x^*_w, vw \in E$ and $w_v = \frac{1}{u} + (1 - \frac{1}{u})x^*_v, v \in V$. To solve the problem, one can use a specialized algorithm for the PCSTP, as the one described in [61], or a heuristic procedure.

Our heuristic procedure builds a tree starting from the edge with the largest weight, where the weight of an edge is defined according to the contribution of the edge itself and of its endpoints to (3.34). Iteratively, the procedure adds edge-vertex pairs to the current tree according to a random order, as long as edge-vertex pairs with a positive contribution can be found. The contribution for adding a vertex

v and an edge vw is again defined as in (3.34). When the procedure stops, if the tree has a positive weight then a violated inequality has been detected.

3.6.2 Separation of Component Inequalities

For integer solutions, any tree T that is contained in the interdicted graph and whose vertex set $V(T)$ exceeds the capacity, produces a violation $|V(T)| - u$ of constraints (3.21). In this case, however, computing the coefficients of each variable x_v, $v \in T$ requires to run a BFS procedure with vertex v as a root. So, adding an inequality for each *intermediate* tree (i.e., trees that do not span a maximal connected subgraph) can be time consuming. For this reason when separating inequalities (3.21) we only consider a spanning tree for each maximal subgraph C in the interdicted graph that exceeds the capacity u. In this case as well, the tree is built according to a random ordering of the vertices in $V(C)$. No separation is performed for fractional solutions, for which a well defined associated optimization problem is lacking.

3.6.3 Separation of Benders Cuts

Benders cuts are separated for integer solutions as discussed in Section 3.3.2: for each connected component C of the interdicted graph whose vertex set $V(C)$ exceeds the capacity, a vertex of maximum degree l is chosen as a root (this choice produces lower values of the coefficients in the Benders cut). A spanning tree rooted at l is constructed by BFS, where vertices are sorted according to a random ordering (again, choice produces lower values of the coefficients in the Benders cut), and the corresponding Benders cut is defined.

3.6.4 Separation of Cover Inequalities

Given a connected subgraph C of the interdicted graph with $|V(C)| > u$, inequalities (3.30) impose either to disconnect the set (by removing at least r vertices, when the component is r-connected) or to remove a number of vertices to reduce the cardinality of the vertex set to u. There are different options for computing the vertex-connectivity r of a connected subgraph, which give rise to different ways of separating constraints (3.30):

1. The vertex-connectivity r of a connected subgraph C can be computed in polynomial time by maximum-flow computations. Even though this procedure runs in polynomial time, it can be time consuming in dense graphs.

2. Borndörfer et al. (1998) [13] proposed a greedy procedure to detect biconnected components. This procedure can be used to detect whether a component is (at least) bi-connected and to derive inequalities (3.30) with $r = 2$.

3. Computing biconnected components ($r = 2$) in a connected undirected graph can be performed in linear time with the sequential algorithm proposed by Hopcroft and Tarjan (1973)[62], during the execution of the DFS. Hence, for each connected subgraph C of the interdicted graph with $|C| > u$, we have as a by-product of DFS a sub-component of cardinality $u + 1$ or $u + 2$, for which we write the corresponding inequality (3.30).

We implemented a separation procedure for integer solution for all three described methods, option 3 being the default one. For fractional solution, we only considered the greedy method described in 2.

3.6.5 Separation of Bin Packing Inequalities

Inequalities (3.9) are separated for integer solutions only. Given an integer solution x^* and the associated interdicted graph defined by the set of interdicted vertices $V(x^*)$, the Bin Packing problem instance associated with the connected components of the latter graph is defined and solved. If the optimal solution to the Bin Packing problem uses more than k bins, an inequality (3.9) is defined for $W = V \setminus V(x^*)$.

3.7. Computational Results

In this computational section, we present the results of the experiments with the aim of assessing the performance of the mathematical models described in the previous sections. We implemented a Branch-and-Cut framework based on formulation (3.8)–(3.11) which has a polynomial number of variables and an exponential number of constraints, namely, the *Bin Packing Inequalities* (3.9) and their feasibility counterpart (3.11). Even though correct, this formulation asks to solve a $\mathcal{NP}$-hard problem to check feasibility of any integer point of the Branch-and-Cut tree. Hence, this basic model is enhanced by four different families of constraints for which we have developed polynomial separation algorithms for integer points: (i) *Component Inequalities* (3.21), (ii) *Degree Inequalities* (3.29), (iii) *Benders cuts* (3.25) and (iv) *Cover Inequalities* (3.30).

The first goal of this computational section is to assess the relative computational performance of each family of inequalities, and their computational interaction when separated in a Branch-and-Cut fashion. Based on the results of these experiments, which are presented in Section 3.7.1, the best (and hence, default) configuration of our newly developed Branch-and-Cut algorithm is determined. The latter is then used in a second set of experiments (cf. Section 3.7.2), in which the performance of the Branch-and-Cut algorithm is compared with the state-of-the-art exact methods for the k-VSP present in the literature.

Benchmark instances. We tested the same four sets of benchmark instances considered in [27]. The first two sets of graphs are obtained from matrix decomposition problems. The considered matrices are the constraint matrices of several `Netlib` and `MIPLIB` instances. There are 55 graphs constituting the `Netlib` data set, with the number of vertices ranging from 51 to 500. The `MIPLIB` data set contains 37 graphs whose number of vertices ranges from 19 to 490 [2]. The other two sets are 40 instances from the second `DIMACS` challenge [35] and 50 `Random` graphs representing hypergraphs generated by the authors of [27]. For the `DIMACS` set, the number of vertices ranges from 23 to 496, whereas for the `Random` set, the number of vertices ranges from 68 to 164. Since [27] considered hypergraphs, we adapted the latter instances to our case by defining a clique for each hyperedge. In summary, our computational study is conducted on a set of 182 graphs with different structures and densities; the exact number of vertices and edges of these graphs are reported in the tables provided in the Appendix.

As far as the values of k (maximum number of shores) and u (maximum capacity of each shore) are concerned, we borrow the same setting from [27] in which $k \in \{4, 8, 12, 16, 24, 32, 64, 218, 256\}$ and $u = \lceil \frac{n}{k} \rceil$. The values of k are clustered into three major categories: (i) `small` $\rightarrow k \in \{4, 8, 12\}$, (ii) `medium` $\rightarrow k \in \{16, 24, 32\}$ and

[2] The constraint matrices determining these graphs have been presolved and reduced by SCIP 3.2.0 with default settings by the authors of [27].

(iii) `large` $\rightarrow k \in \{64, 128, 256\}$. In summary, using 9 different values of k and 182 graphs, we obtained a testbed of 1638 different instances. In the remainder of this chapter, aggregated results for `small`, `medium` and `large` values of k are reported (whereas the detailed results can be found in the Appendix).

Computational Environment. All the reported experiments are performed on a computer equipped with an i7 processor clocked at 3.20 GHz and 64 GB RAM under Linux operating system. We use the `CPLEX 12.7.1` MIP framework to implement our Branch-and-Cut algorithms. `CPLEX` is run in single-threaded mode and all `CPLEX` parameters are set to their default values. A time limit of 30 minutes is set for each tested instance.

3.7.1 Determining the best configuration of the Branch-and-Cut algorithm

As previously discussed, a basic valid formulation for the k-VSP is given by (3.8)-(3.11). Each one of the four families of inequalities: the Component Inequalities (3.21), the Degree Inequalities (3.29), the Benders cuts (3.25) and the Cover Inequalities (3.30) can be used to enhance this basic model. These families of cuts are composed by an exponential number of constraints and thus they are separated within the branching tree for integer solutions. In addition, they can be separated for fractional solutions in order to strengthen the dual bounds and (potentially) improve the computational convergence. In the following, we report results for the following Branch-and-Cut configurations:

- `C`: the *B*&*C* separating the Component Inequalities (3.21) for integer solutions;
- `D`: the *B*&*C* separating the Degree Inequalities (3.29) for integer solutions;
- `B`: the *B*&*C* separating the Benders Cuts (3.25) for integer solutions;
- `CV`: the *B*&*C* separating the Cover Inequalities (3.30) for integer solutions, via the detection of biconnected components. Among the three separation procedures given in Section 3.6.4, after extensive preliminary computational tests, we determined that the best performing way is via the application of the Hopcroft and Tarjan algorithm [62].

Concerning the separation of fractional points for (3.21), recall that this remains an open question (see Section 3.6.2). Regarding the constraints (3.29), we tested the separation of fractional points either in an exact or heuristic fashion. Although improvements were obtained for some specific instances, on average the computational performance was worsened – additional computational effort was needed to solve the LP relaxation at the branching nodes due to a large number of violated cuts detected. We therefore do not report the results for this particular setting. Similar considerations apply to the separation of (3.30) at fractional points, which was performed by means of the greedy procedure proposed in [13]. Also in this case, the average computational performance was worsened.

In all the configurations of the Branch-and-Cut algorithm, Bin Packing inequalities (3.9) are separated at integer points only when no other violated inequalities have been detected. This guarantees that no connected subgraph C with cardinality of the vertex set exceeding the capacity exists in the interdicted subgraph. The associated Bin Packing instances were not challenging and hence, a standard MIP formulation for the Bin Packing problem was used with `CPLEX` as the off-the-shelf

solver. Indeed, the performance of our configurations was not affected by the efficiency of the latter separation procedure which is why we refrained from developing a tailored algorithm for the Bin Packing problem.

In Section 3.5, we presented two additional families of valid inequalities which are in polynomial number: (i) the *Stars Inequalities* (3.31) and (ii) the *Precedence Inequalities* (3.32) and (3.33). Thanks to extensive preliminary experiments, we observed that these inequalities are useful to strengthen the formulation and to speed-up the computational convergence. For this reason, they are always included into our models.

In Table 3.1, we present the results of the computational experiments performed with the previously discussed configurations of the Branch-and-Cut algorithm. Precisely, we report the performance of 5 different configurations: `C`, `D`, `B` and `CV`, i.e., the four basic variants separating the Component, Degree, Benders and Cover Inequalities for integer solutions, respectively. In addition, we report the performance of `C+CV`, which corresponds to the separation of the Component Cuts and of the Cover Cuts for each integer solution. Indeed, while the first three families of cuts have the same structure (i.e., are all associated with trees of the graph G), the latter family is structurally different. Hence, we tried to combine Cover Inequalities with the other inequalities, and we report the results for the best configuration. Table 3.1 is horizontally divided in four sections, the first three reporting aggregated results for the three classes of values for the maximum number of shores k, i.e., `small` $\rightarrow k \in \{4, 8, 12\}$, `medium` $\rightarrow k \in \{16, 24, 32\}$, `large` $\rightarrow k \in \{64, 128, 256\}$. These three sections (546 instances each) report, for each configuration of the Branch-and-Cut algorithm, the total number of instances solved to proven optimality (rows "Opt"), the average computing time in seconds (rows "Avg Time") and the average number of nodes explored by the branching tree (rows "Avg nodes"). Finally the fourth section of the table reports the same information for the entire set of the 1638 instances. All the averages are computed separately for each configuration, by considering only the instances solved to proven optimality by that configuration.

As far as the comparison of the four basic variants (`C`, `D`, `B` and `CV`) is concerned, from the table it emerges that with 1165 instances solved to proven optimality, `C` is the best configuration followed by `B`, which is able to solve 1110 instances, `D` which is able to solve 1068 instances and by `CV`, which only solves 1063 instances. A similar pattern can also be seen for the three different categories of values for k. The number of instances solved to proven optimality and the computational times suggest that the class `small` is the hardest to solve for all our Branch-and-Cut algorithms.

Separating both the Component and the Cover Cuts pays off in terms of the number of instances solved to proven optimality, precisely `C+CV` is able to solve 1180 instances (15 more than `C` alone). The average number of Branch-and-Bound nodes suggests that `CV` explores on average fewer nodes than `C`, especially for small values of k. By combining the two families of cuts, on average the number of explored nodes is reduced, compared with `C` alone.

3.7.2 Comparison with state-of-the-art solution methods

In this section we compare the performances of our best Branch-and-Cut configuration identified in the previous section (i.e., the configuration `C+CV`) with the state-of-the-art exact methods available in the literature for the k-VSP:

- `BP`: the Branch-and-Price algorithm of [27], and

TABLE 3.1: Performance comparison for different configurations of our Branch-and-Cut algorithm.

k		C	D	B	CV	C+CV
small	Opt. (out of 546)	294	226	258	219	305
	Avg Time	66.52	89.38	74.72	78.92	85.12
	Avg Nodes	87355	12375	70370	29922	75221
medium	Opt. (out of 546)	388	360	369	362	392
	Avg Time	53.53	46.39	46.56	32.42	44.34
	Avg Nodes	82145	21012	64327	32320	54149
large	Opt. (out of 546)	483	482	483	482	483
	Avg Time	19.22	17.88	21.82	16.01	19.74
	Avg Nodes	41170	39678	42302	37852	39956
	Total Opt. (out of 1638)	1165	1068	1110	1063	1180
	Total Avg Time	42.58	42.62	42.34	34.56	44.81
	Total Avg Nodes	66472	27608	56148	34334	53786

- `Cplex`: the direct solution of the compact model (3.1), (3.4) -(3.7) via `CPLEX`, a state-of-the-art commercial MIP solver.

For these tests we used the same test-bed of 1638 instances proposed by [27] and described in the previous section. We recall that a time limit of 30 minutes is set for each run as for the experiments reported in [27]. The results of `BP` and `Cplex` are directly borrowed from the tables reported in [27] (the performance of our machine is comparable with the machine used for the experiments of [27], which is equipped with a i7 processor clocked at 3.40 GHz).

The information reported in Table 3.2 summarizes the results of this second set of tests. The table follows the same structure given in Table 3.1, but in addition to the disaggregation concerning the category of k, we also report disaggregated information for each class of instances. All the averages are computed separately for each method, by considering only the instances solved to proven optimality by that method. We discuss now the results for each class of instances separately.

For 360 `DIMACS` instances, `Cplex` is able to solve 200 instances, `BP` 223 instances and `C+CV` 245 instances. For 333 `MIPLIB` instances, `Cplex` is able to solve 107 instances, `BP` 135 instances and `C+CV` 168 instances. For 495 `Netlib` instances, `Cplex` is able to solve 294 instances, `BP` 410 instances and `C+CV` 413 instances. For the 450 `Random` instances, `Cplex` is able to solve 263 instances, `BP` 410 instances and `C+CV` 354 instances.

Summarizing, in terms of the number of instances solved, our Branch-and-Cut algorithm `C+CV` outperforms both `Cplex` and `BP` for the `DIMACS` and `MIPLIB` classes of instances, it has a performance comparable with that of `BP` for the `Netlib` instances, while it is outperformed by `BP` for the `Random` instances. For the category `small` of the k values, `Cplex` remains instead the best option. This is due to the small number of variables of the compact formulation which linearly depends on k. For `medium` and `large` values of k, `Cplex` is largely dominated by `BP` and `C+CV`, which improve their performance for increasing values of k, thus showing a complementary performance with respect to `Cplex`. In particular, `C+CV` is the best option for all classes of instances, when k is `large`.

Finally, performance profiles depicted in Figures 3.3 and 3.4 give a graphical representation of the relative performance of the three compared methods, i.e., `C+CV`, `BP` and `Cplex`. In Figure 3.3, the instances are gathered by class of instances, i.e.,

TABLE 3.2: Performance comparison between Cplex, BP and our best Branch-and-Cut algorithm.

Class	k		Cplex	BP	C+CV
DIMACS	small	Opt. (out of 120)	73	59	66
		Avg Time	194.31	164.37	45.57
	medium	Opt. (out of 120)	58	73	78
		Avg Time	192.77	109.77	28.68
	large	Opt. (out of 120)	69	91	101
		Avg Time	61.57	71.71	33.15
	Total Opt. (out of 360)		200	223	245
	Total Avg Time		148.07	108.68	35.07
MIPLIB	small	Opt. (out of 111)	52	23	45
		Avg Time	232.22	147.41	22.99
	medium	Opt. (out of 111)	25	46	48
		Avg Time	253.00	174.20	2.65
	large	Opt. (out of 111)	30	66	75
		Avg Time	19.35	122.35	61.24
	Total Opt. (out of 333)		107	135	168
	Total Avg Time		177.39	144.29	34.26
Netlib	small	Opt. (out of 165)	138	114	122
		Avg Time	124.51	182.35	36.34
	medium	Opt. (out of 165)	93	141	134
		Avg Time	322.76	66.15	30.52
	large	Opt. (out of 165)	63	155	157
		Avg Time	30.40	24.99	9.97
	Total Opt. (out of 495)		294	410	413
	Total Avg Time		167.06	82.90	24.43
Random	small	Opt. (out of 150)	110	110	72
		Avg Time	247.09	276.42	242.86
	medium	Opt. (out of 150)	65	150	132
		Avg Time	529.22	43.92	82.78
	large	Opt. (out of 150)	88	150	150
		Avg Time	0.35	0.42	0.19
	Total Opt. (out of 450)		263	410	354
	Total Avg Time		234.26	90.38	80.34

Netlib, MIPLIB, DIMACS and Random. In Figure 3.4, the instances are gathered by values of k, i.e., small, medium and large. As proposed in [36], let m be any solution method and i denote an instance of the problem. In addition let $t_{i,m}$ be the time required by method m to solve instance i. The *performance ratio* for a pair (i, m) is defined as

$$r_{i,m} = \frac{t_{i,m}}{\min_{m \in M}\{t_{i,m}\}}$$

where M is the set of the considered methods. For each method $m \in M$, we then define:

$$\rho_m(\tau) = \frac{|\{i \in I : r_{i,m} \leq \tau\}|}{|I|}$$

where I is the set of the instances. The value $r_{i,m}$ represent the worsening (with respect to computing time) incurred when solving instance i using method m instead of the best possible one, whereas $\rho_m(\tau)$ gives the percentage of instances for which the computing time of method m was not larger than τ times the time of the best performing method. For each value of τ in the horizontal axis, the vertical axis reports the percentage of the instances for which the corresponding algorithm spends no more than τ times the computing time of the fastest algorithm. All computing times smaller than 0.1 seconds were scaled to 0.1, which is the granularity of the profile. This way we avoid comparisons between tiny values, which would produce inaccurate conclusions. The curves originate from a point denoting the percentage of instances for which the corresponding algorithm is the fastest, and at the right end of the chart, they show the percentage of instances solved within time limit. The best performing algorithm is graphically represented by the curve in the upper part of the respective figure. The horizontal axis is represented in logarithmic scale.

From Figure 3.3, it emerges that C+CV is the fastest exact method for around 60% of the DIMACS instances, while this is the case for the BP and Cplex for only ≈10% of instances. Even by allowing larger computing times, C+CV outperforms BP and Cplex on this class of instances. For the MIPLIB instances, C+CV is the fastest exact method for 40% of the instances, while this is true for BP (resp., Cplex) for less than 30% (resp., approximately 10%) of the instances. By allowing larger computing times, C+CV outperforms BP and Cplex also on this class of instances. For the Netlib instances, C+CV is the fastest exact method for more than 70% of the instances, while this is true for BP (resp., Cplex) for less than 40% (resp., approximately 10%) of the instances. [3] Even by allowing larger computing times, the fraction of instances solved by C+CV is slightly larger than that of instances solved by BP. For the Random instances, BP is the fastest method for 60% of the instances, followed by C+CV (around 50%) and Cplex (less than 20%). BP is the best method for this class of instances. From Figure 3.3 it also emerges that the hardest set of instances are the MIPLIB ones, since only 50% of these instances can be solved to proven optimality by the best performing method. Instead, almost 70% of the DIMACS, more than 80% of the Netlib and more than 90% of the Random instances, respectively, can be solved by the best exact method.

Figure 3.4, where instances are gathered by values of k, confirms that the hardest instances are the ones of category small, for which more than 60% of the instances can be solved to proven optimality by the best considered exact method. C+CV is the fastest method for around 40% of the instances, while for BP and Cplex this is true for around 20% of the instances. However, by allowing larger computing times, Cplex

[3] These values may sum up to a value larger than 100%, if more than one algorithm is classified as the fastest for a specific instance (because of ties in the computing time).

is the best performing method for this class. The situation is slightly improved for the category `medium`, where more than 70% of the instances can be solved to proven optimality by `C+CV` and `BP`. `C+CV` is the fastest method for almost 60% of the instances, `BP` is the fastest for around 35% of instances and `Cplex` is completely outperformed. For large computing times, `C+CV` and `BP` have a similar performance. As far as the category `large` is concerned, approximately 90% of these instances can be solved by `C+CV` which is also the fastest method for almost 80% of these instances, while this is true for `BP` for around 50% and for `Cplex` for around 10% of the instances. `C+CV` is the best performing method for this class.

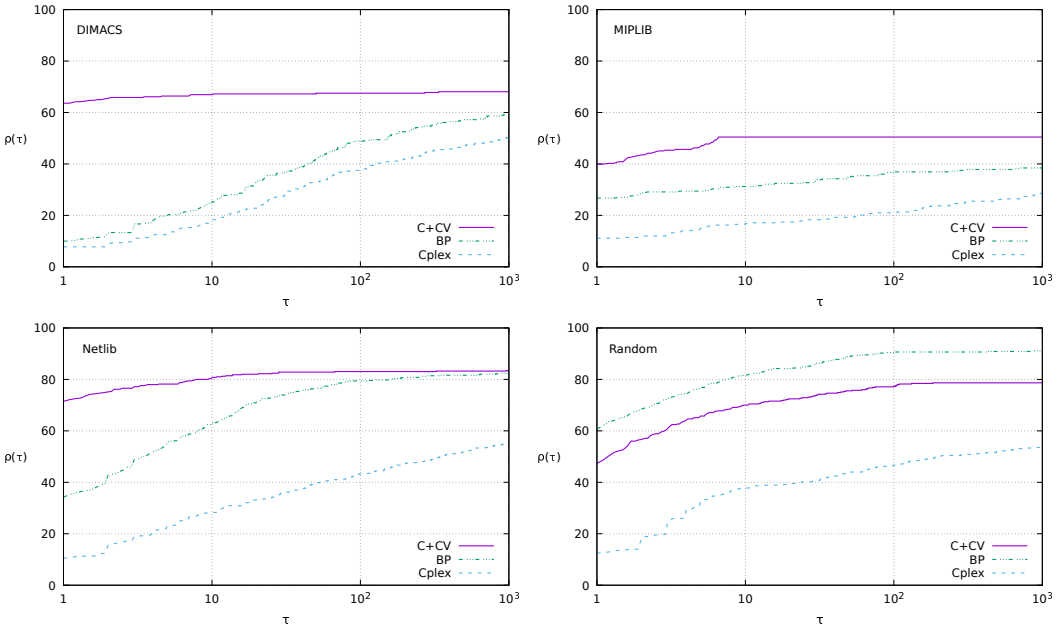


FIGURE 3.3: Performance profiles by class of instances: `DIMACS`, `MIPLIB`, `Netlib` and `Random`.

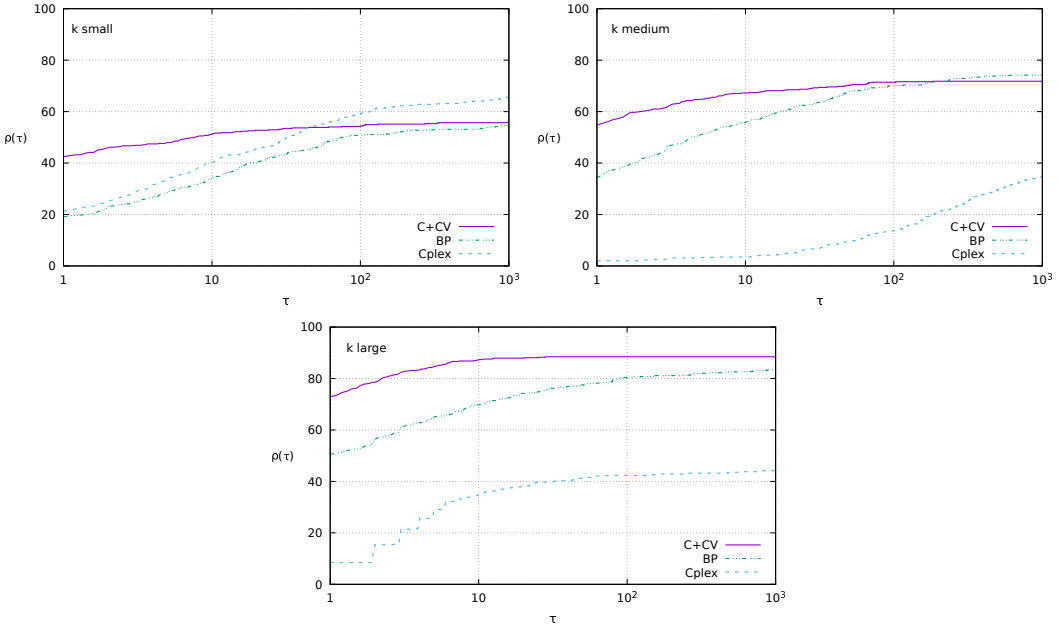


FIGURE 3.4: Performance profiles by values of k: `small` $\rightarrow$ $k \in \{4, 8, 12\}$, `medium` $\rightarrow$ $k \in \{16, 24, 32\}$, `large` $\rightarrow$ $k \in \{64, 128, 256\}$

.

3.8. Conclusions

In this chapter we studied the capacitated k-Vertex Separator problem in which a subset of vertices of minimum cardinality has to be removed from a given graph, so that the size of each connected component in the remaining graph is bounded by u, and all components can be packed into k shores. For this hard problem with applications in detecting critical nodes in communication networks, social networks analysis and matrix decomposition, extended formulations have been well studied in the previous literature, but very little has been known about solving the problem using a canonical IP formulation. The major drawback of the canonical IP formulation is that it requires solving an ($\mathcal{NP}$-hard) bin-packing problem in order to verify feasibility of a solution. To improve the computational efficiency of the underlying IP formulation, we proposed several families of valid inequalities which are derived from a perspective of a two-player sequential game in which a leader removes the vertices, and a follower solves another combinatorial optimization problem that (partially) guarantees feasibility of the solution. Three new families of valid inequalities are proposed, and their effects on the basic IP formulation are studied both from the theoretical and computational perspective.

On a large benchmark set of the instances available in the current literature, we demonstrate that our new Branch-and-Cut approach is competitive with the state-of-the-art Branch-and-Price algorithm and a compact formulation from [27]. In particular, our approach computationally outperforms to the Branch-and-Price algorithm from [27] for large values of the number of shores k and for structured graphs from various applications, while the latter has a better performance for random graphs and average values of the k parameter.

Our computational analysis has revealed that exact approaches for k-Vertex Separator problem can tackle graphs with up to 500 vertices. Solving the problem for graphs with thousands of vertices is a relevant open problem, for which heuristic and approximate methods should be considered as an interesting stream of research.

Finally, we hope that this work raises the awareness on the importance and merit of bilevel optimization for solving difficult combinatorial optimization problems by modeling them as two-player Stackelberg games. Many vertex/edge deletion/insertion problems or graph partitioning problems could benefit from this new modeling paradigm. The same is true for problems that ask for finding the most central or most critical vertices/edges with respect to various centrality measures.

Appendix

In Table 3.3 we report, for each instance from the DIMACS set and for each considered value of k, the optimal solution value of the k-Vertex Separator problem, in parenthesis, and the computational times of C+CV, for instances solved to proven optimality. When the time limit of 30 minutes is reached, we report the values of lower and upper bound at time limit. In the first two columns of the table, we report the number of vertices (n) and the number of edges (m) of the graph. The same information is reported in Tables 3.4, 3.4 and 3.5 for the MIPLIB, Netlib and Random sets of instances, respectively.

