Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Curriculum in Ingegneria Informatica, Gestionale e dell'Automazione

Pricing-based primal and dual bounds for selected packing problems

Ph.D. Dissertation of:
Andrea Pizzuti

Advisor:
Prof. Fabrizio Marinelli

Curriculum Supervisor:
Prof. Claudia Diamantini

XXXII edition - new series

Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Curriculum in Ingegneria Informatica, Gestionale e dell'Automazione

Pricing-based primal and dual bounds for selected packing problems

Ph.D. Dissertation of:
Andrea Pizzuti

Advisor:
Prof. Fabrizio Marinelli

Curriculum Supervisor:
Prof. Claudia Diamantini

XXXII edition - new series

R. & S.

Acknowledgments

Although in my academic path this is the third thesis that I compose, actually it is the first time in which I have decided to insert acknowledgements in one of them. This is not meant to be disrespectful for people who have accompanied me during the Bachelor and Master courses, nor is caused by the lack of wish to thank them for their unshakable support. Simply, this is the earliest time in which I can feel a kind of sincere emotion (viz. sense of satisfaction) in ending the present route, without losing to the compromise of "you should be touched by the incoming goal". For this reason, even if it could sound excessive (or just inappropriate) for a Ph.D. thesis, I want to dedicate few truthful words in acknowledgements.

Firstly, I want to affirm my enormous gratitude for my supervisor Fabrizio. He was a solid and priceless landmark in guiding my research activity since my very first steps into the field of research with my master thesis. Moreover, to me he has always been a lighthouse for his strong and admirable ethic, for which I can only feel the highest of respect and estimation. Finally, and most important, he allowed me to find a path that, even if surrounded by difficulties and uncertainties, I have a sincere desire to continue.

Then, I wish to thank my friends. There is no need to report their names, I know they will naturally understand. Some more than others, each of them supported me since I started my studies at university, when I felt as wandering without a precise reason and questioning about my choices. To you, I owe my (little) serenity, the strength of holding on, love and bonds that go beyond what I ever hoped or deserved. Truly, thank you.

Finally, I want to acknowledge my family. Simply, for everything.

Ancona, Febbraio 2020

Andrea Pizzuti

Abstract

Several optimization problems ask for finding solutions which define *packing* of elements while maximizing (minimizing) the objective function. Solving some of these problems can be extremely challenging due to their innate complexity and the corresponding integer formulations can be not suitable to be solved on instances of relevant size. Thus, clever techniques must be devised to achieve good primal and dual bounds. A valid way is to rely on *pricing*-based algorithms, in which solution components are generated by calling and solving appropriate optimization subproblems. Two main exponents of this group are the *delayed column generation* (CG) procedure and the *sequential value correction* (SVC) heuristic: the former provides a dual bound based on the generation of implicit columns by examining the shadow prices of hidden variables; the latter explores the primal solution space by following the dynamic of approximate prices.

In this thesis we focus on the application of SVC and CG techniques to find primal and dual bounds for selected packing problems. In particular, we study problems belonging to the family of *cutting and packing*, where the classical BIN PACKING and CUTTING STOCK are enriched with features derived from the real manufacturing environment. Moreover, the MAXIMUM γ-QUASI-CLIQUE problem is taken into account, in which we seek for the induced γ-quasi-clique with the maximum number of vertices. Computational results are given to assess the performance of the implemented algorithms.

Sommario

Un ampio insieme di problemi di ottimizzazione richiedono l'individuazione di soluzioni che definiscono un *packing* di elementi, al fine di massimizzare (minimizzare) la relativa funzione obiettivo. Risolvere all'ottimo alcuni di questi problemi può essere estremamente oneroso a causa della loro complessità innata e le corrispondenti formulazioni intere possono non essere appropriate per approcciare istanze di dimensioni rilevanti. Pertanto, il calcolo di bound primali e duali qualitativamente validi necessita della definizione di tecniche ingegnose ed efficaci. Una metodologia valevole consiste nell'impiego di algoritmi basati su *pricing*, in cui le componenti delle soluzioni sono generate tramite la risoluzione di sottoproblemi di ottimizzazione specifici. Due principali esponenti di questa famiglia sono la procedura *delayed column generation* (CG) e l'euristica *sequential value correction* (SVC): la prima fornisce un bound duale basato sulla generazione delle colonne implicite tramite la valutazione dei prezzi ombra di variabili nascoste; la seconda esplora lo spazio delle soluzioni primali seguendo la dinamica di prezzi approssimati.

In questa tesi ci concentriamo sull'applicazione di tecniche SVC e CG per l'individuazione di bound primali e duali per problemi di packing selezionati. In particolare, studiamo i problemi appartenenti alla famiglia del *cutting e packing*, dove i ben noti BIN PACKING e CUTTING STOCK sono arricchiti con caratteristiche derivanti dagli ambienti produttivi reali. Inoltre, studiamo il problema della MAXIMUM γ-QUASI-CLIQUE in cui si cerca la γ-quasi-clique indotta che massimizzi il numero di vertici selezionati. Risultati computazionali sono presentati al fine di validare le performance degli algoritmi implementati.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The definition of many optimization problems is suggested and/or supported by the necessity of solving practical issues encountered in different fields, such as economic, biologic and social environments to be not exhaustive. A broad group of these problems can be ascribed to the family of the SET PACKING PROBLEM (SPP) in which, given a set of elements, the objective claims to maximize (minimize) a cost function associated to the assortments of subsets, so that a formal description of subset feasibility is fulfilled and the solution defines a *packing* of the elements. More specifically, the selection of subsets must be done in order to ensure an empty intersection between subsets so that each element is packed at most once. According to the computational complexity theory, the decision version of the SPP is a classical NP-complete problem and was included in the Karp's 21 NP-complete problems [75]. Thus, the corresponding optimization problems are extremely challenging to be solved.

To give a practical application of SPP, one can refer to the CREW SCHEDULING PROBLEM in airlines. In this problem crews must be assigned to airplanes in the fleet, so that flights can proceed in respect of the compiled timetables and the shifts of crew members meet the specifications of the labor contract. Crews are composed by staff members according to the skills and traits of the employers, such as the competence in piloting different types of aircrafts and the personal affinity with other crew members. Subsets of elements are built on all the possible crew and plane assignments, so that crew members and aircrafts are not shared among different concurrent flights. This corresponds to a packing of crews and planes.

Based on the statement of SPP one can give a general abstraction of problems that emerge from very different environments, skimming from the details that are applications-dependent. However, such details can be used to sketch an informal typology of SPP subfamilies by classifying the elements of the packing and interpreting the meaning behind the requirements of subsets. Among the wide family of SPPs, we focused on problems that are members of two subfamilies: the *geometric packing* problems and the *structure packing* problem. In the geometric packing elements are items characterized by geometric sizes that must be inserted within larger objects of limited sizes. The feasibility of subsets is defined according to physical criteria that reflects the practical rules of a packing, such as the whole containment of items within boundaries or the non-overlapping between items. A noteworthy example of geometric packing is represented by the BIN PACKING PROBLEM (BPP) in which r-dimensional items must be placed into r-dimensional bins and the number of employed bins for the packing is minimized.

With structure packing we refer to the group of SPPs defined on graphs that ask for the search of graph substructures meeting packing constraints. The elements of the problem are generally represented by vertices and edges of the underlying graph, and the feasibility of subsets is established according to the fulfillment of requirements based (for instance) on measures of distance, density and/or connectivity. An essential optimization problem of this category is the NODE PACKING PROBLEM, also known as VERTEX PACKING, MAXIMAL INDEPENDENT SET or STABLE SET problems [98]. Here we look for a maximum subset of vertices such that, for each selected vertex, the set of edges incident on such vertex has an empty intersection with the sets of incident edges on all the other selected vertices; that is, each edge is incident on at most one of the selected vertices.

Our goal in this thesis is to study problems belonging to the SPP family and devise approaches to obtain primal and dual bounds for them. In particular, the methods implemented for SPP solution are the so called *delayed column generation* technique and *sequential value correction* heuristic, which make use of the concept of *reduced costs* (in a direct or indirect way) and are related to the solution of *pricing* subproblems. A formal introduction to these methods is given over the rest of the chapter, passing through notions of linear programming and integer reformulation.

1.1 Some preliminaries on linear programming

Mathematical programming can be intended as a general advanced language employed to describe optimization problems in a formal way. It is possible to classify problems by looking at the "shape" of a mathematical formulation and mathematical programming represents a valid solution methodology.

Before discussing the techniques employed in this thesis, we give some basic elements about general linear programming problem [116]. The elements that compose a linear formulation are a vector of *decision variables* $\mathbf{x} \in \mathbb{R}_n$ used to represent the decision in a problem, an *objective function* $\mathbf{c}^T\mathbf{x}$ to be minimized (maximized) where $\mathbf{c} \in \mathbb{R}_n$ is the cost vector, a set of *constraints* $\mathbf{Ax} = \mathbf{b}$ that defines the *feasible region* expressed compactly, with the matrix of coefficient $\mathbf{A} \in \mathbb{R}_{m\times n}$ and the coefficient vector $\mathbf{b} \in \mathbb{R}_m$. The set $P = \{\mathbf{x} \in \mathbb{R}_n | \mathbf{Ax} = \mathbf{b}\}$

is called *polyhedron*.

Linear program (LP) are characterized by continuous variables, linear constraints and objective function. When the decision variables are subjected to integrality, i.e. $\mathbf{x} \in \mathbb{Z}$, the result is an integer program (IP). Finally, if a formulation presents both continuous and integer variables, a mixed integer program (MIP) is addressed.

The linear programming problem $\mathcal{P}$ in *standard form* can be written as

$$\begin{aligned} \min \mathbf{c}^T\mathbf{x} & \\ \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0}. \end{aligned}$$

A solution $\mathbf{x}^* \in P$ is optimal if it satisfies all the constraints and reaches the global minimum of the objective function.

Given the *primal* LP problem $\mathcal{P}$, the *dual* formulation $\mathcal{D}$ reads as

$$\begin{aligned} \max \mathbf{y}^T\mathbf{b} & \\ \mathbf{y}^T\mathbf{A} &\leq \mathbf{c}^T. \end{aligned}$$

Independently on the chances that one has to find an optimum solution, the dual bounds that can be obtained via linear programming are very useful in practice and can provide a measure of the efficiency or effectiveness of the operational solution.

From the linear programming theory, a solution of $\mathcal{P}$ is optimal if all the *reduced costs* are non-negative, that is

$$\mathbf{c}^T - \mathbf{y}^T\mathbf{A} \geq 0. \tag{1.1}$$

Intuitively, the j-th reduced cost indicates the cost in terms of objective function value to increase variable x_j by a small amount, whereas the j-th component of the dual vector y_j can be interpreted as the *marginal cost* or *shadow price* to be paid for increasing b_j by one unit. As you will see in the next sections, reduced costs have a crucial role in column generation procedure and inspired the sequential value correction heuristic framework.

We conclude this section by introducing an essential result from linear programming theory that is preparatory to the comprehension of the subsequent contents.

A solution vector $\mathbf{x} \in P$ is an *extreme point* of P if it is not possible to express it as a convex combination of two other vectors $\mathbf{v}_1, \mathbf{v}_2 \in P$; that is, for any scalar $\alpha \in [0,1]$ there are no $\mathbf{v}_1, \mathbf{v}_2 \in P$ such that $\mathbf{x} = \alpha\mathbf{v}_1 + (1-\alpha)\mathbf{v}_2$. For any fixed vector $\mathbf{v} \in P$, the *recession cone* is defined as the set of all the directions

$\mathbf{d}$ that start from $\mathbf{v}$ and move indefinitely away from it without leaving P, formally

$$\{\mathbf{d} \in \mathbb{R}_n | \mathbf{A}\mathbf{d} = \mathbf{0}, \mathbf{d} \geq \mathbf{0}\}.$$

Any non-zero vector of the recession cone is called *ray* and those rays that satisfy $n-1$ linear independent constraints to equality are defined as *extreme rays*. The extreme rays of the recession cone associated to a non-empty polyhedron P are referred as the extreme rays of P.

The fundamental result of linear programming theory is the following Weyl-Minkowski theorem, also known as *resolution theorem* [116]:

Theorem 1. *Let P a non-empty polyhedron with at least one extreme point. Given the set of extreme points $\boldsymbol{v}^1, \ldots, \boldsymbol{v}^{|Q|}$ and the set of extreme rays $\boldsymbol{u}^1, \ldots, \boldsymbol{u}^{|R|}$ of P, then each point $\boldsymbol{x} \in P$ can be expressed as*

$$\boldsymbol{x} = \sum_{q \in Q} \beta_q \boldsymbol{v}_q + \sum_{r \in R} \gamma_r \boldsymbol{u}_r \tag{1.2}$$

with $\sum_{q \in Q} \beta_q = 1$, $\boldsymbol{\beta} \in \mathbb{R}^+_{|Q|}$ and $\boldsymbol{\gamma} \in \mathbb{R}^+_{|R|}$.

Informally, theorem 1 states that each point of a polyhedron can be expressed as a convex combination of the extreme points plus a non-negative combination of the extreme rays.

1.2 Dantzig-Wolfe decomposition

The Dantzig-Wolfe decomposition principle was firstly introduced in 1960 [45] in order to cope with large scale linear programming problems by extending the applicability of LP solution techniques. Indeed, the general idea behind decomposition is to exploit the structure of the problem to reformulate it in an equivalent fashion, which is however convenient under certain points of views (e.g. quality of linear relaxation, computational burden in solution). The effectiveness of such technique was largely assessed throughout the years since its first application by Gilmore and Gomory [62, 63]. Thus, the decomposition approach became a standard methodology to whom several papers and book chapters are dedicated [99, 118].

Let us consider the following LP, usually called *compact formulation* ($\mathcal{L}$):

$$\min \mathbf{c}^T \mathbf{x} \tag{1.3}$$

$$\mathbf{A}\mathbf{x} \geq \mathbf{b} \tag{1.4}$$

$$\mathbf{D}\mathbf{x} \geq \mathbf{d} \tag{1.5}$$

$$\mathbf{x} \geq \mathbf{0}. \tag{1.6}$$

Let moreover assume that the polyhedron $P = \{\mathbf{x} \in \mathbb{R}_n^+ | \mathbf{Dx} \geq \mathbf{d}\}$ is non-empty. By theorem 1, we can substitute $\mathbf{x} \in P$ and rewrite formulation $\mathcal{L}$ in the equivalent *extensive formulation* ($\mathcal{R}_{\mathcal{L}}$):

$$\min \sum_{q \in Q} \bar{c}_q \beta_q + \sum_{r \in R} \bar{c}_r \gamma_r \tag{1.7}$$

$$\sum_{q \in Q} \bar{\boldsymbol{a}}_q \beta_q + \sum_{r \in R} \bar{\boldsymbol{a}}_r \gamma_r \geq \boldsymbol{b} \tag{1.8}$$

$$\sum_{q \in Q} \beta_q = 1 \tag{1.9}$$

$$\beta \in \mathbb{R}^+_{|Q|} \tag{1.10}$$

$$\gamma \in \mathbb{R}^+_{|R|} \tag{1.11}$$

where $\bar{c}_q = \mathbf{c}^T \mathbf{v}_q$ and $\bar{\boldsymbol{a}}_q = \mathbf{A}\mathbf{v}_q$ for $q \in Q$, $\bar{c}_r = \mathbf{c}^T \mathbf{u}_r$ and $\bar{\boldsymbol{a}}_r = \mathbf{A}\mathbf{u}_r$ for $r \in R$.

Formulation $\mathcal{R}_{\mathcal{L}}$ has a reduced number of constraints with respect to $\mathcal{L}$ as (1.5) disappeared and only the *convexity constraint* (1.9) is added. Nevertheless, the number of variables may become exponential due to the number of extreme points (rays) and solving $\mathcal{R}_{\mathcal{L}}$ can be impractical. Hence, a column generation procedure is usually employed to solve $\mathcal{R}_{\mathcal{L}}$ by considering only a subset of columns, while missing columns are generated on the fly when needed. Details about column generation algorithm are given later, we now focus on the integer akin to the decomposition for IPs.

1.2.1 Decomposition of IP

Let us introduce the following IP ($\mathcal{I}$):

$$\min \mathbf{c}^T \mathbf{x} \tag{1.12}$$

$$\mathbf{Ax} \geq \mathbf{b} \tag{1.13}$$

$$\mathbf{x} \in X \tag{1.14}$$

where $X = P \cap \mathbb{Z}^+$ and P is defined as above.

When reformulating an IP or an MIP, decomposition can be performed accordingly to two main techniques: *convexification* or *discretization* [83]. The convexification is based on replacing X with its convex hull (conv(X)) and expressing $\mathbf{x}$ in terms of the extreme points and extreme rays of conv(X). However, after applying convexification the integrality constraints on original variables are still required and such variables cannot be put aside. Indeed, it is insufficient to enforce integrality on variables β and γ to obtain an equivalent representation of $\mathcal{I}$, as its optimal integer solution could be an interior point of conv(X). Nevertheless, a remarkable exception is given for $X \subseteq [0,1]^n$ since all its integer points are vertices of conv(X). Generally speaking, to obtain a true integer equivalent of $\mathcal{I}$ one must rely on discretization, which is based on the extension of theorem 1 for the integer case [99]:

Theorem 2. *Let P a non-empty polyhedron and $X = P \cap \mathbb{Z}^+$. Then, there exists a finite set of integer points $\boldsymbol{v}^1, \dots, \boldsymbol{v}^{|Q|} \subseteq X$ and the set of integer rays $\boldsymbol{u}^1, \dots, \boldsymbol{u}^{|R|} \subseteq P$ such that each point $\boldsymbol{x} \in X$ can be expressed as*

$$\boldsymbol{x} = \sum_{q \in Q} \beta_q \boldsymbol{v}_q + \sum_{r \in R} \gamma_r \boldsymbol{u}_r \tag{1.15}$$

with $\sum_{q \in Q} \beta_q = 1$, $\boldsymbol{\beta} \in \mathbb{Z}^+_{|Q|}$ and $\boldsymbol{\gamma} \in \mathbb{Z}^+_{|R|}$.

Theorem 2 allows to express each point of a polyhedron as a convex combination of integer points plus a non negative combination of integer rays. We can then reformulate $\mathcal{I}$ in the equivalent *extensive formulation* ($\mathcal{R}_{\mathcal{I}}$):

$$\min \sum_{q \in Q} \bar{c}_q \beta_q + \sum_{r \in R} \bar{c}_r \gamma_r \tag{1.16}$$

$$\sum_{q \in Q} \bar{\boldsymbol{a}}_q \beta_q + \sum_{r \in R} \bar{\boldsymbol{a}}_r \gamma_r \geq \boldsymbol{b} \tag{1.17}$$

$$\sum_{q \in Q} \beta_q = 1 \tag{1.18}$$

$$\boldsymbol{\beta} \in \mathbb{Z}^+_{|Q|} \tag{1.19}$$

$$\boldsymbol{\gamma} \in \mathbb{Z}^+_{|R|} \tag{1.20}$$

where $\bar{c}_q = \mathbf{c}^T \mathbf{v}_q$ and $\bar{\boldsymbol{a}}_q = \mathbf{A}\mathbf{v}_q$ for $q \in Q$, $\bar{c}_r = \mathbf{c}^T \mathbf{u}_r$ and $\bar{\boldsymbol{a}}_r = \mathbf{A}\mathbf{u}_r$ for $r \in R$.

1.2.2 The case of block-angular structure

It frequently happens that problems present a matrix coefficient $\mathbf{D}$ with a modular structure that can be exploited when applying decomposition techniques. Let K be the set of independent blocks in which $\mathbf{D}$ can be partitioned. Figure 1.1 gives an insight of the block-angular structure.

We can identify $|K|$ disjointed polyhedra $P^k = \{\mathbf{x}^k \in \mathbb{R}^+_{n^k} | \mathbf{D}^k \mathbf{x}^k \geq \mathbf{d}^k\}$ such that $X^k = P^k \cap \mathbb{Z}^+$. Then, $\mathcal{I}$ can be rewritten in

$$\min \sum_{k \in K} (\mathbf{c}^k)^T \mathbf{x}^k \tag{1.21}$$

$$\sum_{k \in K} \mathbf{A}^k \mathbf{x}^k \geq \mathbf{b} \tag{1.22}$$

$$\mathbf{x}^k \in X^k \quad \forall k \in K. \tag{1.23}$$

Proceeding by discretization over each X^k, we attain the integer reformulation

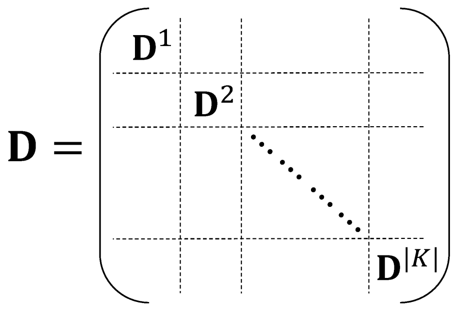

Figure 1.1: Dotted lines separate each subblock, whose non-zero components do not share row and column indices with the non-zero components of other subblocks.

$\mathcal{R}_{\mathcal{I}}$:

$$\min \sum_{k \in K} \left(\sum_{q \in Q^k} \bar{c}^k_q \beta^k_q + \sum_{r \in R^k} \bar{c}^k_r \gamma^k_r \right) \tag{1.24}$$

$$\sum_{k \in K} \left(\sum_{q \in Q^k} \bar{\boldsymbol{a}}^k_q \beta^k_q + \sum_{r \in R^k} \bar{\boldsymbol{a}}^k_r \gamma^k_r \right) \geq \boldsymbol{b} \tag{1.25}$$

$$\sum_{q \in Q^k} \beta^k_q = 1 \qquad \forall k \in K \tag{1.26}$$

$$\beta^k \in \mathbb{Z}^+_{|Q^k|} \qquad \forall k \in K \tag{1.27}$$

$$\gamma^k \in \mathbb{Z}^+_{|R^k|} \qquad \forall k \in K \tag{1.28}$$

where superscript k specifies for each element the relative substructure $k \in K$ (e.g., $\bar{\boldsymbol{a}}^k_q = \mathbf{A}^k \mathbf{v}^k_q$, Q^k et cetera).

We now move the discussion to the column generation topic, a technique that can be necessary to overcome the challenge of solving the LP relaxation of formulations with a huge number of variables as $\mathcal{R}_{\mathcal{I}}$.

1.3 Column generation

A clever way to deal with an overwhelming number of columns in formulations is limiting the mathematical program to a restricted subset of columns, while considering all the remaining ones only implicitly. A suggestion of such strategy was firstly given by Ford and Furkenson [58] in 1958, which was later put into practice by Gilmore and Gomory [62] in the cutting stock problem context by implementing a *delayed column generation* procedure (CG) [83]. It is common to refer to CG with the acceptation of delayed to highlight the peculiarity of the scheme, which implicitly enumerates the columns in place of a full a priori description of them in the formulation.

Given an LP formulation restricted to a subset of columns, namely *restricted master problem*, a CG algorithm performs a search of columns (variables) among the ones that are not explicitly included. The search looks for non-basic variables to be added that have reduced costs able to possibly improve the objective function value. This functioning reflects the selection of pivot elements in the simplex method and lay the groundwork on the linear programming theory. Typically, the search of profitable columns is performed by relying on a *pricing subproblem* that is fed with the values of dual variables taken from the current optimal solution of the restricted master problem. The solutions of the pricing subproblem are used to generate the missing columns, which are added to the restricted master problem. The dual variables are then updated by solving the restricted master problem and the whole loop is iterated, until the global optimality of the restricted master problem is certified by fulfilling condition (1.1).

To formalize, let us indicate $\mathcal{R}_{\mathcal{I}}$ as *master problem*. Moreover, let us call $\tilde{\mathcal{R}}_{\mathcal{I}}$ the restricted master problem obtained by relaxing the integrality constraints in formulation $\mathcal{R}_{\mathcal{I}}$, while restricting variables β^k and γ^k taken into account to the subsets of integer points $\tilde{Q}^k \subseteq Q^k$ and integer rays $\tilde{R}^k \subseteq R^k$ for each $k \in K$. Finally, let us refer to the dual variables of constraints (1.25) and (1.26) as θ and ψ, respectively.

The reduced cost of a column in $\mathcal{R}_{\mathcal{I}}$ for a fixed $k \in K$ is generally expressed as

$$\begin{aligned} \left[(\mathbf{c}^k)^T - \theta \mathbf{A}^k\right] \mathbf{v}_q^k - \psi_k & \qquad \forall q \in Q^k \\ \left[(\mathbf{c}^k)^T - \theta \mathbf{A}^k\right] \mathbf{u}_r^k & \qquad \forall r \in R^k \end{aligned}$$

for integer points q and integer rays r. Given a dual optimal solution (θ^*, ψ^*) of $\tilde{\mathcal{R}}_{\mathcal{I}}$, the k-th pricing problem $Pr(k)$ called to find the most profitable column for block k is:

$$\Omega^k = \min \left[(\mathbf{c}^k)^T - \theta^* \mathbf{A}^k\right] \mathbf{p} \quad - \quad \psi^* \tag{1.29}$$

$$\mathbf{D}^k \mathbf{p} \quad \geq \quad \mathbf{d}^k \tag{1.30}$$

$$\mathbf{p} \quad \in \quad \mathbb{Z}^+_{n^k}. \tag{1.31}$$

For $\Omega^k < 0$ and finite, the optimal solution vector $\mathbf{p}^*$ is an integer point $\mathbf{v}_q^k$ to be added to $\tilde{Q}^k$. The corresponding column β_q^k, with coefficients $\left[\bar{c}_q^k, \bar{\mathbf{a}}_q^k, 1\right]^T$, is generated and added to $\tilde{\mathcal{R}}_{\mathcal{I}}$. When $Pr(k)$ is unbounded, $\mathbf{p}^*$ represents an integer ray $\mathbf{u}_q^k$ to append into $\tilde{R}^k$. In this case, the column γ_q^k is generated with coefficient $\left[\bar{c}_q^k, \bar{\boldsymbol{a}}_q^k, 0\right]^T$ and $\tilde{\mathcal{R}}_{\mathcal{I}}$ is properly updated. Finally, if $\Omega^k \geq 0$ for each $k \in K$, then all reduced costs are non-negative and condition (1.1) is satisfied, so that $\tilde{\mathcal{R}}_{\mathcal{I}}$ is solved to optimality. Clearly, the optimal value of $\tilde{\mathcal{R}}_{\mathcal{I}}$ is a valid lower bound for $\mathcal{R}_{\mathcal{I}}$.

$Pr(k)$ is an IP that can be challenging to solve, thus the performance of the whole CG procedure can be compromised. To accelerate the convergence of the algorithm, one can rely on efficient heuristics or combinatorial algorithms to generate profitable columns, even more than one concurrently for each pricing subproblem. However, (1.1) need to be certified by solving $Pr(k)$ exactly for all $k \in K$.

Unfortunately, it happens quite often that the convergence of the CG procedure requires several iterations, since the values of the dual variables follow an oscillatory dynamic [51] and the optimal solution of $\tilde{\mathcal{R}}_{\mathcal{I}}$ can be degenerate. The phenomenon of the slow convergence is generally indicated as *tailing-off effect*. To overcome this issue and improve the CG performance, several stabilization techniques have been developed, e.g. the box step method [90], bundle methods [69] and primal-dual stabilization strategies [94, 21].

The CG procedure can be embedded in the branch-and-bound framework to find optimal integer solutions of $\mathcal{R}_{\mathcal{I}}$, resulting in the so called *branch-and-price* scheme [15, 128]. Alternatively, algorithms based on CG and designed ad hoc for integer programming problems have been presented in literature. Applications of integer programming CG can be found for the VEHICLE ROUTING PROBLEM [48], the CREW SCHEDULING PROBLEM [50], the TRAVELING SALESMAN PROBLEM [49], the STABLE SET PROBLEM [28] to mention a few. A useful list of these applications is reported in [83].

1.4 Sequential value correction heuristic

The *sequential value correction heuristic* (SVC) is a successful technique largely employed to obtain good primal solutions for BPP and CSP in limited amount of time. Firstly introduced in Mukhachiova and Zalgaller [96], the SVC belongs to the family of sequential heuristics for BPP and CUTTING STOCK PROBLEM (CSP) that construct one pattern per time accordingly to a customized strategy [65, 139, 126, 77]. Indeed, it can be classified as an iterative greedy method in which the space of search is probed following the trajectory described by the dynamic of the so-called *pseudo-prices*. Recall that in the one-dimensional CSP, which generalizes the NP-hard bin packing problem [61], a set of small items I with length l_i have to be cut from large stock items of size l, so that the corresponding demand r_i is fulfilled and the total produc-

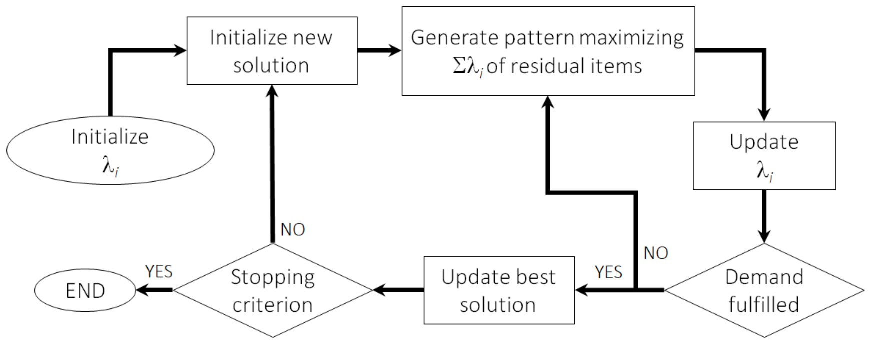

Figure 1.2: SVC flow chart

tion waste is minimized. A more precise formalization of this combinatorial optimization problem will be given in §2, we here contextualize to introduce the basic notation.

In the SVC framework (see figure 1.2), a pseudo-price λ_i is defined as a value associated to each item $i \in I$. After the initialization, a pattern is built by solving an appropriate pricing subproblem in which items are inserted in order to maximize the total sum of the selected item pseudo-prices. Such functioning clearly recall a similarity between the pseudo-prices and the role of the shadow prices in the CG procedure, that guide the generation of new profitable patterns (by which the name pseudo-price derives). Once no more items can be inserted in a pattern, an updating phase modifies the values of the pseudo-prices by perturbing them according to formulae that generally take into account the quality of the last computed pattern. Then, the level of activation of the pattern is decided and if the demand is not completely fulfilled, a new pattern is generated by iterating the process. Otherwise, the solution found is evaluated and it is recorded as best incumbent if improving the objective function. The algorithm ends if the stopping criteria (e.g., time limit, number of iterations or lower bounds) are met, or alternatively the whole process is repeated starting from the initialization of a new empty solution. Typically, solutions found by the SVC algorithm describe a dynamic of the objective function values characterized by a rapid decrease in the first iterations, followed by fluctuations nearby the optimal value for the subsequent ones.

The general description of the SVC heuristic was translated into different implementation across the years and the main differences usually emerge in the initialization and updating phases of the λ_i terms. The definitions of both steps can be specified in a wide spectrum of possibilities that generally reflect the features of the application and are customized on the specific problem to achieve high performances.

The first application of SVC was presented by Mukhacheva and Zalgaller [96], who proposed it in the contexts (among others) of cutting forming problems and problems of cutting totality planning with intensities of their application. For the sake of exemplification, some details of their deterministic implementation are reported. The exploited formulae are based on the measure of trim losses ω^p of pattern $p \in P$ and material consumption of each item $i \in I$. Let λ_i^0 be the starting value of the pseudo-price of item i and λ_i^h the corresponding value after the computation of the h-th pattern. Given a CSP solution computed by means of *first-fit decreasing* (FFD) algorithm with N^{FDD} patterns, the authors initialize the pseudo-prices as

$$\lambda_i^0 = \frac{l_i l}{r_i} \sum_{p=1}^{N^{FDD}} \frac{a_i^p}{l - \omega^p} \qquad i \in I \tag{1.32}$$

where a_i^p gives the number of parts i produced in pattern $p \in P$. Intuitively, for each pattern its trim loss is distributed across the width of the part-types that belong to it, in a way that is proportional to the number of times each item occurs.

After that a pattern p is completed by maximizing the total values of selected items, the pseudo-prices used for the following $p+1$-th pattern are set by

$$\lambda_i^{p+1} = \lambda_i^p (1 - \frac{a_i^p}{r_i}) + \frac{a_i^p}{r_i} \frac{l_i l}{l - \omega^p} \qquad i \in I \tag{1.33}$$

in which the formula defines each new pseudo-price value to be the weighted sum of its previous value and the material consumption norm within the last obtained pattern.

Some years later the SVC scheme was employed again by Mukhacheva et al. [97] in one-dimensional CSP, and by Verkhoturov and Sergeyeva [131] for the two-dimensional CSP with irregular shapes. The latter dealt the issue of the item placement by defining the pseudo-prices as functions of the angle of rotation and such angles are chosen in order to maximize the pseudo-price values. Still, the formulae are deterministic, whereas in the former [97] the method was randomized to diversify the search and prevent cycling. Indeed, the pseudo-price formulation was modified by introducing the random parameter ξ as

$$\lambda_i^{p+1} = \frac{\lambda_i^p \xi (b_i + r_i)}{\xi (b_i + r_i) + a_i^p} + \frac{a_i^p}{\xi (b_i + r_i) + a_i^p} \frac{l_i l}{l - \omega^p} \qquad i \in I \tag{1.34}$$

where b_i is the current residual demand of part-type i. The randomized SVC was embedded in a modified branch-and-bound scheme and, even if it required the tuning of bounds for parameter ξ, (1.34) proved its effectiveness by improving the performance with respect to the deterministic counterpart (1.33).

Relevant subsequent implementations of the SVC can be found in Belov and Scheithauer [18], where the heuristic was embedded in a cutting plane algorithm to solve the one-dimensional CSP with multiple stock widths and the pseudo-prices are initialized with the value of the dual variables of the LP relaxation solution; in [17] by the same authors, in which the SVC is called to solve the residual problem that arises after rounding the LP solutions within a branch-and-cut-and-price scheme. This algorithm is based on strengthening the LP relaxation of branch-and-price nodes by applying Chvátal-Gomory and Gomory mixed-integer cuts [100]. It was applied both to one-dimensional and two-dimensional two-stage guillotine constrained CSP.

In [19] the pseudo-prices updating phase is modified in order to balance the distribution of large items across the sequence of generated patterns. Indeed, the concentration of small items generally gives patterns with a high total price and low trim-loss, that however limits the possibilities to get good patterns with the remaining large items. Inspired by the *max-len-mini-items* heuristic of Kupke [77] that limits the number of items in each pattern, the SVC was thus changed by using pseudo-prices over-proportional to the item widths, so that prices of large items are supported and patterns with mixed combinations of widths are favored. The SVC heuristic in [19] is moreover customized to deal with the multi-objective CSP, in which the minimization of setups and open-stacks is asked along with the reduction of trim-loss. In particular, when the attention is focused on the open-stacks minimization, the pseudo-prices of all items with open order are multiplied by an exponential factor, thus reducing the chance of opening new stacks.

Further employments of the SVC heuristic were presented for the two-dimensional strip-packing problem by Belov at al. [20] and Cui et al. [40] respectively for the orthogonal and non-oriented case, where the latter took into account both the presence and absence of guillotine constraints. More recently, Cui et al. [41] inherited the value correction principle for a sequential pattern-set generation algorithm designed to solve one-dimensional CSP with setup costs. In [33] Chen et al. exploited the SVC scheme to compute both guillotine and non-guillotine patterns for the two-dimensional BPP. Arbib and Marinelli [12] dealt with the minimization of a convex combination of number of bins and maximum lateness in a one-dimensional BPP integrated with scheduling features, in which orders must be fulfilled within prescribed due-dates. The proposed price-and-branch uses primal solutions built by means of an SVC algorithm. Finally, in [34] a CSP with skiving option which derives from paper and plastic film industries is investigated and an SVC heuristic is combined with two integer programming formulations to solve two specifications of the problem.

1.5 Chapter Outline

The rest of the thesis is structured as follows:

- In chapter 2 we focus the discussion on selected geometric packing problems derived from real manufacturing scenarios that are characterized by scheduling aspects. Looking at the representative mathematical programming formulations for these problems, the aim is to provide a useful insight to integrate two perspectives, the packing and the scheduling ones, under a unifying framework.

- Chapter 3 is devoted to a multi-stock CSP arising in woodboard manufacturing systems, in which real aspects are taken into account along with multiple objectives. An SVC algorithm is described in detail, which was designed with the aim of identifying sets of non-dominated solutions. The quality of solutions is discussed by comparison with a benchmark software exploited in a real manufacturing system.

- In chapter 4 we discuss a bi-objective extension of BPP in which items are provided with due-dates and the maximum lateness in the packing process has to be minimized along with the number of employed bins. The problem is coped by means of an SVC heuristic, while bi-objective approximation results are derived from BPP heuristics with guaranteed approximation ratio.

- Chapter 5 deals with a BPP with due-dates that embeds the presence of items processing time in the packing operations. In this context the completion time of a bin is dependent by the multiplicity of the items packed within and the objective is formalized as a convex combination of the number of used bins and the maximum lateness across the process. An integer pattern-based reformulation solved by CG is presented for this problem and evaluated with respect to an assignment compact formulation in terms of continuous bound quality.

- Chapter 6 is related to a structure packing problem called MAXIMUM QUASI-CLIQUE PROBLEM (γ-QCP). Indeed, the clique is a cohesive structure complementary to the independent set that can be seen under the viewpoint of a packing problem. The γ-QCP is a combinatorial optimization problem in which a γ-quasi-clique, that is a clique relaxed on the density constraint, is sought while maximizing the order of the set of selected vertices. We present an original integer reformulation with surrogate relaxation to achieve dual bounds by means of CG procedure. Moreover, the connectivity issue for subgraph structures is discussed and a new sufficient condition on solution connectivity is presented and computationally evaluated.

- Finally, chapter 7 is devoted to the concluding remarks of the thesis and to give some potential directions for future research.

Chapter 2

Cutting processes optimization

The majority of the results presented in this thesis were achieved in the field of bin packing and cutting problems, generally enriched with constraints and features that emerge from real manufacturing scenarios and are closely related to the scheduling of operations. In order to portray the connection that exists in the real environments between packing (cutting) and scheduling aspects, this second chapter want to be a non-exhaustive tentative to integrate the two perspectives under a unifying framework, focusing on representative mathematical programming models.

2.1 Introduction

In general terms, a common feature of packing problems is that they deal with bounded geometric figures (r-dimensional compact sets) that are not deformable, cannot overlap, and must be placed into larger geometric figures, either bounded or unbounded. Due to a mathematical affinity to cutting problems, where figures are components (here called *small items*) to be cut from big plates (*large items*), these problems are gathered into the larger family of *cutting and packing* problems (C&P), including such well known basic models as KNAPSACK, PALLET LOADING, BPP. Among others, the CSP is one of the best known C&P problems in both the practitioner community – for its practical relevance – and the research one – due to the mathematical properties exploited for its solution. Although C&P problems usually emerge in manufacturing environments, related applications include issues derived from different contexts, such as the data packets allocation in telecommunication field [82]. Moreover, BPP and CSP can be seen as special cases of the VEHICLE ROUTING PROBLEM [49].

The basic CSP calls for fulfilling a given demand of small items of various sizes, and possibly shapes, by cutting standard large items of identical size, with the objective of minimizing the amount of unused material. It has almost uncountable variants and extensions that depend on geometrical features, I the dimension of the things to cut (one, two or three), their shape (rectangles, spheres, general convex or non-convex figures), etc. | on the technique used to obtain the shapes, | simple or nested cuts, guillotine cuts in one or more stages, fretwork, part rotation, leftovers reuse allowed or not etc. | on the assortment of either small or large items, | of either small (one, few of many part sizes) or large items (one, many identical, distinct or unbounded stock plates) | and on the type of assignment of small to large items. For a comprehensive typology, see [135]. Whatever is the variant considered, a solution of the CSP is in general a non-ordered set of cutting patterns, each one associated with a positive integer called *activation level* or *run length*. A cutting pattern specifies the small items and their geometrical arrangement that must be cut from each large item, and its run length is the number of large items to be cut in that way.

Now, cutting machines generally implement the same cutting pattern on a single, limited pack of large items at a time and each single cut, as well as the total process, can be organized in many ways that affect system performance. Crucial elements for a correct management of a cutting process are:

- The way each machine implements a cutting pattern: for instance, multiple parallel cut vs single sequential cut: a sequential cutter (e.g., a pantograph milling machine for irregular shapes) implements each pattern by cutting the small items in a sequence, and different sequences may take different time.
- The number, type and duration of *set-ups*, given by slitter repositioning or other types of changeover; set-ups may occur from pattern to pattern, so that solutions with few different patterns can be preferred to ones with many; furthermore, switching time may depend on consecutive pattern pairs.
- The organization of demanded production (and therefore of the patterns it derives from) as desired by downstream plant departments: for example, lots can be required in prescribed sequences or within given due-dates; similarly, large items of different sizes can be subject to different release dates.
- The finite capacity of the buffers designed to accommodate the small items, and so to divide the production by classes (e.g., by order or shape): in this case not all patterns and not all pattern sequences are feasible, because they may produce an excess of some part class.

Summarizing, patterns and run lengths normally give insufficient information to implement a solution, and a subsequent effort to organize the process is normally required to schedule cuts after a CSP solution has been found (*cut-then-schedule*). However, not only is this approach generally sub-optimal, but also traditional CSP objectives may drive patterns and run lengths towards poor or even infeasible choices. With a simultaneous approach that takes those issues into account (*cut-and-schedule*), CSP models are enriched in order to define quality and feasibility of the schedules in which pattern and/or cuts within patterns are organized. In such models, a solution is what we here call a *cutting plan*.

The above discussion suggests that, in industrial applications, the problem of finding an optimal cutting plan combines so many varieties that it is difficult to get a comprehensive picture. Thus, we prefer to focus here on few issues that are common to a large amount of manufacturing environments: (i) reduction of the number of set-ups, (ii) minimization of delays in the delivery of the parts required, and (iii) scheduling of cut-operations when intermediate stacks have a finite capacity. This focus has the advantage of illustrating three different fundamental ways of considering process efficiency without getting lost in implementation details not always worth of consideration from a scientific viewpoint.

Issue (i) deals with machine productivity and is traditionally considered a very important cost source in manufacturing. It originates the so called PATTERN MINIMIZATION PROBLEM (PMP), in which the number of set-ups, that incur at slitter repositioning, has to be minimized.

Issue (ii) arises when material cutting not only has a local dimension of process efficiency, but also one related to customer satisfaction, or to the synchronization with other processes. In the corresponding CSP extension (CSP-DD), lots are provided with individual due-dates and the goal, jointly with material usage optimization, is the minimization of due-date related scheduling functions: typical objectives are the minimization of *maximum* or *weighted tardiness*, or of the number of tardy jobs.

Issue (iii) emerges because industrial production is often organized in lots, that can be either *homogeneous* (namely, formed by parts of the same size/type), or *heterogeneous* (for instance when different part types are grouped to form a particular client order). Real cutting processes often try to limit the variety of unfinished lots simultaneously present in the system. This translates into a problem called STACK-CONSTRAINED CSP (SC-CSP), where feasible plans are allowed to maintain a maximum number of *open stacks* at any time (a stack is regarded as open as far as it is occupied by an uncompleted lot). Minimizing the number of open stacks is recommended in order to reduce the possibility of errors as well as in-process inventory [11], and in some cases (e.g., wood-cutting) it derives from the very technical requirements of machines.

In this chapter we discuss representative models in which optimal/feasible cutting plans are so defined as to cope with the aforementioned CSP extensions. Specifically, after a brief historical excursus that focuses on the main achievements obtained to deal with CSP, we formally define in §2.2 the problem of identifying an optimal cutting plan; then, mixed integer linear programming (MILP) formulations will be respectively reviewed, under a unifying notation, for three representative CSP extensions: PMP in §2.3, CSP-DD in §2.4 and SC-CSP in §2.5. Note that the exposition focuses on the *cut-and-schedule* perspective and the dissertation could be refined by including also the *cut-then-schedule* counterpart. However, this has been considered beyond the purpose of the chapter.

2.1.1 An historical overview

In 1960, the one-dimensional version of the problem, that is, cutting shorter bars of various given length from longer bars of standard length available from stock, received by Kantorovich [74] a first formalization in terms of an integer linear programming model. This model makes use of assignment variables u_{ij} that count how many parts of lot $i \in I$ are produced by stock bar $j \in J$. An extra 0-1 variable w_j for each $j \in J$ tells whether that bar is actually cut or not and, summed up for all the stock bars available, all these extra variables give the amount of material used to cut the required parts.