TABLE 3.3: Features, computational times and optimal solution values (if known) of C+CV for the DIMACS instances.

	n	m	k=4	k=8	k=12	k=16	k=24	k=32	k=64	k=128	k=256
anna	138	493	0.2 (13)	0.4 (15)	0.5 (16)	0.6 (18)	0.4 (22)	0.3 (23)	0.1 (29)	0.1 (39)	0.0 (58)
david	87	406	0.2 (13)	0.3 (16)	0.2 (19)	0.2 (21)	0.1 (25)	0.1 (32)	0.0 (38)	0.0 (51)	0.0 (51)
fpsol2.i.1	496	11654	3.2 (51)	3.1 (57)	10.5 (74)	8.7 (88)	18.2 (95)	14.8 (102)	22.8 (125)	15.0 (145)	5.9 (166)
fpsol2.i.2	451	8691	2.7 (29)	1.2 (34)	1.6 (44)	3.5 (54)	4.5 (63)	5.7 (71)	9.8 (88)	12.0 (114)	8.1 (143)
fpsol2.i.3	425	8688	1.8 (29)	1.9 (36)	1.5 (47)	1.6 (56)	1.7 (65)	2.0 (73)	6.3 (93)	7.4 (114)	3.0 (143)
games120	120	638	(24-46)	(29-57)	(31-58)	(35-69)	(52-72)	(59-75)	(77-86)	0.1 (98)	0.1 (98)
huck	74	301	0.1 (5)	0.0 (11)	0.0 (16)	0.0 (21)	0.0 (23)	0.0 (28)	0.0 (36)	0.0 (47)	0.0 (47)
jean	80	254	0.0 (7)	0.1 (11)	0.0 (14)	0.1 (20)	0.1 (24)	0.0 (28)	0.0 (33)	0.0 (42)	0.0 (42)
miles250	128	387	0.2 (8)	2.0 (13)	190.2 (22)	305.9 (28)	88.3 (37)	0.7 (46)	0.1 (67)	0.0 (84)	0.0 (84)
miles500	128	1170	(23-29)	(37-46)	(47-52)	(60-61)	1087.8 (71)	4.8 (81)	0.4 (96)	0.0 (110)	0.0 (110)
miles750	128	2113	1601.2 (38)	(57-63)	86.7 (69)	8.8 (77)	2.5 (86)	0.7 (96)	0.2 (108)	0.0 (116)	0.0 (116)
miles1000	128	3216	(52-54)	(74-76)	66.7 (85)	5.5 (90)	1.3 (96)	0.4 (104)	0.1 (114)	0.0 (120)	0.0 (120)
miles1500	128	5198	0.7 (64)	2.0 (91)	0.8 (99)	0.4 (104)	0.3 (108)	0.1 (114)	0.1 (120)	0.0 (123)	0.0 (123)
mulsol.i.1	197	3925	0.5 (38)	0.3 (69)	0.3 (74)	0.3 (76)	0.2 (77)	0.3 (79)	0.3 (82)	0.3 (87)	0.2 (97)
mulsol.i.2	188	3885	0.1 (38)	0.1 (54)	0.2 (55)	0.2 (55)	0.4 (57)	0.2 (60)	0.3 (73)	0.2 (77)	0.1 (98)
mulsol.i.3	184	3916	0.1 (39)	0.2 (55)	0.2 (55)	0.2 (55)	0.1 (57)	0.2 (60)	0.2 (74)	0.2 (77)	0.1 (98)
mulsol.i.4	185	3946	0.1 (38)	0.2 (55)	0.2 (55)	0.1 (55)	0.2 (57)	0.2 (60)	0.2 (73)	0.2 (77)	0.1 (99)
mulsol.i.5	186	3973	0.1 (38)	0.2 (55)	0.2 (55)	0.2 (55)	0.2 (57)	0.2 (60)	0.2 (73)	0.2 (76)	0.1 (98)
myciel4	23	71	0.2 (10)	0.1 (11)	0.0 (12)	0.0 (12)	0.0 (12)	0.0 (12)	0.0 (12)	0.0 (12)	0.0 (12)
myciel5	47	236	116.1 (19)	0.5 (20)	0.2 (22)	0.2 (23)	0.1 (24)	0.1 (24)	0.0 (24)	0.0 (24)	0.0 (24)
myciel6	95	755	(29-38)	32.0 (35)	(40-42)	225.2 (44)	4.4 (45)	2.2 (47)	0.6 (48)	0.1 (48)	0.1 (48)
myciel7	191	2360	(42-75)	(55-65)	(61-84)	(64-86)	(73-89)	(81-86)	402.6 (95)	8.3 (96)	0.5 (96)
queen5_5	25	160	503.9 (16)	0.3 (17)	0.2 (19)	0.1 (20)	0.1 (20)	0.0 (20)	0.0 (20)	0.0 (20)	0.0 (20)
queen6_6	36	290	(17-25)	364.7 (26)	1.6 (28)	1.5 (28)	0.2 (28)	0.2 (28)	0.0 (30)	0.0 (30)	0.0 (30)
queen7_7	49	476	(19-35)	(28-36)	(32-38)	(36-38)	16.5 (39)	1.1 (40)	0.0 (42)	0.0 (42)	0.0 (42)
queen8_8	64	728	(22-47)	(34-48)	(39-50)	(45-52)	393.0 (52)	12.6 (54)	0.0 (56)	0.0 (56)	0.0 (56)
queen8_12	96	1368	(30-72)	(44-75)	(54-79)	(59-79)	(65-81)	(69-82)	802.5 (83)	0.3 (88)	0.2 (88)
queen9_9	81	1056	(26-59)	(38-62)	(47-66)	(50-66)	(56-64)	(59-68)	106.0 (69)	0.2 (72)	0.1 (72)
queen10_10	100	1470	(31-75)	(45-81)	(54-82)	(59-84)	(65-85)	(68-85)	1559.1 (87)	0.4 (90)	0.3 (90)
queen11_11	121	1980	(36-90)	(50-101)	(61-99)	(69-101)	(75-104)	(81-105)	(96-107)	1.2 (110)	1.0 (110)
queen12_12	144	2596	(42-108)	(57-121)	(74-120)	(80-123)	(88-125)	(91-126)	(98-128)	(109-128)	6.0 (132)
queen13_13	169	3328	(49-126)	(62-145)	(79-144)	(91-146)	(98-147)	(103-148)	(115-150)	(123-152)	31.2 (156)
queen14_14	196	4186	(56-147)	(68-170)	(90-172)	(99-169)	(111-171)	(116-172)	(125-176)	(139-178)	64.6 (182)
queen15_15	225	5180	(62-168)	(73-195)	(99-199)	(113-198)	(125-200)	(130-199)	(142-204)	(155-206)	264.1 (210)
queen16_16	256	6320	(71-192)	(81-224)	(110-232)	(124-231)	(138-234)	(147-231)	(165-234)	(174-235)	(238-240)
school1	385	19095	(101-267)	(125-311)	(154-292)	(168-295)	(185-301)	(196-307)	(219-307)	(234-318)	(277-328)
school1_nsh	352	14612	(85-245)	(119-303)	(140-269)	(160-269)	(173-279)	(184-275)	(204-281)	(230-291)	(249-300)
zeroin.i.1	211	4100	0.7 (43)	0.5 (49)	0.8 (57)	1.0 (61)	0.5 (70)	0.5 (72)	0.8 (78)	0.5 (84)	0.2 (91)
zeroin.i.2	211	3541	0.2 (28)	0.6 (38)	0.9 (42)	0.7 (46)	0.8 (51)	0.4 (55)	0.4 (62)	0.5 (73)	0.2 (84)
zeroin.i.3	206	3540	0.3 (28)	0.7 (38)	0.6 (42)	0.7 (47)	0.8 (51)	0.8 (54)	0.4 (62)	0.5 (73)	0.2 (83)

TABLE 3.4: Features, computational times and optimal solution values (if known) of C+CV for the MIPLIB instances.

	n	m	k=4	k=8	k=12	k=16	k=24	k=32	k=64	k=128	k=256
30n20b8	490	28234	(136-180)	(208-239)	(229-260)	(245-268)	(259-283)	(270-289)	(305-314)	18.0 (362)	2.1 (413)
50v-10	233	549	(19-22)	(23-25)	435.6 (27)	10.5 (30)	4.2 (35)	0.8 (36)	0.7 (50)	0.7 (50)	0.2 (50)
b-ball	19	143	0.0 (11)	0.0 (11)	0.0 (11)	0.0 (11)	0.0 (11)	0.0 (11)	0.0 (11)	0.0 (11)	0.0 (11)
csched007	271	5427	(53-131)	(79-218)	(95-198)	(104-206)	(114-211)	(125-215)	(146-205)	(170-186)	(180-199)
csched008	271	4956	(50-127)	(75-210)	(91-193)	(100-197)	(109-209)	(121-200)	(145-201)	(163-194)	(178-203)
csched010	272	5321	(49-134)	(72-219)	(92-189)	(101-178)	(112-171)	(123-176)	(147-177)	(168-176)	(182-189)
dfn-gwin-UUM	156	1409	(40-47)	(47-50)	24.6 (47)	5.0 (56)	(71-74)	(81-87)	1303.0 (96)	241.6 (99)	0.1 (101)
eil33.2	32	496	0.0 (24)	0.0 (28)	0.0 (29)	0.0 (30)	0.0 (30)	0.0 (31)	0.0 (31)	0.0 (31)	0.0 (31)
eilB101	100	3921	(61-65)	49.9 (73)	5.5 (77)	1.7 (79)	1.0 (82)	1.0 (84)	0.7 (91)	0.0 (94)	0.0 (94)
ger50_17_trans	498	14021	(111-135)	(138-163)	(152-184)	(151-189)	(185-194)	(197-210)	(255-323)	(287-351)	(300-359)
glass4	392	24768	(120-276)	(191-290)	(219-301)	(232-306)	(255-311)	(263-316)	(304-325)	(329-335)	98.4 (345)
gmu-35-40	357	3461	12.6 (19)	(41-71)	(59-75)	(68-102)	(82-114)	(94-121)	(146-162)	(207-213)	140.1 (239)
gmu-35-50	358	4443	400.3 (28)	(52-78)	(69-85)	(80-100)	(99-111)	(108-124)	(169-172)	35.5 (229)	5.2 (258)
go19	361	1978	(28-177)	(33-240)	(36-251)	(39-244)	(41-249)	(47-272)	(133-266)	(186-260)	(203-265)
harp2	92	999	0.1 (18)	0.1 (18)	0.1 (18)	0.1 (19)	0.1 (19)	0.1 (19)	0.1 (19)	0.1 (19)	0.1 (19)
k16x240	256	600	4.3 (14)	8.6 (15)	1.7 (16)	1.9 (16)	2.2 (16)	1.4 (16)	1.0 (16)	1.0 (16)	0.4 (16)
m100n500k4r1	100	2248	(38-74)	(54-82)	(65-80)	(70-80)	(75-84)	(81-84)	17.8 (86)	0.9 (89)	0.9 (89)
mik.250-1-100.1	100	4950	0.0 (75)	0.0 (87)	0.0 (91)	0.0 (93)	0.0 (95)	0.0 (96)	0.0 (98)	0.0 (99)	0.0 (99)
neos-1228986	241	2915	(18-68)	(43-84)	(65-117)	(74-137)	(88-124)	(98-135)	(132-141)	36.7 (160)	0.1 (161)
neos-1426635	486	6210	(18-67)	(18-164)	(48-76)	(77-173)	(115-257)	(143-217)	(186-264)	(235-281)	(318-325)
neos-1440225	328	8630	(70-96)	(82-242)	(86-177)	(103-215)	(136-219)	(157-211)	(204-224)	(248-273)	(270-286)
neos-777800	475	38862	(142-201)	(216-259)	(236-279)	(253-291)	(274-299)	(295-305)	1331.3 (311)	142.1 (315)	48.1 (317)
neos-911880	83	1704	0.1 (35)	0.0 (35)	0.0 (35)	0.0 (35)	0.0 (35)	0.0 (35)	0.0 (35)	0.0 (48)	0.0 (48)
neos15	492	1680	(19-86)	(25-133)	(26-143)	(33-147)	(35-227)	(37-215)	(81-204)	(167-217)	2.3 (261)
neos788725	433	6960	(62-96)	(66-112)	(74-115)	(86-217)	(120-268)	(130-230)	(180-328)	(222-344)	(255-353)
neos858960	128	2427	1.1 (43)	0.1 (46)	0.1 (46)	0.1 (46)	0.1 (47)	0.1 (48)	0.1 (48)	0.1 (48)	0.1 (48)
noswot	172	1442	(10-27)	(30-52)	(44-68)	(54-77)	(68-82)	(84-97)	0.0 (107)	0.0 (122)	0.0 (147)
ns1766074	110	1755	(34-80)	(50-82)	(60-83)	(68-88)	(74-90)	(80-90)	1078.7 (95)	7.6 (100)	7.5 (100)
p80x400b	474	990	4.5 (7)	0.4 (10)	3.7 (16)	8.0 (21)	5.4 (30)	3.8 (32)	9.8 (43)	3.7 (78)	3.5 (78)
pg	135	2760	0.3 (25)	0.3 (25)	0.3 (25)	0.3 (25)	0.3 (25)	0.3 (25)	0.3 (25)	0.3 (25)	0.1 (35)
pg5_34	225	5100	0.2 (25)	0.2 (25)	0.2 (25)	0.2 (25)	0.2 (25)	0.2 (25)	0.2 (25)	0.1 (25)	0.1 (125)
probportfolio	302	45450	0.3 (226)	0.2 (264)	0.2 (276)	0.2 (283)	0.2 (289)	0.2 (292)	0.2 (297)	0.2 (299)	0.2 (300)
ran14x18	284	756	1.2 (16)	1.2 (16)	(16-20)	1.3 (16)	2.6 (32)	1.9 (32)	1.2 (32)	1.0 (32)	1.2 (32)
ran14x18.disj-8	447	15861	54.2 (96)	20.7 (99)	(103-105)	5.5 (103)	(146-149)	59.5 (165)	8.1 (172)	8.4 (182)	1.4 (195)
ran16x16	288	768	0.1 (16)	1.2 (16)	(19-20)	2.5 (16)	2.3 (32)	2.1 (32)	1.6 (32)	1.3 (32)	1.4 (32)
swath	482	22110	(125-361)	(152-421)	(192-423)	(233-430)	(257-437)	(272-443)	(303-451)	(311-450)	(333-451)
timtab1	332	12582	(91-129)	(121-144)	(140-162)	(155-170)	(174-192)	(187-198)	25.1 (224)	0.3 (259)	0.1 (270)

TABLE 3.5: Features, computational times and optimal solution values (if known) of C+CV for the Netlib instances.

	n	m	k=4	k=8	k=12	k=16	k=24	k=32	k=64	k=128	k=256
adlittle	53	239	0.1 (10)	0.0 (14)	0.0 (16)	0.0 (17)	0.0 (20)	0.0 (23)	0.0 (27)	0.0 (27)	0.0 (27)
agg	164	1694	1.6 (22)	30.6 (44)	19.9 (57)	2.4 (62)	0.6 (78)	0.4 (83)	0.1 (104)	0.0 (114)	0.0 (130)
agg2	280	4010	(42-46)	59.5 (63)	(81-84)	(91-101)	(109-114)	(127-128)	1.2 (147)	0.4 (180)	0.1 (199)
agg3	282	4104	(44-48)	209.2 (66)	(83-89)	(93-105)	(112-119)	917.1 (128)	2.0 (151)	0.3 (180)	0.1 (200)
bandm	180	2379	30.2 (40)	33.8 (52)	4.1 (58)	1.8 (62)	0.6 (70)	0.1 (82)	0.1 (112)	0.1 (126)	0.0 (143)
beaconfd	90	1199	0.3 (26)	0.1 (36)	0.1 (37)	0.0 (38)	0.0 (40)	0.1 (40)	0.0 (40)	0.0 (42)	0.0 (42)
blend	54	548	0.1 (20)	0.1 (26)	0.0 (30)	0.0 (32)	0.0 (36)	0.0 (39)	0.0 (43)	0.0 (43)	0.0 (43)
bnl1	448	2102	(26-55)	(33-61)	(38-77)	(43-79)	(65-71)	(74-77)	1223.5 (107)	1.5 (149)	1.0 (218)
boeing1	284	2751	2.4 (23)	83.9 (41)	20.9 (46)	39.0 (53)	50.3 (64)	12.1 (73)	0.9 (104)	0.8 (142)	0.2 (164)
boeing2	122	740	0.9 (19)	2.4 (25)	4.3 (30)	3.1 (35)	0.5 (37)	0.3 (42)	0.2 (56)	0.1 (71)	0.0 (71)
bore3d	52	615	0.2 (23)	0.1 (28)	0.1 (29)	0.0 (30)	0.0 (32)	0.0 (35)	0.0 (40)	0.0 (40)	0.0 (40)
brandy	113	1613	8.5 (34)	3.3 (42)	2.3 (49)	0.5 (53)	0.1 (60)	0.1 (64)	0.0 (75)	0.0 (84)	0.0 (84)
capri	166	2676	(43-47)	125.0 (57)	2.8 (59)	7.8 (64)	0.7 (73)	0.7 (78)	0.2 (96)	0.1 (105)	0.0 (122)
czprob	475	464	8.8 (3)	18.4 (4)	20.6 (5)	39.4 (6)	110.8 (9)	46.2 (9)	28.0 (11)	28.2 (13)	27.9 (13)
degen2	382	5686	(67-102)	(77-114)	(86-119)	(89-134)	(118-126)	(131-138)	(170-172)	1.1 (200)	1.2 (221)
e226	148	1537	29.4 (29)	1.9 (38)	0.6 (42)	0.3 (50)	0.3 (65)	0.2 (75)	0.0 (90)	0.0 (99)	0.0 (113)
etamacro	307	1489	(30-71)	(36-85)	(39-88)	(42-104)	(52-105)	(60-120)	(106-126)	(140-143)	31.1 (165)
fffff800	306	3886	1.6 (24)	17.7 (43)	8.7 (52)	18.2 (76)	3.7 (110)	2.8 (128)	1.0 (156)	1.0 (166)	0.3 (174)
finnis	350	977	2.8 (11)	282.6 (24)	(31-34)	670.0 (38)	793.9 (45)	70.8 (50)	7.1 (61)	17.0 (92)	3.5 (108)
forplan	104	1153	(33-35)	137.1 (41)	87.9 (47)	16.2 (51)	8.9 (57)	4.2 (59)	0.3 (67)	0.0 (75)	0.0 (75)
gfrdpnc	322	314	0.2 (4)	1.4 (9)	0.8 (11)	1.8 (13)	3.0 (19)	3.7 (23)	2.1 (40)	1.5 (72)	2.1 (92)
grow15	300	2934	745.1 (15)	(28-36)	(43-63)	(54-67)	0.1 (105)	0.0 (150)	0.0 (225)	0.0 (255)	0.0 (269)
grow22	440	4315	(12-24)	(29-35)	(41-49)	(58-104)	(83-108)	0.1 (132)	0.0 (286)	0.0 (352)	0.0 (395)
grow7	140	1371	22.5 (17)	(32-34)	0.0 (56)	0.0 (77)	0.0 (98)	0.0 (105)	0.0 (119)	0.0 (125)	0.0 (132)
israel	163	10628	0.2 (98)	0.1 (118)	0.2 (129)	0.1 (133)	0.0 (137)	0.0 (138)	0.0 (145)	0.0 (147)	0.0 (152)
lotfi	122	528	10.2 (19)	26.3 (25)	11.3 (30)	0.4 (31)	0.9 (37)	0.3 (44)	0.1 (56)	0.0 (67)	0.0 (67)
perold	500	5743	(49-188)	(55-285)	(62-253)	(76-223)	(98-339)	(126-226)	(185-232)	(246-262)	17.6 (305)
pilot4	352	5707	(46-75)	(66-104)	(95-122)	(117-124)	38.6 (122)	156.9 (130)	22.2 (142)	8.0 (178)	0.9 (193)
recipe	55	129	0.0 (1)	0.4 (8)	0.0 (12)	0.0 (15)	0.0 (25)	0.0 (28)	0.0 (38)	0.0 (38)	0.0 (38)
sc105	59	356	0.3 (16)	0.2 (20)	0.1 (25)	0.1 (26)	0.1 (28)	0.1 (34)	0.0 (42)	0.0 (42)	0.0 (42)
sc205	113	1558	(35-40)	3.7 (45)	1.4 (54)	0.7 (56)	0.5 (63)	0.3 (64)	0.1 (71)	0.0 (90)	0.0 (90)
scagr25	221	8050	1641.0 (69)	5.9 (101)	0.5 (113)	0.2 (120)	0.1 (127)	0.1 (132)	0.1 (137)	0.1 (151)	0.0 (176)
scagr7	58	661	0.0 (21)	0.0 (28)	0.0 (32)	0.0 (33)	0.0 (35)	0.0 (39)	0.0 (46)	0.0 (46)	0.0 (46)
scfxm1	242	2057	2.5 (17)	(45-49)	(58-61)	(64-67)	(80-82)	74.1 (91)	0.4 (111)	0.3 (142)	0.0 (168)
scfxm2	485	4231	137.7 (18)	(36-40)	(64-85)	(82-105)	(101-124)	(115-139)	(172-187)	4.1 (228)	1.2 (285)
scorpion	105	502	0.3 (11)	0.2 (12)	0.1 (36)	0.1 (38)	0.1 (41)	0.0 (43)	0.0 (50)	0.0 (70)	0.0 (70)
scrs8	181	1835	(30-35)	(47-50)	(53-56)	355.7 (58)	7.7 (62)	1.3 (66)	0.1 (74)	0.3 (91)	0.1 (117)
scsd1	77	202	0.1 (8)	0.1 (13)	0.1 (16)	0.0 (18)	0.0 (20)	0.0 (25)	0.0 (32)	0.0 (45)	0.0 (45)
scsd6	147	342	(14-16)	(20-22)	(24-28)	(28-31)	(34-35)	458.6 (43)	0.6 (53)	0.2 (62)	0.0 (86)
scsd8	397	1069	(14-25)	(20-63)	(24-71)	(26-92)	(31-104)	(37-125)	(70-149)	(131-168)	31.3 (199)
sctap1	269	706	70.7 (15)	287.4 (22)	111.7 (25)	44.4 (29)	7.9 (35)	1.7 (40)	1.1 (55)	0.6 (71)	0.6 (91)
share1b	102	493	0.1 (7)	0.0 (15)	0.1 (27)	0.0 (31)	0.0 (37)	0.0 (45)	0.0 (60)	0.0 (68)	0.0 (68)
share2b	93	619	0.1 (9)	0.0 (12)	0.0 (37)	0.0 (51)	0.0 (63)	0.0 (65)	0.0 (66)	0.0 (70)	0.0 (70)
shell	252	247	0.1 (4)	0.3 (6)	0.8 (9)	1.4 (10)	1.1 (13)	1.4 (17)	0.6 (27)	1.8 (49)	0.2 (75)
ship04l	313	593	0.3 (5)	2.1 (9)	2.2 (9)	2.5 (11)	3.2 (12)	3.4 (13)	3.9 (14)	4.0 (19)	2.9 (32)
ship04s	213	391	0.2 (4)	1.1 (8)	2.2 (10)	2.0 (10)	0.8 (11)	1.0 (12)	0.9 (17)	1.0 (25)	0.2 (37)
ship08s	284	462	0.2 (4)	0.3 (7)	1.3 (12)	5.3 (14)	4.0 (17)	2.4 (18)	2.1 (23)	2.3 (26)	2.5 (40)
ship12s	344	592	0.2 (3)	0.2 (3)	0.3 (3)	0.6 (11)	1.2 (21)	4.1 (25)	8.9 (37)	3.2 (42)	2.5 (63)
stair	246	11285	(104-116)	33.9 (131)	5.3 (132)	3.8 (138)	2.3 (145)	1.7 (149)	10.7 (168)	0.5 (171)	0.1 (189)
standata	258	411	0.3 (1)	0.4 (3)	1.1 (4)	1.0 (5)	1.0 (9)	1.8 (9)	2.2 (20)	0.7 (28)	0.6 (64)
standmps	360	638	2.4 (8)	8.9 (12)	15.3 (16)	23.6 (19)	20.8 (25)	13.5 (26)	12.0 (38)	17.4 (52)	6.1 (69)
stocfor1	62	272	0.1 (10)	0.0 (13)	0.0 (16)	0.0 (20)	0.0 (24)	0.0 (29)	0.0 (37)	0.0 (37)	0.0 (37)
tuff	137	1464	0.7 (26)	0.7 (36)	3.7 (45)	0.6 (47)	0.2 (58)	0.1 (64)	0.2 (73)	0.2 (79)	0.0 (90)
vtpbase	51	354	0.1 (14)	0.0 (23)	0.0 (25)	0.0 (30)	0.0 (35)	0.0 (40)	0.0 (44)	0.0 (44)	0.0 (44)
wood1p	171	3310	0.1 (28)	0.1 (62)	0.0 (67)	0.0 (78)	0.0 (94)	0.0 (106)	0.0 (135)	0.0 (145)	0.0 (155)

TABLE 3.6: Features, computational times and optimal solution values (if known) of C+CV for the Random instances.

	n	m	k=4	k=8	k=12	k=16	k=24	k=32	k=64	k=128	k=256
grp1_1	68	191	17.6 (15)	12.4 (18)	0.5 (19)	0.3 (20)	0.1 (25)	0.1 (25)	0.0 (29)	0.0 (35)	0.0 (35)
grp1_2	68	169	39.4 (14)	4.9 (16)	0.8 (18)	0.3 (20)	0.1 (22)	0.1 (22)	0.1 (26)	0.0 (36)	0.0 (36)
grp1_3	58	217	(18-21)	(22-24)	43.2 (25)	4.6 (27)	0.4 (28)	0.1 (31)	0.0 (36)	0.0 (36)	0.0 (36)
grp1_4	60	187	50.6 (16)	48.3 (19)	1.1 (21)	0.3 (22)	0.2 (23)	0.1 (28)	0.0 (35)	0.0 (35)	0.0 (35)
grp1_5	75	184	1.0 (12)	0.6 (14)	0.3 (16)	0.3 (20)	0.1 (21)	0.1 (23)	0.1 (28)	0.0 (37)	0.0 (37)
grp2_1	75	230	(17-20)	(21-24)	(24-26)	43.2 (28)	1.6 (29)	0.6 (32)	0.1 (35)	0.0 (46)	0.0 (46)
grp2_2	95	183	3.7 (11)	27.2 (15)	2.2 (17)	1.4 (19)	0.4 (23)	0.2 (27)	0.0 (33)	0.0 (47)	0.0 (47)
grp2_3	87	219	224.2 (16)	(20-22)	1668.2 (23)	511.1 (26)	2.4 (29)	0.5 (32)	0.2 (35)	0.0 (46)	0.0 (46)
grp2_4	98	203	32.1 (14)	135.8 (18)	78.4 (20)	16.8 (22)	1.1 (24)	0.6 (26)	0.2 (35)	0.0 (48)	0.0 (48)
grp2_5	93	160	0.5 (9)	2.1 (13)	1.0 (15)	0.5 (17)	0.3 (21)	0.1 (24)	0.1 (33)	0.0 (43)	0.0 (43)
grp3_1	102	232	517.3 (16)	(19-22)	1041.8 (23)	104.6 (25)	5.1 (28)	1.0 (30)	0.1 (39)	0.0 (55)	0.0 (55)
grp3_2	108	217	1384.5 (15)	1146.5 (19)	1765.4 (23)	46.4 (24)	10.9 (28)	0.8 (30)	0.2 (41)	0.0 (53)	0.0 (53)
grp3_3	122	213	4.5 (11)	9.1 (14)	3.6 (16)	1.2 (18)	1.2 (21)	0.3 (26)	0.2 (42)	0.0 (60)	0.0 (60)
grp3_4	104	217	501.9 (15)	1435.2 (19)	1062.6 (22)	329.4 (24)	5.5 (27)	2.9 (31)	0.1 (40)	0.0 (55)	0.1 (55)
grp3_5	107	216	337.4 (15)	1245.3 (19)	975.8 (22)	100.6 (23)	2.5 (26)	0.8 (29)	0.1 (38)	0.1 (50)	0.0 (50)
grp4_1	142	221	226.3 (12)	(16-17)	(19-20)	446.3 (22)	182.9 (26)	10.9 (28)	0.8 (35)	0.4 (43)	0.1 (63)
grp4_2	125	246	1605.3 (15)	(18-20)	(21-23)	495.7 (25)	36.0 (27)	1.2 (33)	0.3 (45)	0.0 (58)	0.0 (58)
grp4_3	135	187	0.9 (8)	1.3 (11)	1.1 (13)	1.6 (16)	0.8 (18)	0.8 (20)	0.2 (29)	0.2 (34)	0.1 (57)
grp4_4	128	161	0.2 (5)	0.3 (8)	0.3 (9)	0.4 (12)	0.5 (15)	0.2 (18)	0.1 (32)	0.0 (53)	0.0 (53)
grp4_5	126	195	0.5 (7)	2.7 (12)	3.6 (15)	1.8 (18)	1.0 (20)	0.6 (24)	0.2 (40)	0.0 (57)	0.0 (57)
grp5_1	173	219	0.4 (7)	0.5 (9)	1.1 (12)	0.8 (13)	2.3 (17)	1.6 (19)	0.5 (32)	0.5 (43)	0.1 (68)
grp5_2	161	155	0.3 (2)	0.5 (5)	1.0 (6)	0.5 (8)	1.4 (10)	0.3 (12)	0.9 (23)	0.3 (30)	0.1 (54)
grp5_3	158	195	0.4 (7)	1.3 (10)	2.5 (13)	1.6 (15)	2.0 (19)	1.2 (23)	0.7 (31)	0.4 (40)	0.1 (61)
grp5_4	159	192	0.2 (3)	0.5 (6)	0.5 (9)	1.2 (12)	1.5 (17)	0.9 (21)	1.3 (30)	0.3 (43)	0.1 (67)
grp5_5	158	199	0.2 (4)	0.3 (8)	0.4 (9)	0.4 (12)	0.7 (13)	0.3 (18)	0.2 (30)	0.3 (38)	0.1 (64)
grp6_1	69	292	(20-24)	(23-30)	(28-31)	555.6 (31)	1.5 (35)	1.5 (35)	0.3 (39)	0.0 (46)	0.0 (46)
grp6_2	74	266	(19-24)	(23-27)	(25-28)	135.3 (30)	2.8 (31)	0.5 (33)	0.2 (36)	0.0 (44)	0.0 (44)
grp6_3	50	250	(19-22)	544.7 (24)	67.2 (26)	3.0 (27)	0.5 (27)	0.1 (29)	0.0 (33)	0.0 (33)	0.0 (33)
grp6_4	52	275	(20-24)	(25-26)	91.9 (27)	6.6 (28)	0.6 (30)	0.3 (32)	0.0 (37)	0.0 (37)	0.0 (37)
grp6_5	63	297	(21-26)	(24-29)	(28-30)	13.2 (32)	0.7 (33)	0.2 (36)	0.1 (43)	0.0 (43)	0.0 (43)
grp7_1	96	223	(16-19)	(19-25)	(23-26)	440.8 (27)	4.0 (31)	0.8 (34)	0.1 (38)	0.0 (50)	0.0 (50)
grp7_2	77	272	(19-25)	(23-29)	(26-30)	358.7 (31)	5.6 (32)	0.8 (34)	0.2 (40)	0.0 (46)	0.0 (46)
grp7_3	77	349	(21-31)	(26-35)	(28-35)	(34-36)	39.9 (37)	2.1 (38)	0.6 (43)	0.1 (49)	0.1 (49)
grp7_4	87	316	(21-29)	(24-30)	(26-33)	(31-32)	8.8 (35)	0.7 (38)	0.3 (44)	0.0 (55)	0.0 (55)
grp7_5	78	291	(20-25)	(24-28)	(26-30)	320.8 (32)	10.8 (33)	0.8 (35)	0.3 (39)	0.0 (47)	0.0 (47)
grp8_1	115	325	(19-28)	(22-32)	(23-37)	(27-36)	805.4 (39)	10.4 (41)	0.3 (51)	0.0 (66)	0.0 (66)
grp8_2	121	301	(18-23)	(21-28)	(23-29)	(27-31)	482.5 (33)	2.1 (37)	0.2 (49)	0.1 (62)	0.1 (62)
grp8_3	118	302	(18-22)	(21-30)	(24-32)	(26-33)	43.2 (35)	3.1 (38)	0.3 (49)	0.0 (63)	0.0 (63)
grp8_4	122	298	(17-29)	(20-32)	(22-35)	(25-38)	(31-40)	242.7 (41)	0.4 (51)	0.1 (67)	0.1 (67)
grp8_5	108	302	(19-21)	(22-26)	(26-30)	251.4 (29)	6.9 (33)	1.6 (36)	0.1 (45)	0.0 (59)	0.0 (59)
grp9_1	136	340	(19-25)	(23-29)	(24-32)	(27-35)	(35-37)	708.4 (41)	1.2 (48)	0.1 (54)	0.1 (74)
grp9_2	143	237	107.8 (13)	(17-18)	(20-21)	(23-24)	326.3 (29)	48.3 (31)	0.7 (39)	0.5 (46)	0.1 (64)
grp9_3	146	342	(17-24)	(20-31)	(23-33)	(26-35)	(34-37)	62.4 (40)	0.9 (49)	0.4 (58)	0.0 (74)
grp9_4	139	332	(17-24)	(21-28)	(23-31)	(27-33)	411.4 (36)	20.7 (38)	0.4 (45)	0.2 (57)	0.0 (73)
grp9_5	138	297	(17-19)	(20-25)	(23-28)	(25-30)	201.5 (33)	27.0 (36)	0.8 (42)	0.2 (51)	0.0 (69)
grp10_1	168	321	(15-19)	(19-26)	(21-28)	(24-28)	559.7 (32)	29.4 (34)	0.9 (47)	0.5 (57)	0.1 (80)
grp10_2	169	348	(16-22)	(19-28)	(21-29)	(24-34)	(31-35)	552.1 (37)	1.3 (51)	0.3 (62)	0.1 (83)
grp10_3	161	296	(14-17)	(18-21)	(21-23)	1462.4 (23)	106.0 (30)	15.7 (32)	0.7 (43)	0.3 (56)	0.1 (77)
grp10_4	157	281	94.7 (13)	287.3 (17)	478.2 (20)	100.2 (22)	22.8 (27)	1.7 (31)	0.4 (41)	0.6 (49)	0.1 (73)
grp10_5	164	265	134.4 (13)	(16-18)	(19-22)	(22-24)	80.4 (27)	25.2 (29)	0.8 (42)	0.6 (50)	0.1 (71)

Chapter 4

Chance Constrained Problem with Integer Scenario Variables

[1]

4.1. Introduction

Mathematical optimization problems originally arose with a scheme in which the input data are known; but, very soon, the needs of real applications, in which part of the problem setting is not deterministic, led the research to investigate the case of problems with probabilistic constraints [64]. In this kind of problems, a random variable is given and the feasible region of the problem depends on its realization. The decision maker fixes a threshold value, say, α, and the solution must optimize a given objective function while belonging to the feasible region with probability at least $1 - \alpha$. In other words, the problem solver is willing to run the risk of finding a solution that could be infeasible, once the random variable realization takes place. This class of problems is defined as *Chance-Constrained Mathematical Programming* (CCP) problem and it finds application in several fields. See [65] for an application in the water management of a real-life water resource system; in [66], stochastic programming is used to find the optimal vaccination policy for controlling infectious disease epidemics; while, in [67] the authors develop five stochastic programming models to identify cost-effective acid rain control strategies.

Now, we introduce the formal definition of the problem:

Definition 0 *Given a probability value defined by the decision-maker, say, α and a random variable, say, w, a Chance-Constrained Mathematical Programming problem can be expressed as:*

$$max\{cx : Pr(x \in C_x(w)) \geq 1 - \alpha, x \in X\} \qquad \text{(CCP)}$$

Where $C_x(w)$ is a set that depends on the realization of w and X is a set that is described by deterministic constrains.

To be thorough, CCP problem can be generalized to the case in which an unsatisfied realization (i.e., an infeasible solution) can enter a *recovery mode* [68]. In this case, the objective function takes into account also a cost that will be proportional to the infeasibility degree.

In order to obtain a deterministic reformulation of CCP, some assumptions are needed:

[1] The results of this chapter appears in: A. Lodi, E. Malaguti, M. Monaci, G. Nannicini and P. Paronuzzi, "Chance Constrained Problem with Integer Scenario Variables", *Technical Report OR-19-7*, http://or.dei.unibo.it/technical-reports, 2019. [63]

1. the sample space, denoted as Ω, is discrete and finite, and in particular $\Omega = \{w^k : k = 1, \dots, h\}$;
2. the objective function contribution of the normal mode is preferred to the one of the recovery mode.

The first assumption, even if it looks very restrictive, results quite common in practice. Typically, the expected realization of a future event is not very detailed and, often, it is the result of a discretization process. When this is the case, a generic realization w^k is usually called *scenario*. With the second assumption we ensure the two-stage consistency [69].

Now, introducing a set of indicator variables z_h, the problem can be formulated in the following form:

$$\begin{array}{rrrcl} & \max & cx & & \\ & \text{s.t.:} & x & \in & X \\ k = 1, \dots, h & z_k = 0 \Rightarrow & x & \in & C_x(w^k) \\ k = 1, \dots, h & z_k = 1 \Rightarrow & x & \in & \bar{C}_x(w^k) \\ & & \sum_{k=1}^{h} p_k z_k & \leq & \alpha \\ k = 1, \dots, h & & z_k & \in & \{0,1\} \end{array} \tag{CCPR}$$

Where $C_x(w^k)$ and $\bar{C}_x(w^k)$ are the the feasible sets for the normal and recovery modes, respectively.
Under the additional assumption that:

3. all the $C_x(w^k)$'s and $\bar{C}_x(w^k)$'s share the same recession cone;

the problem can be modeled as a MINLP (Mixed Integer Non Linear Program):

$$\begin{array}{rccccccccc} \max & cx & & & & & & & & \\ \text{s.t.:} & Ax & & & & & & & \leq & b \\ & g^1(x, & y^1) & & & & & & \leq & M^1 z_1 \\ & \bar{g}^1(x, & \bar{y}^1) & & & & & & \leq & \bar{M}^1(1 - z_1) \\ & \vdots & & & \ddots & & & & \vdots & \\ & g^h(x, & & & & & y^h) & \leq & M^h z_h \\ & \bar{g}^h(x, & & & & & \bar{y}^h) & \leq & \bar{M}^h(1 - z_h) \\ & & p_1 z_1 & + & \dots & + & p_h z_h & \leq & \alpha \\ & & z_1, & & \dots & & z_h & \in & \{0,1\} \\ & & y^1, & & \dots & & y^h & & \text{integer} \end{array} \tag{CCP-MINLP}$$

In this formulation, we assume that g^k, $\bar{g}^k$ are vectors of convex functions and M^k, $\bar{M}^k$ are vectors of large enough constants. Since there is no objective function contribution associated with variables y, the second stage problems are feasibility problems. But, if necessary, the vector of first-stage variables x can be enlarged in order to consider objective contributions of the second stage problems. Our work refers to the case where all x variables are continuous, but the framework we propose can handle also the integrality requirements on the set X at the cost of additional computational complexity.

The third assumption is necessary because, otherwise, the recession cone of CCP-MINLP and the one of CCPR may differ. Indeed, the recession cone of CCPR is given by the union of the intersection of the recession cones only of the sets $C_x(w^k)$, $\bar{C}_x(w^k)$ that are active, according to z_h variables. On the other hand, the recession cones of inactive sets in CCP-MINLP cannot be deactivated, hence the only unbounded

directions are those that belong to all the sets $C_x(w^k)$, $\bar{C}_x(w^k)$ at the same time (see [70]). This is clearly not the same as the recession cone of CCPR, unless the last introduced condition holds.

As already pointed out, CCP-MINLP corresponds to a Mixed Integer Non Linear Program and it can be implemented within a commercial solver framework. But, this approach might probably result unsuccessful due to two main drawbacks related to this specific formulation of the problem. The first main aspect that must be taken into account is that the size of this formulation increase together with the number of considered scenarios; the second point is linked to the presence of big-M constraints that, generally, tends to make very weak the relaxations of mathematical programs (see, e.g., [71]).

Luedtke in [72] introduced a Benders decomposition approach for chance constrained problems; his algorithm only applies to the case in which the scenario subproblems are described by linear functions and variables must be continuous. Lodi et al. ([3]) proposed a finitely convergent Branch-and-Cut algorithm that applies to the case in which g^k, $\bar{g}^k$ are convex, potentially nonlinear, functions, as in the case we consider, but the scenario variables, namely y^k and $\bar{y}^k$, are restricted to be continuous. In this work, the authors define a master problem in which the indicator variables z_h are kept, but all the big-M constraints are removed. When the master problem produces a solution, say, $\hat{x}$, that is infeasible, then a cutting plan, obtained as outer approximation cut [73], is generated for the sets $C_x(w^k)$ or $\bar{C}_x(w^k)$, depending on the values of the indicator variables.

Our purpose is generalizing the Branch-and-Cut algorithm of [3] to the case represented by CCP-MINLP in which the scenario variables are integer. The main issue that needs to be addressed is that, in opposite with the case considered by Lodi et al., the sets $C_x(w^k)$, $\bar{C}_x(w^k)$ are nonconvex sets due to the integrality constraints of the scenario variables. In this work, we present a Branch-and-Cut approach that also applies to the case in which only some of the components of y^k, $\bar{y}^k$ are restricted to be integer.

4.2. Decomposition algorithm

In the following, we propose a Branch-and-Cut algorithm where we generate cutting planes as outer approximation point cuts, when possible. This approach generalizes the one proposed by Lodi et al. [3] that only applies to the case in which the scenario variables y^k are continuous. From now on, our discussion focuses on the sets $C_{x,y}(w^k)$, defined as the feasible regions of a normal mode scenario subproblem. We can obviously define also $\bar{C}_{x,y}(w^k)$ and equivalently extend the procedure.

We follow a decomposition approach whereby we define a master problem and h subproblems, one for each scenario, involving scenario-dependent constraints. Whenever the solution of the master problem does not satisfy the constraints of some active scenario, cuts are generated for the corresponding set $C_{x,y}(w^k)$. These cuts are then added to the master problem. This basic idea yields an exact algorithm for CCPR.

Given a point $\hat{x}$ from the master, we must answer the question: does there exist a $\hat{y}$ such that $(\hat{x}, \hat{y}) \in C_{x,y}(w^k)$? Let us fix w^k and write $C_{x,y}$ simply. Therefore, for a given scenario k, we can write

$$C_{x,y} = \{(x, y) : g(x, y) \leq 0, y \text{ integer}\} \tag{4.1}$$

where $g(x, y)$ is a vector of convex functions. In addition, let $\text{Cont}(C_{x,y})$ be the continuous relaxation of $C_{x,y}$ and let $\text{Conv}(C_{x,y})$ be the convex hull of $C_{x,y}$.

There are three notable cases in which $\hat{x}$ is not feasible and we want to separate it:

1. $\hat{x} \notin \text{Proj}_x\text{Cont}(C_{x,y})$;
2. $\hat{x} \notin \text{Proj}_x\text{Conv}(C_{x,y})$;
3. $\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y})$.

In the first case, it is sufficient to find an inequality valid for the projection of the continuous relaxation of $C_{x,y}$ and, so, we can straightforwardly apply the same separation procedure used in [3]. In the second case, we necessarily need an inequality valid for the projection of the convex hull of $C_{x,y}$, which exists because we are separating from a convex set. In the last case, such an inequality does not exist, and in order to separate $\hat{x}$, we must perform some type of branching. We remark the difficulty stems from the fact that $C_{x,y}$ is a nonconvex set due to the integrality constraints.

The pseudo-code of our decomposition approach is provided in Algorithm 4.2.0.1. Since the master problem involves the x variables only, the separation routines must find a cut in the x space. In the rest of this section, we provide the separation algorithms for cases 1 and 2 and we discuss different options to perform the branching when case 3 occurs.

4.2.1 Case 1: separation when $\hat{x} \notin \text{Proj}_x\text{Cont}(C_{x,y})$

Define the problem

$$\min_{(x,y)\in\text{Cont}(C_{x,y})} \frac{1}{2}\|x - \hat{x}\|_x^2, \tag{PROJ}$$

where by $\|\cdot\|_x$ we denote the Euclidean distance in the x space only. If $\hat{x} \notin \text{Cont}(C_{x,y})$, the optimal value of (PROJ) must be strictly greater than 0.

Theorem 1 of [3], adapted to our case, states that:

Theorem 1 *Let $\text{Cont}(C_{x,y})$ be a closed set such that $\text{Proj}_x\text{Cont}(C_{x,y})$ is convex, and $\hat{x} \notin \text{Proj}_x\text{Cont}(C_{x,y})$. Let $(\bar{x}, \bar{y})$ be the optimal solution to (PROJ) with positive objective function value. Then, the hyperplane*

$$(\hat{x} - \bar{x})^T(x - \bar{x}) \leq 0$$

separates $\hat{x}$ from $\text{Proj}_x\text{Cont}(C_{x,y})$. This hyperplane is the deepest valid cut that separates $\hat{x}$ from $\text{Proj}_x\text{Cont}(C_{x,y})$, if depth is computed in ℓ_2-norm.

We refer to [3] for a proof of this result.

4.2.2 Case 2: separation when $\hat{x} \notin \text{Proj}_x\text{Conv}(C_{x,y})$

Suppose that $\hat{x} \in \text{Proj}_x\text{Cont}(C_{x,y})$. We want to discover (i) if $\hat{x}$ lies in $\text{Proj}_x\text{Conv}(C_{x,y})$ and, if not, (ii) we aim to define a facet to separate $\hat{x}$. Let n be the dimension of X and remind the following well known theorem:

Theorem 2 (Carathéodory's theorem) *If a point x of R^n lies in the convex hull of a set P, then x can be written as the convex combination of at most $n + 1$ points in P.*

Algorithm 4.2.0.1 Decomposition Algorithm

1: Define a master problem as:

$$\begin{array}{rrcl} \max & cx & & \\ \text{s.t.:} & Ax & \leq & b \\ & \sum_{k=1}^{h} p_k z_k & \leq & \alpha \\ & z & \in & \{0,1\}^h, \end{array} \qquad \text{(MASTER)}$$

2: **repeat**
3: Select an active node and solve its continuous relaxation.
4: If the node is not pruned, then branch on one variable z_k.
5: For each node of the tree with solution $(\hat{x}, \hat{z}), \hat{z} \in \{0,1\}^h$, do:
6: **for** $k \in \{1, \ldots, h\} : \hat{z}_k = 0$ and $\text{Cont}(C_{x,y}(w^k)) \neq \varnothing$ **do**
7: **if** $\nexists (\bar{x}, \bar{y}) \in \text{Cont}(C_{x,y}) : \|\bar{x} - \hat{x}\| \leq \epsilon$ **then**
8: Separate $\hat{x}$ from $\text{Proj}_x \text{Cont}(C_{x,y}(w^k))$ with an inequality $\gamma x \leq \beta$;
9: Add inequality $\gamma x \leq \beta + M z_k$ to MASTER; ▷ Case 1
10: **end if**
11: **end for**
12: **for** $k \in \{1, \ldots, h\} : \hat{z}_k = 0$ and $C_{x,y}(w^k) \neq \varnothing$ **do**
13: **if** $\nexists (\bar{x}, \bar{y}) \in \text{Conv}(C_{x,y}) : \|\bar{x} - \hat{x}\| \leq \epsilon$ **then**
14: Separate $\hat{x}$ from $\text{Proj}_x \text{Conv}(C_{x,y}(w^k))$ with an inequality $\gamma x \leq \beta$;
15: Add inequality $\gamma x \leq \beta + M z_k$ to MASTER; ▷ Case 2
16: **end if**
17: **end for**
18: **if** no inequality has been added and $\hat{x}$ is not feasible for some active scenarios **then**
19: Reject $\hat{x}$ and branch; ▷ Case 3
20: **end if**
21: **if** $(\hat{x}, \hat{z})$ is still feasible **then**
22: update incumbent (lower bound).
23: **end if**
24: **until** there is no node to explore.

In addition to this, remember that a facet of a polytope of dimension n has dimension $n-1$ (i.e., can be written as the affine combination of at most n points).
Now, when solving the following problem:

$$\begin{array}{rrcl} \min & \|\hat{x} - \sum_{i=1}^{n+1} \lambda_i x^i\| & & \\ \text{s.t.:} & \sum_{i=1}^{n+1} \lambda_i & = & 1 \\ i = 1, \ldots, n+1 & \lambda_i & \geq & 0. \\ i = 1, \ldots, n+1 & x^i & \in & \text{Proj}_x C_{x,y}. \end{array} \tag{DIST}$$

we can distinguish two possible results:

1. the objective value is equal to zero, so $\hat{x}$ lies in $\text{Proj}_x \text{Conv}(C_{x,y})$ and, possibly, all λ_i are strictly positive ($\hat{x}$ is a convex combination of the points x_i);
2. the objective value is larger than zero, so $\hat{x}$ does not lie in $\text{Proj}_x \text{Conv}(C_{x,y})$ and at most n out of all λ_i can be strictly positive (they define a facet that separate $\hat{x}$);

Unfortunately, in model DIST both x^i and λ_i are sets of variables, so the formulation is clearly nonlinear and its solution would require an unjustified effort. For this reason, our purpose is providing a procedure that converge to the optimal solution of DIST in a finite number of iterations, within a certain tolerance. The basic idea is to initialize our procedure by finding the closest feasible point to $\hat{x}$, by solving the problem:

$$x^1 \leftarrow \arg_x \min_{(x,y) \in C_{x,y}} \|\hat{x} - x\|_x^2$$

If the distance between this point and $\hat{x}$ is lower than a fixed tolerance, then we consider feasible $\hat{x}$ and the procedure immediately stops without generating any cut. Otherwise, at each iteration, we aim to find a new point in $C_{x,y}$ that, convexly combined with all the previous points, decreases the distance from $\hat{x}$.

For this purpose, we define two new problems. Let j be the current iteration, the first problem has to minimize the distance from the convex combination of the current collection of points to $\hat{x}$. It is defined as follows:

$$\begin{array}{rrcl} \min & \|\hat{x} - \sum_{i=1}^{j} \lambda_i x^i\| & & \\ \text{s.t.:} & \sum_{i=1}^{j} \lambda_i & = & 1 \\ i = 1, \ldots, j & \lambda_i & \geq & 0. \end{array} \tag{MINDIST}$$

In this model only λ_i are variables, while all x_i are known. Once this problem has been solved, we can define $\bar{x} = \sum_{i=1}^{j} \lambda_i x^i$. If the distance between $\hat{x}$ and $\bar{x}$ (i.e., the distance between $\hat{x}$ and a point ling on $\text{Proj}_x \text{Conv}(C_{x,y})$) is lower than a fixed tolerance, say, ϵ, then we state that $\hat{x} \in \text{Proj}_x \text{Conv}(C_{x,y})$, and the procedure stops because we are falling in the third case.

The second problem must identify a descent direction from $\bar{x}$, that decreases the distance to $\hat{x}$. It is defined as follows:

$$\begin{array}{rrcl} \max & (\hat{x} - \bar{x})^T (x_j - \bar{x}) & & \\ \text{s.t.:} & (x_j, y) & \in & C_{x,y}. \end{array} \tag{NEWPOINT}$$

Here, x_j is the only variable. If the optimal solution value of this problem is larger than zero than the convex combination of the new point x_j with the previous ones will decrease the distance from $\hat{x}$. On the contrary, if the optimal solution value is equal to zero but the distance between $\hat{x}$ and $\bar{x}$ is still grater than ϵ, then we state

that $\hat{x} \notin \text{Proj}_x\text{Conv}(C_{x,y})$ and we found a facet to separate $\hat{x}$. It is defined by the hyperplane that is orthogonal to $\hat{x}$ in $\bar{x} = \sum_{i=1}^{j} \lambda_i x^i$.

The pseudo-code of this procedure is provided in algorithm 4.2.2.1.

Algorithm 4.2.2.1 minimizeDistance($\hat{x}, k, \epsilon, \zeta$)

Find x^1 as the integer point in $C_{x,y}$ closest to $\hat{x}$;
$d_1 \leftarrow \|\hat{x} - x^1\|$;
if $d_1 > \epsilon$ **then** ▷ $\hat{x}$ is not a feasible solution for scenario k;
  $\lambda_1 \leftarrow 1$;
  $j \leftarrow 1$;
  repeat
    Solve: $d_j \leftarrow$ MINDIST;
    Define: $\bar{x} \leftarrow \sum_{i=1}^{j} \lambda_i x_i$;
    $j++$;
    Solve NEWPOINT and store the new point x_j;
  until ($d_j \leq \epsilon$ **or** $d_{j-1} - d_j \leq \zeta$)
  if $d_j > \epsilon$ **then**
    Separate $\hat{x}$ from $\text{Proj}_x\text{Conv}(C_{x,y})$ with an inequality $\gamma x \leq \beta$;
    Add inequality $\gamma x \leq \beta + Mz_k$ to MASTER;
    return;
  end if
end if

Proposition 13 *Given a value of two values tolerance $\epsilon \geq 0$ and $\zeta \geq 0$, the value of d_j in Algorithm 4.2.2.1 converges to the optimal solution of DIST in a number of iterations that is finite for $\epsilon > 0$ and $\zeta > 0$, but it may be infinite for $\epsilon = 0$ or $\zeta = 0$.*

4.2.3 Case 3: separation when $\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y})$

When this case occurs, there is no valid cut that can be added. So, we have to reject the current solution, perform some type of branching and generate two new nodes where the current solution is no more feasible. We considered three different options.

Spatial branching on variable x

The first option is performing a spatial branching on $\hat{x}$ in the master. By choosing the right values of branching, one could be able to generated nodes where $\hat{x} \notin \text{Proj}_x\text{Conv}(C_{x,y})$ for at least one scenario and, so, it would be possible to separate the current infeasible solution in both the two new nodes by adding a new (local) cut. In addition to this, the size of the master problem does not change. The drawbacks of this option are all the difficulties of spatial branching, including choosing the appropriate variable for branching. To perform this kind of branching, one can use the information provided by the procedure 4.2.2.1 in order to define a branching that puts in different nodes the different points $x^1, \ldots, x^n$ that define the convex combination of $\hat{x}$. In the following we describe different strategies to select the x variable to which perform the spatial branching. All the strategies exclude from the selection the variables that in the current solution $\hat{x}$ assume the value of one of their bounds, since this would replicate the current node without affecting the solution, and, of course, we assume that the branching value is the value given by the infeasible solution $\hat{x}$. The different possibilities are:

- selecting a random variable;

- selecting the variable that split in the most balanced way the set of points obtained during the separation procedure;
- selecting a variable whose branching generates two nodes in which the distances from the current solution $\hat{x}$ and the nearest feasible point are different (i.e., the starting point of the separation procedure is different in the two nodes);
- selecting a variable whose branching generates two nodes in which the current solution $\hat{x}$ will be discarded by the generation of a cut (in the worst case, this implies to run the separation procedure twice for each variable).

Another possibility is selecting a branching hyperplane, instead of performing the branching on a single variable. This strategy has the drawback of increasing the size of the problem in the generated nodes, since it adds a constraint to the original model, but it could allow to always generate two nodes in which the current solution is discarded by the generation of a cut.

Intern branching on variables y

Another alternative is performing the branching on the integer variables of the scenario subproblem. Because $\hat{x} \in \text{Proj}_x\text{Conv}(C_{x,y})$, we could try to split the set $C_{x,y}$ into two sets, and create two scenarios $C^1_{x,y}, C^2_{x,y}$ to replace the scenario $C_{x,y}$. In the master, there would be two new variables z_{c1}, z_{c2} to indicate whether we are satisfying the constraints for $C^1_{x,y}, C^2_{x,y}$ with the usual convention $z_i = 0 \Rightarrow x \in C^i_{x,y}$, together with the constraint $z_{C1} + z_{C2} \geq 1$ to ensure that the solution belongs to *at most* one among $C^1_{x,y}, C^2_{x,y}$. It is not hard to see that if the scenario has a single binary variable y, this approach works in just one branching step and at most two variables have to be added in the master. But, if both the number of variables y and the values that they can assume increase, then the size of the master problem could become intractable. The drawback, in general, is that this option somehow brings the variables of the subproblems at the level of the master.

Extern branching on variables y

The last possibility is similar to the previous one in the sense that also in this case we use variables y to branch the problem. But, in this case, we handle the branching externally from the solver: whenever case 3 occurs, we build and solve two new instances of the problem. In these instances, that are the nodes of our B&B, we force the selected integer variable y to assume the corresponding feasible values defined by the branching, in addition to all the valid cuts inherited from the master. In this way, all the single addressed problem have an affordable size. The drawback, of course, is the number of the instances that we might have to solve.

4.3. Computational enhancements

In section 4.2 we described the procedure to generate valid cuts for the master problem by checking the feasibility of an integer solution in those scenarios that must be satisfied (i. e., the corresponding z variable is equal to 0 in the current solution).

We specify that explicitly considering the case 1 (Section 4.2.1), in which the solution coming from the master does not belong to the continuous relaxation of the active scenario subproblem, is not necessary for the correctness of the decomposition

algorithm. Nevertheless, we decided to include the generation of those inequalities, since it requires an effort that is almost negligible in term of computational time.

In the following we describe other computational enhancements that can be applied to our algorithm in order to improve its performances.

4.3.1 Cutting strategies

It is worth underling that any cut can be globally added when it is generated in a node of the branching tree whose feasible region has not been affected by spatial branching; while, in the other case, the cut is locally valid and it could be possibly added without any big M coefficient if the the corresponding z variable have been already fixed to 0 by the branching.

In addition to this, one can also evaluate the feasibility of the solution in relation to the intersection of two ore more scenarios, instead of considering only one scenario a time.

On the basis of these observations, we can state that:

- the cuts we add, before spatial branching has been performed, can be globally added to the master problem in the form of $Big - M$ constraints whose activation depends on the sum of the z variables of the involved scenarios;
- the cuts we add, after the spatial branching has been performed, must be locally added to the master problem in the form of $Big - M$ constraints whose activation depends on the sum of the z variables of the involved scenarios.

Observe that in the first case, since we are adding a global cut, we must always include in the constraints all the z variables of the involved scenarios; in the second case, instead, we can possibly exclude from the constraints the z variables that have been already fixed to 0 by the branching (they would be redundant).

This analysis leads to a trade-off in the strategy used to generate cuts: intersecting more scenarios allows to generate stronger cuts, but the problems to be solved are obviously harder, since the number of variables increases, and, in addition to this, these cuts will be valid only when all the z variables of the considered scenarios are equal to 0. In [74], the authors propose different techniques to group different scenarios in order to obtain a stronger lower bound for the CCP. The same grouping techniques may be applied to our solving procedure in order to obtain stronger cuts while taking under control the size of the problems to be solved during the separation procedure.

4.3.2 Storing feasible points

Algorithm 4.2.2.1 aims to determine whether a solution $\hat{x}$ belongs to the projection of the Convex Hull of the feasible region of the problem. The way in which the procedure answers this question is finding, when possible, a collection of feasible points whose convex combination includes $\hat{x}$. So, it is possible to speed up the procedure by storing lists of points that are associated to each scenario, or to each group of scenarios.

During the Branch-and-Cut execution, these lists must be updated any time a spatial branching occurs (points that are no more feasible must be excluded); and, if we consider groups of scenarios, when a scenario within a group becomes active, the corresponding list also must be updated, since the new feasible region would be smaller.

Due to these reasons, the size of the lists must be calibrated: a large number of stored points allows to reduce the number of iteration required by algorithm 4.2.2.1, but it implies a longer time to scan the list of points any time it must be updated.

4.3.3 Solving deterministic problems

Our decomposition algorithm is based on a Branch-and-Cut framework that can be implemented within a commercial integer programming solver such as IBM-ILOG CPLEX. When this is the case, the branching decisions are usually entrusted to the solver that is in charge of handling the generation of new nodes. But, when an integer solution is found by the master problem and we want to determine its feasibility, we can always query the local bounds of the integer variables. In the case at hand, if all the indicator variables z have been already fixed to some integer value, then the problem reduces to be deterministic and there is no reason to run the separation procedure. So, anytime this situation occurs, we directly solve the deterministic formulation of the problem and we provide the solver with the optimal solution we find. This allows us to immediately close the node.

4.4. Computational experiments

At the time of writing this thesis, our research on the topic is still in progress. For this reason, the computational results we will present in this section are related to preliminary experiments that have the main scope of understanding the potentialities of the proposed algorithm. Specifically, our main scope in this phase of the research is analyzing the performances of the separation procedure presented in Section 4.2.2 where we discussed the case in which an infeasible solution produced by the master problem does not belong to the convex hull of the active scenario subproblems, but it is inside their continuous relaxation. So, the results we will present are related to tests in which we did not perform any kind of branching when the third case (Section 4.2.3) occurred (i.e., we stop when a solution belonging to the convex hull of the scenario subproblems is found).

4.4.1 Computational Environment

All the reported experiments are performed on a computer equipped with an i7 processor clocked at 3.20 GHz and 64 GB RAM under Linux operating system. We use the `CPLEX 12.7.1` MIP framework to implement our Branch-and-Cut algorithm. `CPLEX` is run in single-threaded mode and all `CPLEX` parameters are set to their default values.

4.4.2 Implementation details and test instances

In order to test our algorithm, we defined two different probabilistic resource planning problems that can be both represented by a compact formulation. Both the problems use the same data consisting of a set of resources (e.g., server types), denoted by $i \in I := \{1, \dots, n\}$, a set of customer type, denoted by $j \in J := \{1, \dots, m\}$, and a set of scenarios, denoted by $k \in K := \{1, \dots, h\}$. The input parameters are:

- the unit cost c_i of each resource i;
- the demand d_{jk} of each customer type j and scenario k;

- the service rate μ_{ij} of resource i for customer type j.

The objective of the first problem is to minimize the total cost of the resources in a way such that the allocation does not exceed the available resource levels and is sufficient to meet customer demands. The variables of the second stage are general integer. The corresponding model reads as follows:

$$
\begin{array}{lrcll}
\min & \sum_{i\in I} c_i x_i & & & \\
\text{s.t.:} & \sum_{j\in J} y_{ijk} & \leq & f_i(x_i) & i=1\dots n, \quad k=1\dots h \\
& \sum_{i\in I} \mu_{ij} y_{ijk} & \geq & d_{jk}(1-z_k) & j=1\dots m, \quad k=1\dots h \\
& \sum_{k\in K} p_k z_k & \leq & \alpha & \\
& y_{ijk} & \in & \mathbb{N} \qquad i=1\dots n, \quad j=1\dots m, & k=1\dots h \\
& z_k & \in & \{0,1\} & k=1\dots h
\end{array}
\tag{FIRST}
$$

The first set of constraints ensure that the demand allocation does not exceed the resource availability; the second set of constraints impose that the demand of the active scenarios must be satisfied; while the last constraint is the chance constraint.

Also in the second problem, the objective is to minimize the total cost of the resources, but each demand must be satisfied by one resource only, without exceeding the available resource level. In this case, we define the variables of the second stage as binary. The corresponding model reads as follows:

$$
\begin{array}{lrcll}
\min & \sum_{i\in I} c_i x_i & & & \\
\text{s.t.:} & \sum_{i\in I} y_{ijk} & = & 1-z_k & j=1\dots m, \quad k=1\dots h \\
& \sum_{j\in J} \mu_{ij} d_{jk} y_{ijk} & \leq & f_i(x_i) & i=1\dots n, \quad k=1\dots h \\
& \sum_{k\in K} p_k z_k & \leq & \alpha & \\
& y_{ijk} & \in & \{0,1\} \quad i=1\dots n, \quad j=1\dots m, & k=1\dots h \\
& z_k & \in & \{0,1\} & k=1\dots h
\end{array}
\tag{SECOND}
$$

The first set of constraints impose that each customer type in each active scenario has one associated resource; the second set of constraints ensure that the resources are sufficient to satisfy the demands; the last constraint is the chance constraint.

In [72], the author also considers a chance constrained formulation of a resource planning problem inspired by a call center staffing application and the instances he used are available on-line ([75]). We adapted these instances to the problems we consider: in particular, we divided each demand by 10 and we rounded them down and we substituted each non-zero service rate with a random value between 1 and 10. In both formulations, we define $f_i(x_i)$ as a generic convex function, but the results we present in this section are all referred to the linear case in which $f_i(x_i)$ is equal to x_i. We only considered instances with 20 resources ($n = 20$) and 30 customer types ($m = 30$). We considered two sizes for the scenario set ($h \in \{10, 20\}$) and two values for the parameter defining the risk level ($\alpha \in \{0.1, 0.2\}$). For each possible

combination of h and α, we considered 5 different instances. This way, we generated a test-bed of 20 instances for each problem.

4.4.3 Results

In this section we compare the results obtained with the following solution approaches:

- `CONT` applies the separation procedure only for infeasible solutions that do not belong to the continuous relaxation of the scenario subproblems (see Section 4.2.1). Basically, it corresponds to the algorithm proposed in [3].
- `CONV1` and `CONV2` represent the focus of our analysis. Both these approaches include the previous one but they also separate the infeasible solutions that do not belong to the convex hull of the scenario subproblems (see Section 4.2.2). When processing a solution coming from the master, `CONV1` checks the feasibility for one scenario a time, while `CONV2` considers groups of 2 scenarios, defined before the algorithm starts.
- `OPT` uses IBM-ILOG CPLEX 12.7.1 to directly solve the compact formulation of the problems. So, this is the only approach that always finds the optimal solution value.

Since we are considering minimization problems, `CONT`, `CONV1` and `CONV2` provide a lower bound for `OPT`.

Table 4.1 collects the objective values obtained when solving problem FIRST with the four different approaches described above. The first column reports the number of scenarios, the second column reports the different values of the risk level, the third column reports the instance number, columns from the forth to the last one collect the objective function values obtained by `CONT`, `CONV1`, `CONV2` and `OPT`, respectively. Table 4.2 reports the same information for problem SECOND.

We remember that the objective values reported in columns `CONV1` and `CONV2` may correspond to infeasible solution belonging to the convex hull. So, these objective values can be considered optimal only if the same objective value also arises from a node in which the deterministic problem has been solved (see 4.3.3). In both the tables, we mark these cases with an asterisk (*). In addition, we underline the cases in which `CONV1` or `CONV2` do not improve the bound provided by `CONT`.

From table 4.1, we observe that `CONV2` always finds the optimal solution value in all the considered instances, while `CONV1` fails in only one case. Out of 20 instances, algorithm `CONV1` proves the optimality of 16 instances, while `CONV2` proves the optimality of 19 instances. The situation changes a lot when considering problem SECOND. From table 4.2, we observe that, out of 20 instances, `CONV1` finds the optimal solution value in 5 cases, while `CONV2` does it in 7 cases. In four cases, `CONV1` does not increase the bound provided by `CONT` while this circumstance occurs only once for `CONV2`. In addition to this, `CONV2` finds an objective value that is larger than the one found by `CONV1` in almost all the cases (14 out of 20).

We do not report and discuss any computational times, since our algorithm has not been optimized, yet.

TABLE 4.1: Comparison of the objective values obtained with different solution approaches (problem FIRST).

h	α		CONT	CONV1	CONV2	OPT
10	0.1	0	34.28	*45.70	*45.70	45.70
		1	33.69	*43.84	*43.84	43.84
		2	33.49	*43.49	*43.49	43.49
		3	38.89	*48.65	*48.65	48.65
		4	34.74	*44.67	*44.67	44.67
	0.2	0	34.24	*45.15	*45.15	45.15
		1	33.46	*42.74	*42.74	42.74
		2	32.20	*42.68	*42.68	42.68
		3	37.88	*45.43	*45.43	45.43
		4	33.60	41.68	*41.69	41.69
20	0.1	0	34.29	*45.70	*45.70	45.70
		1	34.30	*45.62	45.62	45.62
		2	35.31	45.40	*45.40	45.40
		3	39.42	*49.55	*49.55	49.55
		4	34.16	*43.01	*43.01	43.01
	0.2	0	34.18	*44.43	*44.43	44.43
		1	33.68	*43.92	*43.92	43.92
		2	33.85	*43.50	*43.50	43.50
		3	38.89	48.65	*48.65	48.65
		4	33.60	41.85	*41.85	41.85

TABLE 4.2: Comparison of the objective values obtained with different solution approaches (problem SECOND).

h	α		CONT	CONV1	CONV2	OPT
10	0.1	0	443.46	*464.67	*464.67	464.67
		1	439.84	443.88	451.77	460.74
		2	416.66	*424.47	*424.47	424.47
		3	492.34	499.73	500.45	501.95
		4	458.71	466.83	*468.78	468.78
	0.2	0	438.82	443.46	451.33	453.82
		1	435.48	440.12	443.15	455.32
		2	407.18	407.18	407.18	414.34
		3	479.57	*485.10	*485.10	485.10
		4	435.36	435.39	439.04	450.96
20	0.1	0	443.46	*471.59	*471.59	471.59
		1	444.48	445.40	453.34	470.77
		2	441.91	*456.08	*456.08	456.08
		3	502.78	516.97	*522.84	522.84
		4	448.86	460.65	464.54	474.09
	0.2	0	433.60	433.60	441.65	457.42
		1	437.06	439.84	444.58	462.72
		2	424.47	424.47	431.99	437.33
		3	492.38	496.57	502.53	513.26
		4	443.04	443.04	449.52	461.06

4.5. Conclusions and future work

In this chapter, we addressed the Chance Constrained Problem and we considered the case in which the second-stage variables of its MINLP reformulation are integer. We proposed a decomposition approach based on a Branch-and-Cut framework where we generate cutting planes as outer approximation point cuts, when possible. The difficulty of the problem at hand arises from the fact that the feasible regions of the scenario subproblems, even if described by convex functions, result to be non convex, due to the integrality constraint of the second-stage variables. We proposed a convergent procedure to generate cuts that are valid for the convex hull of these sets, while the case in which the solution produced by the master problem is infeasible but belongs to the convex hull of the scenario subproblems is still an open point.