The assignment integer linear programming formulation report as follows:

$$
\begin{aligned}
\min \sum_{j \in J} w_j & & & & \quad (2.1)\\
\sum_{j \in J} u_{ij} &= r_i & & i \in I & \quad (2.2)\\
\sum_{i \in I} l_i u_{ij} &\leq l w_j & & j \in J & \quad (2.3)\\
u_{ij} &\in \mathbb{Z}^+ & & i \in I, j \in J & \quad (2.4)\\
w_j &\in \{0,1\} & & j \in J. & \quad (2.5)
\end{aligned}
$$

The objective function (2.1) is defined as the minimization of the total number of employed stock bar, under the satisfaction of constraints (2.2) that imposes the required number of parts r_i to be cut for each part-types $i \in I$. Finally, the limited length l of the stock bars obliges variables to obey integer knapsack constraints (2.3).

This model has severe limitations, not only due to the potentially large number of variables (pseudo-polynomial with the number of part types), but also, most of all, to the poor quality of the lower bound obtained by linear relaxation and the strong symmetry shown by feasible solutions. Such issues can be related to the excessive finesse of the grain used to describe solutions, that is actually relevant only when some additional practical aspects are integrated to the basic CSP definition.

One year later, Gilmore and Gomory [62] published a key reformulation of Kantorovich's model, based on Dantzig-Wolfe decomposition. Let a_{ip} be the number of parts of lot $i \in I$ produced with one stock item cut according to pattern $p \in P$, where P is the set of enumerated patterns. By using the integer variables x^p to count the activation level of pattern $p \in P$, the integer reformulation results in:

$$
\begin{aligned}
&\min \sum_{p \in P} x^p && && (2.6)\\
&\sum_{p \in P} a_i^p x^p = r_i && i \in I && (2.7)\\
&x^p \in \mathbb{N} && p \in P. && (2.8)
\end{aligned}
$$

By replacing the knapsack constraints (2.3) by the convex hull of their integer solutions, the new pattern-based model drastically improved the LP lower bound and eliminated solution symmetries. Also, it opened a way to investigate on the so called integer round-up property (IRUP, [16]), that occurs when the integer optimum z^* equals the LP lower bound, i.e., the smallest integer greater than or equal to the optimum value of the LP relaxation (see [86, 85] for details).

Nevertheless, the formulation of Gilmore-Gomory has an exponential number of variables due to the number of different combinations in which a stock item can be cut. Hence, solving its linear relaxation requires the call to a delayed column generation technique, which relies on the solution of a pricing problem to generate profitable columns. In the one-dimensional case, such pricing is modeled as an integer knapsack in which the a_i^p's appear as decision variables p_i and are generated from the pricing solutions.

Finding exactly optimal integer solutions remained a challenge for several years, as the column generation allows computing only the LP optimal value. On the other hand, some heuristic approaches performed very well due to the structural properties of rounding, especially for high-demand instances [119, 134].

A breakthrough was represented by the seminal work of Barnhardt et al. [15] in 1996 that introduced the branch-and-price technique and allowed the solution of one-dimensional CSP instances with hundreds of items [6].

Nowadays, the available approaches make possible to solve pure CSP instances with size that widely exceed the requirements of real production systems. Thus, the attention of the research community move forward, focusing on a wider perspective in which the technological and managerial issues of cutting processes are taken into account. This is also merit of a spreading sensibility that recognizes the solutions of pure CSP as hardly or even impossible to be implemented in practice, given the presence of productive constraints and the impact of significant scheduling-related objectives [73].

2.2 General problem definition

In general terms, the problem of finding a cutting plan can be stated in this way:

Problem 2.2.1. *Find a set of patterns, their run lengths and implementation (pattern assignment to machines and/or time instants) such that:*

- *the demand of all part types is fulfilled;*
- *the output schedule of lots is feasible and meets the desired level of performance;*
- *the total trim-loss is minimized.*

For ease of presentation we will regard lots as homogeneous, and assume that parts are cut from identical stock items. Formulations that generalize to heterogeneous lots and/or stock sizes can however be obtained straightforwardly.

The following list summarizes the notation used throughout the rest of the chapter.

Data:

I: set of lots (jobs), $|I| = n$;

P: set of patterns (implicitly enumerated), $|P| = N$;

l_i: length of the parts belonging to lot $i \in I$ (if one-dimensional);

l: stock items length (if one-dimensional);

r_i: volume (number of parts) of lot $i \in I$;

d_i: due-date of lot $i \in I$;

T: index set of time periods dividing a given planning horizon;

a_i^p: number of parts of lot $i \in I$ produced with one stock item cut according to pattern $p \in P$.

A bar over a symbol makes it binary, giving 1 if > 0 and 0 if ≤ 0. For instance, $\bar{a}_i^p$ means 1 if pattern p produces one or more parts of type i, and 0 otherwise.

Note that the set T of time periods can be a decision variable, as a priori subdivisions can have two types of drawbacks: high computational complexity if too fine-grained, bad approximation if too coarse-grained. Moreover, the time horizon can be divided in periods of non-equal lengths, and these lengths may not be known in advance.

Decision variables:

C_i $\in \mathbb{R}^+$: completion time of lot $i \in I$;

T_i $\in \mathbb{R}^+$: tardiness of lot i: $T_i = \max\{C_i - d_i, 0\}, i \in I$;

$L_{\max}$ $\in \mathbb{R}^+$: maximum lateness of any lot: $L_{\max} = \max_{i \in I}\{T_i\}$;

x_t^p $\in \mathbb{Z}^+$: number of stock items cut in period $t \in T$ according to pattern $p \in P$ (run length of pattern p in period t);

s_{it} $\in \mathbb{Z}^+$: inventory level (number of parts produced and not yet sent out) of lot $i \in I$ at the end of period $t \in T$;

$\theta_{\max}$ $\in \mathbb{Z}^+$: maximum number of lots simultaneously in process throughout the planning horizon;

y_{it} $\in \{0, 1\}$: gets 1 if and only if lot $i \in I$ is still in process in period $t \in T$;

Again, we remove indexes in case of independence, and use a bar to "binarize" variables. In particular:

$\bar{x}_t^p$ gets 1 if and only if pattern p is active in period t;

$\bar{s}_{it}$ gets 1 if and only if one or more parts of type i are still in process during period t

$\bar{T}_i$ $\in \{0, 1\}$ is used to count the number of tardy jobs.

Let $\mathbf{x}$, σ denote vectors collecting variables x_t^p and the remaining variables $(s_{it}, y_{it}$, etc.), respectively. The following formulation $\mathcal{G}$ updates Gilmore and Gomory's model in order to give our problem a general framework:

$$\min \quad \alpha_1 \textit{material-cost}(\mathbf{x}) + \alpha_2 \textit{implementation-cost}(\mathbf{x}, \sigma) \tag{2.9}$$

subject to:

$$\sum_{p \in P} a_i^p \sum_{t \in T} x_t^p \quad = \quad r_i \qquad\qquad j \in J \tag{2.10}$$

$$(\mathbf{x}, \sigma) \quad \in \quad \mathcal{R}. \tag{2.11}$$

The objective function (2.9) describes the convex combination of the two cost items, where $\alpha_1, \alpha_2 \in \mathbb{R}^+$ with $\alpha_1 + \alpha_2 = 1$. Equality (2.10), sometimes relaxed by $\geq$ or by constraining the left-hand side within a prescribed interval, is the demand constraint. Clause (2.11) is a generic form for the implementation (e.g., scheduling) constraints, and region $\mathcal{R}$ varies according to application. For identical stock sizes, the first term of (2.9) has the form

$$\tau = \textit{material-cost}(\mathbf{x}) = \sum_{p \in P} \sum_{t \in T} x_t^p \tag{2.12}$$

and, if (2.10) is not relaxed, measures trim-loss.

When inventory levels are relevant, the demand constraint is replaced by equilibrium equations

$$s_{i,t-1} + \sum_{p \in P} a_i^p x_t^p = r_{it} + s_{it} \qquad\qquad i \in I, t \in T \tag{2.13}$$

that involve inventory variables s_{it}. The first term of the left-hand side and the last of the right-hand side give the amount of parts inherited at the beginning and left at the end of the period. By considering the set T of time periods partitioned into consecutive intervals $[0, d_1], [d_1, d_2], \ldots, [d_{n-1}, d_n], [d_n, d_{n+1}]$, where it is w.l.o.g. assumed $0 < d_1 \leq d_2 \leq \ldots \leq d_n \leq d_{n+1}$, and d_{n+1} is the horizon length; then r_{it}, the amount of parts of type i due by d_t, is r_i for $i = t$ and 0 otherwise.

The above formulation is actually depicted for single machine CSP, in which patterns are sequentially implemented by the same machine. Nevertheless, real cutting processes are commonly carried out by more complex production systems with a structured layout. For instance, cutting operations can be split into two stages if an upstream machine divides each stock item into m parts, and each part is subsequently sent to one of m downstream parallel cutting machines. Another example is given by in-house precut, a practice especially operated by big plants that separately prepare stock material for warehouse, which will be used in consecutive processes. Looking at real cutting systems, the definition of optimal cutting plan gains features typical of scheduling problems with multiple machines. Specifically, a CSP is related to a *parallel machine* scheduling environment when machines separately and simultaneously operate for job fulfillment on independent job portions, and to a *flow-shop* environment when the cutting plan comprises a stream of operations performed by subsequent cutting machines. Although we here focus on single machine CSP, formulation $\mathcal{G}$ can easily be adapted to multi-machine environments by suitably changing the definition of (some of) its components.

2.3 Set-up minimization

The problem of finding, among all the CSP optimal solutions, one that minimizes the number of patterns used and thus, for identical patterns, the set-ups required, was first considered by Vanderbeck [127] and called the PATTERN MINIMIZATION PROBLEM (PMP).

Referring to the general model of §2.2, the problem has overweight *material-cost*$(\mathbf{x})$ in objective (2.9), thence prioritizing trim-loss minimization. Integer variables x^p expressing pattern run lengths need to be coupled with 0-1 pattern activation variables $\bar{x}^p$, that summed up on P give the total number of patterns used:

$$\begin{aligned} \min \sum_{p \in P} \bar{x}^p & & & \qquad (2.14) \\ \sum_{p \in P} x^p & \leq \tau^* & & \\ x^p & \leq U_p \bar{x}^p & & p \in P \\ x^p \in \mathbb{N},\ \bar{x}^p & \in \{0,1\} & & p \in P \end{aligned}$$

plus demand constraints (2.10) written for single period. In (2.14), τ^* indicates the maximum admissible material usage, and the activation constraints uses adequate upper bounds U_p on pattern run lengths.

In alternative, Vanderbeck [127] suggests one binary activation variable per pattern and per run length; that is, $z_x^p = 1$ if and only if pattern p is activated at run length x:

$$\min \sum_{p \in P} \sum_{x=1}^{U_p} z_x^p \qquad (2.15)$$

$$\sum_{p \in P} \sum_{x=1}^{U_p} x z_x^p \leq \tau^* \qquad (2.16)$$

$$z_x^p \in \{0,1\} \qquad p \in P, x = 1, \ldots, U_p$$

subject to demand constraints

$$\sum_{p \in P} \sum_{x=1}^{U_p} x a_i^p z_x^p = r_i \qquad j \in J. \qquad (2.17)$$

Both (2.14)+(2.10) and (2.15)-(2.17) can be derived by discretizing different subsets of constraints in a bilinear model, based on Kantorovich and enriched with pattern activation variables, see [5].

In general, if $\sum_i a_i x_i \leq b$ is valid for an integer program $\mathcal{P}$, and $f(y)$ is non-decreasing and super-additive, then also $\sum_j f(a_i)x_i \leq f(b)$ is valid for $\mathcal{P}$: rank 1 Chvátal-Gomory cuts are for instance derived using $f(y) = \lceil \lambda y \rceil$ for $0 < \lambda < 1$. Cuts to strengthen Vanderbeck's formulation can be derived with this technique from inequalities (2.16) and (2.17). In [127], the super-additive functions have the form $f_p(y) = \lceil \frac{py}{\tau^*} \rceil - 1$ for (2.16) and $f_p(y) = \lceil \frac{pya_i^p}{r_i} \rceil - 1$ for (2.17). The derived cuts dominate Chvátal-Gomory's, can be added without destroying the problem structure, and are separated in pseudo-polynomial time. Alves and Valério de Carvalho [6] propose dual feasible functions (DFF) that dominate the function proposed by Vanderbeck and, consequently, provide sharper cuts. A DFF is a function $f : [0, 1] \to [0, 1]$ such that

$$\sum_{\mu \in H} \mu \leq 1 \Rightarrow \sum_{\mu \in H} f(\mu) \leq 1$$

for any finite set H of real numbers. Let $0 \leq a_i \leq b$ with $b > 0$, and let primal $\mathbf{x}$ fulfil $\sum_{i=1}^n a_i x_i \leq b$, i.e., $\sum_{i=1}^n \frac{a_i}{b} x_i \leq 1$. Rewrite the inequality as

$$\sum_{i=1}^{n} \sum_{k=1}^{x_i} \frac{a_i}{b} \leq 1.$$

Since $\frac{a_i}{b} \leq 1$, by definition of DFF we can replace it by $f(\frac{a_i}{b})$ and stay feasible: therefore

$$\sum_{i=1}^{n} \sum_{k=1}^{x_i} f(\frac{a_i}{b}) = \sum_{i=1}^{n} f(\frac{a_i}{b}) x_i \leq 1$$

is valid.

Comparison were performed on instances with up to 350 items to evaluate the LP relaxations of the two formulations and to point out the role played by the choice of upper bounds U_p [5, 19]. In particular, the LP relaxation of (2.14)+(2.10) is showed to be theoretically tighter, although computational results give only evidence to a limited advantage.

A particular case of PMP, typical of situations with a fixed number of slitters, occurs when a cutting pattern can never produce more than a prescribed number u of small items. For $u = 2$, the feasible patterns are in one-to-one correspondence with the edges of a *compatibility graph* $G = (I, E)$, where $(i, j) \in E$ means that a part of type i and one of type j can be cut from the same stock item.

McDiarmid [93] studied the special case of $G = K_n$, where any two parts fit any stock length, but no three do. In this problem, a crucial role is played by cut schedules. Since every pattern produces two parts, one can regard these two streams of parts as if they were produced by two parallel machines. A non-preemptive strategy is to produce the generic part type i continuously, as soon as the relevant lot is completed. Thus, the CSP is trivial, whereas the PMP (corresponding to maximizing the number of times two jobs end at the same time) is proved to be NP-hard [93]. The last observation leads to formulate the PMP in terms of SET PACKING, see [4].

A natural generalization of this problem describes the set of feasible patterns as the edge set of an undirected graph G_n, whose nodes correspond to part types. The CSP becomes then a b-matching, and hence can be solved in polynomial time. The PMP loses instead the nice properties it has when $G_n = K_n$, but, resorting to a flow model, can still be solved in polynomial time when G_n is a split graph [4].

2.4 The CSP with due-dates

The CSP with due-date related objectives has been considered rather recently. In 2004, Johnson and Sadinlija [73] proposed an integer programming formulation to determine optimal cutting plans with ordered lots subject to due-dates. The model uses a smart trick to get rid of the difficulty of coding 1-machine pattern scheduling, based on the use of x^p to model the length of the p-th pattern, of additional integer variables v^p_{iq} and the "binarized" version $\bar{v}^p_{iq}$. Variables is set to 1 if the "appereance level" of part type i in pattern p is equal to q, case in which v^p_{iq} is set equal to x^p. Let the set Q be defined as the possible number of copies of any item that can simultaneously belong to any pattern.

The formulation states:

$$\min \sum_{p \in P} x^p \tag{2.18}$$

$$\sum_{p \in P} \sum_{q \in Q} q v^p_{iq} \geq r_i \qquad i \in I \tag{2.19}$$

$$\sum_{i \in I} \sum_{q \in Q} q \bar{v}^p_{iq} l_i \leq l \qquad p \in P \tag{2.20}$$

$$\sum_{h \leq p} x^p - M(1 - \sum_{q \in Q} \bar{v}^p_{iq}) \leq d_i \qquad i \in I, p \in P \tag{2.21}$$

$$v^p_{iq} - M \bar{v}^p_{iq} \leq 0 \qquad i \in I, p \in P, q \in Q \tag{2.22}$$

$$\sum_{q \in Q} \bar{v}^p_{iq} \leq 1 \qquad i \in I, p \in P \tag{2.23}$$

$$\sum_{q \in Q} v^p_{iq} - x^p \leq 0 \qquad i \in I, p \in P \tag{2.24}$$

$$M \sum_{q \in Q} \bar{v}^p_{iq} - \sum_{q \in Q} v^p_{iq} + x^p \leq M \qquad i \in I, p \in P \tag{2.25}$$

$$\begin{array}{lll} x^p & \in & \{0,1\} & p \in P \\ \bar{v}^p_{iq} & \in & \{0,1\} & i \in I, p \in P, q \in Q \\ v^p_{iq} & \in & \mathbb{N} & i \in I, p \in P, q \in Q. \end{array}$$

Inequalities (2.19)–(2.21) enforce, respectively, the demand fulfillment, the knapsack constraints and the due-dates satisfaction, assuming that patterns are indexed according to their position in the cutting sequence, e.g. $p = 3$ means that pattern p is cut as third in the sequence with length x^p. The block of constraints (2.22)–(2.25) set the link between variables. Specifically, $\bar{v}^p_{iq}$ switches to 1 whenever i occurs q times in the p-th pattern, and the integer v^p_{iq} variable equals the pattern run length x^p only in this case. The former variables assign part type i to the p-th pattern, hence their sum over all the possible q is ≤ 1, where value 1 is achieved if part type i occurs in the p-th pattern, 0 otherwise. The objective function (2.18) restricts (2.9) to the first term only, asking for the minimization of the total run length. To tighten formulation, further constraints can be embedded to bound the possible values of x^p.

The formulation does not use column generation, and is quite complicated by the frequent recourse to activation constraints and "big constants", a practice that typically produces very weak lower bounds. In fact, the largest model solved with this methodology has just 17 unitary lots.

Reinertsen and Vossen [113] try to overcome this limit by resorting to column generation. Their idea, previously implemented by [11] in a context where the CSP is coupled with inventory management, uses variables x^p_t to define the run length of pattern $p \in P$ during period $t \in T$. Unitary cut times and no set-ups are assumed. The objective is a combination of material and scheduling cost as in (2.9), where the first term is given by (2.12) and

$$\textit{implementation-cost}(\mathbf{x}, \sigma) = \sum_{i \in I} \beta_i T_i$$

defines the weighted sum of tardiness, with $\beta_i \in \mathbb{R}^+$ for each $i \in I$. Demand constraints read as in (2.10), and recall that the set T of time periods is defined by partitioning the planning horizon into consecutive intervals $[0, d_1]$, $[d_1, d_2], \ldots, [d_{n-1}, d_n], [d_n, d_{n+1}]$, where it is w.l.o.g. assumed $0 < d_1 \leq d_2 \leq \ldots \leq d_n \leq d_{n+1}$, and d_{n+1} is the horizon length. Tardiness is treated as a surplus variable in inequalities of the form

$$\sum_{p \in P} \sum_{t=1}^{i} x^p_t \quad \leq \quad d_i + T_i \qquad i \in I \tag{2.26}$$

that try to confine the run length of patterns into the appropriate intervals.

Experiments show that problems with over 200 orders can be tackled in this way. However, the model is exact provided that an *early due-dates* job schedule is optimal. But this is not true in general [8], so the described approach must be regarded as heuristic.

To get an exact formulation for CSP-DD, one can follow [11] and use lot-sizing variables s_{it} to control the inventory level of lot i in period t. Demand constraints (2.10) are replaced by equilibrium equations (2.13), here reported for readers' convenience:

$$s_{i,t-1} + \sum_{p \in P} a_i^p x_t^p \quad = \quad s_{it} + r_{it} \qquad\qquad i \in I, t = 1, \ldots, n+1$$

where r_{it}, the amount of parts of type i due by d_t, is r_i for $i = t$ and 0 otherwise. Time capacity constraints just fit run lengths into each interval:

$$\sum_{p \in P} x_t^p \quad \leq \quad d_t - d_{t-1} \qquad\qquad t = 1, \ldots, n+1. \tag{2.27}$$

Inventory variables s_{it} can get negative values: if $s_{it} < 0$ (backlog), it simply means that lot i is fed by parts produced after d_i. Triggering this event by a suitable 0-1 variable y_{it} allows then to monitor due-dates violations. Remark that the entity of violation is measured by interval lengths, hence is an over-estimate: for instance, if a backlog of lot i is triggered at d_{i+1} but not later, this just implies that $T_i \leq d_{i+1} - d_i$. This approximation can be reduced by a procedure of interval subdivision which, iterated a finite number of times, converges to an exact formulation [8]. Limited optimality gap are achieved by embedding this formulation within a partial enumeration scheme for instances with 20 lots and $r_i \in [1, 100]$.

Finally, the above formulation can be adapted straightforwardly to different objective functions, such as the number of tardy-jobs $\sum_{i \in I} \bar{T}_i$ or the maximum lateness $L_{\max}$ [8].

2.5 The Stack-Constrained CSP

Yanasse and Pinto Lamosa [137] are credited a first attempt, dated 2007, to address the simultaneous generation and sequencing of patterns, with the objective of minimizing trim-loss τ while opening up to a prescribed number $\theta_{\max}$ of stacks. They formulate this integrated problem using (in our notation) pattern run lengths x^p, binary pattern activation variables $\bar{x}^p$, and binary time-indexed variables $\bar{x}_t^p$ that assign patterns $p \in P$ to positions $t \in T = \{1, 2, \ldots, |P|\}$. Pattern activation variables and run lengths are reciprocally constrained by

$$\bar{x}^p \leq x^p \leq U_p \bar{x}^p \tag{2.28}$$

where, as in §2.3, U_p denotes a suitable upper bound to the run length of pattern p.

In this formulation, every pattern is assigned to a certain position via the $\bar{x}_t^p$:

$$\sum_{p \in P} \bar{x}_\tau^p = \sum_{t \in T} \bar{x}_t^q = 1 \qquad q \in P, \tau \in T.$$

Still, not all $\bar{x}^p$ are necessarily 1, that is, not all patterns must be used. This is reported in the following inequality

$$\bar{a}_i^p (\bar{x}_t^p + \bar{x}^p - 1) \leq \bar{s}_{it} \qquad i \in I, t \in T \tag{2.29}$$

involving a binary variable $\bar{s}_{it}$ that takes value 1 if and only if the stack with parts of type i is non-empty at time t. Inequality (2.29) is non-trivial provided that pattern p produces at least one part of type i (i.e., $a_i^p > 0$). Its interpretation is that if p is the t-th pattern of the sequence *and* it is used at non-zero run length, then the i-th stack is non-empty at time t.