Our preliminary computational experiments showed that the bounds provided by the generation of the cuts on the convex hull can be equal to the optimal solution value in some cases, but, in other cases, the gap can be probably closed only by performing some kind of branching, as discussed in Section 4.2.3. So, our future research will move on this direction.

Chapter 5

Integer Optimization with Penalized Fractional Values: The Knapsack Case

[1]

Integer Programming and combinatorial optimization problems require to determine the optimal values for a set of variables, each having a discrete domain. In many cases, variables enforce boolean conditions, and it is quite natural to resort to binary variables. Just to mention a few examples, in the knapsack problem one has to decide whether to insert an item in the knapsack or not. Similarly, in scheduling applications, one is asked to decide if a job should be scheduled on a given machine. Finally, the vehicle routing problem asks to decide if a certain customer must be included in a certain route and if a given edge has to be used in the solution or not. The explosion of new algorithms for binary problems in the last 30 years is motivated by the amount and relevance of applications that can be tackled with these models. It turns out that in many relevant real-world problems decisions can also be taken at a fractional level, and thus decision variables can attain non-integer real values. However, this additional freedom of "splitting" an integer variable and selecting only a fractional part will likely incur additional costs, i.e. "penalties" for deviating from integrality, thus worsening the solution value.

For example, in preemptive scheduling (see, Pinedo [77]), each task may be processed in different phases, until it has finished its execution, to minimize the total makespan. In the Split Delivery Vehicle Routing Problem (see, Archetti and Speranza [78]), the restriction that each customer has to be visited exactly once is removed, i.e., each customer can be served by more than one route, possibly improving the objective function. In most of the cases addressed in the literature, splitting an *item* either produces no additional cost or gives a constant penalty; e.g., Malaguti et al. [79] considered a two-dimensional cutting problem in which raw material has to be cut to produce items, and each cut introduces some constant trim loss. In some applications, the deterioration of the solution induced by splitting cannot be evaluated *a priori*, hence some approximation has to be used; e.g., Lodi et al. [80] considered an applications arising in Mobile WiMAX in which data (items) have to be sent from a base station to users using a unique channel (the knapsack). In this system model, a part of the channel is used to allocate additional information about the packets that are sent. Splitting an item is allowed, but it increases the amount of

[1] The results of this chapter appears in: E. Malaguti, M. Monaci, P. Paronuzzi, and U. Pferschy, "Integer optimization with penalized fractional values: The Knapsack case", *European Journal of Operational Research*, 273(3), 874-888, 2019. [76]

additional information to be sent, i.e., it reduces the available capacity. As the objective is to minimize the amount of overhead while transmitting all data, the problem was formulated to minimize the number of items that are fractioned.

In this chapter we make a step further in the study of integer problems in which splitting is allowed by removing the assumption that the penalty induced by splitting is a constant[2]. In particular, we allow the penalty to be described by an arbitrary function that depends on the fraction of item that is taken, and apply this setting to the simplest combinatorial optimization problem, namely to the 0-1 Knapsack Problem (KP) (cf. Martello and Toth [82] and Kellerer et al. [83]).

In KP we are given a knapsack of capacity W and a set $N = \{1, \dots, n\}$ of items, each item $j \in N$ having a positive weight $w_j \leq W$ and a positive profit p_j. The problem asks for selecting a subset of items with maximum profit whose total weight does not exceed the knapsack capacity. We assume that $\sum_{j \in N} w_j > W$ since otherwise the problem is trivial. As items cannot be split, KP can be modeled by associating, with each item j, a binary variable x_j taking value 1 if the item is selected, and 0 otherwise. Hence, the profit for each item is expressed as $p_j x_j$, and the capacity it consumes is $w_j x_j$. In the *Fractional Knapsack Problem with Penalties* (FKPP) addressed in this chapter, fractions of items are allowed to be selected, but whenever an item is split, a penalty is incurred. Thus, the net profit associated with a fraction $0 < x_j < 1$ of an item j is smaller than (or equal to) $p_j x_j$, while no profit is earned when the item is not selected, and the full item profit p_j is earned for $x_j = 1$. Formally, FKPP is defined as follows: Given a knapsack problem KP as defined above, for each item j there is a function $F_j : [0,1] \to \Re$ such that $F_j(x_j)$ represents the profit earned if item j is taken at some (possibly, fractional) value $x_j \in [0,1]$. We assume that each function $F_j(\cdot)$ has the following shape:

$$F_j(x_j) = \begin{cases} 0 & \text{if } x_j = 0 \\ p_j & \text{if } x_j = 1 \\ p_j x_j - f_j(x_j) & \text{otherwise} \end{cases} \qquad (j \in N) \tag{5.1}$$

where $f_j(x_j) \geq 0$ for $x_j \in [0,1]$ is an arbitrary (even discontinuous) function representing the penalty incurred in case item j is taken at a fractional level x_j. Observe that we allow $f_j(0) > 0$ and/or $f_j(1) > 0$ for some j, as these values are irrelevant for the definition of function $F_j(\cdot)$. In the general case we will not impose any further restrictions on $f_j(\cdot)$ except that function values can be computed in constant time. Thus, FKPP can be formulated as

$$\max \left\{ \sum_{j=1}^{n} F_j(x_j) : \sum_{j=1}^{n} w_j x_j \leq W, \;\; 0 \leq x_j \leq 1 \quad (j \in N) \right\}$$

where each item j has associated a continuous variable x_j indicating the fraction of item j that is selected, and functions $F_j(\cdot)$ are defined according to (5.1). It is easy to see that FKPP is an NP-hard problem since it contains KP as a special case arising for $f_j(x_j) = p_j \; \forall x_j \in (0,1)$, i.e. no item will be split.

A special case of FKPP is given by the *Continuous Knapsack Problem* (CKP), that is the relaxation of KP obtained removing the integrality requirement of the variables. In this case, variables x_j have the same meaning as in FKPP, and both the earned profit and the used capacity are proportional to x_j. Thus, CKP arises when $f_j(x_j) = 0 \; \forall x_j \in [0,1]$ and for each item j. It is well known that this relaxation of KP

[2]Other penalty functions appeared as auxiliary subproblems in Freling et al. [81] and Casazza and Ceselli [4].

can be solved in polynomial time by ordering items according to a non decreasing profit over weight ratio, and inserting them into the knapsack in this order. The first item that cannot be completely inserted in the knapsack (if any), the so-called *critical element* (also known as split or break item), is fractionally inserted in the knapsack, so as to saturate its capacity, and the corresponding fraction of profit is earned.

Literature review. The classic KP is weakly NP-hard, and in practice fairly large instances can be solved to optimality with moderate computational effort. The reader is referred to the books by Martello and Toth [82] and by Kellerer et al. [83] for comprehensive discussion on algorithms, applications and variants of the problem. Despite the wide existing literature on knapsack problems, only few contributions can be found that explicitly take penalties into account.

Freling et al. [81] considered a reformulation of the Multiperiod Single-Sourcing Problem as a Generalized Assignment Problem and noticed that the pricing problem is a knapsack problem with penalties. In this problem, that they called Penalized Knapsack Problem, the objective function includes a penalty that is described by a convex function depending on the total amount of capacity that is used. Observing that the resulting objective function is concave, the authors analyzed the structure of an optimal solution to the continuous relaxation of the problem. They concluded that this structure is similar to that of an optimal solution to the CKP, and proposed a solution algorithm. Ceselli and Righini [84] considered another version of the Penalized Knapsack Problem in which each item has associated a constant penalty and the objective function is given by the total profit minus the largest penalty among the selected items. This problem was extensively studied in Della Croce et al. [85].

Another related problem is the Bin Packing Problem with Item Fragmentation (BPPIF), that was introduced by Mandal et al. [86] to model an application arising in VLSI (Very Large Scale Integration) circuit design. In this problem, one is asked to pack a given set of items in a fixed number of identical bins, while minimizing the number of items that are split among different bins. Casazza and Ceselli [4] formulated BPPIF using a mathematical model with an exponential number of variables, requiring the definition of column generation techniques. It turns out that the pricing problem in this formulation is a special case of FKPP, where the penalty for selecting each item j at a fractional level x_j is defined as a linear function $f_j(x_j) = k_j(1 - x_j)$ for each $j \in N$ and $x_j \in (0, 1)$. Casazza and Ceselli [87] introduced mathematical models and algorithms for many variants of BPPIF. Recently, Byholm and Porres [88] noticed that BPPIF arises in the operation of file systems, and presented approximation and metaheuristic algorithms for its solution. As mentioned, in all these papers there is a constant penalty associated with the splitting of an item for bin packing.

FKPP is also related to the general nonlinear knapsack problem. Bretthauer and Shetty [89] presented a survey concerning algorithms and applications of this problem, and analyzed the general form of the problem and of its most common variants: continuous or integer variables, convex or nonconvex functions, separable or nonseparable functions, bounds on the variables, or generalized upper bound (GUB) constraints. None of these variants, however, can be used to model FKPP.

Paper contribution. To the best of our knowledge, this is the first paper that specifically addresses FKPP in its general settings. Our contributions can be summarized as follows:

1. General structural properties, the special case where all profit functions F_j are convex, and the analogous case where penalties occur as additional weights (instead of profit reduction) are introduced in Section 5.1.

2. In Section 5.2 we propose two different mathematical models and discuss the relation between the two models. The first model has a linear number of variables and constraints, but a non-linear objective function. The second model restricts the weight contribution of each variable to integers and resembles a Multiple-Choice Knapsack Problem (MCKP), albeit it requires a number of variables that is pseudopolynomial in the input size. Moreover, we construct a Fully Polynomial Time Approximation Scheme (FPTAS) for the problem in its general form. Nevertheless, the presence of fractional variables prevents to derive an FPTAS by simple adaptations of similar algorithms for MCKP, and requires a suitable treatment of the items profits.

3. From an algorithmic point of view we first report in Section 5.4 the dynamic program recently proposed by Ceselli and Casazza [4] for the optimal solution for the special case in which all profit functions are convex. Then an improved algorithm with lower computational complexity is presented which partitions the dynamic programming iterations into phases, each addressing a subproblem of suitable size. Finally, Section 5.5 presents some fast and simple heuristic algorithms for the approximate solution of FKPP.

4. Section 5.6 reports the outcome of an extensive computational study on the performance of the proposed models and algorithms. To this end we developed benchmark instances derived from the KP literature using different shapes of the penalty functions. It turns out that our newly developed, improved dynamic programming scheme delivers the best performance among all solution approaches for the instances of the convex case.

5.1. Structural results

In this section we impose some natural assumptions on the input data, and describe some properties of any optimal FKPP solution that will be used in the next section for modelling and solving the problem. Furthermore, we will introduce the relevant special case where all $F_j(\cdot)$ are convex.

Assumption 1 *All item weights w_j and the capacity value W are integers.*

This assumption is usually imposed for knapsack-type problems and holds without loss of generality. If there are rational weight coefficients, one can multiply all weights and the capacity by the smallest common multiple of the denominators and thus reach integer values.

Proposition 14 *For each item $j \in N$ we can replace $F_j(\cdot)$ with $\widetilde{F}_j(\cdot)$ in the objective function of FKPP, where $\widetilde{F}_j(x_j) = \max\{F_j(y) : y \in [0, x_j]\} \quad \forall x_j \in [0, 1]$.*

Proof. Let us denote by $I = (n, W, (F_j), (w_j))$ an instance of FKPP. We define another FKPP instance, say, $\widetilde{I} = (\widetilde{n}, \widetilde{W}, (\widetilde{F}_j), (\widetilde{w}_j))$ with $\widetilde{n} = n$ items, capacity $\widetilde{W} = W$ and, for each item $j = 1, \ldots, n$, weight $\widetilde{w}_j = w_j$ and profit function $\widetilde{F}_j(x_j) = \max_{y \le x_j}\{F_j(y)\}$.

Since I and $\widetilde{I}$ differ for the objective function only, a solution x is feasible for I if and only if it is feasible for $\widetilde{I}$. In addition, by definition of $\widetilde{F}_j(\cdot)$, we have $\widetilde{F}_j(x_j) \geq F_j(x_j)\ \forall j$, i.e., moving from I to $\widetilde{I}$ cannot reduce the value of solution x.

We now prove that, given an optimal solution $\widetilde{x}$ (say) for $\widetilde{I}$, it is possible to construct a solution x that has the same value in I. For each item j, set

$$x_j := \min\{y : F_j(y) = \widetilde{F}_j(\widetilde{x}_j)\}. \tag{5.2}$$

Note that, by definition of $\widetilde{F}_j(\cdot)$, we have $x_j \leq \widetilde{x}_j$ for each item j, i.e., x is a feasible solution for instance I. In addition (5.2) ensures that $F_j(x_j) = \widetilde{F}_j(\widetilde{x}_j)\ \forall j$, i.e., solutions x and $\widetilde{x}$ have the same value according to profit functions $F_j(\cdot)$ and $\widetilde{F}_j(\cdot)$, respectively. This implies that x is indeed optimal for I. □

It must be noted that the computation of $\widetilde{F}_j(\cdot)$ is not necessarily possible in polynomial time but depends on the properties of $F_j(\cdot)$. Thus, also mapping an optimal solution for $\widetilde{F}_j(\cdot)$ back into a solution for $F_j(\cdot)$, as required by (5.2), may be impossible in polynomial time. However, bearing these computational caveats in mind, the proposition justifies our second assumption:

Assumption 2 *For each item $j \in N$, function $F_j(\cdot)$ is non-decreasing in $[0, 1]$.*

We now observe that, while an optimal solution to CKP includes at most one fractional item (namely, the critical item), this is not the case for FKPP. Indeed, there are generic FKPP instances with n items for which the optimal solution splits *all* items.

Proposition 15 *There are instances of FKPP for arbitrary n, where all n items are split in the optimal solution.*

Proof. Consider the following instance with n items, knapsack capacity $W = M$ for some large value M, and all items identical with $p_j = w_j = M - 1$ for $j = 1, \ldots, n$. The profit function $F_j(x_j) = F(x_j) = (M-1)x_j - f(x_j)$ for $j = 1, \ldots, n$ is defined as the following piece-wise linear (non-convex) function (see Figure 5.1):

$$F(x_j) = \begin{cases} 0 & \text{if } 0 \leq x_j \leq \frac{2}{M-1} \\ \frac{M}{M-2n}(M-1)x_j - \frac{2M}{M-2n} & \text{if } \frac{2}{M-1} < x_j \leq \frac{1}{n}\frac{M}{M-1} \\ \frac{M}{n} & \text{if } \frac{1}{n}\frac{M}{M-1} < x_j \leq \frac{M-2}{M-1} \\ (M-1)(M-1-\frac{M}{n})x_j - (M-1)(M-2-\frac{M}{n}) & \text{if } \frac{M-2}{M-1} < x_j \leq 1 \end{cases}$$

Choosing an item j with $x_j \geq \frac{M-2}{M-1}$ leaves a residual capacity ≤ 2 which could only be filled by items i with $x_i \leq \frac{2}{M-1}$, but then these items i would contribute zero profit and the resulting solution has a total profit at most $p_j x_j \leq M - 1$.

For choosing an item j with $x_j < \frac{M-2}{M-1}$ it is always better to set x_j to the lower end of the interval where profit is constant, i.e. $x_j = \frac{1}{n}\frac{M}{M-1}$. Taking this fractional choice for all n items yields an optimal solution value of M (and total weight M). Choosing a value even smaller than $\frac{1}{n}\frac{M}{M-1}$ for some items does not offer any opportunities for improving the contribution of other items by increasing their values, since an increase of profit starts only for item values strictly greater than $\frac{M-2}{M-1}$, which leads us back to the case we settled at the beginning of the proof. □

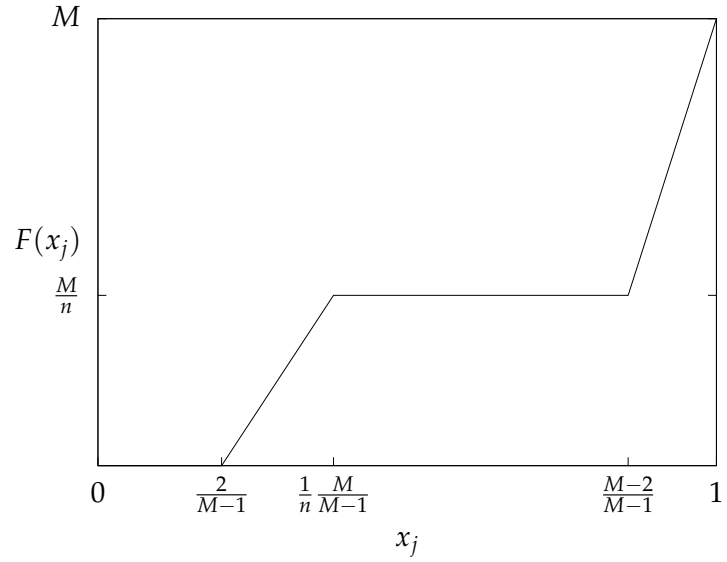


FIGURE 5.1: Example of a generic profit function that leads to an optimal solution with all fractional items.

5.1.1 FKPP with convex objective function

For several applications, it is reasonable to assume that the penalty functions $f_j(\cdot)$ are concave. This means that taking a smaller part of an item and selecting it (or its complement) will incur a rather small penalty, as it may correspond to a minor trimming operation. However, splitting an item closer to the middle, and thus deviating more significantly from integrality, requires a more involved effort and causes a higher penalty. Hence, we will consider the special case where all functions $f_j(\cdot)$ are *concave* and thus all profit functions $F_j(\cdot)$ are *convex*. Clearly, this case also includes the most plausible cases of linear and constant penalty functions. Note that the special case of FKPP resulting from BPPIF (Ceselli and Casazza [4]) contains linear penalty functions and thus falls into this category of convex profit functions.

We now show that for this case an optimal solution exists in which at most one item is taken at a fractional level. This fact has pleasing modeling and algorithmic consequences, discussed next.

Proposition 16 *If all profit functions $F_j(\cdot)$ are convex, there exists an optimal solution for FKPP that has at most one fractional item.*

Proof. Let x^* be an optimal solution for FKPP that includes two items, say h and k, at a fractional level. We will show that x^* cannot be unique. Let $\varepsilon > 0$ be a positive value and define $\varepsilon_h = \varepsilon / w_h$ and $\varepsilon_k = \varepsilon / w_k$. Now consider two solutions y and z as follows:

$$y_j = \begin{cases} x_j^* & \text{if } j \neq h,k \\ x_h^* + \varepsilon_h & \text{if } j = h \\ x_k^* - \varepsilon_k & \text{if } j = k \end{cases} \qquad (j \in N)$$

and

$$z_j = \begin{cases} x_j^* & \text{if } j \neq h,k \\ x_h^* - \varepsilon_h & \text{if } j = h \\ x_k^* + \varepsilon_k & \text{if } j = k \end{cases} \qquad (j \in N)$$

Given their definition, these solutions satisfy the capacity constraint, as ε units of capacity are moved from h to k or vice-versa. In addition, as both x_h^* and x_k^* are strictly positive and smaller than 1, solutions y and z are feasible if ε is chosen such that $\varepsilon_h \leq \min\{1 - x_h^*, x_h^*\}$ and $\varepsilon_k \leq \min\{1 - x_k^*, x_k^*\}$. Finally, $\varepsilon > 0$ implies $\varepsilon_h > 0$ and $\varepsilon_k > 0$, i.e., y and z are distinct from x^* and $x^* = \frac{1}{2}y + \frac{1}{2}z$. As all profit functions are convex we have that:

$$\sum_{j \in N} F_j(x_j^*) \leq \frac{1}{2} \sum_{j \in N} F_j(y_j) + \frac{1}{2} \sum_{j \in N} F_j(z_j)$$

implying that:

$$\sum_{j \in N} F_j(x_j^*) \leq \max \left\{ \sum_{j \in N} F_j(y_j), \sum_{j \in N} F_j(z_j) \right\}$$

i.e., at least one between y and z yields a profit not smaller than that of x^*. Thus, another optimal solution may be defined either increasing h and decreasing k or decreasing k and increasing h, until one of the two variables hits either the lower or the upper bound. □

Since we assumed the knapsack capacity W and all the item weights w_j ($j = 1, \ldots, n$) to be integer, the following result is a direct consequence of Proposition 16.

Proposition 17 *If all profit functions $F_j(\cdot)$ are convex, there exists an optimal solution to FKPP where $w_j x_j$ is integer for each item j.*

Proof. Proposition 16 ensures that an optimal FKPP solution, say x^*, exists in which at most one item is split. Let k be such item. Since we assume all weights and the capacity be integer, all items $j \neq k$ have an integer value of $w_j x_j^*$, and the residual capacity for item k is an integer number. As function $F_k(\cdot)$ is non-decreasing, there is an optimal solution where item k is taken at the largest possible value, which is indeed an integer. □

We conclude this section observing a relevant property of function $F_j(\cdot)$ ($j \in N$), in case it is convex. Indeed, under convexity hypothesis, $F_j(0) = 0$ implies that there must exist a value $x'_j \in [0, 1)$ such that $F_j(x) \leq 0$ for $x \in [0, x'_j]$ and $F_j(x)$ is strictly increasing for $x \in [x'_j, 1]$. Thus, the profit function $\widetilde{F}(\cdot)$ introduced in Proposition 14 is defined as $\widetilde{F}_j(x) = F_j(x)$ for $x \geq x'_j$ and $\widetilde{F}_j(x) = 0$ for $x < x'_j$. This means that, if we use function $\widetilde{F}_j(\cdot)$ instead of $F_j(\cdot)$, an optimal solution $\widetilde{x}_j$ obtained for the latter can be mapped to $x_j = 0$ if $F_j(\widetilde{x}_j) \leq 0$ and to $x_j = \widetilde{x}_j$ otherwise. In other words, there is no need to explicitly compute x'_j, and, given an optimal solution for $\widetilde{F}_j(\cdot)$, one can compute variables x_j according to (5.2) in linear time.

5.1.2 Penalty in terms of weight

We have defined FKPP by considering a penalty in the profit of fractional items. It seems natural to also consider the case in which the penalty for fractional items is enforced in terms of additional weight, instead of reduced profit. In other words, we can consider that a fraction x_j of a given item j produces a profit $p_j x_j$, but consumes a capacity $G_j(x_j) \geq w_j x_j$, where $G_j(x_j)$ is a (possibly discontinuous) function defining the weight of fraction x_j, once the penalty is considered. Define, $G_j(0) = 0$ and $G_j(1) = w_j$, for all $j \in N$. In this section we show that the fractional knapsack

problem with weight penalty can be reduced to FKPP for a suitable penalty function. To this aim we introduce, for each item j, set

$$S_j = \{x_j \in [0,1] : \nexists y > x_j,\ G_j(y) \leq G_j(x_j)\} \tag{5.3}$$

that contains all those points that are not dominated, i.e., for which no point exists that provides a larger profit and consumes the same capacity (or less). In the following, we show that, for each $j \in N$, only points $x_j \in S_j$ are relevant in the definition of the weight function $G_j(\cdot)$.

Proposition 18 *There exists an optimal solution x of the fractional knapsack with penalty in terms of weight such that $x_j \in S_j$ for each item $j \in N$.*

Proof. Let x be an optimal solution of the problem. By contradiction, assume there exists an item j (say) such that $x_j \notin S_j$. This means that there exists y such that $G_j(y) = G_j(x_j)$ and $y > x_j$. Setting $x_j = y$ does not increase the amount of capacity required by item j, while it increases the objective function value; this contraditcs the assumption that x is an optimal solution. □

Based on this proposition, for each item j we can define a reduced weight function whose domain is S_j; it turns out that this reduced function is strictly increasing on its domain. Setting aside the theoretical issue that the reduced function $G_j(\cdot)$ may be difficult to compute, this justifies the following assumption.

Assumption 3 *For all $j \in N$, function $G_j(\cdot)$ is a* strictly *increasing function and hence it is invertible.*

Note that, if $G_j(\cdot)$ is discontinuous for some j, then $G_j^{-1}(\cdot)$ is not defined for some weight values. However, we can complete the definition of each $G_j^{-1}(\cdot)$ for the whole domain by setting $G_j^{-1}(w) := \max\{x_j : G_j(x_j) \leq w\}$ for all $w \in [0, w_j]$. Similarly to the discussion in Section 5.1, the computational complexity for computing $G_j^{-1}(w)$ depends on the properties of $G_j(\cdot)$.

The main result of this section follows:

Proposition 19 *A fractional knapsack problem with penalty in terms of weight, described by invertible functions $G_j(\cdot)$ for all $j \in N$, can be reduced to a FKPP.*

Proof. Let $I = (n, W, (p_j), (G_j))$ be an instance of fractional knapsack with penalty in terms of weight. For each item j, let $w_j = G_j(1)$ denote its maximum weight. We define an instance $\widetilde{I} = (\widetilde{n}, \widetilde{W}, (\widetilde{F}_j), (\widetilde{w}_j))$ of FKPP with $\widetilde{n} = n$ items and capacity $\widetilde{W} = W$. In addition, for each item $j = 1, \ldots, n$, we define a weight $\widetilde{w}_j = w_j$ and profit function $\widetilde{F}_j(\widetilde{x}_j) = p_j G_j^{-1}(w_j \widetilde{x}_j)$.

We now show that every feasible solution for I has associated a feasible solution for $\widetilde{I}$ with the same value, and viceversa. Let x be a feasible solution for I, i.e., such that each item j has weight $G_j(x_j)$ and gives a profit $p_j x_j$. We now define a solution $\widetilde{x}$ by setting $\widetilde{x}_j = G_j(x_j)/w_j$, i.e., item j has weight $G_j(x_j)$ and gives a profit $\widetilde{F}_j(\widetilde{x}_j) = p_j G_j^{-1}(w_j \widetilde{x}_j) = p_j G_j^{-1}(w_j G_j(x_j)/w_j) = p_j x_j$.

Similarly, let $\widetilde{x}$ be a feasible solution for $\widetilde{I}$, i.e. each item j has weight $w_j \widetilde{x}_j$ and profit $\widetilde{F}_j(\widetilde{x}_j)$. We define a solution x by setting $x_j = \widetilde{F}_j(\widetilde{x}_j)/p_j$, i.e., item j has weight $G_j(x_j) = G_j(\widetilde{F}_j(\widetilde{x}_j)/p_j) = G_j(p_j G_j^{-1}(w_j \widetilde{x}_j)/p_j) = w_j \widetilde{x}_j$ and profit $p_j x_j = p_j \widetilde{F}_j(\widetilde{x}_j)/p_j = \widetilde{F}_j(\widetilde{x}_j)$. □

5.2. Mathematical models for FKPP

In this section we present two alternative mathematical models for FKPP. The first model is general, though it may be non-linear depending on the form of the penalty functions. Conversely, the second model is an Integer Linear Program (ILP) restricted to integral weight contributions of all items. Recalling Proposition 17, this applies e.g. for the case when all $F_j(\cdot)$ functions are convex.

5.2.1 General model

The first model ($MGEN$) has continuous variables x_j to denote the fraction of item $j \in N$ in the solution. In addition we introduce, for each item j, two binary variables α_j and β_j, that are used to handle the profit function for $x_j = 0$ and $x_j = 1$, respectively. Denoting by $\Delta_j^0 = f_j(0)$ and $\Delta_j^1 = f_j(1)$, the formulation reads:

$$(MGEN) \qquad \max \sum_{j \in N} \left(p_j x_j - f_j(x_j) + \Delta_j^0 \alpha_j + \Delta_j^1 \beta_j \right) \tag{5.4}$$

$$\sum_{j \in N} w_j x_j \leq W \tag{5.5}$$

$$\alpha_j \leq 1 - x_j \qquad j \in N \tag{5.6}$$

$$\beta_j \leq x_j \qquad j \in N \tag{5.7}$$

$$0 \leq x_j \leq 1 \qquad j \in N \tag{5.8}$$

$$\alpha_j, \beta_j \in \{0,1\} \qquad j \in N. \tag{5.9}$$

The objective function (5.4) takes into account both the linear profit and the penalty of each item and is possibly increased by Δ_j^0 or Δ_j^1 when item j is not selected or fully inserted into the knapsack, respectively. While (5.5) is the capacity constraint, inequalities (5.6) and (5.7) set the correct values of the α_j and β_j variables, allowing each such variable to be 1 only in case the associated item is not taken or fully taken, respectively.

This model has polynomial size for what concerns both the number of variables and constraints. All constraints in the model are linear, and all possible nonlinearities appear in the objective function only. Thus, the computational tractability of the model depends on the shape of the penalty functions $f_j(\cdot)$.

5.2.2 An ILP model for integer weight contributions

The second model ($MINT$) assumes the integrality of the used weight for each item j. It samples the function $F_j(x_j)$ for all relevant values of variable x_j, and introduces a binary variable for each such value. In particular, for each item $j \in N$ and possible weight $k = 1, \ldots, w_j$, we introduce a binary variable y_{jk} that takes value 1 if item j uses k units of capacity, i.e., if $w_j x_j = k$. For each pair (j,k), we denote by t_{jk} the net profit obtained by taking $\frac{k}{w_j}$ units of item j, namely:

$$t_{jk} = F_j\left(\frac{k}{w_j}\right) = \begin{cases} 0 & \text{if } k = 0 \\ p_j & \text{if } k = w_j \\ p_j \frac{k}{w_j} - f_j(\frac{k}{w_j}) & \text{otherwise.} \end{cases} \qquad (j \in N)$$

As an illustrative example, for an item j with $w_j = 3$ we define 3 variables y_{j1}, y_{j2}, y_{j3} with weights $1, 2, 3$ and profits $t_{j1} = F_j(1/3) = p_j/3 - f_j(1/3)$, $t_{j2} = F_j(2/3) = 2p_j/3 - f_j(2/3)$ and $t_{j3} = F_j(1) = p_j$.

Recall that, for the case where all profit functions $F_j(\cdot)$ are convex, Proposition 17 ensures that an optimal solution exists in which each item uses an integer number of units of capacity. Thus, *MINT* gives an optimal solution for the convex case. Of course, *MINT* can be also applied for the general case where it will yield a possibly sub-optimal, approximate solution, which will be discussed in Section 5.2.3.

Using the above definitions, the second model is:

$$(MINT) \qquad \max \sum_{j \in N} \sum_{k=1}^{w_j} t_{jk} y_{jk} \tag{5.10}$$

$$\sum_{k=1}^{w_j} y_{jk} \leq 1 \qquad j \in N \tag{5.11}$$

$$\sum_{j \in N} \sum_{k=1}^{w_j} k\, y_{jk} \leq W \tag{5.12}$$

$$y_{jk} \in \{0,1\} \qquad j \in N; k = 1, \ldots, w_j. \tag{5.13}$$

Objective function (5.10) takes into account both the profit and the penalties for the selected items. Constraints (5.11) ensure that each item is associated with a unique weight value, whereas (5.12) imposes the capacity constraint.

Observe that the number of y_{jk} variables is $\sum_{j \in N} w_j$, i.e., pseudopolynomial in the input size. Thus, we may expect this model to be extremely challenging to be solved for instances with large item weights. However, this model is a pure Integer Linear Program (ILP), and can thus be solved using the rich machinery offered by modern commercial ILP solvers. Moreover, it is easy to see that *MINT* corresponds to a *Multiple-Choice Knapsack Problem* (MCKP) with inequality constraints, where each subset of items in MCKP corresponds to the possible choices for cutting an integer weight from an item in FKPP. Thus, one could also solve *MINT* by applying specialized algorithms for MCKP. Note that number of items in the resulting instance of MCKP, say $\widetilde{n}$, is pseudopolynomial, namely $\widetilde{n} = \sum_{j \in N} w_j$ which is in $O(nw_{\max})$, where $w_{\max} := \max\{w_j : j = 1, \ldots, n\}$ denotes the maximum weight of an item. Thus, the classical dynamic programming scheme described in Kellerer et al. [83, Sec. 11.5]) would take $O(nw_{\max}W)$ time.