Limiting the number of stacks that are open at any time t corresponds to requiring

$$\sum_{i \in I} \bar{s}_{it} \leq \theta_{\max} \qquad t \in T. \tag{2.30}$$

Inequalities (2.28), (2.29) and (2.30) are exponentially many: there are, in fact, $|P|^2$ inequalities of type (2.29). Also, the number of variables is $> |P|^2$, because of $\bar{x}_t^p$. These numbers can be quite large even for small instances: for example, $n = 20$ part types and 1000 patterns originate $1,041,980$ variables and $20,025,001$ constraints. To handle such large numbers, [137] resort to primal heuristics coupled with lagrangian relaxation.

A more compact formulation, where the number of constraints increases polynomially with n, was then proposed by [9]. The formulation is based on the idea of *feasible track*, a 0-1 matrix **B** with n rows that has the *Consecutive Ones Property* (C1P) and exactly $\theta_{\max}$ 1's per column. The C1P requires that the column of **B** can be permuted so that, in any row, no 0 occurs between two 1. Every matrix of this form can be constructed as follows: (i) choose, for the first column, an arbitrary 0-1 vector with $\theta_{\max}$ 1's; (ii) add columns one by one, switching exactly one 1 of the previous column to 0 and one 0 to 1, with the condition that once a 1 is switched to 0 it never becomes 1 again. It is easy to see that any matrix so obtained has $\tilde{n} = n - \theta_{\max} + 1$ columns. Associate every column $\mathbf{b}_t$ of **B** with a period $t \in T = \{1, \ldots, \tilde{n}\}$, and assign, through the usual run lengths x_t^p, patterns $p \in P$ to periods in T. The following property holds:

Theorem 3. *A pattern sequence opens no more than $\theta_{\max}$ stacks if and only if the patterns* $\mathbf{a}$ *of the sequence that are assigned to period t fulfil* $\bar{\mathbf{a}} \leq \mathbf{b}_t$.

Proof. See [9].

By Theorem 3, a feasible sequence is encoded by integer variables x_{pt} fulfilling the $n\tilde{n}$ (thus polynomially many) inequalities

$$\sum_{p \in P} a_i^p x_t^p \leq r_i b_{it} = r_i(\phi_{it} - \psi_{it}) \qquad i \in I, t \in T. \tag{2.31}$$

In the right-hand side of inequality (2.31), the entries b_{it} of the feasible track are written as the difference of two non-decreasing sequences of 0-1 variables ϕ_{it}, ψ_{it}, where the former dominates the latter: this is a way to let the 1s of row $b_i = (\phi_{it} - \psi_{it})_{t \in T}$ occur consecutively, for every $i \in I$. For instance, the difference between ϕ-sequence 00011111 and the ψ-sequence 00000011 is the b-sequence 00011100. Variables $\phi_{it}, \psi_{it} \in \{0, 1\}$ thus must obey the linear constraints

$$\phi_{it} \leq \phi_{i,t+1} \qquad \psi_{it} \leq \psi_{i,t+1} \qquad \psi_{it} \leq \phi_{it}$$

for all i and t, and the limit on open stacks can be written

$$\sum_{i=1}^{n} (\phi_{it} - \psi_{it}) \leq \theta_{\max} \qquad t \in T.$$

Delayed pattern generation can be used to compute the linear relaxation. Even if the number of variables is still non-negligible, the model can be exploited to obtain good feasible solutions on instances with up to 20 lots.

In [92], Matsumoto et al. studied a one-dimensional CSP from the paper tube industry that deals with error-inducing issues that can affect the handling activities performed by human operators. These translate into three constraints of more general significance: (i) setups of each stock roll type should be made at most once; (ii) the minimum difference in the lengths of pieces simultaneously located on the worktable should be not smaller than a certain threshold Δ; (iii) up to a maximum number of stacks $\theta_{\max}$ should be opened at the same time. The authors introduce the concept of *cutting group* to manage (ii) and (iii) and develop a 0-1 ILP reformulation that is heuristically solved by means of a tabu search. Computational experiments are presented to show the quality of the approach on both random and real instances, with $n \leq 50$ and mean demand of part types up to 52.

In the special case in which every pattern cuts at most two parts, if all pairs of part types correspond to feasible patterns, then the CSP admits a trivial solution consisting of a non-preemptive cut schedule, see §2.3. With this schedule, no more than 2 distinct part types are in-process at any time: so at most 2 stacks are maintained open throughout production. The property still holds if the compatibility graph G_n is a threshold graph, but does not hold in general for a split graph [4].

2.6 Conclusion and future development

Implementing a cutting plan requires much more information than that obtained by just solving a CSP, as pattern schedules significantly affect the efficiency (and sometimes the practical feasibility) of CSP solutions. To cope with this issue, several extensions were defined in the CSP literature, each one integrating a particular aspect. In this chapter we gave a general outlook to the problem of defining an optimal cutting plan, seen as a CSP solution that complies with given scheduling requirements. We then briefly reviewed, in the light of an agile shared notation, MILP formulations of three main problem species: the PATTERN MINIMIZATION PROBLEM, the CSP with due-date related objectives, and the stack-constrained CSP.

This can be seen as a starting point for further extension of the study and additional cutting problems could be surveyed to take into consideration non-negligible aspects in industrial processes, such as multi-machine environments and pattern-dependent cut time. This study can be also deepened by looking at various solution methods proposed in literature, both with a *cut-then-schedule* and *cut-and-schedule* approach.

Chapter 3

An SVC for a rich and real two-dimensional woodboard cutting problem

[1]

In this chapter we deal with a rich version of CSP arising from woodboard manufacturing systems, in which real aspects are taken into account along with multiple objectives. Due to the complexity of the problem, an SVC algorithm was designed and implemented to heuristically identify a set of non-dominated solutions. Results are then given by comparing the SVC procedure with the previous software adopted by a leader company in production of cutting machines.

3.1 Introduction

The CSP is a well-known model of real-world optimization problems arising in manufacturing and can be summarized in a very simple way: *find an efficient way of cutting small objects from large ones*.

Although clear in its statement and elegant in its formal mathematical definition, numerous variants and extensions were proposed in literature to refine the CSP definition and shrink the gap between the theory and the practice, suggested by the real productive conditions that typify manufacturing systems. Referring to the extensive survey proposed by Wäscher et al. [135], variants arise whenever some assumptions defining general properties of the problem are replaced with different ones, such as problems with multiple objectives [133], on-line versions [67] or stochastic problems [46]. Extensions

[1]The contents of this chapter appears in: C. Arbib, F. Marinelli, A. Pizzuti and R. Rosetti: A heuristic for a rich and real two-dimensional woodboard cutting problem. In Proceedings of the 7th International Conference on Operations Research and Enterprise Systems - Volume 1: ICORES, ISBN 978-989-758-285-1, (2018) pages 31-37. DOI: 10.5220/0006534500310037

instead are born if the problem is enriched with additional aspects that enlarge the boundary of the problem beyond the essence of cutting, touching issues related to a more general perspective of the planning of operations. In this sense, examples can be found in the pattern minimization problem [127], pattern sequencing problem [136] or lot-sizes management [101].

Any variant (extension) is regarded of interest for the academia as far as it requires a non-standard model and/or a novel methodological approach. Unfortunately, it is often hard to encapsulate reality into a single elegant mathematical formulation. Moreover, it is generally recognized that the success of an optimization method depends on the concurrency of transversal skills: one is indeed mathematical modeling, but it is not the only one. A fundamental role is for instance played by the bi-directional interface that must be developed to feed the model with data and to present the solutions to decision-makers: a development that normally requires more IT than math skills.

On the other hand, the more a model is close to operational decision, the more attention is to be devoted to process details. Disregarding such details (which is often done in research papers in order to capture the mathematical essence of the problem) would simply cause the software product not to work, and the management not to buy it. For example, edge trimming is generally neglected in CSP models; but a cutting pattern that does not take it into account, may not be realizable in practice.

While a specification on edge trimming can easily be dealt with after prototype development, this is not the case of other apparently irrelevant issues. For example, blade thickness expressed in fractions of millimeters can increase the precision required of the packing algorithm, with important effects on its efficiency if, as usual, it has pseudo-polynomial time complexity.
A twofold challenge has therefore to be addressed:

- *On software design.* Technical issues | be they expressed as constraints or utility criteria | have both a local impact on the feasibility of cutting patterns (number of stages, item rotation etc.) and a global one on the cutting plan as a whole (number of patterns, cut sequence etc.). Pattern generation algorithms can differ a lot from each other in order to take care of the former issues, and the latter can affect the whole software architecture.

- *On decision-making.* A hierarchical problem decomposition based on the various utility criteria (e.g. first minimize trim-loss, then setups, then open stacks) can output low-quality or even infeasible solutions. On the other hand, combining different objectives into a single goal may produce solutions not appreciated by all the involved decision makers.

We here propose to address this challenge by a single architectural choice in which patterns are generated via sequential value correction heuristic and then used to provide a collection of *non-dominated solutions*.

We refer to problems with a level of detail that covers industrial specification as a whole and puts the solution method in a condition to operate in the plant as *rich*. Specifically, this chapter is devoted to a rich MULTIPLE-STOCK-SIZE TWO-DIMENSIONAL CSP (M2D-CSP) compliant with the specification of a family of wood cutting machines produced by *SCM Group* [117].

The M2D-CSP can be presented as follows. Let I be a set of n distinct rectangles (part types). Each $i \in I$ has width w_i and height h_i, and must be cut in r_i copies from a stock set J of m different (and larger) rectangles. Each $j \in J$ has width W_j and height H_j, and is available in m_j copies. The cut should be done to minimize used stock area; other objectives | such as reduction of employed stocks, setup minimization etc. | can however be selectively or entirely considered. Given these additional criteria, our M2D-CSP can be considered as an extension |and also a variant| of the classical CSP in the sense of [135]. Our problem has these main features:

- Cuts proceed parallel to stock item from edge to edge (*guillotine cut*).
- Items (either produced or in stock) can be orthogonally rotated.
- 2 and (simplified) 3-stage cuts are possible (figure 3.1): in a first stage, a configuration $\mathbf{c} = \langle$stock type, first cut orientation (horizontal or vertical)$\rangle$ is selected, and multiple cuts are done orthogonally to orientation; then the stock item is rotated by 90° and the stage is repeated.
- Precut is possibly allowed as a pre-processing step.

Part types admit overproduction if explicitly indicated, but demand is in general to be fulfilled exactly. Indeed, relaxing demand constraints by overproduction can yield solutions with very small trim-loss; but a substantial number of excess items are cut, increasing too much the necessary amount of stock items and consequently production costs.

3.2 Literature review

Real cutting processes often deal with heterogeneous stock types. The relevant M-CSP is handled in various ways. In [18] an exact cutting plane approach is proposed for one-dimensional CSP with multiple stock widths. The same problem is solved heuristically in [109]. In [138], the two-dimensional CSP in the wood industry is addressed and a heuristic algorithm is proposed for the multiple stock sizes case.

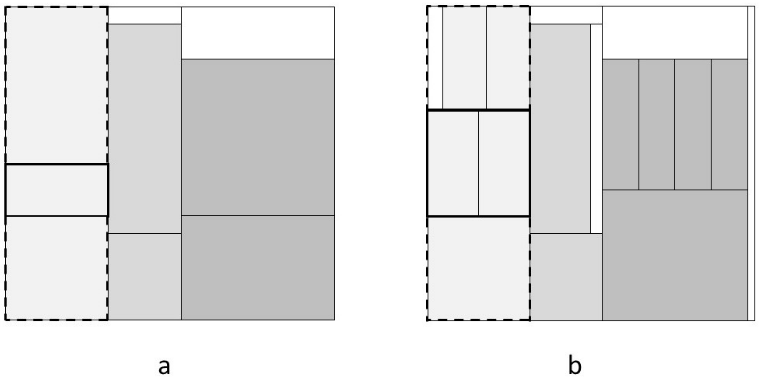

Figure 3.1: a) 2-stage pattern; b) (simplified) 3-stage pattern. Dotted lines delimit sections, bold lines highlight the boundary of strips.

Trim-loss minimization is not the only criterion considered in an industrial cutting process: indeed, setup costs and the number of stacks maintained open throughout the process can affect process quality as well. Examples of CSP seeking both setup minimization and open stack limitation can be found in [19], where a sequential heuristic for the one-dimensional case is proposed, and in [101], where the problem is formulated as a non-linear concave minimization problem and solved through global or local procedures. Metaheuristics and an exact approach that separately address the CSP with setup minimization are reported respectively in [124] and [5]; moreover CSP with open stack limitation is considered in [10] and [137].

Although industrial applications require consideration of several practical issues, these are taken into account by very few papers; and in those cases, very few additional issues are regarded simultaneously. Here is a list of exceptions (to the best of our knowledge):

- [84] describes a truncated Branch-and-Price algorithm to solve an M2D guillotine CSP in woodboard cutting industry, where orthogonal rotation of items and boards is allowed, and stacked boards can be cut concurrently. The objective is to minimize a weighted combination of stock usage, number of cutting cycles and of 3-stage cuts.

- [129] presents a 1D-CSP arising in a plastic manufacturing facility, with five objective functions hierarchically ranked, some technical parameters related to the employed cutting machines and order priorities. A GRASP algorithm with a call to a sequential heuristic procedure is implemented.

- [35] proposes a mathematical formulation for a rich 1D-CSP in the metal industry. The various technical constraints taken into account significantly affect the total cost and involve trim-loss, material reusability and cut time. Approximation algorithms based on dynamic programming are devised.

3.3 A sequential value correction heuristic

As pointed out in §3.3.1, the basic idea behind the SVC scheme consists in assigning a value (the pseudo-price) to each part type and generate cutting pattern sequentially, each one with the parts not yet allocated that maximize the total value of the pattern. After that a pattern has been computed, pseudo-prices are conveniently updated and the process reiterated. In this application, generated solutions are evaluated to fill a pool of non dominated solutions. Dominance relationship are based on the following minimization criteria: total used stock area, total trim-loss, number of used stocks, number of distinct setups, and total number of precuts (Notice that used area, trim-loss and number of used stocks measure distinct performance due the overproduction features and the presence of stocks of different sizes).

The whole procedure is composed by three main steps, each one optimizing a distinct performance indicator by resorting to Algorithm 1 which describes an SVC.
The algorithm can be summarized as following:

- Step 1: SVC solves the M2D-CSP with non-batched demand, seeking used area minimization.

- Step 2: The second step tries to reduce the number of setups by demand rounding. In particular, the demand of each part type is aggregated by increasing batch dimension (parameter *size*) and SVC is called to find solutions with reduced number of setups. Starting from $size = 2$, the procedure is iterated for increasing values of *size*, chosen as the largest integer in $[size + 1, \max_{i \in I} r_i]$ that minimizes the area associated to the residual demand $\mathbf{b}'$. Step 2 terminates when SVC does not find any new non-dominated solution.

- Step 3: in this phase SVC searches for solutions with reduced number of precuts. A suitable opportunity threshold δ is used to trigger decreasing chances of employing a precut. The procedure iterates by updating δ until new generated solutions have no precut.

Algorithm 1 Sequential value correction heuristic

```
1: procedure SVC($\mathbf{r}, size, \delta$)
2:     $\lambda_i \leftarrow S_i$
3:     $cuts \leftarrow 0$
4:     for $i \leftarrow 1 \dots N$ do
5:         $g \leftarrow random(1, \rho)$
6:         $\mathbf{b} \leftarrow \mathbf{r}$
7:         $\mathcal{S} = \emptyset$
8:         repeat
9:             $\mathbf{r}' \leftarrow \left\lfloor \frac{\mathbf{b}}{size} \right\rfloor$
10:            $\mathbf{b}' \leftarrow \mathbf{b} \bmod size$
11:            while $\mathbf{r}' > 0$ do
12:                $P = \emptyset$
13:                for $\mathbf{c} \in C$ do
14:                    $p[\mathbf{c}] \leftarrow getPattern(\mathbf{c}, \lambda, \delta, \mathbf{r}')$
15:                    $P = P \cup \{p[\mathbf{c}]\}$
16:                $(a, cuts) \leftarrow getBestPattern(P)$
17:                $fillingPattern(a, cuts)$
18:                $\mathcal{S} \leftarrow a$
19:                $h \leftarrow random(1/g, g)$
20:                $\lambda \leftarrow updatePrices(a, \lambda, h)$
21:            $\mathbf{b} \leftarrow \mathbf{b}'$
22:            $size = 1$
23:        until $\mathbf{b} > 0$
24:        $patternSeq(\mathcal{S}, cuts)$
25:        if $notDominated(\mathcal{S})$ then
26:            $Sol = Sol \cup \{\mathcal{S}\}$
```

3.3.1 SVC Heuristic

The SVC heuristic is the core of the algorithm and is implemented to build the CSP solutions.

Let I' be the set formed by the items of I and their rotated counterparts. In its general framework, SVC starts by defining a pseudo-price λ_i for each part type $i \in I'$. Each price is initialized at the item area S_i. Part type demand is batched by *size*, that is, stock items are stacked and each pattern is activated at multiples of *size*. A partial CSP solution is obtained by sequentially building patterns through subroutine *getPattern*: the subroutine attempts to solve subproblems where the sum of prices is maximized under bounded knapsack and compatibility constraints. For each configuration $\mathbf{c} = \langle$stock type, first cut orientation$\rangle$ in the set C of all possible configurations, a cutting pattern is generated and added to a set P. The most profitable pattern $a \in P$ is then selected using *getBestPattern*, which also set the corresponding activation level *cuts*. The *fillingPattern* function is implemented to insert unplaced items in empty sections, given the residual demand and the value of *cuts* of the chosen pattern. Subsequently, prices λ_i are updated by promoting items with high current demand r'_i and small multiplicity in a. A complete CSP solution $\mathcal{S}$ is finally obtained by combining the partial solution computed for demand $\mathbf{r}'$ with the one found for the residual demand $\mathbf{b}'$.

In general, open stacks are not an objective to be minimized, but are limited by the number of unloading stations that equip the cutting machine. Although the pattern generation procedure already respects the limited availability of unloading stations, an attempt at reducing the stacks opened by the current solution $\mathcal{S}$ is made by the *patternSeq* procedure. The function *patternSeq* implements a simple pattern sequencing heuristic that compares each pattern with all the subsequent ones, and swaps pattern pairs that allow the largest decrease of open stacks.

The current solution is added to a pool (*Sol*) if it fulfills all the constraints and is not dominated by any other. The process is iterated a prescribed number N of times unless certain halting conditions hold (e.g., user stop or minimum used area reached). Finally, the pool *Sol* is filtered and reduced to a frontier of non-dominated solutions.

It may happen that *Sol* consists of partial solutions only, due to insufficient stock availability. In that case, non dominated solutions are limited to those that fulfills the largest demand.

For the sake of conciseness, in the following we shall assume $\mathbf{c} = \langle j, H \rangle$, i.e., the cut of the first stage is always parallel to the height H_j of the stock item: the arguments presented maintain validity when replacing "horizontal" with "vertical" and "H_j" with "W_j".

3.3.2 Patterns Generation and Selection

For a given configuration $\mathbf{c} = \langle j, H \rangle \in C$, function *getPattern* creates a pattern that depends on the current prices λ and demands $\mathbf{r}'$. First, for each part type $i \in I'$ a vertical section of width w_i is defined, and a bounded integer knapsack problem is solved. The knapsack capacity is set equal to the stock height H_j and element sizes correspond to the heights h_i of the part types selected. Clearly, only part types $k \in I'$ with $w_k \leq w_i$, demand $r'_k > 0$ and prices $\lambda_k > 0$ are considered. Actually, each knapsack element describes a horizontal *strip* containing a single item k (2-stage patterns), or multiple copies of the same part-type (3-stage patterns) wide at most w_i. In the latter case, the maximum multiplicity m_k is given by $\lfloor w_i / w_k \rfloor$, and prices are computed accordingly.

Let Q_c be the set of sections generated so far, each one provided with a certain activation level defined with respect to the current part type demand $\mathbf{r}'$. The pattern is obtained by solving a further bounded knapsack problem that optimally selects sections in Q_c. In particular, the knapsack capacity is equal to W_j and knapsack elements correspond to sections, where the size and value of element q are respectively equal to the width and price of section q (the price of a section is the sum of the prices of items it contains). The resulting solution provides a subset of sections $\bar{Q}_c$ that may present overproduction. In order to preserve the structure of the knapsack problem, only activation levels of the elements (sections) are explicitly considered, whereas part type multiplicities in sections are neglected; moreover, the simultaneous presence of part types and rotated part types is not accounted for. It follows that part type overproduction is not prevented and a post-processing may be required to exactly fulfill part type demands. Sections $q \in \bar{Q}_c$ are sorted by non-increasing order of the ratio $\frac{V_q}{S_q}$, where V_q and S_q are respectively the price and the area of section q.

Sections are possibly added to the pattern p following the prescribed order, and demand $\mathbf{r}'$ is accordingly updated. In particular, a section is discarded if it causes demand overproduction, or overflow of the allowed limit on open stacks. In that case, the resulting pattern p may present a residual stock area of width R_j that can be further filled. Therefore, the whole procedure, i.e. sections generation and selection, is repeated in order to generate a sub-pattern for the configuration $\mathbf{c} = \langle \bar{j}, H \rangle \in C$, where stock item $\bar{j}$ has width R_j and height H_j. The sub-pattern is placed next to p and merged with it, and the process reiterated until the residual stock area cannot be filled any longer. Pattern p is then added to the set P.

Once patterns are generated for each $\mathbf{c} \in C$, the procedure *getBestPattern* selects the most profitable pattern $a \in P$, that is, a pattern that maximizes the total price of the sections it contains. Pattern a is finally added to the partial CSP solution $\mathcal{S}$ with activation level $cuts$, computed according to the current demand $\mathbf{r}'$ and stock items availability. The pattern generation process is iterated until the whole demand is fulfilled.

Since the presence of real-valued prices, all the above bounded integer knapsack are solved by encoding the integer variables as binary variables and then by using an own implementation of the algorithm for the 0-1 knapsack described in [91].

Precut Policy

For each pattern p, function *getPattern* generates also a pattern p' that contains a *precut*, and then evaluates the opportunity of replacing p with p'. For a given pattern p, the algorithm scans all the cuts of the second stage (those required to separate strips), choosing one whose quota h is closest to the middle of H_j, see Figure 3.2.

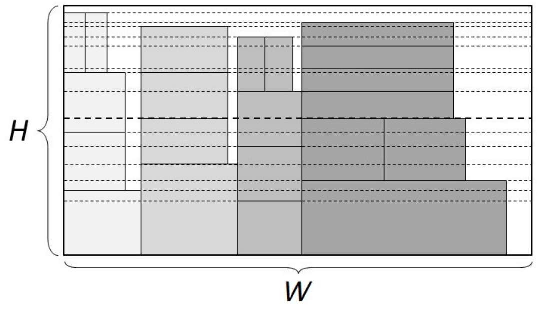

Figure 3.2: Dotted lines indicate the potential cuts of the second stage. The bold line refers to the cut closest to the middle of the stock height H_j.

Stock item j (and pattern p) is split into new stock items j_l and j_u (sub-patterns p_l and p_u) of sizes $h \times W_j$ and $(H_j - h) \times W_j$, respectively; the items that cross quota h are removed and the sub-pattern p_u deleted, see Figure 3.3. Stock item j_u is used to build a new pattern p'_u (by the same procedure described above), and a pattern with precut p' is obtained by merging p'_u and p_l, see Figure 3.4. If the difference between the reduced costs of p and p' is larger than a given threshold δ, then the pattern with precut is selected; otherwise *getPattern* selects p.

The value of δ is set to zero in Steps 1 and 2 of the heuristic, and is progressively increased in Step 3, where the algorithm aims at producing solutions with a decreasing number of precuts.

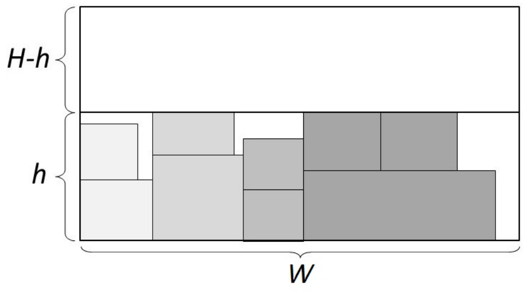

Figure 3.3: sub-pattern p_l

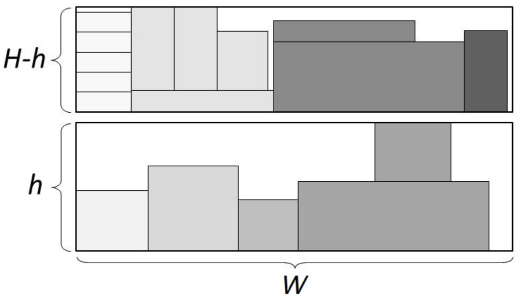

Figure 3.4: sub-patterns p_l and p'_u

3.4 Computational results

Table 3.1: Results without precut on ten real instances.

		SVC heuristic				Company software		
Name	n	Area	N	#patterns	CPU time	ΔArea (%)	ΔN (%)	Δpatterns (%)
I_1	58	2403.12	304	69	26.69	1.31	1.32	2.90
I_2	83	1454.8	251	102	6.67	1.19	1.20	-6.86
I_3	141	1512.21	508	165	121.14	1.58	1.57	0.61
I_4	500	221.32	29	29	103.52	2.40	34.48	34.48
I_5	500	273.84	36	36	-	-	-	-
I_6	747	3158.81	620	546	634.72	2.13	1.94	-14.65
I_7	759	495.92	96	96	760.08	5.45	5.21	5.21
I_8	869	2923.78	563	269	972.06	-	-	-
I_9	951	354.72	67	67	34.63	-	-	-
I_10	972	903.11	179	178	494.59	-	-	-
Average	**-**	**1370.16**	**265.30**	**155.70**	**350.46**	**2.34**	**7.62**	**3.61**

The SVC was implemented in C++ and experiments were performed on a Intel® Core2 Duo E8500 3.16 GHz with 8Gb RAM. The SVC heuristic has been tested on ten real instances provided by SCM Group, with $n \in [58, 972]$ and $\rho = 1.15$. Results have been compared to those provided by the software previously adopted by the company and developed by a national software house devoted to the design of cutting optimization software. In order to be consistent, the non-dominated solution using the minimum total stock area has been promoted to the *best* solution and compared to the single solution provided by the company own software.

Table 3.1 and 3.2 show the results for the case with disabled and allowed precut, respectively. For each instance, the SVC performance is evaluated in terms of total used stock area (Area), number of used stocks (N), total number of patterns (#patterns) and CPU time. The last three columns show the percentage gaps with the results obtained by the company software (positive values mean an improvement obtained by SVC; the symbol '-' means that the CPU time limit of three hours has been reached by the company software).

On average, the SVC heuristic reduces the total used stock area by 2.34%

Table 3.2: Results with precut on ten real instances.

		SVC heuristic				Company software		
Name	n	Area	N	#patterns	CPU time	ΔArea (%)	ΔN (%)	Δpatterns (%)
I_1	58	2403.12	304	80	5.92	1.64	1.64	-18.75
I_2	83	1454.8	251	105	34.55	0.79	0.80	-6.67
I_3	141	1509.24	507	152	332.28	1.18	1.18	-8.55
I_4	500	220.78	29	29	139.09	-	-	-
I_5	500	273.3	36	36	471.14	-	-	-
I_6	747	3158.27	620	524	632.56	2.08	1.13	-10.11
I_7	759	494.83	96	96	104.28	-	-	-
I_8	869	2908.4	566	264	549.39	-	-	-
I_9	951	354.72	67	67	84.11	-	-	-
I_10	972	903.33	179	177	1332.55	-	-	-
Average	**-**	**1368.08**	**265.50**	**153**	**368.587**	**1.42**	**1.19**	**-11.02**

without precuts (1.42% with precuts), while the percentage trim-loss is reduced by a factor between 1.5 and 4. The number of used stocks is also reduced by 7.62% (1.19 %) on the mean. When precut is disabled, an average reduction of 3.61% in the number of distinct patterns is achieved, although a meaningful mean increase of 11.02% arises when precut is allowed. However, such worsening is mainly due to the greater material usage efficiency attained by the SVC *best* solution: in most of the cases, SVC computes non-dominated solutions requiring comparable used material and less number of patterns. The CPU time required to solve real instances is about 6 minutes on the mean, with just few instances that exceed 10 minutes.

The heuristic was also tested on 52 benchmark instances with $n \in [10, 250]$. With the precut option disabled (enabled) the algorithm reduces trim-loss in 17.3% (15.4%) of the cases. By using the continuous lower bound of the total requested area, the optimality of the best solution with respect to the material usage has been certified in the 80.8% (82.7%) of the cases. Independently of the precut policy, in 17.3% of the cases a solution with one pattern less is found, and only 1.9% of the instances require an additional pattern.

3.5 Conclusion and future development

Mathematical programming is a powerful tool that can be used to model many relevant industrial problems. However, those mathematical formulations are often challenging and, in some cases, real process constraints are hard to be modeled. This is also true for many CSP variants that try to simplify resolution by encoding just a restricted number of real-world constraints. Software applications based on those CSP formulations may then happen to be inappropriate, and companies are continuously looking for tools able to deal with the whole process features and specification.

This chapter addressed such an issue and presented an SVC heuristic to solve a rich M2D-CSP that fulfills the specification of a family of wood cutting machines produced by *SCM Group*. The objectives of used stock area minimization and of additional criteria, as well as the fulfillment of open stack constraints, were considered. Computational tests demonstrated the improvement achieved on the previous software solution, in terms of both solution quality and CPU time requirements.

Further research is needed to include additional optimization criteria, e.g., the minimization of cutting times and the management of left-overs, and to provide an exact formulation of the pricing problem when the whole set of specification is considered. Also, cutting processes are strongly affected by operations scheduling: the formulation of a bi-objective rich M2D-CSP that simultaneously minimizes a scheduling function (such as the maximum lateness or the weighted sum of tardiness) should then deserve investigation.

3.6 Acknowledements

We gratefully acknowledge dr. eng. Federico Ratti, Stefano Oliva and Simone Cera of SCM Group for the fruitful discussions and constructive remarks that helped us design and refine the algorithm described in this chapter.

Chapter 4

An SVC heuristic for two-dimensional bin packing with lateness minimization

[1]

In several manufacturing environments the raw material does not represent the main cost element to be taken into account, since other factors have relevant impact on the total cost of operations. A non-negligible aspect that deserves attention is the management of delay in productive activities, which is strictly related to the crucial requirement of fulfilling the order due-dates.

In order to model a manufacturing system in which both the schedule and the efficiency of operations are optimized, in this chapter we study a bi-objective two-dimensional BPP in which we concurrently seek the minimization of the maximum lateness and the number of employed bins. We present bounds and derive approximation results under the bi-objective perspective from approximation algorithms for the single-objective BPP. Then, an SVC heuristic is proposed to solve the problem and compared to benchmark algorithms from literature.

4.1 Introduction

The r-dimensional Bin Packing Problem (rBP) is a well-known combinatorial optimization problem [81] that calls for packing a given set I of n r-dimensional items into a minimum number of identical r-dimensional bins. An instance of the two-dimensional basic version (2BP) consists of rectangular items described by positive integer widths $w_1, \dots, w_n$ and heights $h_1, \dots, h_n$ and a (sufficient number of) rectangular bins of size $W \times H$, with $W \geq w_i$ and

[1]The contents of this chapter develop the ones reported in: F. Marinelli, A. Pizzuti: A Sequential Value Correction heuristic for a bi-objective two-dimensional bin-packing. Electronic Notes in Discrete Mathematics - Volume 64 (2018) pages 25-34. https://doi.org/10.1016/j.endm.2018.01.004

$H \geq h_i$, $i = 1, \ldots, n$. In general, a packing is feasible if items are completely contained in the bins and do not overlap each other, but a number of further placement rules and restrictions have been investigated in the last years (see e.g. [135] for a comprehensive survey). In this work we deal with the *orthogonal* 2BP, where item edges must be parallel to bin edges. Both the *oriented* and the *non-oriented* versions are considered: in the latter case, $90°$ item rotation is allowed. Following a common notation, we will denote these problems as 2OBP, 2RBP, where O, R stand for *oriented* and *rotation*, respectively.

A 2BP solution that is optimal with respect to the efficiency of bin allocation, may not be so attractive when its quality depends also on the way items are allocated over time: if bins are packed in sequence and the allocation of each item $i \in I$ is seen as a task due by a date d_i, one can also be interested in the optimization of due-date related objective functions (e.g., weighted tardiness, number of tardy jobs, maximum lateness). Indeed, scheduling issues in packing problems are receiving increasing attention. For instance: [12] investigates the minimization of a convex combination of number of bins and maximum lateness; [8] and [113] propose Integer Linear Programming formulations to minimize the number of bins *and* the weighted tardiness in one-dimensional cutting stock. Other papers that deal with this topic are [22, 68, 110, 111, 140].

The mainstream applications of 2BP and of its cutting counterpart are truck loading and warehousing in logistics, and cutting optimization in manufacturing. In these areas, the interest for scheduling issues needs not be recalled. More applications of 2OBP can be found in telecommunications: in [82] the authors model the downlink subframe allocation problem in Mobile WiMAX technology as a 2OBP where items are group of data packets and bins represent time/frequency slots. A related packing-scheduling problem is addressed in [52], referred to QoS in a UMTS system with data packets obeying to deadlines or minimizing delays.

In this chapter we consider a bi-criteria 2BP with due dates: we will refer to this problem as 2BP-DD in general, and as 2OBP-DD, 2RBP-DD when orientation is significant. The chapter develops earlier ideas formulated in [87]: problems are described in §4.2 with a discussion of some meaningful properties; a Sequential Value Correction heuristic (SVC-DD) for the problems addressed is proposed in §4.4; an improved dual bound for the scheduling objective function is provided in §4.5; finally, in §4.6, non-dominated solutions computed by the SVC-DD heuristic are analyzed and compared to those obtained by the algorithms proposed in [22] and [110].

4.2 Problem and basic properties

In 2BP-DD, we assume as in [22] that

Assumption 1. *Filling each bin according to the relevant packing pattern requires a known constant time $\tau \in \mathbb{Z}^+$ that does not depend on the pattern.*

Assumption 2. *The completion time C_i of any item i contained in the k-th bin of the sequence equals $k\tau$.*

Rephrasing Assumption 2, all the items of a bin are simultaneously released as soon as the bin has been packed.

Our objective calls for the minimization of both the number N of bins used and the maximum lateness $L_{\max}$. By Assumption 2, the number of bins used is proportional to $\frac{1}{\tau}$ and to the largest completion time $C_{\max} = \max_{i \in I}\{C_i\}$ of the items in I. The lateness $L_{\max}$ is defined in the literature as the largest violation of a due date, and can be a negative number. For our purposes, however, we regard negative $L_{\max}$ as equivalent to $L_{\max} = 0$, and so define $L_{\max} = \max_{i \in I}\{L_i, 0\}$, where $L_i = C_i - d_i$ is the lateness of item i, and d_i is its due-date.

A feasible packing of value $(N, L_{\max}) = (z, \ell)$ is said to be (strictly) *non-dominated* if for any feasible packing of value $(\bar{z}, \bar{\ell})$ one has either $z < \bar{z}$ or $\ell < \bar{\ell}$ (or both). Moreover, a pair (z, ℓ) is said to be *weakly non-dominated* if there is no feasible pair $(\bar{z}, \bar{\ell})$ such that $\bar{z} < n$ and $\bar{\ell} < \ell$. For example, take $\pi_1 = (20, 5), \pi_2 = (18, 6), \pi_3 = (18, 5), \pi_4 = (21, 6)$: in this set π_1, π_2, π_3 are all weakly non-dominated, π_3 is strictly non-dominated and weakly dominates π_1, π_2 and finally π_4 is strictly dominated by π_1, π_3 and weakly dominated by π_2. The general problem (viz., regardless of item orientation) can then be formalized in this way:

Problem 4.2.1. *Given a set I of n items of size $w_i \leq W, h_i \leq H$, each due by a specific date d_i, $i \in I$, find all the item allocations to bins whose values $(N, L_{\max})$ are strictly non-dominated.*

Let us now focus on some properties of non-dominated solutions of Problem 4.2.1: specifically, on the dependence of $L_{\max}$ on the filling time τ (§4.2.1), and on upper bounds that can be obtained exploiting the relation between N and $L_{\max}$ (§4.2.2).

4.2.1 Dependence on τ

Non-dominated solutions of 2BP-DD depend on τ in a non-trivial way. Suppose to draw the Pareto region for given due-dates d_i and variable τ. As a general observation, the smaller the τ, the smaller the minimum $L_{\max}$, while the minimum N does not depend on τ and is therefore unchanged. But, contrary to intuition, the relative positions of Pareto-optima are not necessarily preserved as τ decreases. In fact, let I be a 2BP-DD instance with bin processing time $\tau > 1$ and due dates $d_i, i \in I$. Let then x^1 and x^2 be two solutions of

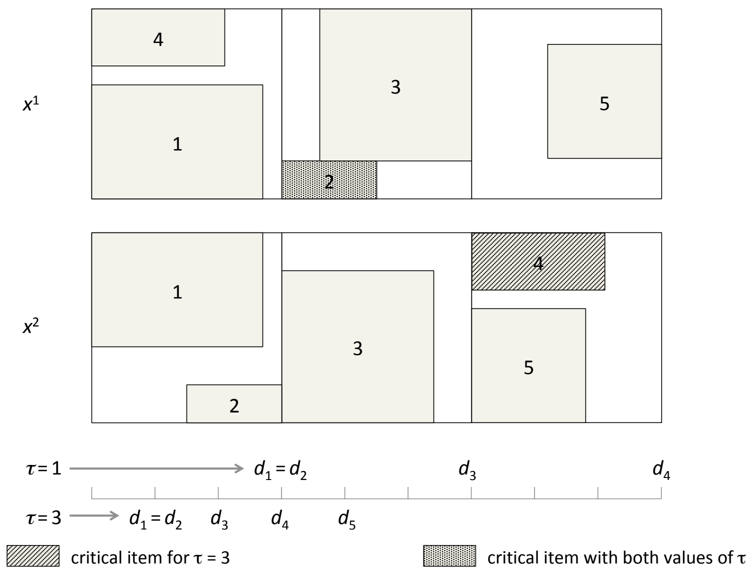

Figure 4.1: How the Pareto region changes with τ, *ceteris paribus*.

$\mathcal{I}$ with $N = z$ bins and minimum $L_{\max}$ of

$$\ell^1 = k\tau - d_p < h\tau - d_q = \ell^2$$

for some k, h, p and q: that is, p and q are the *critical items* determining the minimum $L_{\max}$ in the respective solutions, p (q) is processed in the k-th (in the h-th) bin, and x^1 weakly dominates x^2. Suppose that p and q still determine the minimum $L_{\max}$ in x^1 and x^2 when bin processing time is altered to $\tau' = \alpha\tau$. It is easily seen that x^2 now weakly dominates x^1 for α such that

$$(\alpha - 1)\tau(h - k) < \ell^1 - \ell^2. \tag{4.1}$$

More explicitly (see Figure 4.1), take an instance with $n = 5$ items of size $(w_1, h_1) = (9, 6)$, $(w_2, h_2) = (5, 2)$, $(w_3, h_3) = (8, 8)$, $(w_4, h_4) = (7, 3)$, $(w_5, h_5) = (6, 6)$ and due-dates $d_1 = d_2 = 1$, $d_3 = 2$, $d_4 = 3$ and $d_5 = 4$. Suppose $\tau = 3$ and $W = H = 10$. Consider two solutions x^1 and x^2, both fitting in $N = 3$ bins and sequenced in this way: x^1 has items 1 and 4 packed in the first bin, items 2 and 3 in the second, and item 5 in the third; in x^2, instead, the first bin is filled with items 1 and 2, item 3 is packed in the second, and the remaining items in the third. Then, x^1 weakly dominates x^2 since $\ell^1 = 2\tau - d_2 = 5$ and $\ell^2 = 3\tau - d_4 = 6$. However x^2 weakly dominates x^1 for

$0 < \alpha \leq \frac{2}{3}$: e.g., for $\alpha = \frac{1}{3}$, the minimum $L_{\max}$ become $\ell^1 = 1$ and $\ell^2 = 0$.

Due-date scaling by τ is however an easier matter. In fact, an instance I with $\tau > 1$ is clearly equivalent to I' with $\tau' = 1$ and fractional due-dates d_i/τ. When due-dates are restricted to be integers, the above transformation is of course not allowed but, as we will immediately show, the dependence on τ can be controlled and turns out to be not excessive. Construct in fact a scaled instance I' with due dates

$$d_i' = d_i - (d_i \mod \tau)$$

rounded down to the closest multiple of τ. Take two non-dominated solutions x of I and x' of I' with z bins and minimum $L_{\max}$ of values ℓ, ℓ', respectively. Then

Proposition 1. $\ell' < \ell + \tau$.

Proof. Let C_i, L_i and C_i', L_i' be the completion time and the lateness of item i in solutions x, x'. Therefore

$$C_i - d_i' = L_i + (d_i \mod \tau)$$

is the lateness of item $i \in I$ in solution x re-computed according to the scaled due dates. Because x' is $L_{\max}$-optimal in the scaled problem,

$$\ell' = \max_i \{C_i' - d_i'\} \leq \max_i \{C_i - d_i'\} \leq \max_i \{L_i + (d_i \mod \tau)\} \leq$$

$$\leq \ell + \max_i \{(d_i \mod \tau)\} < \ell + \tau.$$

□

Proposition 1 can easily be extended to $1 < \tau' < \tau$, obtaining in general $\ell' < \ell + \frac{\tau}{\tau'}$. It basically states that the increase of $L_{\max}$ is strictly limited to the time required to pack a single bin. Moreover, it is easy to see that the frontier of the strictly non-dominated solutions of I can be obtained by the set of all the weakly non-dominated solutions of I'.

4.2.2 Upper bounds to N

Clearly, minimizing N does not correspond to minimizing $L_{\max}$: Example 2.3 in [12] shows that a solution with a minimum number N^* of bins and $L_{\max} = N^* - 1$ can be converted into one with $N^* + 1$ bins and $L_{\max} = 0$. However, a strong correlation between the two objectives exists due to Assumption 2. Indeed, given a solution of value (z, ℓ), let

$$I_\ell = \{i \in I : L_i < \ell\}$$

denote the set of items delayed less than ℓ. For any $i \in I_\ell$, define

$$\Delta_i(\ell) = \left\lfloor \frac{\ell - L_i}{\tau} \right\rfloor$$

that is, $\tau\Delta_i(\ell)$ is the largest delay of item i that does not affect the lateness bound ℓ: in symbols, $L_i + \tau \cdot \Delta_i(\ell) \leq \ell \, \forall i \in I_\ell$. Let C_i, an integer multiple of τ, denote the completion time of $i \in I$ in a solution of value (z, ℓ). The two-stage approach of [110] embeds the same idea behind Proposition 2.

Proposition 2. *For any solution of 2BP-DD with lateness* $L_{\max} = \ell$,

$$N \leq \max_{i \in I_\ell} \{C_i/\tau + \Delta_i(\ell)\}.$$

Proof. By the above definition, in any solution with lateness ℓ, item i is delayed at most $\tau\Delta_i(\ell)$. So its completion time cannot be more than $C_i + \tau\Delta_i(\ell)$, and therefore the number N of bins of any such solution is at most $\max_{i \in I_\ell}\{C_i/\tau + \Delta_i(\ell)\}$. □

Solutions of the single-objective 2BP can be exploited to obtain bounds to non-dominated solutions of 2BP-DD. Let items be sorted by early due date first (EDD), that is, $d_1 \leq \ldots \leq d_n$, and let $(z_{EDD}, \bar{\ell})$ be the value of a 2BP-DD solution computed by a Next Fit algorithm regardless of lateness optimization. Then:

Proposition 3. *Any non-dominated solution has* $N \leq z_{EDD}$ *bins.*

Proof. Let $J = \{i \in I : \frac{C_i}{\tau} + \Delta_i(\bar{\ell}) \geq z_{EDD} + 1\}$. Due to the Next Fit rule, if the item in J with the smallest due-date is packed in the k-th bin, then bins from $k+1$ to z_{EDD} contain only items of J. The statement is trivial for $J = \emptyset$. Suppose $J \neq \emptyset$. If there exists a permutation of the items in the first k bins with $L_{\max} < \bar{\ell}$, then an additional bin is useless. Otherwise, let $h \in I$ be an item defining $\bar{\ell}$ (namely, $L_h = \bar{\ell}$). A better solution can only be achieved by exchanging h with another item i packed in one of the previous bins of the current solution. However, $d_i \leq d_h$ holds for all these items and the exchange would imply $L_{\max} \geq \bar{\ell}$. □

In other words, Proposition 3 says that N and $L_{\max}$ trade off only for N up to a given threshold, beyond which $L_{\max}$ increases with N.