5.2.3 Comparison between *MGEN* and *MINT*

In the convex case both the general model *MGEN* and model *MINT* are valid. The following proposition states the relationship between the associated continuous relaxations.

Proposition 20 *When functions $F_j(\cdot)$ are convex for all $j \in N$, the continuous relaxation of model (5.4)–(5.9) dominates the continuous relaxation of model (5.10)–(5.13).*

Proof. Let (x^*, α^*, β^*) be an optimal solution to the continuous relaxation of model *MGEN*. Observe that, in any optimal solution, variables α and β will be at the maximum value that is allowed, namely $\alpha_j^* = 1 - x_j^*$ and $\beta_j^* = x_j^*$ for each $j \in N$. Thus, the contribution of each item j to the objective function is

$$p_j x_j^* - f_j(x_j^*) + \Delta_j^0(1 - x_j^*) + \Delta_j^1 x_j^* \leq$$

$$p_j x_j^* - \left(\Delta_j^0(1-x_j^*) + \Delta_j^1 x_j^*\right) + \Delta_j^0(1-x_j^*) + \Delta_j^1 x_j^* = p_j x_j^*$$

where the first inequality holds by convexity of $-f(x_j)$.

Consider now a solution, say y^*, for the second model, defined as follows: for each item $j \in N$, let

$$y_{jk}^* = \begin{cases} 0 & \text{if } k < w_j \\ x_j^* & \text{if } k = w_j \end{cases}$$

It is easy to see that this solution satisfies constraints (5.11) and (5.12), i.e., y^* is a feasible solution to the continuous relaxation of $MINT$. The associated value is $p_j y_{jw_j}^* = p_j x_j^*$ which concludes the proof. □

We can also explicitly state the following property of the continuous relaxation of $MINT$.

Proposition 21 *Let $\bar{y}_{jk}$ be an optimal solution of the LP-relaxation of $MINT$. Then $\bar{y}_{jk} = 0$ for $k < w_j$.*

Proof. Comparing the efficiencies e_{jk} (profit per unit of weight) we have for $k < w_j$:

$$e_{jk} = \frac{p_j \frac{k}{w_j} - f_j(\frac{k}{w_j})}{k} = \frac{p_j}{w_j} - \frac{1}{k} f_j(\frac{k}{w_j}) \leq \frac{p_j}{w_j} = e_{jw_j}$$

Observe that (5.11) impose that, for each item j, the sum of the associated y variables is at most 1. Thus, for each item j, it is always advantageous to consider variable y_{jw_j} (i.e., the one with highest efficiency) and set it to the largest possible value. □

Although the associated continuous relaxation of $MINT$ provides weaker bounds than its counterpart for $MGEN$, the former can be computed much more efficiently. It follows from Proposition 21 that the LP-relaxation of $MINT$ is equivalent to the LP-relaxation of the underlying knapsack problem KP and thus can be solved by the classical linear time algorithm (see, Kellerer et al. [83, Sec. 3.1]) in $O(n)$ time[3]. For the continuous relaxation of $MGEN$ no combinatorial algorithm is available for the general case.

As observed above, one could obtain an approximation of the general case by solving model $MINT$ and thus implicitly adding the restrictions that $w_j x_j$ is integer for each item j. This might seem attractive since the MCKP-type model $MINT$ can be expected to be much easier to solve than the general $MINT$, as illustrated by our experiments in Section 5.6. Unfortunately, the quality of this approximation may be quite low in the worst case.

For a given instance I, let us denote by $z^*(I)$ and $z_{INT}(I)$ the optimal solution value and the value of the solution computed by model $MINT$, respectively. The following result determines the maximum percentage deviation of the latter with respect to the optimum when nonconvex profit functions are considered.

Theorem 3 *Let I be an instance of FKPP with arbitrary profit functions $F_j(\cdot)$ for each item j. Then, we have $\frac{z_{INT}(I)}{z^*(I)} \leq \frac{1}{2}$ and the ratio can be tight.*

Proof. For ease of notation, we omit the indication of the instance I at hand. Let z^C be the optimal solution value of the problem in which items can be split with no

[3]Note that employing the linear time algorithm for the LP-relaxation of MCKP (see, Kellerer et al. [83, Sec. 11.2]) would give an $O(nw_{\max})$ pseudopolynomial algorithm.

penalties. Similarly, denote by z^{KP} the same value for the problem in which they cannot be split at all. We have

$$z^C \geq z^* \geq z_{INT} \geq z^{KP} \geq \frac{z^C}{2}, \tag{5.14}$$

where the last inequality derives from the observation that z^{KP} corresponds to a KP whose continuous relaxation is z^C. Thus, both z_{INT} and z^* belong to the interval $[\frac{z^C}{2}, z^C]$, implying that the ratio between these two values cannot be larger than $\frac{1}{2}$.
To prove that the ratio is tight, consider a family of instances defined according to an integer parameter k, as follows: there are $n = k^2 + 1$ items and the capacity is $C = k + 1$. The first item has profit $p_1 = k$, weight $w_1 = 1$, and a profit function such that $F_1(x_1) = 0$ for $x_1 < 1$. Each remaining item $j = 2, \ldots, n$ has $p_j = k + 1$, $w_j = k + 1$, and profit function $F_j(x_j) = (k + 1)x_j$ for $0 \leq x_j \leq \frac{1}{k(k+1)}$ and $F_j(x_j) = \frac{1}{k}$ for $x_j \geq \frac{1}{k(k+1)}$.
The optimal solution completely packs the first item and fills the residual capacity by taking each remaining item j at a level $x_j = \frac{1}{k(k+1)}$. It is easy to see that each such item requires a capacity equal to $\frac{1}{k}$ and that the resulting profit is $z^* = p_1 + k^2\frac{1}{k} = k + k = 2k$. The heuristic solution is composed by a unique item $j \in [2, n]$ that is taken completely, thus yielding a profit $z_{INT} = k + 1$. Observe that an equivalent solution is obtained taking item 1 and using the residual capacity equal to k to pack the remaining items. In this case, forcing each item to have an integer $w_j x_j$ value implies that k items are taken at a level $x_j = \frac{1}{k+1}$, thus producing a total profit equal to $k + k\frac{1}{k} = k + 1$. Thus we have $\frac{z_{INT}}{z^*} = \frac{k+1}{2k}$, i.e, the ratio is arbitrarily close to $\frac{1}{2}$ for sufficiently large k. □

Inequality (5.14) also shows the following bound on the effect of allowing fractional values.

Corollary 1

$$z^{KP} \leq z^* \leq 2\, z^{KP}$$

5.3. An FPTAS for the general case

For an NP-hard problem the best algorithmic approach one can hope for is a Fully Polynomial Time Approximation Scheme (FPTAS). This means that for any given accuracy $\varepsilon > 0$, the algorithm computes a feasible solution with a relative deviation of at most ε from the (unknown) optimal solution. Its running time must be polynomial in $1/\varepsilon$ and size of the encoded input. This allows a trade-off between accuracy and running time: Permitting a larger deviation a rough estimation of the optimal solution value is reached within a short computation time. Allocating more running time, almost optimal solutions can be guaranteed. We can derive an FPTAS for FKPP by employing another approximate representation of FKPP as a Multiple-Choice Knapsack Problem (MCKP) different from Section 5.2.2.

For each item j, we define a function $\varphi_j : [0, p_j] \to [0, 1]$ such that $\varphi_j(p) = \inf\{x : F_j(x) \geq p\}$. This function is well defined since all F_j are monotonically increasing, though it may not be easy to evaluate. If F_j is strictly increasing and continuous then φ_j is the inverse function F_j^{-1}. According to our definition, if a profit function is constant on a certain interval, i.e. $F_j(x) = c$ for all $x \in [a, b] \subseteq [0, 1]$, then $\varphi_j(c) = a$.

Note that in all cases we are aware of, an FPTAS for knapsack-type problems is derived from an exact, pseudopolynomial dynamic programming scheme. Usually, the profit values are divided by a suitable constant (depending on ε) and the resulting values are rounded down. Thus, the order of magnitude of all profits is reduced. This *scaling* of the profit space results in a polynomial running time at a certain loss of optimality. For FKPP the situation is different since the continuous, non-discrete nature of the decision variables and the resulting item weights and profits does not seem to permit an exact dynamic programming approach for the general case. Of course, it is well-known that the MCKP instance implied by $MINT$ can be solved to optimality in pseudopolynomial time and can be approximated by an FPTAS, but the resulting solution may deviate considerably from the optimal solution for the general case of FKPP, as illustrated by Theorem 3.

For our general FPTAS we are given an accuracy parameter ε and a scaling parameter K to be defined later. We partition the profit range $[0, p_j]$ of every item j into equal-sized intervals of width K. Thus, the profit range is approximated by a sequence of equidistant discrete values $0, K, 2K, \ldots, \lfloor \frac{p_j}{K} \rfloor K$, and the final value p_j (if it is not a multiple of K). This guarantees that for any profit $p \in [0, p_j]$, there exists an integer number $i \in \{0, 1, \ldots, \lfloor \frac{p_j}{K} \rfloor\}$ such that $iK \leq p < (i+1)K$.

For any instance I of FKPP, we define a corresponding instance I^A of MCKP with the same capacity W. Each item j in I gives rise to a subset N_j of $\lfloor \frac{p_j}{K} \rfloor + 1$ items in I^A. For $i \in \{0, 1, \ldots, \lfloor \frac{p_j}{K} \rfloor\}$ there is an item in N_j with profit i and weight $\varphi_j(iK)w_j$. Note that the sequence of profits in each subset is completely regular, while the sequence of weights may contain large jumps and also subsequences of identical values if F_j is discontinuous. Every feasible solution of I^A directly implies a feasible solution of I by setting $x_j = \varphi_j(iK)$ if item i was selected from subset N_j.

Our FPTAS consists of running dynamic programming by profits, i.e. determining for every profit value the smallest weight of a solution reaching this profit. In this way one can solve I^A to optimality and report the associated feasible solution for I. The running time of the underlying standard algorithm (see, Kellerer et al. [83, Sec. 11.5]) is a product of the number of items, and an upper bound on the objective function. Setting $p_{\max} := \max\{p_j : j = 1, \ldots, n\}$, considering that the total number of items in I^A is $\sum_{j \in N}(\lfloor \frac{p_j}{K} \rfloor + 1) \leq n\, p_{\max}/K + n$, and stating a trivial upper bound on the objective function value of I^A as $n\, p_{\max}/K$, its running time can be bounded by $O((np_{\max}/K)^2)$. Now, choosing (similar to the classical FPTAS for KP) $K := \frac{\varepsilon p_{\max}}{n}$ the running time is fully polynomial, namely $O(n^4/\varepsilon^2)$. Note that we do not consider possible improvements of this complexity as our aim is to establish the existence of an FPTAS.

Theorem 4 *There is an FPTAS for FPKK if, for each item $j \in N$ and possible profit $p \in [0, p_j]$, value $\varphi_j(p)$ can be computed in polynomial time.*

Proof. It remains to show that the algorithm yields an ε-approximation.[4] Consider an instance I with optimal solution x^*, solution value z^*, and the corresponding instance I^A of MCKP. For each value x_j^*, determine an integer number i_j such that $i_j K \leq F_j(x_j^*) < (i_j + 1)K$. Now define the following solution, obtained by setting $x_j^{*A} := \varphi_j(i_j K)$ for each item j. Clearly, x^{*A} is a feasible solution in I^A; let us denote by z^{*A} its objective function value in I^A. Observe that x^{*A} is also feasible in I, and

[4] It should be noted that the direct application of standard arguments as given in Kellerer et al. [83, Sec. 2.6] does not work since the optimal solution for I may have an almost arbitrary profit if translated into a feasible solution of I^A.

its associated value is Kz^{*A}. The difference between z^* and Kz^{*A}, i.e. moving from I to I^A and back to I, can be bounded by

$$z^* - Kz^{*A} \leq nK = \varepsilon p_{\max} \leq \varepsilon z^* \tag{5.15}$$

since the profit contributed by each item j will be reduced by less than K.

Now consider the solution to I^A computed by our algorithm, i.e., an optimal solution obtained running the dynamic programming algorithm, and let $z^A \geq z^{*A}$ be its objective value in I^A. As the outcome of our FPTAS, this implies a feasible solution, say x', for I with objective function value $z' = Kz^A$. Thus, we have from (5.15)

$$(1-\varepsilon)z^* \leq Kz^{*A} \leq Kz^A = z'$$

i.e., the produced solution yields an ε-approximation of z^* for any $\varepsilon > 0$. □

5.4. Dynamic Programming algorithms for the convex case

In the following sections, we present dynamic programming (DP) algorithms that can be used for computing an optimal solution to FKPP in case that an optimal solution exists with at most one fractional item. By Proposition 16 this applies for the broad class of instances where profit functions are convex. Our algorithms will use a DP algorithm for KP as a black box. Thus, we firstly review two classical DP algorithms for KP. Then, we describe the DP approach proposed in [4] to solve FKPP. Finally, we present a new algorithm that yields improvements with respect to the algorithm by [4] both from a theoretical and from a computational viewpoint.

5.4.1 Dynamic Programming algorithms for KP

The basic idea of DP algorithms for KP is to solve, for each item $j = 1, \ldots, n$ and capacity value $c = 1, \ldots, W$, the KP instance defined by item set $\{1, \ldots, j\}$ and capacity c. Denoting by $T(j, c)$ the value of this solution, we can determine an optimal solution to the original instance as $T(n, W)$.

There are two main procedures that can be used to compute the T entries. The first one, also known as Bellman recursion (see, Bellman [90]), is based on the computation of *all* the $n \times W$ entries. The computation of each entry can be done in constant time, yielding an $O(nW)$ complexity. The second scheme is known as *Dynamic Programming with Lists* and goes back to Nemhauser and Ullmann [91] (see also Kellerer et al. [83, Sec. 3.4]). The basic idea is that an optimal solution can be determined without computing all the $T(j, c)$ entries since many (j, c) pairs may be redundant. This consideration may reduce the memory requirement and improve the computational performance of the resulting algorithm. In fact, under certain input distributions it was shown in [92] to run in expected polynomial time. On the other hand, there is no effect in reducing the worst-case complexity and, if the number of non-redundant entries is comparable to nW, then the overhead needed to handle the lists could be significant.

In the following, we will assume that a black box procedure `DP-KP` implementing one of the schemes above is available. Observe that both algorithms require some initialization step; conceptually, this corresponds to setting $T(0, c) = 0$ for each $c = 1, \ldots, W$. However, one can easily generalize the schemes simply setting $T(0, c) = S(c)$ for each $c = 1, \ldots, W$, where S is some starting vector to be properly defined

in case some profit is obtained even when no items are taken ($S = \underline{0}$ means that a zero vector is used). In addition, note that the DP algorithm returns, as a byproduct, the optimal solution value for *all* possible capacities $c = 1, \ldots, W$. Given an item set N, a capacity value W, and a vector S containing a starting profit value for each capacity entry, function DP-KP(N, W, S) computes and returns a vector $\overline{T}$ such that $\overline{T}(c) = T(|N|, c)$ for each $c = 1, \ldots, W$.

5.4.2 A Dynamic Programming algorithm for FKPP

The DP algorithm proposed in [4] is based on the following observation: if no item is fractionally selected in an optimal solution, then an optimal solution of FKPP corresponds to the optimal solution of KP. Otherwise, one can exploit the fact that only one item is taken at a fractional level, guess this item and solve a KP associated with the remaining items. In case a DP algorithm is used to solve the KP instance, one can easily compute *a posteriori* the amount of capacity to be used for the split item and derive an optimal FKPP solution. The complete algorithm $DP1$ is given in Algorithm 5.4.2.1, where each z_j denotes the profit of the optimal solution in case item j is split. The version of this algorithm, where DP with Lists is used, will be referred by $DP2$.

Algorithm 5.4.2.1 $DP1$

compute the optimal solution when no item is split
set $\overline{T} :=$ DP-KP$(N, W, \underline{0})$ and $z^* := \overline{T}(W)$
for all items $j \in N$ **do**
  apply DP without item j
  set $\overline{T} :=$ DP-KP$(N \setminus \{j\}, W, \underline{0})$ and $z_j := \overline{T}(W)$;
  complete the solution by splitting item j in the best possible way
  for $c = 1$ to $w_j - 1$ **do**
    if $\overline{T}(W - c) + F_j(\frac{c}{w_j}) > z_j$ **then**
      $z_j := \overline{T}(W - c) + F_j(\frac{c}{w_j})$
    end if
  end for
  if $z_j > z^*$ **then**
    $z^* := z_j$
  end if
end for
return z^*

Algorithm $DP1$ can be executed in $O(n^2W)$ time. Indeed, computing the optimal solution when no item is split requires the execution of a dynamic programming for KP. As to the case in which an item is split, there are n iterations. At each iteration j, one has to run the dynamic programming for KP with item set $N \setminus \{j\}$ and capacity W, which requires $O(nW)$; given that, the associated z_j value can be computed in $O(w_j)$ time. Thus, the overall complexity of $DP1$ is $O(n^2W)$. The same considerations apply for $DP2$.

Proposition 22 *If all profit functions $F_j(\cdot)$ are convex, FKPP is weakly NP-hard.*

Proof. It is easy to see that FKPP is in NP. As pointed out in the Introduction, it contains KP as a special case, which arises when no item j can be split, e.g., $f_j(x_j) = p_j \quad \forall x_j \in (0, 1)$ $(j \in N)$. Thus, FKPP is NP-hard, but can be solved is

pseudopolynomial time as shown by the analysis of algorithm $DP1$. □

5.4.3 An improved Dynamic Programming algorithm for FKPP

In this section we introduce a new DP algorithm for FKKP having an improved computational complexity with respect to the scheme given in the previous section. The resulting algorithm $IDP1$ (see Algorithm 5.4.3.1) takes as input an integer parameter k that will be defined later. As for the previous approach, this algorithm considers one item at a time as the split item, computes the optimal KP solution without this item, and completes the solution in the best possible way. The main difference is that the KP instances are solved in an incremental way. In particular, items are partitioned into r subsets, each containing at most k items. For each item j that belongs to subset L_i, we compute vector $\overline{T}_1$, containing, for every capacity value $c = 1, \ldots, W$, the optimal solution value of the KP instance defined by all items not in L_i and by a capacity value c. Each such a solution is then completed considering all items in L_i but j. This can be done executing again procedure DP-KP, using $\overline{T}_1$ as the vector of starting values, thus obtaining a vector $\overline{T}_2$ that contains, for every capacity value $c = 1, \ldots, W$, the optimal solution value of KP instance with all items but j and capacity equal to c. At this point, we can proceed as in $DP1$, i.e., considering all fractional levels for item j, and computing the optimal value of the FKPP solution. As stated in the following Theorem 5, this allows a reduction of the computational complexity of the algorithm. Employing an implementation of this approach based on DP with Lists will give an algorithm denoted by $IDP2$.

Algorithm 5.4.3.1 $IDP1$

compute the optimal solution when no item is split
set $\overline{T} :=$ DP-KP$(N, W, \underline{0})$ and $z^* := \overline{T}(W)$
partition item set N into r subsets L_i such that $|L_i| \leq k$ for each $i = 1, \ldots, r$
for $i = 1$ to r **do**
  guess the set L_i of items that contains the split item
  set $\overline{T}_1 :=$ DP-KP$(N \setminus L_i, W, \underline{0})$
  for all items $j \in L_i$ **do**
    apply DP to the residual items (but item j) starting from the optimal values associated with item set $N \setminus L_i$
    set $\overline{T}_2 :=$ DP-KP$(L_i \setminus \{j\}, W, \overline{T}_1)$ and $z_j := \overline{T}_2(W)$
    complete the solution splitting item j in the best possible way
    for $c = 1$ to $w_j - 1$ **do**
      if $\overline{T}_2(W - c) + F_j(\frac{c}{w_j}) > z_j$ **then**
        $z_j := \overline{T}_2(W - c) + F_j(\frac{c}{w_j})$
      end if
    end for
  end for
  if $z_j > z^*$ **then**
    $z^* := z_j$
  end if
end for
return z^*

Theorem 5 *Algorithm $IDP1$ can be executed in $O(n^{3/2}W)$ time.*

Proof. Consider an iteration of the algorithm, associated with subset (say) L_i. The first DP execution takes $O((n-k)W)$ time. Then, k executions of the DP algorithm are needed, trying all items in L_i as split item. Each execution requires $O(kW)$ time and produces a solution in which item j is not taken. Trying all possible $w_j - 1$ ways to complete this solution with item j takes $O(W)$ additional time.
Thus, the complexity of each iteration is $O\big(nW - kW + k[kW + W]\big)$. Executing r iterations yields a total complexity equal to $O(rnW - rkW + rk^2W + rkW)$ time.
Taking $k = \lfloor\sqrt{n}\rfloor$ and $r = \lceil n/k\rceil \approx \sqrt{n}$ we obtain the claimed complexity equal to $O(\sqrt{n}\,nW - nW + \sqrt{n}\,nW + nW) = O(\sqrt{n}\,nW)$. □

Again, a similar reasoning applies for the list based $IDP2$.

5.5. Heuristics

In this section we present three simple heuristic algorithms that provide approximate solutions for the general FKPP.

5.5.1 First heuristic algorithm

The first heuristic ($H1$) exploits the similarity between FKPP and KP. The procedure, described in Algorithm 5.5.1.1, first computes an optimal solution of the KP instance obtained when items cannot be split. Then, it fills the residual capacity (if any) using a fraction of some residual item. To this aim, all items that are not packed in the KP solution are tried, and the one returning the maximum profit is selected.

Algorithm 5.5.1.1 $H1$

solve KP and let x be an optimal solution
set $z^H := \sum_{j\in N} p_j x_j$ and $\bar{c} := W - \sum_{j\in N} w_j x_j$
let $j = \arg\max_{i:x_i=0}\{F_i(\bar{c}/w_i)\}$
set $x_j := \bar{c}/w_j$ and $z^H := z^H + F_j(x_j)$
return z^H

5.5.2 Second heuristic algorithm

The heuristic algorithm $H1$ requires the solution of a KP. As this problem is NP-hard, though solvable efficiently in practice, we developed a second algorithm based on the approximate solution of the knapsack problem. In this algorithm, that will be denoted as $H2$, we used the classical `GREEDY` procedure described in Kellerer et al. [83], that returns a KP solution that is maximal with respect to inclusion. This solution is possibly improved using a fraction of some unpacked item in the same way it happens for $H1$.

5.5.3 Third heuristic algorithm

Our third heuristic produces an initial KP solution by applying a variant of the `GREEDY` procedure, called `GREEDY-SPLIT` in Kellerer et al. [83], that packs items into the knapsack until the critical item is found (and then terminates). The residual capacity (if any) is filled in an iterative way, selecting at each iteration the unpacked item that can be packed with a maximum profit. The complete algorithm is given in Algorithm 5.5.3.1.

Algorithm 5.5.3.1 *H3*

execute the `GREEDY-SPLIT` algorithm for KP and let x be the resulting solution
set $z^H := \sum_{j \in N} p_j x_j$ and $\bar{c} := W - \sum_{j \in N} w_j x_j$
while $\bar{c} > 0$ **do**
  let $j = \arg\max_{i: x_i = 0} \{F_i(v_i) : v_i = \min(1, \bar{c}/w_i)\}$
  set $x_j := \min(1, \bar{c}/w_j)$, $\bar{c} = \bar{c} - x_j w_j$ and $z^H := z^H + F_j(x_j)$
end while
return z^H

5.6. Computational experiments

In this section we report the outcome of our computational experiments on FKPP. In Section 5.6.1 we first give some details about the implementation of the algorithm, while Section 5.6.2 describes our benchmark instances. Sections 5.6.3, 5.6.4 and 5.6.5 report the results of the exact methods for different classes of problems, while Section 5.6.6 reports the outcome of the experiments concerning the heuristic solution of FKPP.

5.6.1 Settings

All experiments were performed single thread on a computer equipped with an Intel(R) Core(TM) i7-6900K processor clocked at 3.20 GHz and 64 GB RAM under GNU/Linux Ubuntu 16.04. Each run was assigned a time limit of one hour. All DP algorithms and the heuristic algorithms were implemented in *C++*, while models *MGEN* and *MINT* were solved using the state-of-the-art commercial solver CPLEX 12.7.1.

As to model *MINT*, we also solved it using a combinatorial exact algorithm, namely algorithm *MCKP* by Pisinger [93]. This algorithm is considered the state-of-the-art in the solution of Multiple-Choice Knapsack Problems, and its code is publicly available at `www.diku.dk/~pisinger/codes.html`. Given a FKPP instance, we defined an associated MCKP instance as follows: each item j in FKPP corresponds to a subset of items of MCKP, and each possible fraction of item j with weight equal to k in FKPP corresponds to an item with weight k in subset j of the MCKP instance, possibly removing MCKP items with negative profit. Algorithm *MCKP* is designed for problems with integer positive data and for the version of the problem with the equality constraint, i.e., the case in which exactly one item must be selected from each subset of items. Thus, we had to implement the following transformation:

1. all profit values were multiplied by a large factor, possibly rounding the obtained values;
2. for each subset j of items, a dummy item with zero profit and weight was added;
3. the profit and the weight of each item were increased by one, the capacity was increased by n.

The results obtained by solving the resulting MCKP instance will be denoted by *MCKP* in the following. Computational experiments showed that *MCKP* largely outperforms, in terms of computing time, the direct application of our commercial solver on model *MINT*. For this reason, we do not report the results obtained using the latter method in our analysis.

5.6.2 Benchmark instances

To the best of our knowledge, there is no FKPP benchmark in the literature. Thus, we generated a large set of problems derived from KP instances from the literature. We now describe the way KP instances were selected, and discuss later how each KP problem was used to generate a FKPP instance.

KP instances

To define our benchmark, we used the KP instances introduced by Pisinger [94]. In particular, several classes of instances have been obtained with different types of correlation between profits and weights. The instances generator, publicly available at `www.diku.dk/~pisinger/codes.html`, takes as input the class number, a positive parameter R that defines the range in which weights are generated, and the total number of items n.
For our experiments we considered only the six classes (11, 12, 13, 14, 15, 16) that are denoted as *hard* by Pisinger [94] for KP algorithms, and used five different values of R (namely, $R = 10^3, 10^4, 10^5, 10^6$ and 10^7) and five different values of n ($n = 20, 50, 100, 200$, and 500). It turned out that the generator returned integer overflow when generating instances of class 16 with $R \geq 10^5$; thus, we disregarded the corresponding instances. For each class, R and n, we generated one problem, producing a set of 135 KP instances.

Penalty functions

As to the penalty functions, we tested continuous functions expressed by polynomials with degree at most 2, i.e., *linear* or *quadratic* penalty functions. Recall that $\Delta_j^0 = f_j(0)$ and $\Delta_j^1 = f_j(1)$ represent, for a given item j, the value of the penalty function for $x_j = 0$ and $x_j = 1$, respectively. Thus, the general form of the penalty function is

$$f_j(x_j) = -k_j x_j^2 + (\Delta_j^1 - \Delta_j^0 + k_j)x_j + \Delta_j^0 \tag{5.16}$$

where k_j is a parameter that defines the slope of the function. Given (5.16), the profit function for each item j reads as follows

$$F_j(x_j) = \begin{cases} 0 & \text{if } x_j = 0 \\ p_j & \text{if } x_j = 1 \\ p_j x_j + k_j x_j^2 + (\Delta_j^0 - \Delta_j^1 - k_j)x_j - \Delta_j^0 & \text{otherwise} \end{cases} \qquad (j \in N) \tag{5.17}$$

The linear case. When $k_j = 0$, $\forall j \in N$, all profit functions are linear, and all methods described in Sections 5.2 and 5.4 can be used to determine an optimal solution. In this case, we consider three different penalty functions, depending on the values Δ_j^0 and Δ_j^1, as follows:

1. *Constant* penalty: if $\Delta_j^0 = \Delta_j^1 = \Delta_j$, the penalty function for item j is constant, and the associated profit function is given by

$$F_j(x_j) = \begin{cases} 0 & \text{if } x_j = 0 \\ p_j & \text{if } x_j = 1 \\ p_j x_j - \Delta_j & \text{if } 0 < x_j < 1 \end{cases} \qquad (j \in N)$$

2. *Increasing* penalty: if $\Delta_j^0 = 0$ and $\Delta_j^1 > 0$, the penalty function for item j is negligible for small fractions x_j while it assumes its maximum for values of x_j close to 1. In this case the profit function is not continuous in $x_j = 1$:

$$F_j(x_j) = \begin{cases} p_j & \text{if } x_j = 1 \\ p_j x_j - \Delta_j^1 x_j & \text{if } 0 \leq x_j < 1 \end{cases} \qquad (j \in N)$$

3. *Decreasing* penalty: if $\Delta_j^0 > 0$ and $\Delta_j^1 = 0$, the penalty function for item j assumes its maximum value for $x_j = 0$ and decreases to 0 for $x_j = 1$. In this case the profit function is not continuous in $x_j = 0$:

$$F_j(x_j) = \begin{cases} 0 & \text{if } x_j = 0 \\ (p_j + \Delta_j^0)x_j - \Delta_j^0 & \text{if } 0 < x_j \leq 1 \end{cases} \qquad (j \in N)$$

The quadratic case. If $k_j \neq 0$ the objective function (5.17) is quadratic. In particular, it corresponds to a convex function if $k_j > 0$, whereas it is a concave function in $(0, 1)$ if $k_j < 0$ (while it is not concave neither convex when 0 and 1 are considered). In the first case Proposition 16 applies, thus both model *MINT* and the DP algorithm can be used to derive an optimal solution. In this case, however, we could not use model *MGEN* since our solver does not support the maximization of convex functions. In the concave case, instead, model *MINT* and the DP algorithm do not provide an optimal solution, whereas model *MGEN* asks for the minimization of a convex quadratic integer program, which can be tackled by our solver (the continuous relaxation of *MGEN* has concave objective function).

Figure 5.2 shows the different shapes of the profit function in the linear and quadratic cases inside the interval $[0, 1]$. They are compared with the linear profit function corresponding to CKP.

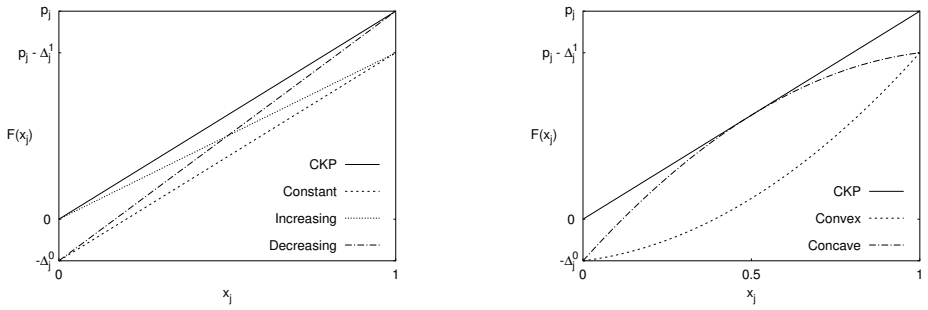


FIGURE 5.2: Linear profit functions (on the left) and convex and concave profit functions (on the right) of FKPP compared with CKP.

Our benchmark includes, for each KP instance, 5 FKPP problems defined according to the penalty functions described above: convex, concave, constant, increasing and decreasing. In all cases, but for the one with increasing penalty, we set $\Delta_j^0 = 0.1p_j$ for each item j. Similarly, all instances but those with decreasing penalty shape, $\Delta_j^1 = 0.1p_j$ for each item j. For the quadratic concave case, we set $k_j = 0.4p_j$, as this choice defined a curve that is a tangent to the linear function $p_j x_j$ for $x_j = 0.5$. By analogy, we set $k_j = -0.4p_j$ in the convex case.

5.6.3 Results on linear instances

Table 5.1 reports the results for the considered FKPP instances with linear penalty functions ($k_j = 0$). The first and the second columns of the table report the range R and the number of items n of the instances, respectively. Each row summaries the results of eighteen different instances (fifteen for $R \geq 10^5$): one instance for each class (as done by Pisinger [94]) and for each type of linear penalty function (constant, increasing and decreasing). Then the table is vertically divided into six sections, each associated with a model or an algorithm. For each algorithm we report the percentage of instances solved to optimality and the average computing time (for instances solved to optimality only). As algorithms $IDP1$ and $IDP2$ can solve all instances to proven optimality, we report only the associated computing times. Finally, row *Avg* collects the averages of the above values.