4.3 Approximating non-dominated solutions

In multicriteria optimization, several methods were defined to exactly identify the set of non-dominated solutions. Generally speaking, such methods can be distinguished in scalarization-based (e.g., weighted combination, ϵ-constraint, Tchebycheff metric) and non-scalarizing (e.g., lexicographic, max-ordering), see [122]. Whatever method is adopted, providing a complete description of the Pareto set is computationally challenging, and an approximated description of the Pareto set via (meta)heuristics is often an acceptable compromise [55]. Nevertheless, any approximation raises the question of how to evaluate the quality of the frontier found, and which approximation measures are the most adequate. In the literature, several measures were proposed [141], but just few of them consider the worst case analysis.

To the best of our knowledge, there are two main approaches to measure the worst case approximation of algorithms for p-criteria problems: one [54], discussed in §4.3.1, is suitable for heuristics that produce a single solution to be compared to the whole Pareto frontier, the other [103] is used for algorithms that produce a set of solutions that do not dominate each other. In particular, according to [103], an algorithm $\mathcal{H}$ is ϵ-approximated if, for every non-dominated solution x^*, there exists a solution $x_{\mathcal{H}}$ such that

$$f_j(x_{\mathcal{H}}) \leq (1+\epsilon) f_j(x^*)$$

holds for each criterion j. However, this kind of approximation cannot be adopted in our case (and in general when the optimal value of one of the criteria can be zero) because the ratio $L_{\max}(x_{\mathcal{H}})/L_{\max}(x^*)$ is not well-defined.

If instead we consider the absolute error $E_A(\mathcal{H}) = L_{\max}(x_{\mathcal{H}}) - L_{\max}(x^*)$, we observe as expected that most ϵ-approximate algorithms $\mathcal{H}_{2BP}$ for 2BP can perform arbitrarily bad, i.e., the bound $E_A(\mathcal{H}_{2BP}) \leq \tau(1+\epsilon)N(x^*) - \tau$ is tight. One exception is the Next Fit algorithm, where items are sorted by EDD, which is 3-approximate for 2BP. Indeed:

Proposition 4. *With the Next Fit heuristic, the absolute error of* $L_{\max}$ *is bounded by* $3\tau N(x^*) - \tau$

Proof. see [112], Corollary 3. □

4.3.1 Single solution approximation

The framework proposed by [54] relies on two norm-based definitions of approximation ratio. Given an algorithm $\mathcal{H}$ that provides a solution $x_{\mathcal{H}}$, its performance is measured as

$$R_1(x_{\mathcal{H}}, x^*) = \frac{|\|f(x_{\mathcal{H}})\| - \|f(x^*)\||}{\|f(x^*)\|} \quad \text{or} \quad R_2(x_{\mathcal{H}}, x^*) = \frac{\|f(x_{\mathcal{H}}) - f(x^*)\|}{\|f(x^*)\|}$$

where x^* is a non-dominated solution, $f(x) \in \mathbb{R}_p$ denotes the objective function vector and $\|\cdot\|$ is a monotone norm. Note that R_1 compares the norms of the vectors, whereas R_2 evaluates the difference of vector components.

We then say that an algorithm $\mathcal{H}$ is R_i-approximated with $\mu \in \mathbb{R}^+$ if $x_{\mathcal{H}}$ fulfils

$$R_i(x_{\mathcal{H}}, x^*) \leq \mu \tag{4.2}$$

for all non-dominated solutions x^*. As (4.2) gives a tighter definition of approximation for $i = 1$ than for $i = 2$, an algorithm $\mathcal{H}$ that is R_1-approximated with some $\mu \in \mathbb{R}^+$ is also R_2-approximated with the same μ. Note that this definition covers two types of approximation, one of which derives from taking a single (heuristic, but even non-dominated) solution to represent the whole Pareto-frontier.

Of course, the approximation ratio depends in general on the norm used. In the following we will refer to the general norm $\|.\|_q$. Observe that for $q = 1$, R_1-approximation reads in our case

$$\frac{N(x_{\mathcal{H}}) + L_{\max}(x_{\mathcal{H}}) - N(x^*) - L_{\max}(x^*)}{N(x^*) + L_{\max}(x^*)} \leq \mu$$

that corresponds to the μ-approximation in the ordinary sense of the single objective problem where one wants to minimize $N + L_{\max}$. As noticed in [12] (Proposition 2.1), this problem can be approximated by heuristics for the traditional 2BP problem. In fact, 2BP can be 1-approximated both in the *oriented* [71] and the *non-oriented* [66] case. The algorithm of [66] finds a feasible solution within $2N^*$ bins in polynomial time by performing a separation between large and small items based on their area; it exploits as subroutines the Steinberg's algorithm [120] and the Next Fit Decreasing Height (NFDH) [37] procedure designed for strip packing, whereas the algorithm of [13] is used to guarantee the approximation ratio in the asymptotic case. The same absolute factor for the *oriented* case is attained by the algorithm of [71], which strongly relies on the PTAS for 2D knapsack problem presented in [14] and the techniques for rectangle packing described in [72]. The area and the sizes of the items are concurrently combined to identify a more extensive separation, while the Steinberg's and NFDH algorithms are exploited again and the asymptotic approximation ratio is ensured through the algorithm of [70] for large optimal values.

Therefore, $\frac{z_{2BP} - N^*}{N^*} \leq 1$, where N^* is the minimum feasible number of bins and z_{2BP} is the number of bins used by the approximating algorithm $\mathcal{H}_{2BP}$. For $\tau = 1$ this guarantees an R_1-approximation of Problem 4.2.1 with $\mu = 3$ under the $\|.\|_1$ norm, see [12].

The definition of 1-approximation trivially implies $\frac{z_{2BP} - (N^* + k)}{N^* + k} \leq 1$ for any

$k \in \mathbb{N}$. Now note that one can explore all the possible z-values of the Pareto-optimal set by varying k, with at most one non-dominated solution for any given k.

So let $\mathcal{H}_{2BP}$ be a 1-approximate algorithm for 2BP returning a solution of value (z_{2BP}, ℓ_{2BP}), and let $(\bar{z} = N^* + k, \bar{\ell})$ be the value of a non-dominated solution of 2BP-DD, where N^* denotes the minimum number of bins of 2BP and $k \in \mathbb{N}$. For the following two results it is useful to recall that, for $\tau \geq 1$, the maximum lateness of any solution can never exceed its total completion time, i.e., for any 2BP-DD solution of value (z, ℓ) one has $\ell \leq \tau z$; moreover, $\mathcal{H}_{2BP}$ is 1-approximating, hence $z_{2BP} \leq 2N^*$, and therefore $z_{2BP} \leq 2(N^* + k)$ clearly holds for $k \geq 0$ and any solution of $\mathcal{H}_{2BP}$.

Proposition 5. *$\mathcal{H}_{2BP}$ is R_1-approximated for 2BP-DD under $\|.\|_q$ with*

$$\mu = 2\sqrt[q]{1+\tau^q} - 1$$

for any $\tau \geq 1$.

Proof. Writing (4.2) for 2BP-DD and $i = 1$, we obtain:

$$(1-\mu)\sqrt[q]{(N^*+k)^q + \bar{\ell}^q} \;\leq\; \sqrt[q]{z_{2BP}^q + \ell_{2BP}^q} \;\leq\; (1+\mu)\sqrt[q]{(N^*+k)^q + \bar{\ell}^q}.$$

The left inequality holds for any $\mu \geq 1$. As previously recalled, $\ell \leq \tau z$ for any 2BP-DD solution: hence $\ell_{2BP}^q \leq \tau z_{2BP}^q$. Bounding z_{2BP} via the 1-approximation ratio of $\mathcal{H}_{2BP}$ implies, as observed, $z_{2BP} \leq 2(N^* + k)$: the right inequality then reduces to

$$\sqrt[q]{(1+\tau^q) z_{2BP}^q} \;\leq\; \sqrt[q]{2^q(1+\tau^q)(N^*+k)^q} \;\leq\; (1+\mu)\sqrt[q]{(N^*+k)^q + \bar{\ell}^q}.$$

Without loss of generality, we set $\bar{\ell} = 0$ to tighten the right-hand side. Indeed, any valid approximation result still holds for any $\bar{\ell} \geq 0$. Then, the inequality becomes

$$\sqrt[q]{2^q(1+\tau^q)(N^*+k)^q} \;\leq\; (1+\mu)\sqrt[q]{(N^*+k)^q}$$

and the result is easily derived. □

Proposition 6. *$\mathcal{H}_{2BP}$ is R_2-approximated for 2BP-DD under $\|.\|_q$ with*

$$\mu = \sqrt[q]{1 + 2^q\tau^q}$$

for any $\tau \geq 1$.

Proof. Adapting (4.2) with $i = 2$ to our case we obtain

$$\sqrt[q]{|z_{2BP} - N^* - k|^q + |\ell_{2BP} - \bar{\ell}|^q} \;\le\; \mu \sqrt[q]{(N^* + k)^q + \bar{\ell}^q}. \tag{4.3}$$

Using the 1-approximation ratio of $\mathcal{H}_{2BP}$, we have

$$-(N^* + k) \;\le\; z_{2BP} - (N^* + k) \;\le\; (N^* + k)$$

which implies

$$|z_{2BP} - (N^* + k)|^q \;\le\; (N^* + k)^q.$$

Plugging this inequality into (4.3) we get

$$\sqrt[q]{(N^* + k)^q + |\ell_{2BP} - \bar{\ell}|^q} \;\le\; \mu \sqrt[q]{(N^* + k)^q + \bar{\ell}^q}.$$

Now two possibilities arise:

1. if $\ell_{2BP} < \bar{\ell}$, then clearly $(\bar{\ell} - \ell_{2BP})^q \le \bar{\ell}^q$ and the inequality holds for any $\mu \ge 1$.

2. if $\ell_{2BP} \ge \bar{\ell}$, then we use $\bar{\ell} = 0$ to tighten the inequality; then raising a power of both sides we obtain:

   $$\ell_{2BP}^q \;\le\; (\mu^q - 1)(N^* + k)^q.$$

   Bounding ℓ_{2BP}^q as in Proposition 5, we further get

   $$|\tau z_{2BP}|^q \;\le\; 2\tau(N^* + k)^q \;\le\; (\mu_2^q - 1)(N^* + k)^q$$

   from which the result is straightforward.

□

4.4 A sequential value correction heuristic

In an optimization model with exponentially many variables, solving a pricing problem is fundamental for generating potentially advantageous columns. In the 0-1 LP formulation we refer to for the 2BP problem, columns correspond to optional bin fillings, and the advantage of choosing a particular filling is measured by summing the *shadow prices* of the items that fill the bin. Such prices are the components of an optimal dual solution that correspond to the items at any iteration of the column generation procedure.

In 2BP, solving the pricing problem amounts to solve a two-dimensional knapsack problem, by far more complex than the one-dimensional knapsack used for pricing in 1BP. To save CPU time, one can avoid the LP that is necessary to resolve in order to get the exact shadow prices, and resort to a heuristic evaluation of their values, called *pseudo-pricing*, that we will soon describe. The idea behind a sequential value correction (SVC) algorithm for BP is then quite simple: given a *pseudo-price* p_i for each item $i \in I$, bins are packed sequentially, each one with the items | not yet allocated | that maximize the sum of pseudo-prices. After a solution has been computed, pseudo-prices are conveniently updated and the process iterated until some halting criterion is fulfilled. In our implementation (here named SVC-DD and described in Algorithm 2), the algorithm ends either when a convenient number P of solutions has been computed or when both the number of bins and the maximum lateness of the current solution $\mathcal{S}_k$ close the gap with the relevant lower bounds n_{LB} and ℓ_{LB} (function check_optimality()).

Our implementation (here named SVC-DD and described in Algorithm 2) differs from those proposed in [20, 42], mainly in how bins are filled and pseudo-prices updated. Moreover, in our case the output is a set of reciprocally non-dominated solutions in terms of the number N of bins filled and the maximum lateness $L_{\max}$.

Besides the efficiency and quality of solutions computed by SVC-DD (see §4.6) it is worth noting that SVC-DD does not use any parameter except the two (P and P_{inner}) for controlling the total number of solutions generated. Therefore, differently from the algorithms for 2RBP-DD proposed in [22] and [110], it needs neither a demanding preliminary parameter tuning nor a sensitive analysis.

Let $\mathcal{D}$ be the generalized item set that for 2OBP-DD coincides with I, while for 2RBP-DD also includes the 90° rotated items $h_i \times w_i$ for every $i \in I$. Moreover, let z_{LB} and ℓ_{LB} be lower bounds to N and $L_{\max}$, respectively.

4.4.1 Pseudo-price update

Due to the bi-criteria objective, pseudo-prices reflect both the packing and scheduling quality of the solutions generated. Assume items sorted by nondecreasing due-dates, i.e., $d_1 \leq \ldots \leq d_n$. Initially, the function init_prices() sets the pseudo-prices to

$$p_i := \frac{WH}{d_i + 1} + w_i h_i \qquad i \in \mathcal{D} \tag{4.4}$$

where the first term promotes urgent items, while the second increases the price of large items.

Algorithm 2 SVC-DD

```
**Input:** $\mathcal{D}, W, H, z_{LB}, \ell_{LB}$
Output: solutions $\mathcal{S}_1, \dots, \mathcal{S}_P$
$\mathbf{p} \leftarrow$ init_prices$(\mathcal{D}, W, H)$
**for** $k \leftarrow 1, \dots, P$ **do**
    $\mathcal{S}_k \leftarrow \emptyset$
    $\bar{\mathcal{D}} \leftarrow \mathcal{D}$
    **repeat**
        $\mathcal{B} \leftarrow \emptyset$
        $[S, l(r_{\min}), q(r_{\min})] \leftarrow$ init_skyline()
        **repeat**
            $\tilde{\mathcal{D}} \leftarrow$ select_items$(\bar{\mathcal{D}}, l(r_{\min}), q(r_{\min}))$
            $\mathcal{K}^* \leftarrow$ KP_Conflict$(\tilde{\mathcal{D}}, l(r_{\min}), q(r_{\min}))$
            $\mathcal{B} \leftarrow$ add_items$(\mathcal{K}^*)$
            $[S, l(r_{\min}), q(r_{\min})] \leftarrow$ update_skyline$(\mathcal{K}^*)$
            $\bar{\mathcal{D}} \leftarrow$ remove_items$(\mathcal{K}^*)$
        **until** $(l(r_{\min}), q(r_{\min})) \neq (W, H)$
        $\mathcal{S}_k \leftarrow \mathcal{S}_k \cup \{\mathcal{B}\}$
    **until** $(\bar{\mathcal{D}} \neq \emptyset)$
    **if** check_optimality$(\mathcal{S}_k, n_{LB}, \ell_{LB})$ **then return**
    **if** $(i-1) \equiv 0 (\mod P_{inner})$ **then** $\delta \leftarrow$ random$(0, 1]$
    $\mathbf{p} \leftarrow$ update_prices$(\mathcal{S}_k, \mathbf{p}, z_{LB}, \ell_{LB}, \delta)$
```

After the current solution $\mathcal{S}_k$ has been computed, pseudo-prices are updated in function update_prices(). First, the value of p_i, $i \in \mathcal{D}$, is increased by a factor accounting for the unused space of the current solution $\mathcal{S}_k$:

$$p_i := p_i \left(1 + \delta \cdot \frac{z_{LB}}{m}\right)^{\frac{\omega(i)-\bar{\omega}}{mWH}} \tag{4.5}$$

where $\omega(i)$ gives the value of the waste within the bin $\mathcal{B} \in \mathcal{S}_k$ in which item i is packed, and $\bar{\omega} = \frac{WHz_{LB} - \sum_{i \in D} w_i h_i}{z_{LB}}$ is a lower bound to the relative waste. Since the residual of a bin generally increases with the bin relative position in the sequence, the formula promotes late packed items with bad patterns, thus ensuring a suitable mix of the search space. The exponent and the constant term in the base are small fractional numbers that prevent pseudo-prices from diverging quickly.

Also, the pseudo-price of the tardiest item h (and possibly of its rotated counterpart) is further increased by a factor proportional to the current maximum lateness:

$$p_h := p_h \left\{ 1 + \left[1 + \delta \cdot \left(\frac{\ell_{LB}}{\tau \cdot z_{LB}} \right) \right] \cdot \left[\frac{(C_h - \ell_{LB})/\tau}{d_h + 1} \right] \right\}. \tag{4.6}$$

where as usual C_h is the completion time of h in the current solution $\mathcal{S}_k$. Coefficient δ is randomly selected in $(0, 1]$ to perturb pseudo-price dynamics, and is renewed each P_{inner} main iterations. The fractional factor $\left(1 + \delta \cdot \frac{\ell_{LB}}{\tau \cdot n_{LB}}\right) > 1$ is generally very close to 1, whereas the second component of the product linearly grows with the current lateness of item h. Again, numbers are so designed as to induce relatively small increments of the current pseudo-price.

Update (4.5) is omitted if the number z of bins in the current solution reaches the corresponding lower bound, thus being certified optimal.

4.4.2 Bin filling

The bin filling procedure here described inserts items of $\mathcal{D}$ into the current bin $\mathcal{B}$ in a bottom-up fashion by iterated solution of one-dimensional knapsack problems. At the end of each step, the upper borders of the items inserted describe a curve that we call *skyline*.

Focusing on a generic step of the procedure, let $\bar{\mathcal{D}} \subseteq \mathcal{D}$ be the set of items not yet allocated, and let $S = \langle r_1, \dots, r_{|S|} \rangle$ (the *skyline*) be an ordered list of segments associated with the bin $\mathcal{B}$ being filled, see Figure 4.2-4.3. Each segment r_j of S (a *roof* of the skyline) is described by a quota $q(r_j)$ and a length $l(r_j)$; moreover, each endpoint of r_j is delimited by a *wall*, that is a vertical segment joining the endpoint to the adjacent roof, or the vertical edge of the bin, if adjacent. The skyline defines a border within $\mathcal{B}$ above which unpacked items can be placed (see Figure 4.2-4.3; we assume w.l.o.g. that adjacent roofs have distinct quotas: consecutive roofs at the same quota are merged into a single one).

At the beginning, i.e., when $\mathcal{B}$ is empty, the function init_skyline() sets the skyline to a single “ground" roof r_1 with $q(r_1) = 0$ and $l(r_1) = W$; the bin filling process iteratively places unpacked items on the lowest roof of the skyline. Specifically, let $r_{\min}$ be the roof at minimum quota in the current skyline

1. If no item can be placed on top of $r_{\min}$ | viz., $h_i > H - q(r_{\min})$ for each $i \in \bar{\mathcal{D}}$ | the procedure ends.

2. Otherwise (Figure 4.2) function KP_Conflict() defines and solves a 0-1 one-dimensional knapsack problem (KP), with knapsack capacity $l(r_{\min})$ and items in $\tilde{\mathcal{D}} = \{i \in \bar{\mathcal{D}} | w_i \leq l(r_{\min}) \text{ and } h_i \leq H - q(r_{\min})\}$:

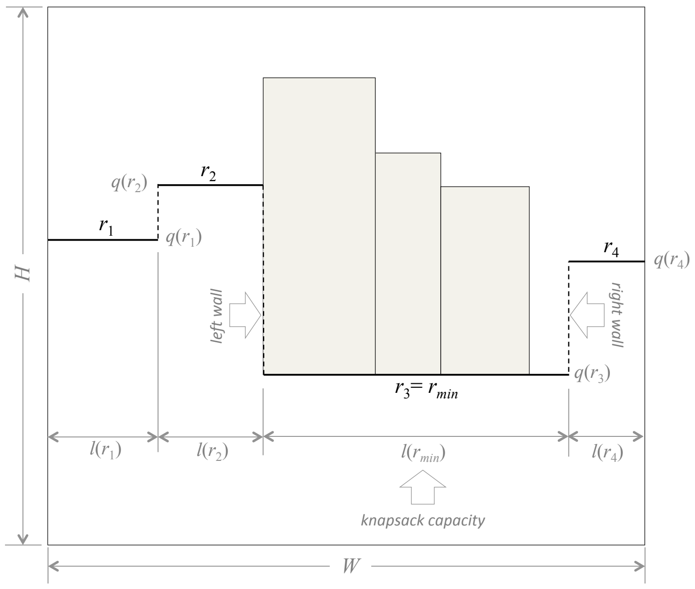

Figure 4.2: A skyline, with indication of a roof at minimum quota and of the relevant knapsack; the items in a knapsack solution are ranked by non increasing heights, starting from the highest wall of $r_{\min}$ | in this case, the left one.

the items of an optimal solution $\mathcal{K}^* \subseteq \tilde{\mathcal{D}}$ of KP are placed on top of $r_{\min}$, sorted by non-increasing heights and placed in this order starting from the highest among the delimiting walls.

3. Then (Figure 4.3) function update_skyline() updates the skyline by adding a new roof for each item of $\mathcal{K}^*$ and possibly enlarging one roof adjacent to $r_{\min}$ of an amount corresponding to the unused knapsack capacity. The skyline update involves a new $r_{\min}$, and the procedure is iterated going back to step (1).

Items in $\mathcal{K}^*$ (and possibly their rotated counterparts) are removed from $\bar{\mathcal{D}}$ and the process iterated.

For 2OBP-DD, KP is a standard 0-1 knapsack problem. For 2RBP-DD, however, it becomes a knapsack problem with compatibility constraints: in fact, at most one item of $\mathcal{D}$ out of $w_i \times h_i$, $h_i \times w_i$ can be allocated to the knapsack. In our implementation, we use a standard knapsack solver and handle conflicting items heuristically: while the solution $\mathcal{K}^*$ contains conflicting items,

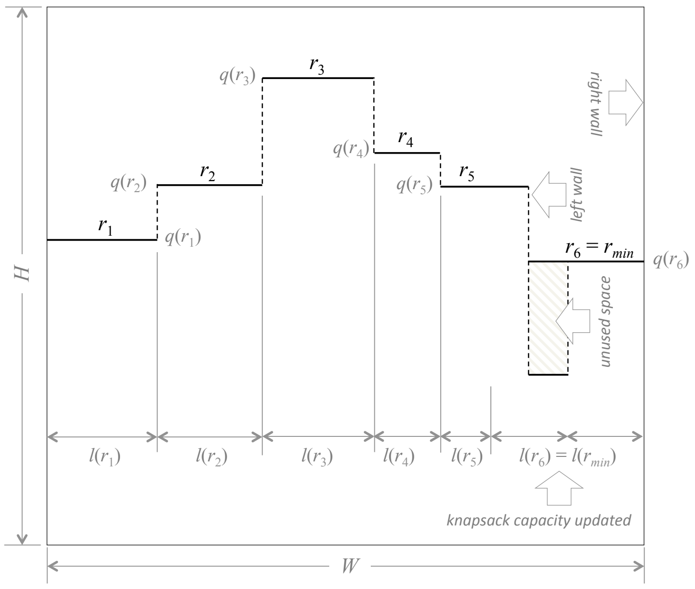

Figure 4.3: The skyline updated by adding the new roofs, renumbering, and enlarging r_6 to its left with the unused space (dashed area). After update, it turns out in this example $r_{\min} = r_6$.

we remove from $\tilde{\mathcal{D}}$ the one with smallest pseudo-price per surface unit, i.e., a conflicting item i with minimum $p_i/(w_i h_i)$; then we solve KP again. Although potentially demanding in terms of CPU time, such a simple strategy performs very well since conflicts seldom occur (roughly, in 4% of the problems solved).

4.5 Dual bounds

The strong correlation determined by Assumption 2 between the minimum number N of bins and the minimum $L_{\max}$ still holds for dual bounds, and lower bounds to N can be exploited to build lower bounds to $L_{\max}$ (see [22]).

A simple lower bound for 2OBP can easily be derived by considering the two one-dimensional bin packing problems with bins of sizes W and H and items of sizes $w_1, \ldots, w_n$ and $h_1, \ldots, h_n$, respectively. Tighter bounds can be derived by using *dual feasible functions* (DFF) as described in [7]:

Definition 1. *A discrete function $f : [0, C] \to [0, C']$ (with C, $C' \in \mathbb{Z}$) is* dual feasible *if*

$$\sum_{x \in S} x \leq C \Rightarrow \sum_{x \in S} f(x) \leq f(C) = C'$$

for any finite discrete set S.

Although such bounds are not valid for 2RBP, which is a problem less constrained than 2OBP, they can still be exploited provided that the instances of 2RBP are adequately relaxed, as proposed in [36]: given an instance of 2RBP with item set $\mathcal{I}$ and square bins, construct an instance of 2OBP with the item set $\mathcal{D}$ obtained as described in §4.4, i.e., by expanding I with the 90° rotated counterpart of each item. Then, for any lower bound N^o_{LB} to the 2OBP-optimum of $\mathcal{D}$, $N^r_{LB} = \left\lceil \frac{N^o_{LB}}{2} \right\rceil$ is a lower bound to the 2RBP-optimum of $\mathcal{I}$. The proposed approach can be employed also when bins of $\mathcal{I}$ are rectangular, by transforming them into squares and introducing an appropriate number of dummy items to avoid over-relaxation. However, the authors of [36] show that N^r_{LB} dominates known lower bounds of 2RBP only in the case of square bins.

A lower bound to $L_{\max}$ proposed in [22] is based on the joint consideration that (i) the EDD rank $\sigma = \langle 1, \ldots, n \rangle$ is optimal for $\min L_{\max}$ 1-machine scheduling, (ii) the minimum time at which the k-th item in σ is scheduled in a combined packing-scheduling problem is at least τ times the minimum number of bins necessary to accommodate the first k items in the rank, $\mathcal{F}_k = \{1, \ldots, k\}$, or anyway no less than a lower bound N^k_{LB} to this value. Therefore, a lower bound to L_k is

$$L^k_{LB} = \tau N^k_{LB} - d_k$$

and the bound to $L_{\max}$ is computed as

$$L_{LB} = \max_{k=1,\ldots,n} \{L^k_{LB}\}. \tag{4.7}$$

The tighter the N^k_{LB}, the more L_{LB} becomes effective, and we note here that L^k_{LB} can be further tightened under simple conditions even when N^k_{LB} is the optimal number of bins. In fact, L_{LB} is obtained by computing all the terms appearing in (4.7), as if item k were packed in the N^k_{LB}-th bin, even when no feasible solution with $L_{\max} = L^k_{LB}$ can place k in that bin.

Consider the following simple example: there are $n = 8$ items, with sizes and due dates as in Table 4.1, to be packed into bins of size $L \times H = 1 \times 10$ and scheduled in time slots of length $\tau = 1$. Items can be packed using four bins, thus $N^n_{LB} = 4$ and formula (4.7) returns $L_{LB} = 0$. No EDD rank can be packed in four bins, though (see Figure 4.4): either the completion time increases by one time unit as in the leftmost schedule; or, as in the rightmost, compressing

Table 4.1: Data for an example of lower bound to $L_{\max}$.

item	length l_i	height h_i	due date d_i
1	1	1	1
2	1	2	1
3	1	3	1
4	1	4	1
5	1	6	4
6	1	7	4
7	1	8	4
8	1	9	4

the items in less bins has the effect of 'dragging' urgent items towards the end of the schedule, so making an EDD rank impossible. The shortest EDD schedule and the tightest packing (which does not admit an EDD schedule) respectively give $L_{\max} = 1$ and $L_{\max} = 3$, hence $L_{\max}$ cannot be less than 1 time unit.

Generalizing the above example, we can improve L_{LB}. Let $\sigma = \langle 1, \ldots, n \rangle$ be an EDD rank. Call items h and k *compatible* if they can share the same bin and, for $h < k$, let $\bar{N}^i_{LB}$ be a lower bound to the tightest packing of

- $\{1, \ldots, i\}$ for $1 \leq i \leq h$ (that is $\bar{N}^i_{LB} = N^i_{LB}$)
- $\{1, \ldots, i, k\}$ for $h < i < k$

Finally, let $N^{i,\text{EDD}}_{LB}$ be a lower bound to the tightest packing of the first i items of σ, subject to the condition that bins can be ordered respecting the EDD rank expressed by σ. Then

Proposition 7. *Suppose that, for some $k \leq n$, $N^{k,\text{EDD}}_{LB} > N^k_{LB} = N^{k-1}_{LB}$. Let h be the closest item that is compatible with k in the EDD rank σ, and*

$$L' = \max \left\{ \begin{array}{ll} \tau N^i_{LB} - d_i & 1 \leq i \leq h \\ \tau \bar{N}^i_{LB} - d_i & h < i < k \end{array} \right\}$$

$$L'' = \max \left\{ \begin{array}{ll} \tau N^i_{LB} - d_i & 1 \leq i < k \\ \tau (N^i_{LB} + 1) - d_i & i = k \end{array} \right\}$$

Then

$$\bar{L}^k_{LB} = \min\{L', L''\} \tag{4.8}$$

is a lower bound to $L_{\max}$.

Proof. If $N^k_{LB} = N^{k-1}_{LB}$, then the tightest packing of $\{1, \ldots, k\}$ necessarily places the k-th element with one compatible with k, say $h < k$. Consider the sequence

$$\bar{\sigma} = \langle 1, \ldots, h-1, h, k, h+1, \ldots, k-1 \rangle$$

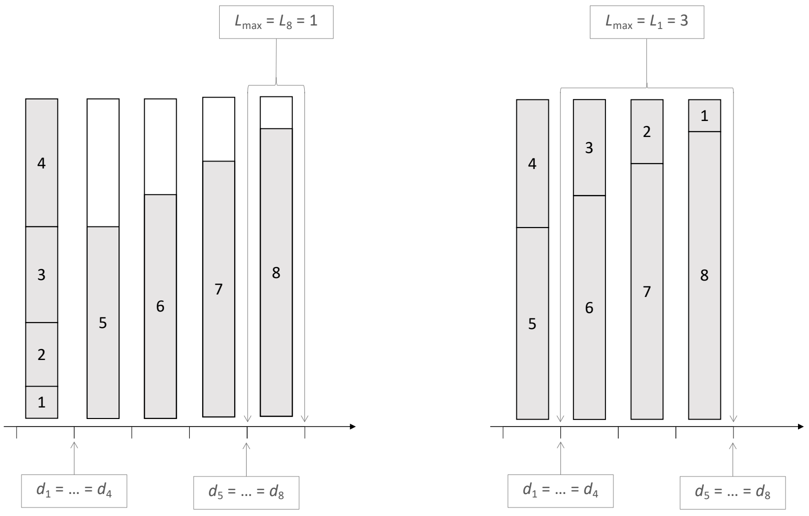

Figure 4.4: $L_{\max}$ obtained with the tightest packing of an EDD rank (left) and the tightest packing at all.

which is compliant with this packing. As h and k are paired in the same bin, they can regarded as a single item due by $\min\{d_h, d_k\} = d_h$, therefore $\bar{\sigma}$ is an EDD rank on $k-1$ items. Then, using (4.7) and recalling the definition of $\bar{N}^i_{LB}$ we obtain:

$$\begin{array}{lcll} L_{\max} & \geq & \tau N^i_{LB} - d_i & 1 \leq i \leq h \\ L_{\max} & \geq & \tau \bar{N}^i_{LB} - d_i & h < i \leq k-1 \end{array}$$

where, to alter item completion times as least as possible, h is the closest item in σ that is compatible with k. Note that h and k are completed at the same time and $d_h \leq d_k$: so we do not need to specify a bound for $i = k$.

In alternative, we can bound L_i using σ, with N^i_{LB} as an (optimistic) completion time of $\{1, \ldots, i\}$ for $i < k$, and $N^{k,\text{EDD}}_{LB} \geq N^k_{LB} + 1$ for $i = k$:

$$\begin{array}{lcll} L_{\max} & \geq & \tau N^i_{LB} - d_i & 1 \leq i < k \\ L_{\max} & \geq & \tau (N^i_{LB} + 1) - d_i & i = k \end{array}$$

Choosing the best of the two cases above, we finally obtain (4.8). $\square$

By Proposition 7 we get an improved lower bound $\bar{L}_{LB}$ which is generally more effective in presence of large items. At the expense of additional CPU time, the result can be further reinforced by considering the items with both sides exceeding half the sides of the bin:

$$I_L = \{i \in I : \min(w_i, h_i) > \max(\frac{W}{2}, \frac{H}{2})\}$$

For the 2BP, pairs in I_L cannot be packed in the same bin. When computing L^i_{LB}, for each $1 \leq i \leq h$ the bound on the number of bins N^i_{LB} can be tightened by assuming items in I_L as packed in different bins, thus reducing the area available to pack the remaining items in such bins.

As described in [110], a lower bound to $L_{\max}$ can also be obtained by relaxing the geometrical constraints of a MILP formulation for the maximum lateness minimization into DFFs-based inequalities which ensure that the total area of items inside each bin is dual feasible. Lower bounds obtained in this way are generally good, see §4.6, but the approach calls for the solution of a MILP which is usually time-consuming and poorly scalable.

We close this section by observing that the specific structure of L_{LB} also allows to infer the cardinality of the Pareto frontier. Let $\bar{x} = (\bar{z}, \bar{\ell})$ be a feasible solution.

Proposition 8. *If $\bar{z} = N_{LB} + h$ and $\bar{\ell} \leq \max\{L_{LB}, \tau(\bar{n} + k) - d_n\}$, with $h \geq 0$ and $k \geq 1$, then the set of strictly non-dominated solutions contains at most $h + k$ points.*

Proof. Let $\tilde{x} = (\bar{z} + k, \tilde{\ell})$ be a feasible solution. $\tilde{\ell} \geq \tau(N_{LB} + h + k) - d_n$ by (4.7) therefore $\tilde{x}$ is dominated by $\bar{x}$, and there are at most $h + k$ non-dominated solution with less than $\bar{z} + k$ bins. □

4.6 Computational results

The SVC-DD algorithm was coded in C++ and compiled with Microsoft® C/C++ Optimizing Compiler (version 19.11.25447). Parameters P and P_{inner} were set to 10^3 and 10^2, respectively. Computational tests were performed on a Intel® Core(TM) i7-7500U 2.90 GHz with 16Gb RAM.

The experiments were conducted on two sets $\mathcal{I}_R$ and $\mathcal{I}_B$ of instances. The former derives from 28 industrial orders with up to 360 items with sizes $w_i \in [30, 2928]$, $h_i \in [24, 1820]$ and rectangular large bins with sizes $W \in [2200, 5600]$, $H \in [1163, 2100]$. For each instance, Table 4.2 reports the number m of items, the sizes W and H of the bin, the ratio $\frac{w_i h_i}{WH}$ between bin and item areas (min, max, and mean value) and the ratio $\frac{w_i}{h_i}$ between item widths and heights (min, max, and mean value).

Table 4.2: details of $\mathcal{I}_R$-instances. The complete set of instances is available from the author.

ID	n	W, H	$\min_{i\in\mathcal{I}} \frac{w_ih_i}{WH}$	$\max_{i\in\mathcal{I}} \frac{w_ih_i}{WH}$	$avg_{i\in\mathcal{I}} \frac{w_ih_i}{WH}$	$\min_{i\in\mathcal{I}} \frac{w_i}{h_i}$	$\max_{i\in\mathcal{I}} \frac{w_i}{h_i}$	$avg_{i\in\mathcal{I}} \frac{w_i}{h_i}$
r_1	11	2800, 2070	0.047	0.59	0.30	1.38	3.28	2.83
r_2	14	2800, 2070	0.081	0.54	0.30	0.63	4.38	2.27
r_3	24	2800, 2070	0.020	0.22	0.08	0.56	5.62	2.29
r_4	27	2800, 2070	0.045	0.53	0.27	0.68	4.60	2.39
r_5	28	3500, 1750	0.012	0.09	0.04	0.61	3.41	1.51
r_6	30	2800, 2070	0.008	0.19	0.04	0.38	5.69	2.27
r_7	33	2800, 2070	0.052	0.50	0.24	0.67	11.32	3.13
r_8	42	2800, 2070	0.011	0.16	0.06	0.55	5.87	2.70
r_9	49	2550, 2070	0.049	0.54	0.28	0.58	6.28	2.47
r_{10}	61	4250, 1860	0.010	0.42	0.08	0.50	7.51	2.49
r_{11}	63	2550, 2100	0.142	0.53	0.35	2.16	8.07	3.50
r_{12}	67	2750, 2100	0.012	0.38	0.13	0.69	21.82	3.28
r_{13}	69	5600, 1163	0.013	0.22	0.07	0.27	4.07	1.41
r_{14}	83	2800, 2070	0.003	0.08	0.02	0.60	9.10	2.81
r_{15}	84	2440, 1830	0.005	0.27	0.06	0.43	17.89	2.33
r_{16}	86	2550, 2070	0.193	0.50	0.36	2.31	5.97	3.46
r_{17}	87	2550, 2070	0.057	0.63	0.32	0.53	12.60	3.12
r_{18}	98	2740, 1840	0.004	0.30	0.04	0.18	29.35	3.75
r_{19}	119	2800, 2070	0.001	0.35	0.10	0.11	21.33	4.55
r_{20}	141	2440, 1220	0.003	0.25	0.08	0.14	16.67	2.56
r_{21}	164	2740, 1840	0.003	0.70	0.08	0.12	34.92	5.39
r_{22}	186	2750, 2090	0.003	0.29	0.05	0.12	31.60	3.85
r_{23}	215	2200, 1220	0.001	0.33	0.05	0.07	58.67	6.04
r_{24}	218	2456, 2070	0.150	0.53	0.35	1.67	6.38	3.44
r_{25}	238	2550, 2100	0.038	0.69	0.23	0.50	7.07	2.39
r_{26}	257	2740, 1840	0.001	0.80	0.06	0.20	36.00	4.27
r_{27}	314	2740, 1840	0.002	0.79	0.08	0.12	36.00	5.84
r_{28}	352	2850, 2100	0.053	0.18	0.06	0.80	15.13	14.93

The latter was kindly provided by the Authors of [22], who added due dates to five hundred benchmark instances reported in [23] (classes I-VI) and [80] (classes VII-X). Each class I to X includes fifty instances grouped by ten in five subsets, each one with $n \in \{20, 40, 60, 80, 100\}$ items and square bins. For each class, Table 4.3 reports the bin size, the assortment of item sizes and, according to [22], the number of instances with $N_{LB} > 1$ (column "# inst."). The other entries in row i give an indication of the frequency of item sizes of class i: for example, 0.7 in the last row and column means that 70% of the items were generated with w, h uniformly chosen in $[1, \frac{1}{2}W]$ and $[1, \frac{1}{2}H]$, respectively.

Due-dates for both $\mathcal{I}_R$ and $\mathcal{I}_B$ instances were generated by randomly pick integers in the interval $[\tau + 1, \tau\beta N_{LB}]$ with $\beta \in \{0.6, 0.8, 1.0\}$ and the N_{LB} employed in [22]: hence $\mathcal{I}_R$ and $\mathcal{I}_B$ consist of three different due-dates groups A, B, C, amounting on the whole to eighty-four $\mathcal{I}_R$-instances and one thousand five hundred $\mathcal{I}_B$-instances.

Although SVC-DD was specifically designed for 2RBP-DD, it performs well also as a bin packing heuristic. Indeed, when SVC-DD is used to solve 2BP (that requires skipping (4.6) and omitting the due-date related term in (4.4)), competitive results are achieved in comparison to the bin packing benchmark

Table 4.3: instances in I_B: bin size and assortment of item sizes for each class

Class	# inst.	W, H		distribution of item sizes						
			w h	$U(1,10)$ $U(1,10)$	$U(1,35)$ $U(1,35)$	$U(1,100)$ $U(1,100)$	$U(\frac{2}{3}W, W)$ $U(1, \frac{1}{s}H)$	$U(1, \frac{1}{2}W)$ $U(\frac{2}{3}H, H)$	$U(\frac{1}{2}W, W)$ $U(\frac{1}{2}H, H)$	$U(1, \frac{1}{2}W)$ $U(1, \frac{1}{2}H)$
I	50	10		1						
II	39	30		1						
III	50	40			1					
IV	39	100			1					
V	35	100				1				
VI	50	300				1				
VII	50	100					0.7	0.1	0.1	0.1
VIII	50	100					0.1	0.7	0.1	0.1
IX	50	100					0.1	0.1	0.7	0.1
X	50	100					0.1	0.1	0.1	0.7

heuristics reported in [42] (see Table 4.4): solutions are in fact obtained in similar running time (0.94 seconds on average), and the number of bins is on average 0.28% worse than that obtained with algorithm SVC2BPRF of [42]. In two cases (classes II, IV) the improvement on SVC2BPRF is strictly positive, and in two cases (class IV and IX) SVC-DD matches all the benchmark heuristics.

Table 4.4: SVC-DD, SVC2BPRF and best H solutions of 2BP on I_B: n mean values

Class	SVC-DD	SVC2BPRF	best H
I	19.46	19.44	19.44
II	2.50	2.54	2.48
III	13.56	13.54	13.54
IV	2.42	2.5	2.42
V	17.32	17.24	17.24
VI	2.26	2.24	2.18
VII	15.34	15.14	15.14
VIII	15.38	15.22	15.22
IX	42.38	42.38	42.38
X	9.88	9.86	9.86
Average	14.05	14.01	13.99

4.6.1 Comparison to other approaches

SVC-DD vs. MXGA

In the first part of our experiments, we compare SVC-DD on data-set I_B to:

- the multicrossover genetic heuristic (MXGA) proposed in [22];
- the hybrid constraint and integer linear programming approach (CPMIP) described in [110].

Both algorithms above are specifically designed for 2RBP-DD. For a fair comparison of results, we use the same value of τ ($= 100$) and solution analysis as in [22, 110], adopting the performance indicators (primal-dual gaps)

$$G_N = 100\frac{(z - N_{LB})}{N_{LB}} \qquad\qquad G_L = 100\frac{(\ell - L_{LB})}{L_{LB}}.$$

that refers to a solution (z, ℓ) with z bins and maximum lateness equal to ℓ. In particular, for each instance we take a solution (π_z) achieving the minimum N and one (π_ℓ, possibly different from π_z) with minimum $L_{\max}$, and then measure their quality by G_N and G_L. For the sake of fairness, we run SVC-DD by using the same N_{LB} [47] and L_{LB} (4.7) of the benchmark algorithms. We indicate with SVC-DD* the case in which the algorithm uses the dual bound for 2RBP [36] and the improvement (4.8) of Proposition 7. Note that formulas (4.5) and (4.6) are dependent by the employed dual bounds, thus primal solutions computed by SVC-DD* can be different w.r.t. the ones found by SVC-DD when using [47] and (4.7).

Table 4.5 shows the aggregated results of MXGA and SVC-DD. Each row (i.e., for each class of instances and due-date groups A, B, C), reports the gaps G_N and G_L of π_z and π_ℓ averaged on the class populations indicated in Table 4.3.

Numerical results show that SVC-DD largely improves the mean gaps in the large majority of instances. Accordingly, the gap percentage improvement $(G_\bullet^{\text{MXGA}} - G_\bullet^{\text{SVC-DD}})/G_\bullet^{\text{MXGA}}$ is across all the groups of instances on average 26.1% in G_N and 31.6% in G_L for π_n, 25.4% in G_N and 37.1% in G_L for π_ℓ. Moreover, SVC-DD finds solutions with a minimum number of bins in all the instances of classes II. When considering SVC-DD*, our solutions further improve on the mean G_N between 26.23% (π_ℓ - group C) and 36.59% (π_n - group A), G_L within 26.82% (π_n - group B) and 36.55% (π_ℓ - group C). Additionally, SVC-DD* finds the *ideal* point (a solution that achieves both absolute minima N and $L_{\max}$) in all the instances of class IX.

About CPU time, though requiring the repeated solution of 0-1 knapsack problems, SVC-DD has a mean running time of 1.46 sec. This value is two order of magnitude less than the CPU times reported in [22], a speedup that cannot be just ascribed to hardware configuration.

SVC-DD vs. CPMIP

SVC-DD can be just partially compared to CPMIP, since [110] focuses on $L_{\max}$ minimization and therefore gives gaps G_L but no detail on packing quality. Thus, we focus the comparison on G_L for solutions π_ℓ.