		MGEN		*DP1*		*IDP1*	*DP2*		*IDP2*	*MCKP*	
R	n	*% Opt.*	*Time*	*% Opt.*	*Time*	*Time*	*% Opt.*	*Time*	*Time*	*% Opt.*	*Time*
10^3	20	100.00	0.01	100.00	0.00	0.00	100.00	0.00	0.00	100.00	0.00
10^3	50	100.00	0.11	100.00	0.01	0.00	100.00	0.00	0.00	100.00	0.00
10^3	100	100.00	1.10	100.00	0.03	0.00	100.00	0.02	0.01	100.00	0.00
10^3	200	100.00	7.30	100.00	0.08	0.01	100.00	0.06	0.01	100.00	0.00
10^3	500	100.00	10.00	100.00	1.06	0.10	100.00	1.31	0.13	100.00	0.02
10^4	20	100.00	0.01	100.00	0.01	0.01	100.00	0.00	0.00	100.00	0.29
10^4	50	100.00	0.31	100.00	0.06	0.02	100.00	0.01	0.01	100.00	0.04
10^4	100	100.00	42.85	100.00	0.22	0.05	100.00	0.07	0.03	100.00	0.05
10^4	200	77.78	1208.05	100.00	0.87	0.13	100.00	0.37	0.06	100.00	0.10
10^4	500	55.56	0.49	100.00	4.66	0.46	100.00	3.48	0.37	94.44	0.20
10^5	20	100.00	0.01	100.00	0.10	0.04	100.00	0.00	0.00	100.00	22.72
10^5	50	100.00	0.06	100.00	0.40	0.13	100.00	0.01	0.01	100.00	29.42
10^5	100	100.00	0.75	100.00	1.58	0.35	100.00	0.33	0.14	100.00	62.46
10^5	200	80.00	0.97	100.00	6.21	0.92	100.00	2.92	0.66	100.00	28.97
10^5	500	60.00	76.46	100.00	38.47	3.45	100.00	27.86	3.09	100.00	0.76
10^6	20	100.00	0.01	100.00	0.95	0.40	100.00	0.00	0.01	100.00	831.63
10^6	50	100.00	0.27	100.00	5.87	1.80	100.00	0.07	0.06	93.33	1102.64
10^6	100	100.00	0.85	100.00	21.61	4.25	100.00	3.86	1.68	86.67	256.69
10^6	200	80.00	1.21	100.00	84.95	12.10	100.00	33.94	8.32	100.00	143.40
10^6	500	86.67	673.10	100.00	531.08	47.37	100.00	395.27	42.46	86.67	466.65
10^7	20	100.00	0.01	100.00	12.10	5.35	100.00	0.04	0.08	0.00	-
10^7	50	100.00	0.37	100.00	78.15	22.85	100.00	0.20	0.19	0.00	-
10^7	100	100.00	1.96	100.00	298.58	60.47	100.00	18.08	12.98	0.00	-
10^7	200	100.00	2.63	100.00	1161.82	167.09	100.00	477.00	129.27	0.00	-
10^7	500	100.00	12.90	0.00	-	650.15	60.00	0.22	701.85	0.00	-
Avg		93.60	81.67	96.00	93.70	39.10	98.40	38.61	36.06	78.44	147.30

TABLE 5.1: Average computing time (seconds) over 6 classes of instances with linear profit function.

Computational experiments show that the DP algorithms have a much better performance than the direct application of the ILP solver on model $MGEN$: the computing times for the model are often orders of magnitude larger than those of the DP algorithms, and the number of instances solved to optimality is slightly over the 80%. The improved DPs ($IDP1$ and $IDP2$) are, on average, the best performing algorithms for the larger instances having 500 items. For these instances, the computing times are one order of magnitude smaller than those of the corresponding non-improved versions. In addition, $IDP1$ and $IDP2$ are the only methods that can solve all the instances within the time limit. Finally, as to $MCKP$, results show that the performance of this algorithm are only marginally affected by the number of items, whereas it strongly deteriorates when increasing the maximum size R of the

items: actually, *MCKP* is unable to solve any instance with $R = 10^7$.

Figures 5.3, 5.4 and 5.5 report the same results, for constant, increasing and decreasing penalty functions, respectively, using performance profiles (in logarithmic scale). Following the guidelines suggested by Dolan and Moré [36], performance profiles are defined as follows. Let m be any solution method and i denote an instance of the problem. In addition let $t_{i,m}$ be the time required by method m to solve instance i. We define *performance ratio* for pair (i, m) as

$$r_{i,m} = \frac{t_{i,m}}{\min_{m' \in M}\{t_{i,m'}\}}$$

where M is the set of the considered methods. Then, for each method $m \in M$, we define:

$$\rho_m(\tau) = \frac{|\{i \in I : r_{i,m} \leq \tau\}|}{|I|}$$

where I is the set of the instances. Intuitively, $r_{i,m}$ denotes the worsening (with respect to computing time) incurred when solving instance i using method m instead of the best possible one, whereas $\rho_m(\tau)$ gives the percentage of instances for which the computing time of method m was not larger than τ times the time of the best performing method.

The performance profiles clearly show that the DP algorithms are not influenced by the considered penalty function: they "sample" the value of the profit function for integer weight values and associate the profit to the corresponding item fraction. Instead, the performances of the other two methods depend on the penalty function: model *MGEN* has a good performance for about 40% of the instances with increasing penalty functions, for which it is the fastest method, but then it struggles in solving the instances with constant penalty function. On the contrary, *MCKP* turns out to be the fastest method for almost 40% of the instances with constant penalty, but it has low performance in the remaining two cases. Among DP algorithms, *IDP*2 turns out to be the most efficient method: actually, this is the fastest algorithm for almost 30% of the instances and turns out to be the best algorithm for all the hard instances that require a large computing time.

5.6.4 Results on convex instances

Table 5.2 reports the results for the considered FKPP instances with convex profit function ($k_j < 0$, $j \in N$). The table is organized as Table 5.1 though it does not include results for model *MGEN*, that cannot be optimized using our MIP solver.

The results are somehow similar to those of the linear case, and confirm that the DP algorithms are not really dependent on the shape of the profit function. Conversely, algorithm *MCKP* has a much more unpredictable behavior that, in any case, deteriorates when increasing the value of R.

Figure 5.6 reports the performance profile for these instances, showing that *DP*2, *IDP*2 and *MCKP* are the best methods for 30% of the instances each, while *IDP*1 being the fastest method for 10% of the instances. The fact that *DP*2 can be the best method, i.e., even better that its improved counterpart *IDP*2, is due to implementation details that produce some slowdown in the latter; typically these effects are negligible, but they are evident for easy instances that are solved within fractions of a second.

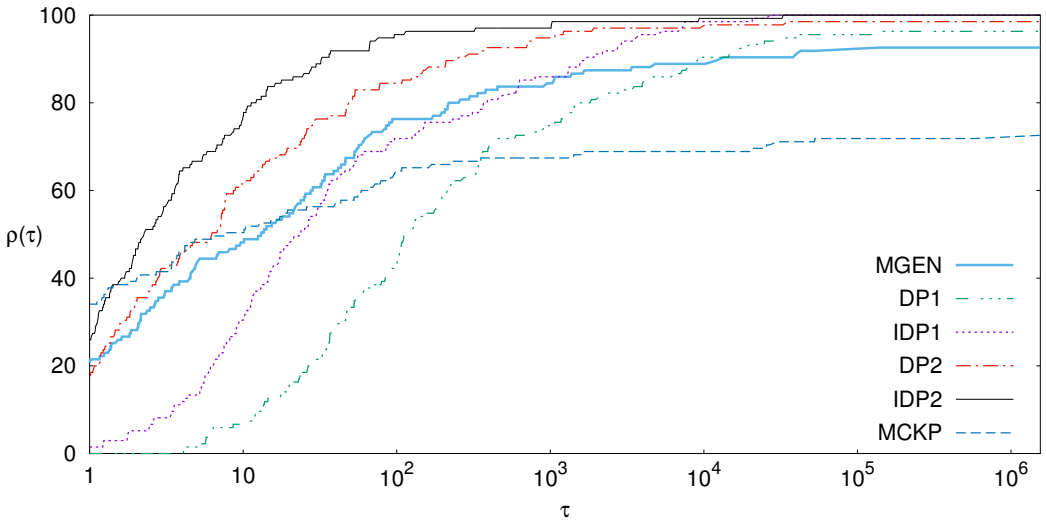


FIGURE 5.3: Performance profile of exact methods for FKPP - Constant penalty function.

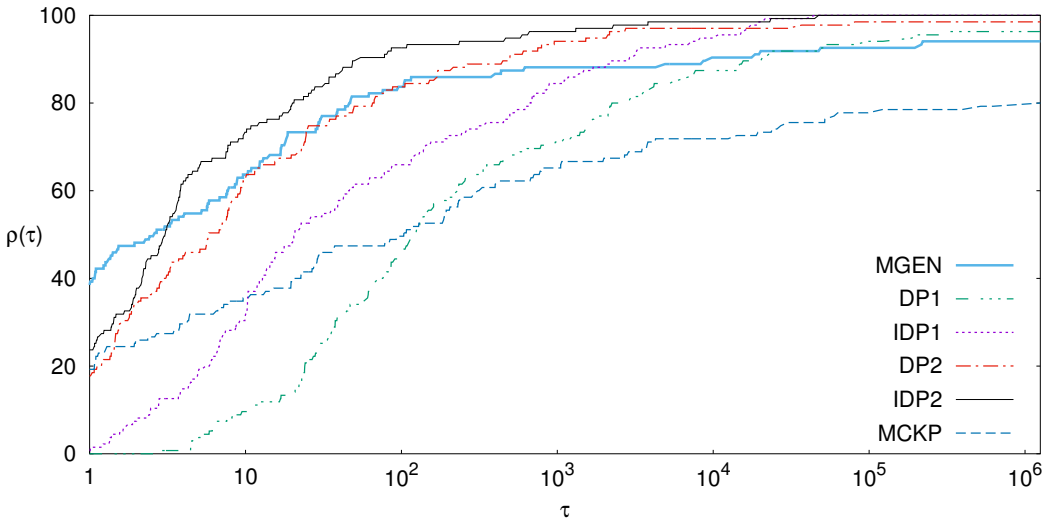


FIGURE 5.4: Performance profile of exact methods for FKPP - Increasing penalty function.

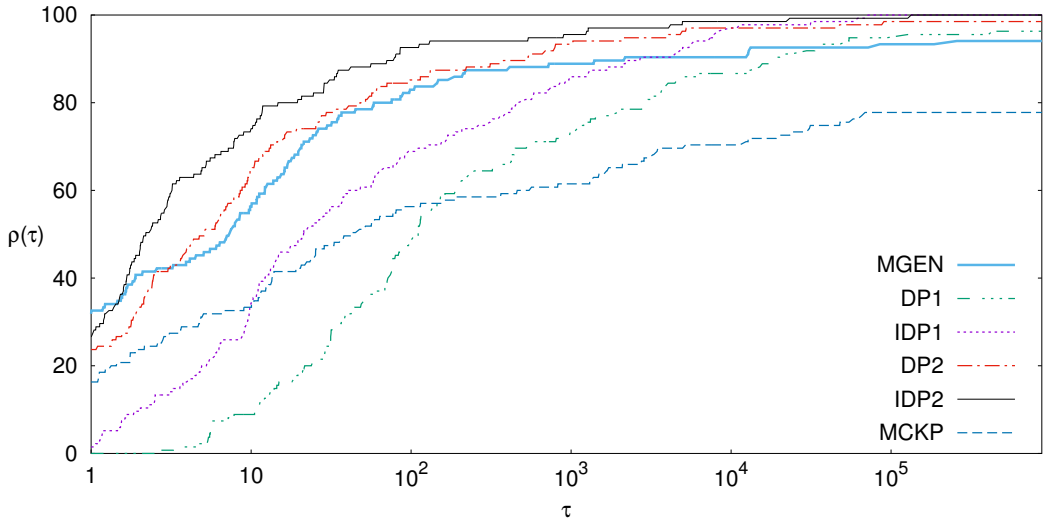


FIGURE 5.5: Performance profile of exact methods for FKPP - Decreasing penalty function.

		DP1		*IDP1*	*DP2*		*IDP2*	*MCKP*	
R	*n*	*% Opt.*	*Time*	*Time*	*% Opt.*	*Time*	*Time*	*% Opt.*	*Time*
10^3	20	100.00	0.00	0.00	100.00	0.00	0.00	100.00	0.00
10^3	50	100.00	0.01	0.00	100.00	0.00	0.00	100.00	0.00
10^3	100	100.00	0.03	0.00	100.00	0.02	0.00	100.00	0.00
10^3	200	100.00	0.08	0.01	100.00	0.07	0.02	100.00	0.00
10^3	500	100.00	1.09	0.12	100.00	1.30	0.15	100.00	0.01
10^4	20	100.00	0.01	0.00	100.00	0.00	0.00	100.00	0.16
10^4	50	100.00	0.05	0.02	100.00	0.01	0.01	83.33	0.01
10^4	100	100.00	0.22	0.04	100.00	0.07	0.04	100.00	0.02
10^4	200	100.00	0.79	0.12	100.00	0.36	0.07	100.00	0.04
10^4	500	100.00	4.67	0.48	100.00	3.50	0.36	100.00	0.11
10^5	20	100.00	0.08	0.03	100.00	0.00	0.00	100.00	17.64
10^5	50	100.00	0.39	0.17	100.00	0.01	0.01	80.00	6.14
10^5	100	100.00	1.59	0.36	100.00	0.32	0.15	80.00	0.37
10^5	200	100.00	6.16	0.97	100.00	2.96	0.66	100.00	0.32
10^5	500	100.00	38.44	3.47	100.00	28.52	3.11	60.00	0.35
10^6	20	100.00	0.94	0.40	100.00	0.01	0.01	60.00	1006.89
10^6	50	100.00	5.30	1.60	100.00	0.08	0.05	80.00	421.53
10^6	100	100.00	20.74	4.09	100.00	3.18	1.54	80.00	3.80
10^6	200	100.00	83.88	12.00	100.00	31.16	8.32	100.00	17.54
10^6	500	100.00	593.05	52.29	100.00	507.75	43.01	80.00	54.57
10^7	20	100.00	11.61	5.17	100.00	0.05	0.08	0.00	-
10^7	50	100.00	71.74	21.21	100.00	0.17	0.20	0.00	-
10^7	100	100.00	282.98	56.63	100.00	16.62	13.03	0.00	-
10^7	200	100.00	1114.09	160.22	100.00	357.51	129.78	0.00	-
10^7	500	0.00	-	678.84	60.00	0.23	701.45	0.00	-
Avg		96.00	93.25	39.93	98.40	38.16	36.08	72.13	76.47

TABLE 5.2: Average computing time (seconds) over 6 classes of instances with convex objective function. Observe that the times required by the different DP algorithms are almost identical to those of the linear case.

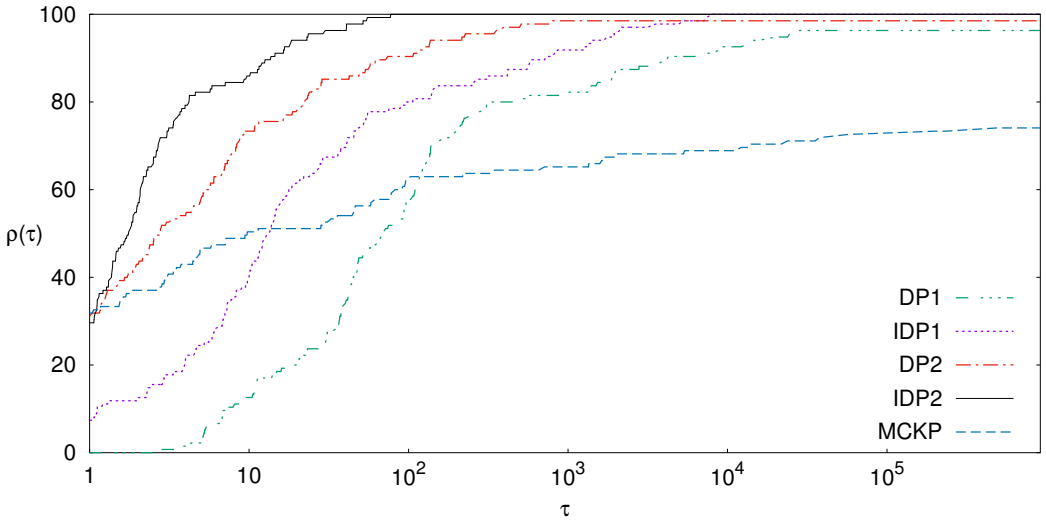


FIGURE 5.6: Performance profile of exact methods for FKPP - Quadratic convex penalty function.

5.6.5 Results on concave instances

As to profit functions that are concave in $(0,1)$, the only available method for computing an optimal solution is the application of our solver to model *MGEN*. Table 5.3 reports the computing times and the percentage of instances solved to optimality for these instances.

	$n = 20$		$n = 50$		$n = 100$		$n = 200$		$n = 500$	
R	*% Opt.*	*Time*	*% Opt.*	*Time*	*% Opt.*	*Time*	*% Opt.*	*Time*	*% Opt.*	*Time*
10^3	100.00	0.75	83.33	614.84	33.33	0.44	33.33	5.74	0.00	-
10^4	100.00	1.32	66.67	0.94	33.33	0.66	33.33	6.24	16.67	165.84
10^5	100.00	0.93	80.00	19.21	40.00	0.68	20.00	3.99	20.00	29.63
10^6	100.00	0.81	80.00	77.98	20.00	1.06	20.00	4.99	20.00	13.16
10^7	100.00	14.39	40.00	287.85	20.00	4.12	20.00	5.14	20.00	10.93
Avg	100.00	3.64	70.00	200.16	29.33	1.39	25.33	5.22	15.33	54.89

TABLE 5.3: Average times (seconds) over 6 classes of instances with quadratic concave objective function.

The results in Table 5.3 show that solving the problem with concave profit is much more challenging than in the linear case: *MGEN* is able to consistently solve all the instances with 20 items, with an average computing time of 3.64 seconds, but it fails in solving 30% of the instances with $n = 50$. Remind that, in the linear case, the same algorithm was able to solve all the instances with $n \leq 100$. Results are even worse for larger instances: only 15% of the instances with 500 items are solved to proven optimality within the given time limit.

5.6.6 Results of heuristic algorithms

Finally, we report the results obtained by executing the heuristic algorithms of Section 5.5 on our instances. We compare these heuristics with a trivial approach that solves the associated KP instance and takes no item at a fractional level. All these algorithms require a negligible computing time for the considered instances, thus times are not reported in what follows.

We evaluate the quality of a heuristic solution as follows

$$\%prof = 100 \times \frac{z^H}{z^*} \tag{5.18}$$

where z^* and z^H denote the optimal and the heuristic solution values, respectively. Figure 5.7 plots the quality of the heuristic solutions for all the instances with linear and convex profit functions, as for all these instances the optimal solution value is known. The figure shows the trend with increasing number of items (on the left) and range (on the right).

From Figure 5.7 (left side), we see that $H1$ finds the best solutions, on average, for instances with 20 items; for larger instances the best algorithm is $H2$. $H3$ has a trend which is similar to that of $H2$, though it performs better than the latter on instances with 20 items only. Finally we observe that the quality of the heuristic solutions improves when increasing the number of items: for $n = 20$, the best performing algorithm (namely, $H1$) has an average profit that is about 1% smaller than the optimal one, while for $n = 500$ this gap is considerably reduced, and the profits of the heuristic solutions are almost the same as the profits of the optimal solutions. This is due to the fact that, for large number of items, the optimal KP solution value is very close to the optimal FKPP solution value (see the plot of algorithm KP), hence the way the spare capacity is filled becomes less crucial. From the practical viewpoint this is good news: for smaller instances, it is important to compute optimal solutions, which can be obtained in short computing time; for large instances, where computing optimal solutions may be time consuming, heuristic solutions are indeed near-optimal. Figure 5.7 (right side) shows a similar behavior concerning the KP solutions. However, in this case, the trend associated with the other heuristic solutions is much more irregular. This plot points out the obvious dominance of $H1$ over KP. In addition, also in this case $H2$ and $H3$ have similar performances, that are better than $H1$ for instances with small R, but become even worse than KP for instances with $R \geq 10^5$.

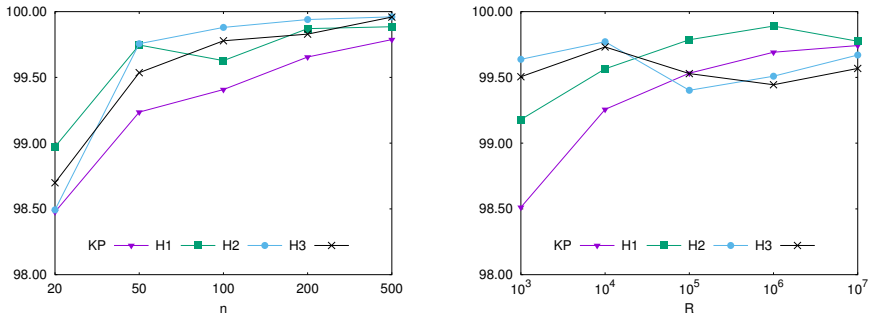


FIGURE 5.7: Values of $\%prof$ of the heuristic solutions with increasing number of item (on the left) and with increasing range (on the right).

As to instances with concave profit functions, we could solve to optimality only instances with 20 items. Table 5.4 reports, for the KP and the three heuristic algorithms, the average values of $\%prof$, and the percentage of instances for which each algorithm computed an optimal solution.

In this case the best algorithm is clearly $H3$, which finds an optimal solution in about 44% of the cases and gives, on average, a profit which is 99.93% times the optimal one.

	KP		*H*1		*H*2		*H*3	
R	*% Opt.*	*%prof*	*% Opt.*	*%prof*	*% Opt.*	*%prof*	*% Opt.*	*%prof*
10^3	16.67	95.77	33.33	96.95	16.67	98.88	33.33	99.72
10^4	0.00	96.76	0.00	97.30	33.33	99.53	50.00	99.98
10^5	20.00	99.19	20.00	99.74	20.00	98.54	40.00	99.96
10^6	40.00	98.45	40.00	99.42	20.00	99.37	60.00	99.99
10^7	20.00	99.15	20.00	99.15	20.00	99.23	40.00	99.98
Avg	19.33	97.86	22.67	98.51	22.00	99.11	44.67	99.93

TABLE 5.4: Average percentage profit and optimal solutions for the heuristic algorithms, concave profit function (20 items).

5.7. Conclusions

We considered integer optimization problems in which the integrality of the variables can be relaxed at the expenses of some penalty in the objective function, a situation that has relevant applications in many contexts. Different from previous approaches from the literature, that always considered the case in which the penalty is constant, we allow the penalty to be dependent on the fractional quantity. As a case study, we concentrated on the Knapsack Problem (KP), which is the simplest integer optimization problem and for which many relevant properties are known. Introducing a penalty associated with fractional variables, we thus defined the Fractional Knapsack Problem with Penalties (FKPP), that we modelled using two mathematical formulations. We studied some properties of the problem, and analyzed the performances of the two models depending on the penalty functions used. From an approximability perspective, we showed that FKPP admits a Fully Polynomial Time Approximation Scheme (FPTAS), independently from the form of the penalty function. From an algorithmic point of view, we introduced two dynamic programming algorithms and three fast heuristic algorithms. We performed an extensive computational analysis to illustrate the performances of all the approaches on a large benchmark set derived from the literature. Our experiments show that we are able to efficiently solve medium-sized FKPP instances, and that the hardness of the instances is closely related to the shape of the penalty function used. Moreover, it seems that FKPP instances are consistently much harder that their KP counterparts, which can be solved in an almost negligible time using combinatorial algorithms from the literature. This suggests that some combinatorial algorithm tailored for FKPP might allow the solution of much larger instances in a reasonable computing time.

Chapter 6

Models and heuristic algorithms for a real-world lot sizing and distribution problem

[1]

Production and distribution are two critic activities in supply chain management. They consist on the coordinate set of actions allowing the match between industries and their markets, vendors and buyers, suppliers and purchasers. With the increase of competition and globalization, their role shifted from operative to tactical and from a separate local vision to an integrated management of these functions. Nowadays, competition is among collaborative interconnected and optimized supply chain networks. Each of them links, organically, companies to their markets and strategic suppliers.

Following this trend, production and distribution planning rises as an integrated decisional science to provide strategies and quantitative methods to fulfill the market demand, answering the research and industrial question "What to produce and deliver, when and how?". Behind such question, the following multiple dimensions of production and distribution planning emerge:

- Dynamic dimension, i.e., time drives the decisions of producing and distributing according to the market due dates and order delivery time;
- System dimension, i.e., the supply chain configuration (layers, entities, routes, etc.) constraints and guides the decisions;
- Goal dimension, i.e., the goals to pursue as efficiency, cost, service level, green and social aspects, etc;
- Stakeholder dimension, i.e., the decision-making process is tailored on the interests of the supply chain actors.

The convergence of the introduced multiple dimensions is hard, making production and distribution planning a wide, complex and dynamic problem to solve. The scientific literature agrees to adopt an integrated and stepwise approach following the impact of decisions and the horizon of the plan. Strategic, tactical and operative clusters of decision are thus introduced. The strategic level focuses on long-term decisions over a multiple-year time horizon. At this level, the key elements of the supply chain system are fixed, e.g., overall structure, plant capacity, shipping modes,

[1] The results of this chapter appears in: V.Bo, M. Bortolini, E. Malaguti, M. Monaci, C. Mora, and P. Paronuzzi, "Models and Algorithms for Integrated Production and Distribution Problems", *Technical Report OR-19-8*, http://or.dei.unibo.it/technical-reports, 2019. [95]

etc. Generally, the temporal dimension is neglected, and a single large time window is used. The inclusion of the temporal aspect of supply chains is the focus of the tactical level, where a shorter planning time horizon, e.g., one or two semesters, is typically considered. At this level, decisions on capacity and production allocation among plants, network flows, shipping and storage modes are taken from an integrated perspective minimizing costs and optimizing the supply chain service level. Finally, daily scheduling and routing activities are among the most important decisions of the operative level.

This paper, driven by an industrial instance and application, aims at contributing to the tactical cluster of decisions. The goal is to provide the analytical model and a solution approach to support planners to the best allocation of production among plants and to the distribution flow definition within a multi-product, multi-plant and multi-period context. An aggregated cost function includes relevant drivers affecting the product total cost, while production and distribution constraints link the model to the industrial practice. We present a solution approach based on heuristic and metaheuristic optimization algorithms. To assess its effectivity and capability to guarantee the expected level of performance (in terms of computing time), we computationally test the approach to full-scale real-world instances from industry and to a large benchmark of realistic instances.

The paper is organized as follows. In Section 6.1 we review the relevant literature on the production scheduling area, both in terms of supply chain management and for what concerns optimization algorithms. Section 6.2 gives a formal definition of the problem at hand and introduces a mathematical formulation based on a Mixed Integer Linear Programming (MILP) model. In Section 6.3 we present a fast heuristic approach based on a greedy approach. In Section 6.4 we propose a metaheuristic approach based on the ruin-and-recreate paradigm. Finally, in Section 6.5 we give the outcome of computational experiments for the proposed approaches on two real-world test-cases and on a large set of realistic instances that are derived from the real ones. Conclusions are drawn in Section 6.6.

6.1. Literature review

As already mentioned, we consider different aspects of the supply chain, but we identified the lot sizing as the core problem of our topic. Given the amount of an independent demand, the lot sizing is the problem of deciding which equipment must be used and in which period the production must be performed in order to satisfy the demand, while minimizing the total required cost. It has been addressed for the first time in 1913 in a seminal paper by Harris [96], who introduced the definition of *Economic Order Quantity* (EOQ). The vastness of the possible applications of this problem results evident from its definition; consequently, the number of possible variants is also huge. For these reasons, understanding which is the specific version of the lot sizing problem that one is dealing with is a crucial task that is, at the same time, not straightforward. The tertiary study proposed by Glock et al. [97] results very useful for a first orientation. According to the definitions of this tertiary study, our case can be defined as an extension of the multi-item Capacitated Lot Sizing Problem (CLSP) and the model we propose in Section 6.2 corresponds to a dynamic (demand is not constant) and deterministic (demand is known) model. The same paper indicates the work of Maes and Van Wassenhove [98] and the one of Karimi et al. [99] as the two main reviews about this topic. Moreover, other interesting reviews are reported: the research of Buschkühl et al. [100], that focuses on the

multi-level version of the problem, and two papers by Jans and Degraeve [101] and [102], that discuss on meta-heuristic approaches and on possible extensions of the problem, respectively. The Capacitated Lot Sizing Problem aims to determine the amount and the period of the production of items in a considered planning horizon. The production quantity is constrained by a finite capacity in each period. The problem involves linear production costs, that increase as the produced quantity does, fixed set up costs, that occur any time a machine starts or switch the production in some period, and holding costs, that occur when a product is stored in the inventory from a period to the next one. The single-item version of the problem is shown to be NP-hard in [103] even when the demand is stationary and there are no holding costs. The proof provided by the authors is based on the reduction from the decision problem of checking the feasibility of a 0-1 equality constrained knapsack problem, that is an NP-complete problem. The multi-item version is *strongly* NP-hard, as shown in [104]. In this case, the authors make use of a pseudo-polynomial reduction of the Three Partition Problem, that is a strongly NP-complete problem.

Anyhow, all the reviews state that, due to the complexity of the problem, there are few attempts in the literature that try to make use of an exact algorithm ([105], [106], [107]). On the other hand, there are many contributions based on a heuristic approach, that are split them in two different categories in [99]: common-sense heuristics and mathematical programming-based heuristics. The optimization algorithm that will be introduced in Sections 6.3 - 6.4 belongs to the former class and. Using the terminology of [99], it includes both a *period-by-period* constructive heuristic (see Section 6.3), that builds a feasible solution by considering one demand at a time starting from the first period, and an *improvement* heuristic (see Section 6.4), that improves a given feasible solution. As for the works that also adopt this kind of procedures, in addition to the papers cited in the review of Karimi et al. ([108], [109], [110], [111], [112] as *period-by-period* constructive heuristic, and [113], [114], [115] as *improvement* heuristic) we also report the more recent studies of Li et al. [116] and of Toscano et al. [117].

As already mentioned, our problem is an extension of the CLSP. Compared with the basic problem, several additional aspects appear in our application, namely: setup times, multiple parallel machines, stockout costs, transportation costs (both from the company plants to the customers and from the company plants to external warehouses), availability of different capacity configurations, and minimum lot size. All these aspects appear separately in many papers from the literature but, to the best of our knowledge, there are no previous studies on problems that simultaneously involve all these aspects. Setup times were addressed by Jans and Degraeve [118], [119], who introduced lower bounds and a branch-and-price algorithm, and by Hindi et al. [120], who proposed a smoothing heuristic procedure based on the Lagrangian relaxation of the problem. The multiple machines case hase been addressed by Kang et al. [121], whose approach is based on a column generation/branch-and-bound methodology. Sandbothe and Thompson [122], [123] and Aksen et al. [124] addressed the lot-sizing problem with stockouts but they considered the the single-item version of the problem only. The transportation aspect was considered by Chandra and Fisher [125], who showed that considering the production and the distribution problems in an integrated system leads to better results than solving the two problems separately. Haq et al. [126] presented a case study where they formulated an integrated production-inventory-distribution model; similar integrated models were also proposed by Bhutta et al. [127] and by Jolayemi and Olorunniwo [128], while the extension of the model to the capacitated case has been introduced by Rao [129] for a single-item environment and it has been subsequently

generalized to the multiple item case by Rajagopalan and Swaminathan [130] and by Bradley and Arntzen [131]. Different problems including the minimum lot size constraint were addressed using optimization algorithms. Anderson and Cheah [132] proposed a heuristic algorithm based on a Lagrangian relaxation of the problem, Constantino [133] studied the problem from the polyhedral viewpoint, while Mercé and Fontan [134] solved the problem using a MILP-based heuristic. A mathematical programming-based heuristics for the dynamic multi-item capacitated lot-sizing problem under random period demands was given by Tempelmeier [135]. The algorithm is based on a heuristic column generation procedure combined with an ABC heuristic that extends the algorithm introduced by Maes and Van Wassenhove [109] for the deterministic CLSP. Armentano et al. [136] used a branch-and-bound procedure to solve the multi-item, single-level CLSP with setup times reformulated as a minimum cost network flow problem. Finally, we mention the dynamic multi-level capacitated lot sizing problem addressed by Chen [137], where the setup state of a resource may be used in consecutive time periods (*setup carryover*). The problem is solved using a variant of the fix-and-optimize approach presented in [138], dividing the problem in a number of smaller subproblems that are iteratively solved in a heuristic way.

6.2. Problem Statement

We consider the problem of a company that faces a multiperiod demand for the assortment of different products. The demand of each period originates from a set of customers with a geographic distribution, and the products can be obtained from different plants, from which they are shipped to the customers. Plants are equipped with production lines, each one with the capability to produce, in each period, a subset of the products after a suited setup; for each product, there is a minimum production quantity that can be considered. Production consumes resources that can be available at company, plant or line level. Production lines work according to shift configurations, that define the number of production hours during a period and that can be changed only at the beginning of some periods. Each plant is associated with an internal warehouse, where the production of a period can be stored before it is shipped to a customer at a later period. Each plant can also rent additional space from external warehouses, if needed. The company wants to decide, for each period, the shift configuration of each line and the production level of each product, in order to minimize the total cost, determined by configuration and setup costs for the lines, production, storage and shipment costs for the demands that are satisfied, and penalty costs for the demands that are not satisfied. A formal definition of the problem and a mathematical model are given in what follows.

6.2.1 Sets and variables

In our formulation we consider the following sets

- $\mathcal{K} = \{1, \ldots, K\}$: set of products;
- $\mathcal{J} = \{1, \ldots, J\}$: set of customers;
- $\mathcal{I} = \{1, \ldots, I\}$: set of plant;
- $\mathcal{M} = \{1, \ldots, M\}$: set of lines. Each line m is associated with a specific plant $i(m) \in \mathcal{I}$, that is, for each plant $i \in \mathcal{I}$ we define the set of associated lines

as $M(i)$. Compatibility between products and lines is expressed defining, for each product $k \in \mathcal{K}$, a set $C(k)$ of lines that can produce k;

- $H(m)$: set of configurations for each line $m \in \mathcal{M}$. Each shift configuration for a line has associated a cost and determines the capacity of the line. Without loss of generality, we assume that a larger cost corresponds to a larger capacity, and that configurations are sorted by increasing cost;
- $\mathcal{F} = \{1, \ldots, f\}$: set of product families; each product k belongs to one family $f_k \in \mathcal{F}$;
- $\mathcal{T} = \{1, \ldots, T\}$: set of time intervals. Set $\mathcal{T}$ is partitioned into π subsets $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_\pi$, that are called *multiperiods*. Each multiperiod includes all the time intervals between two consecutive periods in which line configurations can be changed;
- $\mathcal{RP}_i = \{1, \ldots, RP_i\}$: set of resources for plant i;

and decision variables specified as follows

- x_{kmt} = number of products k obtained on line m during period t ($k \in \mathcal{K}, m \in C(k), t \in \mathcal{T}$). These integer variables encode the main decision for the company, that is defining about production over time periods and lines;
- z_{kjt} = unsatisfied demand for product k at period t for customer j ($k \in \mathcal{K}, j \in \mathcal{J}, t \in \mathcal{T}$). These integer variables define the amount of customers demand that the company does not satisfy;
- f_{kjit} = amount of product k shipped from plant i to customer j at period t ($k \in \mathcal{K}, j \in \mathcal{J}, i \in \mathcal{I}, t \in \mathcal{T}$). These integer variables define the way in which customers demand is satisfied by the company;
- $y_{mhv} = \begin{cases} 1 & \text{if line } m \text{ operates in configuration } h \text{ during multiperiod } v \\ 0 & \text{otherwise} \end{cases}$
  ($m \in \mathcal{M}, h \in H(m), v = 1, \ldots, \pi$).
  These binary variables define the shift configuration (working hours) of a line during each multiperiod v.
- w_{kit} = amount of product k stored in the internal warehouse associated with plant i during period t ($k \in \mathcal{K}, i \in \mathcal{I}, t \in \mathcal{T}$);
- s_{kit} = amount of product k stored in the external warehouse associated with plant i during period t; ($k \in \mathcal{K}, i \in \mathcal{I}, t \in \mathcal{T}$);
- ϕ_{kit} = amount of product k shipped from plant i to the associated external warehouse during period t ($k \in \mathcal{K}, i \in \mathcal{I}, t \in \mathcal{T}$);
- $\alpha_{mft} = \begin{cases} 1 & \text{if line } m \text{ processes at least one product of family } f \text{ during period } t \\ 0 & \text{otherwise} \end{cases}$
  ($m \in \mathcal{M}, f \in \mathcal{F}, t \in \mathcal{T}$).
  These binary variables are used to denote the product families that are processed on a line during a specific period, and to allocate the associated setup costs.

6.2.2 Cost function

The total cost to be minimized takes into account different components: the costs for operating the lines, the production and operational costs for the satisfied demands, and the opportunity costs (penalties) for the demands that are not satisfied. In particular, the overall cost is defined as the sum of the following terms:

C^{PR} is the direct production cost, defined as

$$C^{PR} = \sum_{k \in \mathcal{K}} \sum_{m \in \mathcal{M}} \sum_{t \in \mathcal{T}} c^{PR}_{km} x_{kmt}$$

where c^{PR}_{km} is the cost for producing a unit of product k on line m;

C^{NS} is the opportunity cost for the unsatisfied demand, defined as

$$C^{NS} = \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{J}} \sum_{t \in \mathcal{T}} c^{NS}_{kt} z_{kjt}$$

where c^{NS}_{kt} is the opportunity cost for the unsatisfied demand of a unit of product k at period t;

C^{TC} is the shipping cost to the customers, defined as

$$C^{TC} = \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} c^{TC}_{kji} f_{kjit}$$

where c^{TC}_{kji} is the shipping cost for a unit of product k to customer j from plant i or from an internal/external warehouse associated with the plant;

C^{DC} is the configuration cost for the product lines, defined as

$$C^{DC} = \sum_{m \in \mathcal{M}} \sum_{h \in H(m)} \sum_{v=1}^{\pi} |\mathcal{T}_v| \, c^{DC}_{mh} y_{mhv}$$

where c^{DC}_{mh} is the shift cost for line m in configuration h;

C^{SI} is the storage cost in the internal warehouses, defined as

$$C^{SI} = \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} c^{SI}_{ki} w_{kit}$$

where c^{SI}_{ki} is the storage cost for a unit of product k in the internal warehouses at plant i;

C^{SE} is the storage cost in the external warehouses, defined as

$$C^{SE} = \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} c^{SE}_{ki} w_{kit}$$

where c^{SE}_{ki} is the storage cost for a unit of product k in the external warehouses at plant i;

C^{SU} is the setup cost, defined as

$$C^{SU} = \sum_{m \in \mathcal{M}} \sum_{f \in \mathcal{F}} \sum_{t \in \mathcal{T}} c^{SU}_{m} \alpha_{mft}$$

where c_m^{SU} is the cost for a setup on line m.

C^{TE} is the shipping cost to the external warehouses, defined as

$$C^{TE} = \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} c_{ki}^{TE} \phi_{kit}$$

where c_{ki}^{TE} is the shipping cost for a unit of product k to the external warehouses at plant i.

6.2.3 Constraints

The production schedule is subject to a number of different constraints, that we present grouped according to their meaning and structure.

Demand

$$\sum_{i \in \mathcal{I}} f_{kjit} + z_{kjt} = d_{kjt} \qquad k \in \mathcal{K}, j \in \mathcal{J}, t \in \mathcal{T} \tag{6.1}$$

These constraints impose that the demand is either satisfies by shipping the required amount of products from some plant, or it is canceled and computed as missed demand. The demand is expressed in terms of an order d_{kjt} for a product k to be delivered at customer j at time period t. Observe that we allow solutions where a customer is served shipping products from different plants.

Lot-sizing balancing constraints

$$\sum_{m \in M(i)} x_{kmt} + s_{ki,t-1} + w_{ki,t-1} = \sum_{j \in \mathcal{J}} f_{kjit} + s_{kit} + w_{kit} \qquad k \in \mathcal{K}, i \in \mathcal{I}, t \in \mathcal{T} \tag{6.2}$$

$$\sum_{k \in \mathcal{K}} w_{kit} \leq W_i \qquad i \in \mathcal{I}, t \in \mathcal{T} \tag{6.3}$$

These constraints generalize the classical lot-sizing formulation to the case of multiple products, plants and warehouses. Constraints (6.2) impose, for each product, plant, and period, a balance between the availability of the product, given by the production and the stocked quantities, and the stocked quantities at the end of the period plus the quantity that is shipped to customers. Constraints (6.3) define, for each plant and period, the available capacity W_i of the internal warehouse associated with the plant. Note that there is no similar constraint for external warehouses, as we assume they have unbounded capacities.

Resources

$$\sum_{k \in \mathcal{K}} \tau_{km} x_{kmt} + \sum_{f \in \mathcal{F}} t_m^{SU} \alpha_{mft} \leq \sum_{h \in H(m)} Q_{mh} y_{mhv} \qquad m \in \mathcal{M}, v = 1, \ldots, \pi, t \in \mathcal{T}_v \tag{6.4}$$

$$\sum_{h \in H(m)} y_{mhv} = 1 \qquad m \in \mathcal{M}, v = 1, \ldots, \pi \tag{6.5}$$

$$\sum_{k \in \mathcal{K}} \sum_{m \in M(i)} b_{kmr}^{P} x_{kmt} \leq B_{irt}^{P} \qquad i \in \mathcal{I}, r \in \mathcal{RP}_i, t \in \mathcal{T} \tag{6.6}$$

These constraints limit the use of available lines and resources according to their availabilities.

Constraints (6.4) impose an upper bound on the maximum working time for each line m in each period t. Indeed, the working time of the line is determined by the production times τ_{km} for each product k obtained on the line plus the total setup time, which is equal to t_m^{SU} for each different family that is produced. The right-hand-side gives the availability of the line, where each coefficients Q_{mh} represents the available working time for line m and shift configuration h. These constraints are paired with constraints (6.5) that impose to select exactly one shift configuration for each line and multiperiod. Finally, constraints (6.6) describe the consumption of resources available at plant level and impose, for each plant, resource and period, that the amount of resource used cannot exceed the availability. In these constraints, each b_{kir}^P is the amount of resource r used by line m to produce one product of type k, while B_{irt}^P denotes the availability of resource r at plant i during time period t. In a very similar way, it is possible to express the constraints (if any) that regulate the consumptions of resources that are available at company or line level.

Setup, minimum production

$$\sum_{k \in \mathcal{K}: f_k = f} x_{kmt} \leq BIGM\, \alpha_{mft} \qquad m \in \mathcal{M}, t \in \mathcal{T}, f \in \mathcal{F} \tag{6.7}$$

$$(x_{kmt} = 0) \quad \vee \quad (x_{kmt} \geq \rho_{kmt}) \qquad k \in \mathcal{K}, m \in \mathcal{M}, t \in \mathcal{T} \tag{6.8}$$

Constraints (6.7) model the setup that are incurred for each additional line, period and product family. In these constraints, $BIGM$ is a large enough constant that is used to model the logical implication forcing an α variable to take value 1 in case some products belonging to a certain family are produced. Observe that we consider a planning problem, thus we do not model the specific setup time and cost as a function of the production sequence, and adopt instead average values for setup times and costs. Logical constraints (6.8) impose that, for each product, line, and period, either there is at least a minimum production level ρ_{kmt} of the product, or there is no production at all for that product.

Shipment to external warehouses

$$\phi_{kit} \geq s_{kit} - s_{ki,t-1} \qquad k \in \mathcal{K}, i \in \mathcal{I}, t \in \mathcal{T} \tag{6.9}$$

Finally, constraints (6.9) model the flow of products to the external warehouses, for each product, plant and period. In particular, a positive value for a ϕ variable corresponds to some increase in the amount of products that stocked in an external warehouse.

Mathematical formulation Finally the overall Mixed Integer Formulation reads

$$\min\ C^{PR} + C^{NS} + C^{DC} + C^{SI} + C^{SE} + C^{TE} + C^{TC} + C^{SU} \tag{6.10}$$

$$(6.1) - (6.9)$$

$$f_{kjit} \geq 0 \qquad k \in \mathcal{K}, j \in \mathcal{J}, i \in \mathcal{I}, t \in \mathcal{T} \tag{6.11}$$

$$z_{kjt} \geq 0 \qquad k \in \mathcal{K}, j \in \mathcal{J}, t \in \mathcal{T} \tag{6.12}$$

$$x_{kmt} \geq 0 \qquad k \in \mathcal{K}, m \in C(k), t \in \mathcal{T} \tag{6.13}$$

$$w_{kit} \geq 0 \qquad k \in \mathcal{K}, i \in \mathcal{I}, t \in \mathcal{T} \tag{6.14}$$

$$s_{kit} \geq 0 \qquad k \in \mathcal{K}, i \in \mathcal{I}, t \in \mathcal{T} \tag{6.15}$$

$$y_{mhv} \in \{0,1\} \qquad m \in M, h \in H(i), v = 1, \ldots, \pi \tag{6.16}$$

$$\alpha_{mft} \in \{0,1\} \qquad m \in M, f \in \mathcal{F}, t \in \mathcal{T} \tag{6.17}$$

$$\phi_{kit} \geq 0 \qquad k \in \mathcal{K}, i \in \mathcal{I}, t \in \mathcal{T} \tag{6.18}$$

where the objective function (6.10) is the sum of the costs, all variables associated with physical quantities are defined as continuous nonnegative, and activation variables associated with line configurations and setups are binary.

The model has a polynomial number of variables and constraints. Note that constraints (6.8) are not linear, in that they model a disjunctive argument. However, it is well known that they can be rewritten in an equivalent form that is linear with respect to the decision variables. This requires the addition to the model of binary variables and $BIGM$ coefficients, possibly leading to formulations that have a weak linear programming relaxation. However, nowadays, that many commercial MILP solvers allow formulations like (6.8), and adopt specific techniques to strengthen formulations that include very large coefficients (see, e.g., [139]).

6.3. A constructive algorithm

In this section we present an iterative constructive heuristic that is used to produce a feasible solution for the problem; possibly, this solution may be used as a starting point for the metaheuristic algorithm that will be described in the next section.

Our algorithm has a *primal* nature, in that it starts with a feasible solution and maintains feasibility at each iteration. Initially, no production takes place and each line operates in each multiperiod according with the configuration having minimum cost. Then, the algorithm considers one demand at a time, and determines the best policy for the current demand, according to a greedy strategy. For the current demand, the algorithm determines the "best" line and time period in which production must take place, the amount of products to be produced, and updates the production schedule accordingly. In case some change of configuration is required, the new configuration is maintained for all time periods of the multiperiod.

Remind that each demand has associated a product, a customer, and a time period. Demands are initially sorted by increasing time period, breaking ties according to the type of product and by the customer. Let k, j and t denote the product, the customer and the period, respectively, associated with the current demand. Then, we consider all lines $m \in C(k)$ that are compatible with products of type k and all time periods $\tau \leq t$. For each pair (m, τ) we determine the associated *production level*, i.e., the maximum amount $e(km\tau)$ of product k that can be produced for satisfying the current order. The total cost for pair (m, τ) is determined by

- the direct production cost;
- the shipping cost to the customer;
- possibly, the setup cost;
- possibly, the cost for changing configuration;
- possibly, the storage cost and the shipping cost to external warehouses.

The first two costs are related with production and shipping of the products, respectively. The setup cost is incurred only in case no product belonging to familty f_k is scheduled on line m in time period τ. As to the configuration cost, we take into account only the incremental cost that is incurred when the configuration for line m has to be changed to the next one allowing some additional capacity. In the score computation, only a fraction of this additional cost is taken into account, the value of the fraction being a parameter called $ConfigParam \in [0,1]$ that will be described later. Finally, storage costs are considered in case $\tau < t$, and may be associated with costs for shipping products to an external warehouse in case not all products can be stored in the internal warehouse in all the time interval $[\tau, t]$.

Due to several constraints, e.g., capacity of the line or minimum amount of production on the line, the production level depends on each pair (m, τ). Thus, to have a fair comparison, we score each pair (m, τ) according to the cost per unit of product, i.e., we divide the total cost by the production level $e(km\tau)$, and select the pair $(\overline{m}, \bar{t})$ with minimum score.

If no line can produce product k in any time period $\tau \leq t$, or the cost per unit of product is larger than the cost cost related to the out-of-stock of product k, the current demand is refused. In case $e(k\overline{m}\bar{t}) < d_{kjt}$ we define a new (dummy) order with demand $d'_{kjt} = d_{kjt} - e(k\overline{m}\bar{t})$ and iterate the process. Note that the requirement on the minimum production, see constraints (6.8), may impose a production that is strictly larger than d_{kjt}. In this case, some units of product have to be stored in the warehouse, and used in some subsequent time period (the associated storage costs being not taken into account in this phase).

The pseudo-code of the algorithm is given in Algorithm 6.3.0.1.

Algorithm 6.3.0.1 Algorithm Greedy

1: sort the demands by nondecreasing time interval, breaking ties by product and by customer;
2: **for all** demand **do**
3: determine the "best" line $\overline{m}$, period $\bar{t}$ and production level $e(k\overline{m}\bar{t})$;
4: produce $e(k\overline{m}\bar{t})$ units of product k on line $\overline{m}$ in period $\bar{t}$;
5: possibly update the configuration of line $\overline{m}$ in the multiperiod containing $\bar{t}$;
6: **if** $e(k\overline{m}\bar{t}) < d_{kjt}$ **then**
7: define a new demand $d'_{kjt} = d_{kjt} - e(k\overline{m}\bar{t})$;
8: **else if** $e(k\overline{m}\bar{t}) > d_{kjt}$ **then**
9: store $e(k\overline{m}\bar{t}) - d_{kjt}$ units of product k in the warehouse associated with the plant $i(\overline{m})$;
10: **end if**
11: **end for**

6.4. A metaheuristic approach

Preliminary computational experiments showed that the constructive heuristic of Section 6.3 is very fast but may produce solutions in which the use of some lines is not fully optimized. Thus, we developed a metaheuristic approach that is based on the ruin-and-recreate paradigm (see, Schrimpf et al. [5]). The idea of a ruin-and-recreate algorithm is to determine feasible solutions by (i) destroying a considerable part of a feasible solution (*ruin* phase), and (ii) applying a rebuilding procedure (*recreate* phase) to complete the solution.

In our algorithm, the initial solution is produced by the constructive heuristic, and three different procedures are used to destroy the solution:

1. remove all production associated with a certain line in a given multiperiod;
2. remove all production associated with a certain product in all time periods;
3. remove all production associated with a certain line in all time periods.

The first ruin procedure is used to repair situations in which the configuration of the line is not optimized in the first time periods of a multiperiod. Typically this happens when a change of configuration is incurred at some intermediate period of the multiperiod, preventing all previous time intervals to take advantage of the increased capacity of the line. The procedure considers one pair (line m, multiperiod v) at a time, removes all the production of line m in multiperiod v, and repairs the solution through multiple executions of the constructive procedure of Section 6.3. In particular, at each execution we guess the configuration for the line m in multiperiod v, and try to reschedule all the demands that are unsatisfied (either unscheduled in the initial solution or removed by the ruin phase) without changing configuration of the lines. The best solution found is used to replace the incumbent, in case some it produces some improvement.

The second ruin procedure considers one product k at a time, removes the production of all products of type k on all lines and periods, and reschedules unsatisfied demands by applying again the constructive heuristic (without changing the configuration of the lines).

Similarly, the third ruin procedure considers each line m at a time, removes all the production on the current line, and reschedules unsatisfied demands using the constructive heuristic (without changing the configuration of the lines).

It should be noted that, in all cases, the repair phase is obtained running the constructive procedure of Section 6.3 to a restricted problem in which the configuration of all lines in all the multiperiods is fixed. For this reason, configuration costs are not taken into account in this phase.

The pseudo-code of the metaheuristic algorithm is given in Algorithm 6.4.0.1.

Algorithm 6.4.0.1 Algorithm Ruin-and-Recreate

```
1: determine an initial feasible solution using Algorithm 6.3.0.1
2: repeat
3:     for all line m and multiperiod v do
4:         remove all production on line m during multiperiod v;
5:         for all configuration h ∈ H(m) do
6:             set configuration h for line m in multiperiod v;
7:             schedule unsatisfied demands without changing the configuration of the lines;
8:             possibly update the incumbent;
9:         end for
10:    end for
11:    for all product k do
12:        remove all production of type k;
13:        schedule unsatisfied demands without changing the configuration of the lines;
14:        possibly update the incumbent;
15:    end for
16:    for all line m do
17:        remove all production on line m;
18:        schedule unsatisfied demands without changing the configuration of the lines;
19:        possibly update the incumbent;
20:    end for
21: until stopping conditions
```

In our implementation, the algorithm is halted after a maximum number *IterNum* of iterations or when an iteration with no improvement is encountered.

6.5. Computational experiments

In this section we evaluate the computational performance of our approach on a set of real and realistic instances derived from two industrial problems in the food and beverage industry. The objective of our experiments is threefold:

- first, we want to tune the algorithm parameters and assess the performance of the two steps of our approach, in terms of solution quality and computing time;
- second, we want to compare the overall performance of our approach with those of a state-of-the-art MILP solver directly executed on mathematical model (6.10)–(6.18);
- and finally, we are interested in analyzing the structure of the obtained solutions for the industrial problems in both the original case and under some realistic business scenarios.

Real-world instances. We considered two real instances, denoted as A and B in the following, that describe real-world situations arising from the food industry. In these instances the demand to be satisfied is specified in terms of *orders*, where an order is given by an a request for an amount d_{kjt} of product k by customer j at period t.

Table 6.1 reports the main characteristics of the two instances, expressed in terms of number of products ($|\mathcal{K}|$), customers ($|\mathcal{J}|$), plants ($|\mathcal{I}|$), lines ($|\mathcal{M}|$), different shift configurations ($|\mathcal{H}|$), product families ($|\mathcal{F}|$), time periods ($|\mathcal{T}|$) and multiperiods ($|\approx|$). Finally, the last column ($|\mathcal{O}|$) gives the total number of orders that have to be satisfied.

	$\lvert\mathcal{K}\rvert$	$\lvert\mathcal{J}\rvert$	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{M}\rvert$	$\lvert\mathcal{H}\rvert$	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\approx\rvert$	$\lvert\mathcal{O}\rvert$
A	887	3	6	40	5	160	41	11	27,760
B	541	7	9	31	7	152	52	14	26,024

TABLE 6.1: Main characteristics of the real-world instances.

Realistic instances. Starting from the two real-world instances, we produced an additional benchmark of 80 *realistic* instances. The realistic instances were obtained by slightly modifying, in a random way, the original input data. In particular, for each real-world instance, we generated:

- 10 *similar* instances in which each order d_{kjt} has been replaced by a new order in the range $[0.9d_{kjt}, 1.1d_{kjt}]$;
- 10 *varied* instances in which each order d_{kjt} has been replaced by a new order in the range $[0.7d_{kjt}, 1.3d_{kjt}]$.

All the remaining input parameters were left unchanged.

Experimental setting. All procedures were implemented in C++ and all experiments were executed on a computer with a 3.20 GHz Intel Core I7 processor and 64 GB of RAM. Our algorithm is sequential in nature and cannot take advantage from the parallelism of our hardware. However, considering the orders according to different sortings produces different solutions. Thus, we implemented a parallel version of the algorithm in which 4 threads are used, each running the same algorithm on the same input, but with a different sequence for processing the orders. Then, the best of the four solutions is returned. Preliminary experiments in which orders are considered according to (common-sense) deterministic rules produced solutions of the same quality as using random sortings.

6.5.1 Parameter Tuning

Our solution approach includes 2 main parameters:

- Parameter *ConfigParam*, which defines the fraction of additional cost for the activation of a shift configuration that is allocated to the current order;
- Parameter *IterNum*, which determines the number of iterations of local search to be performed after the first phase of the algorithm.

We performed the tuning of our algorithm on the original instances using 7 different values for *ConfigParam*, namely 0.00, 0.05, 0.10, 0.20, 0.30, 0.40 and 0.50.

Our first order of business was to determine the best values of *ConfigParam* for the two real-world instances. To this aim, we considered only the values of the solutions produced by the algorithm, without considering the computing time.

Table 6.2 reports the total cost for different values of *ConfigParam*. The entries in the table report the best among the solutions found by the 4-threads algorithm after the execution of the constructive algorithm and one iteration of Local Search. It is worth underlining that parameter *ConfigParam* is used during the first phase of the algorithm only; thus, there would not be any additional information in considering more than one iteration of local search.

	0.00	0.05	0.10	0.20	0.30	0.40	0.50
A	119.15	118.43	118.17	118.29	119.57	120.09	122.62
B	263.44	252.66	248.13	255.64	265.97	280.48	288.84

TABLE 6.2: Effect of parameter *ConfigParam* on real-world instances.

Results in Table 6.2 suggest that *ConfigParam* = 0.10 is the best value for the two real-world instances. For the first instance, while the average solution value (for the considered values of the parameter) is equal to 119.47, using *ConfigParam* = 0.10 produces a solution with value 118.17, with a saving equal to 1.09%. For the second instance, the saving is more relevant and is equal to 6.37%.

As to the second parameter, we performed the following experiment. We considered instance A and run the algorithm for a (potentially very large) number of iterations, halting the procedure when no improvement is obtained after the last iteration of local search. Figure 6.1 reports the cost of the best solution found as a function of the computing time. The first point of the curve corresponds to the solution at the end of the first phase, while each subsequent point is the solution cost obtained after an additional iteration of the local search. The picture shows that a good trade-off between computing time and quality of the solution is obtained with 2 iterations of local search. As most of the computing time is spent in the local search phase, the computational effort grows almost linearly with the number of local search iterations. For all the reported result, the computational time with this configuration of the algorithm is approximately 90 seconds, for instance A, and 160 seconds, for instance B.

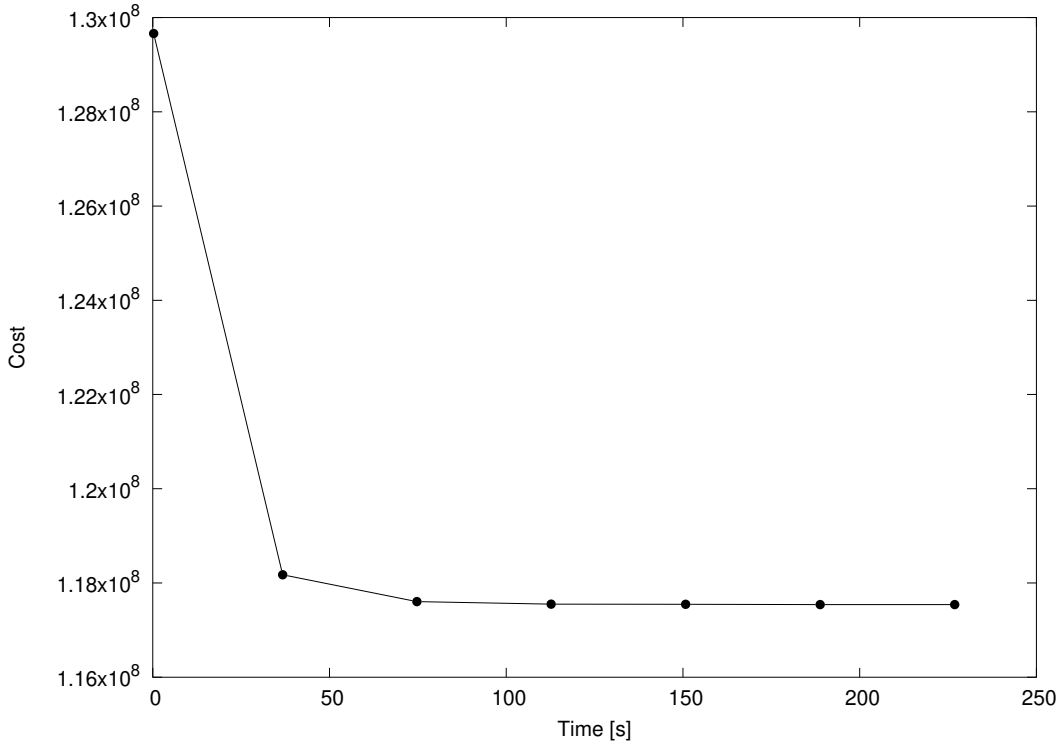


FIGURE 6.1: Cost of a solution as a function of the computing time for subsequent iterations of Local Search.

6.5.2 Comparison with MIP

In this section we compare the performances of our algorithm (in its best configuration, as described in the previous section) with those of a general-purpose commercial MILP solver. To this aim, we considered both the real-world and the realistic instances, and used IBM-ILOG Cplex 12.7.1 (CPLEX, in the following) to solve the mathematical formulation of Section 6.2.3. CPLEX was run on 4 threads with a limit of 10 hours of computing time; all the remaining parameters are set to their default values.

Table 6.3 compares the values of the gap obtained through our heuristic algorithm (HEUR, in the following) with those obtained by CPLEX. The first line of the table is associated with the real-world instances, while the remaining lines refer to the realistic instances. For each instance and algorithm we report the *gap* of the best solution found, computed as $\%gap = 100\frac{U-L}{L}$, where U is the solution value and L denotes the best lower bound known. All figures in the table are computed using the best lower bound produced by CPLEX at the time limit as L. In one case only, CPLEX got all the available 64 GB of RAM of the machine and the process has been automatically killed by the system. We denote this case with an asterisk (*) and we report the gap reached until that moment. Due to the complexity of the problem, we preferred to remove constraints 6.8 when solving the mathematical model with CPLEX. Thus, the reported results in columns CPLEX refer to solutions that are typically infeasible from this point of view, i.e., the comparison is by design in favour of CPLEX.

The results show that, on the real-world instances, our heuristic produces solutions whose gap is very close to that of the best solution found by CPLEX with a huge computational effort. Results are similar when realistic instances are considered, though CPLEX does not take the minimum production constraint into account.

6.5.3 Solution Structure and Scenario analysis

Finally, in this section we discuss the cost structure of the solutions computed by our method and we analyze how the cost structure of the solutions changes under possible scenarios. Figure 6.2 compares the *as-is* solution (column 1) with the following *what-if* situations:

2. the use of external warehouses is forbidden;
3. the distribution costs are doubled;
4. the second most important plant is closes;
5. the scarce resource of the plant is reduced to the 70% of its value;
6. the scarce resource of the plant is available in an infinite quantity;
7. all the operation times of the lines are reduced to the 90% of their value.

The first hypothetic scenario has a total cost that is slightly larger than the as-is solution; since the external warehouses are not available, the inventory costs decreases, but the amount of undelivered orders increases. Even if the distribution costs are doubled (scenario 3), the amount of delivered orders remain the same; this can be noticed from the fact that the Out of Stock costs does not change, if compared with the as-is situation, while the total distribution costs are actually doubled. The most important negative impact on the Out of Stock costs is given by scenario 4, where

TABLE 6.3: Comparison of the gap obtained by our algorithm (HEUR) and CPLEX

		A		B	
Type	Problem	HEUR	CPLEX	HEUR	CPLEX
-	-	12.92	9.11	15.98	16.16
Similar	1	13.08	10.45	16.65	16.18
	2	12.88	11.04	16.34	12.22
	3	12.79	10.20	16.78	15.07
	4	13.06	11.80	16.17	15.38
	5	12.82	9.35	16.33	17.85
	6	13.08	12.78	16.51	14.04
	7	12.95	10.66	16.68	16.47
	8	13.16	14.45	16.39	19.58
	9	13.19	10.12	16.38	13.87
	10	12.88	10.87	16.27	15.35
Varied	1	13.02	10.88	16.40	*19.93
	2	13.10	12.26	16.45	16.30
	3	13.59	11.18	16.86	15.75
	4	13.41	10.09	15.93	13.96
	5	13.26	10.28	16.24	15.74
	6	13.32	26.51	16.36	16.34
	7	13.40	12.40	17.04	14.36
	8	12.81	11.89	16.48	15.77
	9	13.40	9.41	16.94	13.96
	10	12.45	9.65	16.81	13.31

we suppose that the second important plant is closed. From scenarios 5 and 6, we can observe the importance of the plant resource availability: when it is decreased (scenario 5), then a larger part of the demand remains unsatisfied; when it is available in an infinite quantity, then the production is able to meet more than half of the demand that was unsatisfied in the as-is solution and the inventory costs slightly decrease. Decreasing the operation times of the lines allows to save part of the production costs (scenario 7).

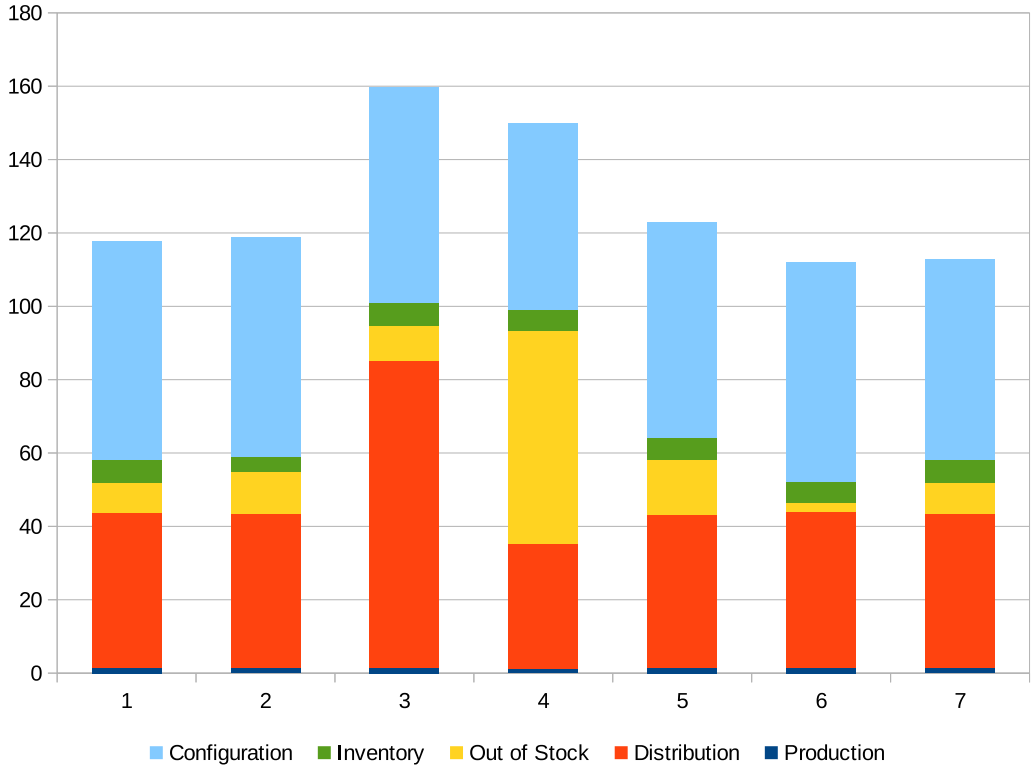


FIGURE 6.2: Different *what-if* scenarios.

6.6. Conclusions

In this chapter we addressed a lot sizing and distribution problem that arose from a real-world application. The problem includes several products, whose production must be scheduled on lines belonging to different plants, and several customers, whose demand must be satisfied taking into account the transportation costs from the plants to the customer locations.

After providing the formal definition of the problem based on a MILP formulation, we introduced both an iterative constructive heuristic algorithm and a metaheuristic approach based on the ruin-and-recreate paradigm.

An extensive analysis of the results of computational experiments on two real-world test-cases and on a large set of realistic instances proved that the approximations obtained in short computing time by our heuristic approach are very close to those that can be achieved by running a state-of-the-art commercial solver on the mathematical model for a very long computing time.

Chapter 7

Conclusions and future work

In this thesis, we faced the idea of decomposing a problem from different perspectives. In order to be successfully applied, decomposition techniques typically require the existence of a specific structure in the formulation of the problem at hand. Then, detecting this structure may turn out to be essential for the purpose of approaching the problem with the most suitable method, but, at the same time, it may be a difficult task if this structure is not evident from the most natural formulation of the problem. In this sense, we highlighted the decomposition as a problem itself and, in Chapters 2 and 3, we introduced two different problems belonging to the family of the Critical Node Detection Problems: the k-Vertex Cut problem and the capacitated k-Vertex Separator problem. In both cases, we extrapolated a bilevel point of view that allowed us to improve the best known results for several difficult classes of instances. Pursuing this direction, the future research may investigate how similar problems could benefit from this new modeling paradigm.

In Chapter 4, we dealt with the class of Chance Constraint Mathematical Programming problems and we took into consideration the case in which the variables representing future decisions have to assume integer values. Stochastic optimization problems, like this one, are often formulated with models whose coefficient matrix reveals a blocks structure that makes the problem very appealing to be effectively addressed through decomposition techniques. So, in this case, we introduced the decomposition with its most common acceptation, that is, as a very effective solution method. Indeed, we presented a branch-and-cut framework based on a decomposition approach whereby we defined a master problem and one subproblems for each possible realization of the random variable. Computational experiments showed that the outer approximation point cuts, added by separating infeasible solutions from the convex-hull of the subproblems, turned out to be functional in providing valid bounds for the considered test-bed instances. However, our research on this topic is still in progress and, on the basis of these promising preliminary results, we count on improving the performances of the separation procedures and implementing the spatial branching needed to definitively close the current existing gaps.

Another class of problems in which decomposition algorithms have been widely applied in the combinatorial optimization field is the family of packing problems. Even though this is one of the most investigated topic in the literature, the case in which items can be fragmented at the expense of some penalty has been taken into account only in the recent few years. In Chapter 5, we applied this concept to the simplest and most iconic packing problem, namely the 0-1 Knapsack Problem, and we introduced the Fractional Knapsack Problem with Penalties. This problem finds several applications in practice, but it also arises as the subproblem to be solved in Dantzig-Wolfe decomposition methods for packing problems where item fragmentation is allowed. The Fully Polynomial Time Approximation Scheme and the dynamic programming algorithm, described in Sections 5.3 and 5.4, represent our

main contributions on this subject. Future researches may reveal how these results could be applied to similar problems that arise, like in this case, as the consequence of decomposition approaches, or how they could be extended to general integer programs where variables can potentially take fractional values.

Bibliography

[1] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs", *Operations research*, vol. 8, no. 1, pp. 101–111, 1960.

[2] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems", *Numerische Mathematik*, vol. 4, no. 1, pp. 238–252, 1962, ISSN: 0945-3245.

[3] A. Lodi, E. Malaguti, G. Nannicini, and D. Thomopulos, "Nonlinear chance-constrained problems with applications to hydro scheduling", *Technical Report OR-15-9*, 2015, http://or.dei.unibo.it/technical-reports.

[4] M. Casazza and A. Ceselli, "Mathematical programming algorithms for bin packing problems with item fragmentation", *Computers & Operations Research*, vol. 46, pp. 1–11, 2014.

[5] G. Schrimpf, J. Schneider, H. Stamm-Wilmbrandt, and G. Dueck, "Record breaking optimization results using the ruin and recreate principle", *Journal of Computational Physics*, vol. 159, pp. 545–566, 2 2016.

[6] F. Furini, I. Ljubić, E. Malaguti, and P. Paronuzzi, "On integer and bilevel formulations for the k-vertex cut problem", *Mathematical Programming Computation*, pp. 1–32, 2019.

[7] D. Kempe, J. Kleinberg, and É. Tardos, "Influential nodes in a diffusion model for social networks", in *Automata, Languages and Programming*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 1127–1138, ISBN: 978-3-540-31691-6.

[8] F. Furini, I. Ljubić, P. San Segundo, and S. Martin, "The maximum clique interdiction problem", *European Journal of Operational Research*, vol. 277, no. 1, pp. 112–127, 2019.

[9] M. Lalou, M. A. Tahraoui, and H. Kheddouci, "The critical node detection problem in networks: A survey", *Computer Science Review*, vol. 28, pp. 92 – 117, 2018, ISSN: 1574-0137.

[10] C. de Souza and E. Balas, "The vertex separator problem: Algorithms and computations", *Mathematical Programming*, vol. 103(3), pp. 609–631, 2005.

[11] D. Cornaz, F. Furini, M. Lacroix, E. Malaguti, A. R. Mahjoub, and S. Martin, "The vertex *k*-cut problem", *Discrete Optimization*, vol. 31, pp. 8–28, 2019.

[12] W. Ben-Ameur and M. D. Biha, "On the minimum cut separator problem", *Networks*, vol. 59(1), pp. 30–36, 2012.

[13] R. Borndörfer, C. Ferreira, and A. Martin, "Decomposing matrices into blocks", *SIAM Journal on Optimization*, vol. 9, no. 1, pp. 236–269, 1998.

[14] A. Berger, A. Grigoriev, and R. v. d. Zwaan, "Complexity and approximability of the k-way vertex cut", *Networks*, vol. 63(2), pp. 170–178, 2014.

[15] F. Barahona, "On the k-cut problem", *Operations Research Letters*, vol. 26, no. 3, pp. 99 –105, 2000, ISSN: 0167-6377.

[16] O. Goldschmidt and D. S. Hochbaum, "A polynomial algorithm for the k-cut problem for fixed k", *Mathematics of Operations Research*, vol. 19, no. 1, pp. 24–37, 1994.

[17] D. R. Karger and R. Motwani, "An nc algorithm for minimum cuts", *SIAM Journal on Computing*, vol. 26, no. 1, pp. 255–272, 1997.

[18] A. Gupta, E. Lee, and J. Li, "An FPT algorithm beating 2-approximation for k-cut", in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '18, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2018, pp. 2821–2837, ISBN: 978-1-6119-7503-1.

[19] H. Saran and V. Vazirani, "Finding k cuts within twice the optimal", *SIAM Journal on Computing*, vol. 24, no. 1, pp. 101–108, 1995.

[20] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakiss, "The complexity of multiterminal cuts", *SIAM Journal on Computing*, vol. 23(4), pp. 864–894, 1994.

[21] S. Chopra and M. R. Rao, "On the multiway cut polyhedron", *Networks*, vol. 21(1), pp. 51–89, 1991.

[22] N. Garg, V. V. Vazirani, and M. Yannakakis, "Multiway cuts in directed and node weighted graphs", *In Automata, Languages and Programming*, pp. 487–498, 1994.

[23] ——, "Multiway cuts in node weighted graphs", *Journal of Algorithms*, vol. 50(1), pp. 49–61, 2004.

[24] D. Marx, "Parameterized graph separation problems", *Theoretical Computer Science*, vol. 351(3), pp. 394–406, 2006.

[25] D. Cornaz, Y. Magnouche, A. R. Mahjoub, and S. Martin, "The multi-terminal vertex separator problem: Polyhedral analysis and branch-and-cut", *Conference on Computers & Industrial Engineering (CIE45)*, pp. 857–864, 2015.

[26] W. Ben-Ameur, M.-A. Mohamed-Sidi, and J. Neto, "The k-separator problem", *In Computing and Combinatorics*, pp. 337–348, 2013.

[27] M. Bastubbe and M. Lübbecke, "A branch-and-price algorithm for capacitated hypergraph vertex separation", *Technical Report, Optimization Online*, 2017.

[28] J. Fukuyama, "NP-completeness of the planar separator problems", *Journal of Graph Algorithms and Applications*, vol. 10, no. 2, pp. 317–328, 2006.

[29] T. N. Bui and C. Jones, "Finding good approximate vertex and edge partitions is NP-hard", *Information Processing Letters*, vol. 42, no. 3, pp. 153 –159, 1992.

[30] E. Balas and C. C. de Souza, "The vertex separator problem: A polyhedral investigation", *Mathematical Programming*, vol. 103(3), pp. 583–608, 2005.

[31] J. Magnouche, "The multi-terminal vertex separator problem : Complexity, polyhedra and algorithms", 2017.

[32] G. Brown, M. Carlyle, J. Salmerón, and K. Wood, "Defending critical infrastructure", *INFORMS Journal on Applied Analytics*, vol. 36, no. 6, pp. 530–544, 2006.

[33] M. Fischetti, I. Ljubić, M. Monaci, and M. Sinnl, "Interdiction games under monotonicity", *INFORMS Journal on Computing*, vol. 31, no. 2, pp. 390–410, 2019.

[34] *Color02/03/04: Graph coloring and its generalizations*, http://mat.gsia.cmu.edu/COLOR03/, Accessed: 2018-07-20.

[35] *Dimacs implementation challenges*, http://dimacs.rutgers.edu/archive/Challenges/, Accessed: 2018-07-20.

[36] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles", *Mathematical Programming*, vol. 91, no. 2, pp. 201–213, 2002.

[37] F. Furini, I. Ljubić, E. Malaguti, and P. Paronuzzi, "Casting light on the hidden bilevel combinatorial structure of the k-vertex separator problem", *Technical Report OR-19-6*, 2019, http://or.dei.unibo.it/technical-reports.

[38] C. de Souza and E. Balas, "The vertex separator problem: A polyhedral investigation", *Mathematical Programming*, vol. 103(3), pp. 583–608, 2005.

[39] ——, "The vertex separator problem: Algorithms and computations", *Mathematical Programming*, vol. 103(3), pp. 609–631, 2005.

[40] H. N. Djidjev, "Partitioning planar graphs with vertex costs: Algorithms and applications", *Algorithmica*, vol. 28, no. 1, pp. 51–75, 2000.

[41] N. Garg, H. Saran, and V. Vazirani, "Finding separator cuts in planar graphs within twice the optimal", *SIAM Journal on Computing*, vol. 29, no. 1, pp. 159–179, 1999.

[42] R. Lipton and R. Tarjan, "A separator theorem for planar graphs", *SIAM Journal on Applied Mathematics*, vol. 36, no. 2, pp. 177–189, 1979.

[43] M. Heath, E. Ng, and B. Peyton, "Parallel algorithms for sparse linear systems", *SIAM Review*, vol. 33, no. 3, pp. 420–460, 1991.

[44] R. J. Lipton and R. E. Tarjan, "Applications of a planar separator theorem", in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 1977, pp. 162–170.

[45] M. Bergner, A. Caprara, A. Ceselli, F. Furini, M. Lübbecke, E. Malaguti, and E. Traversi, "Automatic dantzig–wolfe reformulation of mixed integer programs", *Mathematical Programming*, vol. 149, no. 1, pp. 391–424, 2015.

[46] S. Shen and J. C. Smith, "Polynomial-time algorithms for solving a class of critical node problems on trees and series-parallel graphs", *Networks*, vol. 60, no. 2, pp. 103 –119, 2012.

[47] S. Shen, J. C. Smith, and R. Goli, "Exact interdiction models and algorithms for disconnecting networks via node deletions", *Discrete Optimization*, vol. 9, no. 3, pp. 172 –188, 2012.

[48] D. Cornaz, F. Furini, M. Lacroix, E. Malaguti, A. R. Mahjoub, and S. Martin, "The vertex k-cut problem", *Discrete Optimization*, vol. 31, pp. 8 –28, 2019.

[49] M. Baïou and F. Barahona, "Stackelberg bipartite vertex cover and the preflow algorithm", *Algorithmica*, vol. 74, no. 3, pp. 1174–1183, 2016.

[50] L. Brotcorne, M. Labbé, P. Marcotte, and G. Savard, "Joint design and pricing on a network", *Operations Research*, vol. 56, no. 5, pp. 1104–1115, 2008.

[51] C. Casorrán, B. Fortz, M. Labbé, and F. Ordóñez, "A study of general and security Stackelberg game formulations", *European Journal of Operational Research*, vol. 278, no. 3, pp. 855–868, 2019.

[52] K. J. Cormican, D. P. Morton, and R. K. Wood, "Stochastic network interdiction", *Operations Research*, vol. 46, no. 2, pp. 184–197, 1998.

[53] S. Dempe and A. B. Zemkoho, "The bilevel programming problem: Reformulations, constraint qualifications and optimality conditions", *Math. Program.*, vol. 138, no. 1-2, pp. 447–473, 2013.

[54] M. Fischetti, I. Ljubić, M. Monaci, and M. Sinnl, "A new general-purpose algorithm for mixed-integer bilevel linear programs", *Operations Research*, vol. 65, no. 60, pp. 1615–1637, 2017.

[55] T. Kleinert, M. Labbé, F. Plein, and M. Schmidt, "There's no free lunch: On the hardness of choosing a correct big-m in bilevel optimization", *Operations Research*, 2019.

[56] L. Lozano and J. Smith, "A value-function-based exact approach for the bilevel mixed integer programming problem", *Operations Research*, vol. 65, no. 3, pp. 768–786, 2017.

[57] S. Tahernejad, T. Ralphs, and S. DeNegre, "A Branch-and-Cut Algorithm for Mixed Integer Bilevel Linear Optimization Problems and Its Implementation", *Mathematical Programming Computation (to appear)*, 2019. [Online]. Available: http://coral.ie.lehigh.edu/~ted/files/papers/MIBLP16.pdf.

[58] A. Baggio, M. Carvalho, A. Lodi, and A. Tramontani, "Multilevel approaches for the critical node problem", Canada Excellence Research Chair in Data Science for Real-Time Decision-Making, Tech. Rep. DS4DM-2017-012, 2018.

[59] J. S. Borrero, O. A. Prokopyev, and D. Sauré, "Sequential interdiction with incomplete information and learning", *Operations Research*, vol. 67, no. 1, pp. 72–89, 2019.

[60] M. Fischetti, M. Leitner, I. Ljubić, M. Luipersbeck, M. Monaci, M. Resch, D. Salvagnin, and M. Sinnl, "Thinning out Steiner trees: A node-based model for uniform edge costs", 2014.

[61] M. Leitner, I. Ljubić, M. Luipersbeck, and M. Sinnl, "A dual ascent-based branch-and-bound framework for the prize-collecting steiner tree and related problems", *INFORMS Journal on Computing*, vol. 30, no. 2, pp. 402–420, 2018.

[62] J. Hopcroft and R. Tarjan, "Algorithm 447: Efficient algorithms for graph manipulation", *Communications of the ACM*, vol. 16, no. 6, pp. 372–378, 1973.

[63] A. Lodi, E. Malaguti, M. Monaci, G. Nannicini, and P. Paronuzzi, "Chance constrained problem with integer scenario variables", *Technical Report OR-19-7*, 2019, http://or.dei.unibo.it/technical-reports.

[64] A. Charnes, W. W. Cooper, and G. H. Symonds, "Cost horizons and certainty equivalents: An approach to stochastic programming of heating oil", *Management Science*, vol. 4, no. 3, pp. 235–263, 1958.

[65] J. Dupačová, A. Gaivoronski, Z. Kos, and T. Szántai, "Stochastic programming in water management: A case study and a comparison of solution techniques", *European Journal of Operational Research*, vol. 52, no. 1, pp. 28 –44, 1991.

[66] M. W. Tanner, L. Sattenspiel, and L. Ntaimo, "Finding optimal vaccination strategies under parameter uncertainty using stochastic programming", *Mathematical Biosciences*, vol. 215, no. 2, pp. 144–151, 2008.

[67] T. Watanabe and H. Ellis, "Stochastic programming models for air quality management", *Computers & Operations Research*, vol. 20, no. 6, pp. 651–663, 1993.

[68] X. Liu, S. Küçükyavuz, and J. Luedtke, "Decomposition algorithms for two-stage chance-constrained programs", English, *Mathematical Programming*, vol. 157, no. 1, pp. 219–243, 2014, ISSN: 0025-5610.

[69] S. Takriti and S. Ahmed, "On robust optimization of two-stage systems", *Mathematical Programming*, vol. 99, pp. 109–126, 2004.

[70] R. G. Jeroslow, "Representability in mixed integer programming, I: Characterization results", *Discrete Applied Mathematics*, vol. 17, no. 3, pp. 223–243, 1987.

[71] P. Bonami, A. Lodi, A. Tramontani, and S. Wiese, "On mathematical programming with indicator constraints", *Mathematical Programming*, vol. 151, no. 1, pp. 191–223, 2015.

[72] J. Luedtke, "A branch-and-cut decomposition algorithm for solving chance-constrained mathematical programs with finite support", *Mathematical Programming*, vol. 146, no. 1, pp. 219–244, 2014. DOI: 10.1007/s10107-013-0684-6. [Online]. Available: https://doi.org/10.1007/s10107-013-0684-6.

[73] M. Duran and I. Grossmann, "An outer-approximation algorithm for a class of mixed-integer nonlinear programs", *Mathematical Programming*, vol. 36, pp. 307–339, 1986.

[74] Y. Deng, S. Ahmed, J. Lee, and S. Shen, "Scenario grouping and decomposition algorithms for chance-constrained programs", *Available on Optimization Online http://www. optimizationonline. org/DB HTML/2017/02/5853. html*, 2017.

[75] J. "Luedtke, "Online supplement to: A branch-and-cut decomposition algorithm for solving chance-constrained mathematical programs", 2012, http://homepages.cae.wisc.edu/~luedtkej.

[76] E. Malaguti, M. Monaci, P. Paronuzzi, and U. Pferschy, "Integer optimization with penalized fractional values: The knapsack case", *European Journal of Operational Research*, vol. 273, no. 3, pp. 874–888, 2019.

[77] M. Pinedo, *Scheduling. Theory, Algorithms, and Systems*. New York: Springer-Verlag, 2012.

[78] C. Archetti and M. G. Speranza, "Vehicle routing problems with split deliveries", *International Transactions in Operational Research*, vol. 19, no. 1-2, pp. 3–22, 2012.

[79] E. Malaguti, R. M. Durán, and P. Toth, "Approaches to real world two-dimensional cutting problems", *Omega*, vol. 47, pp. 99–115, 2014.

[80] A. Lodi, S. Martello, M. Monaci, C. Cicconetti, L. Lenzini, E. Mingozzi, C. Eklund, and J. Moilanen, "Efficient two-dimensional packing algorithms for mobile WiMAX", *Management Science*, vol. 57, pp. 2130–2144, 2011.

[81] R. Freling, E. Romeijn, D. R. Morales, and A. Wagelmans, "A branch and price algorithm for the multi-period single-sourcing problem", *Operations Research*, vol. 51, no. 6, pp. 922–939, 2003.

[82] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Chichester, New York: John Wiley & Sons, 1990.

[83] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, 2004.

[84] A. Ceselli and G. Righini, "An optimization algorithm for a penalized knapsack problem", *Operations Research Letters*, vol. 34, no. 4, pp. 394–404, 2006.

[85] F. Della Croce, U. Pferschy, and R. Scatamacchia, "New exact approaches and approximation results for the penalized knapsack problem", *Discrete Applied Mathematics*, vol. to appear, 2018, doi:10.1016/j.dam.2017.11.023.

[86] C. Mandal, P. Chakrabarti, and S. Ghose, "Complexity of fragmentable object bin packing and an application", *Computers & Mathematics with Applications*, vol. 35, no. 11, pp. 91–97, 1998.

[87] M. Casazza and A. Ceselli, "Exactly solving packing problems with fragmentation", *Computers & Operations Research*, vol. 75, pp. 202–213, 2016.

[88] B. Byholm and I. Porres, "Fast algorithms for fragmentable items bin packing", Turku Centre for Computer Science, Tech. Rep. 1181, 2017.

[89] K. M. Bretthauer and B. Shetty, "The nonlinear knapsack problem–algorithms and applications", *European Journal of Operational Research*, vol. 138, no. 3, pp. 459–472, 2002.

[90] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[91] G. Nemhauser and Z. Ullmann, "Discrete dynamic programming and capital allocation", *Management Science*, vol. 15, pp. 494–505, 1969.

[92] R. Beier and B. Vöcking, "Random knapsack in expected polynomial time", *Journal of Computer and System Sciences*, vol. 69, no. 3, pp. 306–329, 2004.

[93] D. Pisinger, "A minimal algorithm for the multiple-choice knapsack problem", *European Journal of Operational Research*, vol. 83, no. 2, pp. 394–410, 1995.

[94] ——, "Where are the hard knapsack problems?", *Computers & Operations Research*, vol. 32, no. 9, pp. 2271–2284, 2005.

[95] V. Bo, M. Bortolini, E. Malaguti, M. Monaci, C. Mora, and P. Paronuzzi, "Models and algorithms for integrated production and distribution problems", *Technical Report OR-19-8*, 2019, http://or.dei.unibo.it/technical-reports.

[96] F. Harris, "How many parts to make at once", *Factory, the Magazine of Management*, vol. 10, pp. 135–136, 1913.

[97] C. H. Glock, E. H. Grosse, and J. M. Ries, "The lot sizing problem: A tertiary study", *International Journal of Production Economics*, vol. 155, pp. 39–51, 2014.

[98] J. Maes and L. Van Wassenhove, "Multi-item single-level capacitated dynamic lot-sizing heuristics: A general review", *Journal of the Operational Research Society*, vol. 39, no. 11, pp. 991–1004, 1988.

[99] B. Karimi, S. F. Ghomi, and J. Wilson, "The capacitated lot sizing problem: A review of models and algorithms", *Omega*, vol. 31, no. 5, pp. 365–378, 2003.

[100] L. Buschkühl, F. Sahling, S. Helber, and H. Tempelmeier, "Dynamic capacitated lot-sizing problems: A classification and review of solution approaches", *Or Spectrum*, vol. 32, no. 2, pp. 231–261, 2010.

[101] R. Jans and Z. Degraeve, "Meta-heuristics for dynamic lot sizing: A review and comparison of solution approaches", *European journal of operational research*, vol. 177, no. 3, pp. 1855–1875, 2007.

[102] ——, "Modeling industrial lot sizing problems: A review", *International Journal of Production Research*, vol. 46, no. 6, pp. 1619–1643, 2008.

[103] M. Florian, J. K. Lenstra, and A. Rinnooy Kan, "Deterministic production planning: Algorithms and complexity", *Management science*, vol. 26, no. 7, pp. 669–679, 1980.

[104] W.-H. Chen and J.-M. Thizy, "Analysis of relaxations for the multi-item capacitated lot-sizing problem", *Annals of operations Research*, vol. 26, no. 1, pp. 29–72, 1990.

[105] I. Barany, T. J. Van Roy, and L. A. Wolsey, "Strong formulations for multi-item capacitated lot sizing", *Management Science*, vol. 30, no. 10, pp. 1255–1261, 1984.

[106] G. D. Eppen and R. K. Martin, "Solving multi-item capacitated lot-sizing problems using variable redefinition", *Operations Research*, vol. 35, no. 6, pp. 832–848, 1987.

[107] J. M. Leung, T. L. Magnanti, and R. Vachani, "Facets and algorithms for capacitated lot sizing", *Mathematical programming*, vol. 45, no. 1-3, pp. 331–359, 1989.

[108] P. Eisenhut, "A dynamic lot sizing algorithm with capacity constraints", *AIIE transactions*, vol. 7, no. 2, pp. 170–176, 1975.

[109] J. Maes and L. N. Van Wassenhove, "A simple heuristic for the multi item single level capacitated lotsizing problem", *Operations Research Letters*, vol. 4, no. 6, pp. 265–273, 1986.

[110] H. Gu *et al.*, "Planning lot sizes and capacity requirements in a single stage production system", *European Journal of Operational Research*, vol. 31, no. 2, pp. 223–231, 1987.

[111] W. J. Selen and R. M. Heuts, "A modified priority index for günther's lot-sizing heuristic under capacitated single stage production", *European journal of operational research*, vol. 41, no. 2, pp. 181–185, 1989.

[112] Ö. Kirca and M. Kökten, "A new heuristic approach for the multi-item dynamic lot sizing problem", *European Journal of Operational Research*, vol. 75, no. 2, pp. 332–341, 1994.

[113] A. Dogramaci, J. C. Panayiotopoulos, and N. R. Adam, "The dynamic lot-sizing problem for multiple items under limited capacity", *AIIE transactions*, vol. 13, no. 4, pp. 294–303, 1981.

[114] R. Karni and Y. Roll, "A heuristic algorithm for the multi-item lot-sizing problem with capacity constraints", *IIE Transactions*, vol. 14, no. 4, pp. 249–256, 1982.

[115] W. W. Trigeiro, "A simple heuristic for lot sizing with setup times", *Decision Sciences*, vol. 20, no. 2, pp. 294–303, 1989.

[116] Y. Li, Y. Tao, and F. Wang, "An effective approach to multi-item capacitated dynamic lot-sizing problems", *International Journal of Production Research*, vol. 50, no. 19, pp. 5348–5362, 2012.

[117] A. Toscano, D. Ferreira, and R. Morabito, "A decomposition heuristic to solve the two-stage lot sizing and scheduling problem with temporal cleaning", *Flexible Services and Manufacturing Journal*, vol. 31, no. 1, pp. 142–173, 2019.

[118] R. Jans and Z. Degraeve, "Improved lower bounds for the capacitated lot sizing problem with setup times", *Operations Research Letters*, vol. 32, no. 2, pp. 185–195, 2004.

[119] Z. Degraeve and R. Jans, "A new dantzig-wolfe reformulation and branch-and-price algorithm for the capacitated lot-sizing problem with setup times", *Operations research*, vol. 55, no. 5, pp. 909–920, 2007.

[120] K. S. Hindi, K. Fleszar, and C. Charalambous, "An effective heuristic for the clsp with set-up times", *Journal of the Operational Research Society*, vol. 54, no. 5, pp. 490–498, 2003.

[121] S. Kang, K. Malik, and L. J. Thomas, "Lotsizing and scheduling on parallel machines with sequence-dependent setup costs", *Management science*, vol. 45, no. 2, pp. 273–289, 1999.

[122] R. A. Sandbothe and G. L. Thompson, "A forward algorithm for the capacitated lot size model with stockouts", *Operations Research*, vol. 38, no. 3, pp. 474–486, 1990.

[123] ——, "Decision horizons for the capacitated lot size model with inventory bounds and stockouts", *Computers & operations research*, vol. 20, no. 5, pp. 455–465, 1993.

[124] D. Aksen, K. Altınkemer, and S. Chand, "The single-item lot-sizing problem with immediate lost sales", *European Journal of Operational Research*, vol. 147, no. 3, pp. 558–566, 2003.

[125] P. Chandra and M. L. Fisher, "Coordination of production and distribution planning", *European Journal of Operational Research*, vol. 72, no. 3, pp. 503–517, 1994.

[126] A. N. Haq, P. Vrat, and A. Kanda, "An integrated production-inventory-distribution model for manufacture of urea: A case", *International Journal of production economics*, vol. 25, no. 1-3, pp. 39–49, 1991.

[127] K. S. Bhutta, F. Huq, G. Frazier, and Z. Mohamed, "An integrated location, production, distribution and investment model for a multinational corporation", *International Journal of Production Economics*, vol. 86, no. 3, pp. 201–216, 2003.

[128] J. K. Jolayemi and F. O. Olorunniwo, "A deterministic model for planning production quantities in a multi-plant, multi-warehouse environment with extensible capacities", *International Journal of Production Economics*, vol. 87, no. 2, pp. 99–113, 2004.

[129] M. Rao, "Optimal capacity expansion with inventory", *Operations Research*, vol. 24, no. 2, pp. 291–300, 1976.

[130] S. Rajagopalan and J. M. Swaminathan, "A coordinated production planning model with capacity expansion and inventory management", *Management Science*, vol. 47, no. 11, pp. 1562–1580, 2001.

[131] J. R. Bradley and B. C. Arntzen, "The simultaneous planning of production, capacity, and inventory in seasonal demand environments", *Operations Research*, vol. 47, no. 6, pp. 795–806, 1999.

[132] E. J. Anderson and B. S. Cheah, "Capacitated lot-sizing with minimum batch sizes and setup times", *International Journal of Production Economics*, vol. 30, pp. 137–152, 1993.

[133] M. Constantino, "Lower bounds in lot-sizing models: A polyhedral study", *Mathematics of Operations Research*, vol. 23, no. 1, pp. 101–118, 1998.

[134] C Mercé and G Fontan, "Mip-based heuristics for capacitated lotsizing problems", *International Journal of Production Economics*, vol. 85, no. 1, pp. 97–111, 2003.

[135] H. Tempelmeier, "A column generation heuristic for dynamic capacitated lot sizing with random demand under a fill rate constraint", *Omega*, vol. 39, no. 6, pp. 627–633, 2011.

[136] V. A. Armentano, P. M. França, and F. M. de Toledo, "A network flow model for the capacitated lot-sizing problem", *Omega*, vol. 27, no. 2, pp. 275–284, 1999.

[137] H. Chen, "Fix-and-optimize and variable neighborhood search approaches for multi-level capacitated lot sizing problems", *Omega*, vol. 56, pp. 25–36, 2015.

[138] S. Helber and F. Sahling, "A fix-and-optimize approach for the multi-level capacitated lot sizing problem", *International Journal of Production Economics*, vol. 123, no. 2, pp. 247–256, 2010.

[139] P. Belotti, P. Bonami, M. Fischetti, A. Lodi, M. Monaci, A. Nogales-Gómez, and D. Salvagnin, "On handling indicator constraints in mixed integer programming", *Computational Optimization and Applications*, vol. 65, pp. 545–566, 2016.