According to Table 4.6, primal solutions π_ℓ provided by SVC-DD are significantly better in all the instances, except class IX, with reduced G_L values w.r.t. CPMIP. On the mean, the gap percentage improvement is 34.2% on group A, 18.7% on group B and 16.4% on group C.

The Authors of [110] propose two different lower bounds to $L_{\max}$, the tightest of which computed via a Mixed Integer Linear Program (MILP). For 256

Table 4.5: MXGA and SVC-DD solutions of 2RBP-DD on I_B: G_N and G_L gaps (%)

	MXGA				SVC-DD				SVC-DD*			
	π_z		π_ℓ		π_z		π_ℓ		π_z		π_ℓ	
Class (group A)	G_N	G_L	G_N	G_L	G_N	G_L	G_N	G_L	G_N	G_L	G_N	G_L
I	3.5	13.9	4.1	12.4	3.8	13.2	4.5	11.8	0.3	5.9	1.0	3.6
II	2	11.2	2	11.1	0.0	0.4	0.0	0.4	0.0	0.4	0.0	0.4
III	6.8	23.6	7.7	22	5.8	19.8	6.4	18.0	2.3	9.8	2.7	8.8
IV	4.7	17.2	4.7	17.1	0.7	3.0	0.7	3.0	0.7	3.0	0.7	3.0
V	6.2	20.2	7	17.9	5.7	16.7	6.3	15.5	1.8	8.3	2.5	5.7
VI	9.3	16.5	9.3	16.6	2.0	2.7	2.0	2.7	2.0	2.7	2.0	2.7
VII	7.5	28.2	8.9	23.5	7.7	23.8	9.1	20.5	6.7	21.5	8.3	18.0
VIII	7.9	28.3	8.8	23.3	7.9	22.4	8.6	21.3	6.6	18.8	7.2	17.8
IX	0.7	1.7	0.7	1.7	0.7	1.7	0.7	1.7	0.0	0.0	0.0	0.0
X	7.1	26.4	7.9	23.8	6.4	18.1	6.8	16.4	5.4	15.4	5.8	13.8
Average	5.6	18.7	6.1	16.9	4.1	12.2	4.5	11.1	2.6	8.6	3.0	7.4
Class (group B)												
I	3.4	28.7	4.6	24.2	4.0	28.2	5.8	21.4	0.4	16.9	2.1	7.5
II	2	34.1	2.7	34	0.0	1.9	0.0	1.9	0.0	1.9	0.0	1.9
III	7.1	55	8.7	46.2	6.4	38.9	8.4	34.5	2.8	24.7	4.8	18.4
IV	4.7	35.9	4.7	36	0.7	6.2	0.7	6.2	0.7	6.2	0.7	6.2
V	6	40.7	8.1	35.5	5.8	41.1	7.6	29.1	1.9	20.7	3.8	11.5
VI	10.3	37.7	11	37.7	2.0	6.0	2.0	6.0	2.0	6.0	2.0	6.0
VII	7.8	66.7	10.2	52.2	8.1	47.8	10.0	37.3	7.1	45.8	9.1	32.6
VIII	7.8	59.4	9	49.4	8.3	46.8	10.0	37.2	7.2	39.9	8.9	31.6
IX	0.7	2.4	0.7	2.4	0.7	3.8	0.7	3.8	0.0	0.0	0.0	0.0
X	7.1	60	8.6	53.5	6.8	39.4	7.7	31.9	5.8	30.3	6.8	26.0
Average	5.7	42.1	6.8	37.1	4.3	26.0	5.3	20.9	2.8	19.2	3.8	14.2
Class (group C)												
I	3.4	124.4	5.4	93	4.1	117.8	6.5	64.6	0.7	53.1	3.1	19.9
II	2	152.7	4	149.5	0.0	3.2	0.0	3.2	0.0	3.2	0.0	3.2
III	7	147.6	9.1	124.9	6.6	125.8	10.8	84.2	3.1	71.6	6.7	41.1
IV	4.7	154.2	5.3	153.2	0.7	15.2	0.7	15.2	0.7	15.2	0.7	15.2
V	6	145.1	8.8	105	6.6	96.4	9.1	71.2	2.4	57.0	5.3	28.9
VI	11	241.1	11	241.2	2.0	58.4	2.0	58.4	2.0	58.4	2.0	58.4
VII	7.8	263	10.5	209.6	8.6	167.7	12.1	132.6	7.9	116.1	11.4	89.4
VIII	8.1	328.6	10.1	273.3	8.7	226.3	10.2	169.7	7.3	202.6	8.8	135.1
IX	0.7	9.9	0.7	9.9	0.6	26.0	0.7	16.2	0.0	0.0	0.0	0.0
X	7	418.8	9	318.5	6.8	253.3	8.4	188.5	5.9	173.2	7.5	126.7
Average	5.8	198.5	7.4	167.8	4.5	109.0	6.1	80.4	3.0	75.0	4.5	51.8

instances out of 1500 in their experiment, the MILP was not able to return a valid bound within one hour of CPU time. A valid bound for the remaining 1244 instances is computed within a mean running time of 74.91 seconds, and in 361 cases it strictly improves (by 31.30% on average) that reported in [22]. Nonetheless, the bounds proposed in [110] do not dominate L^n_{LB} as computed in [22] and with formula (4.7).

The improved version of L^n_{LB} achieved by Proposition 7 was computed in 0.63 seconds on average and resulted tighter than L^n_{LB} of (4.7) in 39 instances (2.60% of the cases), with an improvement on this subset that ranges from 0.36% to 223.53% and an average of 22.25%. When computing L^n_{LB} by dual feasible functions, the bound L^n_{LB} of [22] strictly improves in 505 cases by 9.45% on average. Using Proposition 7, this bound is further improved from 0.27% to 35.42% (6.97% on average) in 1.93% of the cases.

Although being valid for any dimension of rBP, Proposition 7 naturally loses effectiveness as the dimension grows due to the correlation to $|\mathcal{I}_{\mathcal{L}}|$. A

Table 4.6: MXGA, CPMIP and SVC-DD π_ℓ solutions of 2RBP-DD on I_B: G_L gaps (%)

	MXGA	CPMIP	SVC-DD	SVC-DD*
Class (group A)	G_L	G_L	G_L	G_L
I	12.4	13.9	11.8	3.6
II	11.1	2.3	0.4	0.4
III	22	22.1	18	8.8
IV	17.1	10.5	3	3
V	17.9	19.1	15.5	5.7
VI	16.6	9.3	2.7	2.7
VII	23.5	27.7	20.5	18
VIII	23.3	26.5	21.3	17.8
IX	1.7	1.7	1.7	0.0
X	23.8	20.2	16.4	13.8
Average	16.9	15.3	11.1	7.4
Class (group B)				
I	24.2	23.1	21.4	7.5
II	34	2.9	1.9	1.9
III	46.2	41.7	34.5	18.4
IV	36	14.3	6.2	6.2
V	35.5	33.3	29.1	11.5
VI	37.7	12.5	6	6
VII	52.2	48.7	37.3	32.6
VIII	49.4	46.5	37.2	31.6
IX	2.4	2.4	3.8	0.0
X	53.5	40.9	31.9	26
Average	37.1	26.6	20.9	14.2
Class (group C)				
I	93	69.6	64.6	19.9
II	149.5	7.2	3.2	3.2
III	124.9	91.2	84.2	41.1
IV	153.2	35.6	15.2	15.2
V	105	81.1	71.2	28.9
VI	241.2	76.9	58.4	58.4
VII	209.6	156.7	132.6	89.4
VIII	273.3	232.2	169.7	135.1
IX	9.9	10.5	16.2	0.0
X	318.5	214.5	188.5	126.7
Average	167.8	97.6	80.4	51.8

wider impact is expected to emerge in the one-dimensional case.

The best gaps G_L obtained in [110] by means of the best known lower bounds, averaged on due-date types, are 11.9% for due-date group A, 20.5% for group B and 68.5% for group C. Our best gaps G_L reached with SVC-DD* are 7.5% in group A, 14.6% in group B and 54.8% in group C. Summarizing, primal solutions π_ℓ found by SVC-DD are generally better than the ones computed by CPMIP; when embedding different lower bounds, although a straightforward comparison has a limited meaning, the gaps observed with SVC-DD* are significantly tighter than CPMIP's. Moreover, SVC-DD* finds an optimal $L_{\max}$ in 620 cases out of 1500 vs. 586 cases in which CPMIP does the same | but with a mean running time of 30.2 seconds, roughly fifteen times that of SVC-DD*.

4.6.2 Pareto-analysis

Measuring solution quality by gaps G_N and G_L has inherent limits, as 2BP-DD is a multi-objective problem. Moreover, for relatively large due-dates and τ, G_L results very data-sensitive: for example, with $\tau = 100, L_{LB} = 5$ and $d_i = 95$, gap G_L jumps from 0% (when item i is assigned to the first bin) to 2000% (when i is allocated one bin later). In order to assess the performance of SVC-DD we then preferred two measures, that we derived from papers on multi-objective evaluation: $\bar{R}_1$ and G_A. The former uses the ratio $R_1(x_{\mathcal{H}}, x^*)$ proposed by [54], see §4.3.1; the latter is a slight modification of the *Space Covered Measure* (SCM) presented by [141]. In both cases we resort to the notion of *ideal* solution value, intended as the point $x_{id} = (z_{id}, \ell_{id}) \in \mathbb{N}_2$ with $z_{id} = N_{LB}$ and $\ell_{id} = L_{LB}$.

Let $\bar{X} = \{\bar{x}_i = (\bar{z}_i, \bar{\ell}_i), i = 1, \ldots, p\}$ be the set of non-dominated solutions computed through a heuristic for 2BP-DD. Indicator $\bar{R}_1$ is the minimum of $R_1(\bar{x}_i, x_{id})$ computed with norm $\|.\|_2$ among all solution values in $\bar{X}$:

$$\bar{R}_1 = \min_{i \in \bar{X}} R_1(\bar{x}_i, x_{id}) = \frac{|\sqrt{\bar{z}_i^2 + \bar{\ell}_i^2} - \sqrt{N_{LB}^2 + L_{LB}^2}|}{\sqrt{N_{LB}^2 + L_{LB}^2}}$$

While $\bar{R}_1$ is constructed after the differences of solution value norms from the norm of the ideal point, indicator G_A attempts at evaluating the area that underlies the Pareto-frontier (as approximated by solutions in $\bar{X}$). Let $x_{nad} = (N_{UB}, L_{UB})$ be the *nadir* point, where N_{UB} and L_{UB} are valid upper bounds to N and $L_{\max}$ respectively. A frontier is evaluated by two areas computed as sums of rectangles, see Fig. 4.5: the first area underlies x_{id} and is given by $A_{id} = N_{LB}(L_{UB} - L_{LB}) + L_{LB}N_{UB}$; the second is associated with $\bar{X}$ and amounts to $A_{\bar{X}} = \sum_{i=0}^{p} \bar{\ell}_i(\bar{z}_{i+1} - \bar{z}_i)$, where solution values are sorted by increasing N and $\bar{z}_0 = \bar{\ell}_{p+1} = 0$, $\bar{\ell}_0 = L_{UB}$, $\bar{z}_{p+1} = N_{UB}$. The quality of a heuristic frontier $\bar{X}$ is then measured through the percentage gap:

$$G_A = 100\frac{A_{\bar{X}} - A_{id}}{A_{id}}$$

G_A increases as the heuristic frontier steps further from x_{id}, and takes into account both the quality and cardinality of $\bar{X}$. Note that G_A mainly differs from the SCM of [141] in how areas are computed, but can be employed to evaluate any bi-objective optimization algorithm. Clearly, both $\bar{R}_1$ and G_A equal zero only when x_{id} is the value of a feasible packing, which implies that the optimal frontier consists of just one solution of value x_{id} (proving the existence of such packing is NP-complete).

Since $\bar{R}_1$ is more meaningful when solution values are comparable to each

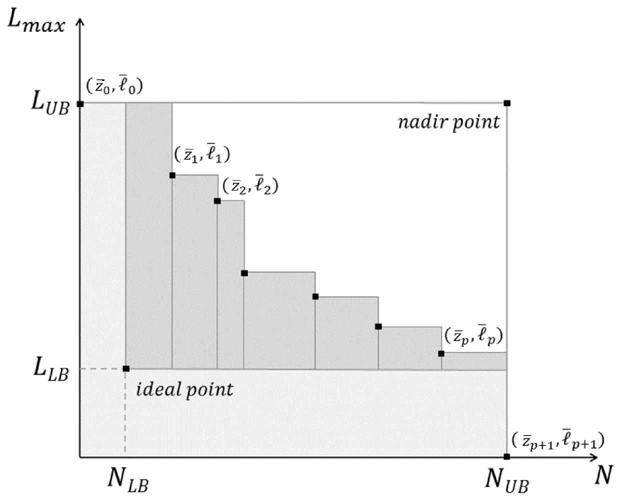
L_{max}
$(\bar{z}_0, \bar{\ell}_0)$
L_{UB}
nadir point
$(\bar{z}_1, \bar{\ell}_1)$
$(\bar{z}_2, \bar{\ell}_2)$
$(\bar{z}_p, \bar{\ell}_p)$
L_{LB}
ideal point
$(\bar{z}_{p+1}, \bar{\ell}_{p+1})$
N_{LB}
N_{UB}
N

Figure 4.5: A_{id} = light grey area; $A_{\bar{X}} - A_{id}$ = dark grey area; $\bar{x} = (\bar{z}_i, \bar{\ell}_i), i = 1, \ldots, p$, are the solution values in the heuristic frontier $\bar{X}$.

Table 4.7: SVC-DD* solutions of 2RBP-DD on I_B: $\bar{R}_1$, $G_A(\%)$ and structure of Pareto-frontiers.

	Group A					Group B					Group C				
Class	$\bar{R}_1$	G_A	$\#m$	$\#s$	$\#opt$	$\bar{R}_1$	G_A	$\#m$	$\#s$	$\#opt$	$\bar{R}_1$	G_A	$\#m$	$\#s$	$\#opt$
I	0.01	0.66	8	40	22	0.02	0.96	16	29	20	0.02	1.33	20	22	19
II	0.00	0.07	0	50	46	0.00	0.20	0	50	42	0.00	0.16	0	50	39
III	0.04	2.84	3	30	13	0.04	3.80	13	22	14	0.05	4.60	20	12	11
IV	0.01	0.90	0	49	28	0.01	1.19	0	49	26	0.01	1.50	0	49	29
V	0.03	1.97	6	29	18	0.03	2.57	15	22	16	0.04	3.28	25	15	11
VI	0.02	1.00	0	49	25	0.02	1.54	0	49	22	0.02	1.59	0	47	18
VII	0.09	6.76	12	8	2	0.10	8.20	14	6	1	0.10	9.66	21	3	0
VIII	0.09	6.61	5	11	0	0.09	8.13	12	7	1	0.09	9.36	13	5	0
IX	0.00	0.00	0	50	50	0.00	0.00	0	50	50	0.00	0.00	0	50	50
X	0.07	5.19	3	25	7	0.08	6.65	6	15	5	0.07	7.29	10	12	5
Average	0.04	2.60	-	-	-	0.04	3.32	-	-	-	0.04	3.88	-	-	-
Total	-	-	37	341	211	-	-	76	299	197	-	-	109	265	182

other, the computational results reported next are obtained by setting $\tau = 1$ and normalizing due-dates accordingly.

We performed the above described Pareto-analysis on both I_B and I_R instances. Table 4.7 shows the results achieved by SVC-DD* on I_B instances. The table reports $\bar{R}_1$ and G_A for each class and due-date group, with G_A referred to the nadir point $x_{nad} = (z_{EDD}, \tau \cdot z_{EDD})$. The table also gives

- how many times ($\#m$) the set $\bar{X}$ of reciprocally non-dominating solutions found by SVC-DD* has more than one point,
- how many times ($\#s$) Proposition 8 certifies that the optimal frontier consists of a single solution,

- how many instances (*#opt*) out of *#s* are solved to optimality by SVC-DD* (i.e., the cases in which there is a feasible ideal point).

By Proposition 8, the Pareto-optimal set consists of very few solutions: in our tests we found a single non-dominated solution in 1278 out of 1500 cases, two in 210 cases and three in the remaining 6 cases. In 905 out of 1278 cases we proved that the Pareto-optimal set consists of a single solution. SVC-DD* found the ideal point, and therefore certified optimality, in 590 cases, whereas a solution with $\bar{z} = N_{LB}$ bins was found in the remaining 315 cases.

Table 4.8: SVC-DD* solutions of 2OBP-DD on I_B: $\bar{R}_1$, $G_A(\%)$ and structure of Pareto-frontiers.

	Group A					Group B					Group C				
Class	$\bar{R}_1$	G_A	*#m*	*#s*	*#opt*	$\bar{R}_1$	G_A	*#m*	*#s*	*#opt*	$\bar{R}_1$	G_A	*#m*	*#s*	*#opt*
I	0.06	4.01	8	32	16	0.06	5.05	17	22	17	0.06	5.39	21	10	5
II	0.00	0.22	0	50	45	0.00	0.42	0	50	33	0.00	0.42	0	50	33
III	0.09	7.38	6	21	10	0.10	8.99	14	14	8	0.10	10.09	27	5	4
IV	0.03	1.95	0	48	24	0.04	3.70	0	46	17	0.03	3.10	0	47	21
V	0.09	6.73	10	19	14	0.09	8.01	19	15	9	0.09	8.91	27	15	9
VI	0.04	3.03	0	48	22	0.04	2.55	2	46	19	0.04	4.12	0	46	14
VII	0.19	13.74	8	32	14	0.19	16.55	14	26	13	0.18	17.59	22	21	10
VIII	0.20	14.51	5	32	9	0.19	16.60	9	25	8	0.18	18.37	15	20	7
IX	0.01	0.36	0	49	48	0.01	0.44	0	49	49	0.01	0.51	3	46	45
X	0.10	7.92	3	21	10	0.11	9.93	11	16	8	0.10	11.11	18	13	7
Average	0.08	5.98	-	-	-	0.08	7.22	-	-	-	0.08	7.96	-	-	-
Total	-	-	40	352	212	-	-	86	309	181	-	-	133	273	155

The features of $\bar{X}$ do not change substantially for the oriented problem 2OBP-DD (see Table 4.8): SVC-DD* found a single non-dominated solution in 1241 cases, two in 250 cases, three in 8 cases and four in a single case. In 934 cases of 1241 only a single non-dominated solution exists in the Pareto-frontier: for 548 instances our algorithm returned the ideal point, whereas in the other 386 cases the lower bound on the number of employed bins was reached.

On the other hand, the mean values of $\bar{R}_1$ and G_A (referred to the same ideal point as 2RBP-DD) more than double in all the due-date groups A,B and C, while global mean CPU time decreases by almost 36% (from 1.46 to 0.95 sec).

Let us now describe the algorithm performance on the set I_R of real industrial instances. Table 4.9 shows the results for due-date group C (similar results are obtained on groups A and B). Columns *#items*, *#sol* and *CPU* report the number of items per instance, the number of non-dominated solutions in $\bar{X}$ and the CPU time spent, respectively.

Real instances appear a bit more challenging than artificial ones: the mean values of $\bar{R}_1$ and G_A are 0.09 and 8.86, respectively: more than two times the value observed for I_B. On the other hand, the mean CPU time is 8.31 seconds: more than five times that required for I_B.

As in I_B, the vast majority of the frontiers found by SVC-DD* consists of a single point, but rather than a characteristic of optimal frontiers, in I_R this feature seems to be related to some inability of the algorithm in diversification. Indeed, we were able to certify a single-point optimal frontier in just 7 out of 28 cases, and could certify optimality in one case only. This observation suggests, as future work, to try improving SVC-DD* by local search. An intuitive way to populate $\bar{X}$ can rely on exploring solutions with increasing N and decreasing $L_{\max}$: starting from a non-dominated packing, the search could anticipate the critical item for which $L_{\max} = \ell$, while enforcing a controlled delay on items with strictly positive $\tau\Delta_i(\ell)$, see §4.2.2.

Table 4.9: SVC-DD* solutions of 2RBP-DD on I_R and due-date type C.

Name	n	$\bar{R}_1$	G_A	#sol	CPU	Name	n	$\bar{R}_1$	G_A	#sol	CPU
r_1	11	0.00	0.00	1	< 0.005	r_{15}	84	0.00	1.49	1	2.83
r_2	14	0.02	2.13	1	0.22	r_{16}	86	0.08	4.74	1	4.03
r_3	24	0.56	39.10	1	0.38	r_{17}	87	0.08	8.99	1	3.64
r_4	27	0.14	14.01	1	0.53	r_{18}	98	0.00	1.49	1	3.72
r_5	28	0.01	0.00	1	0.86	r_{19}	119	0.01	3.49	2	4.70
r_6	30	0.00	0.16	1	0.59	r_{20}	141	0.10	11.75	1	6.38
r_7	33	0.13	12.65	1	0.78	r_{21}	164	0.08	8.91	1	8.17
r_8	42	0.02	2.97	1	0.72	r_{22}	186	0.01	2.66	1	10.84
r_9	49	0.08	8.52	1	1.39	r_{23}	215	0.01	4.32	1	12.98
r_{10}	61	0.21	19.22	1	1.53	r_{24}	218	0.07	7.85	3	23.00
r_{11}	63	0.05	4.90	2	2.31	r_{25}	238	0.09	8.39	3	32.84
r_{12}	67	0.13	14.14	1	1.97	r_{26}	257	0.01	2.72	1	18.66
r_{13}	69	0.21	23.45	1	3.53	r_{27}	314	0.05	7.58	1	27.52
r_{14}	83	0.09	13.47	1	3.06	r_{28}	352	0.21	19.04	1	55.58

Finally, the differences between oriented and non-oriented results are less evident than in I_B: for 2OBP-DD (see Table 4.10), $\bar{R}$ and G_A (referred to the same ideal point as 2RBP-DD) respectively increase up to 0.12 and 11.76 on the mean, whereas the mean CPU time decreases by only 13.96% (from 8.31 to 7.15 seconds).

4.7 Conclusions

In this chapter we considered a bi-objective extension of an orthogonal two-dimensional bin packing problem, where items are equipped with due-dates, and we wish to minimize both the maximum lateness of the items and the number of bins required to pack them all. We discussed some basic properties of non-dominated solutions and their dependence by the packing time τ. Moreover, following the definition of approximation ratios defined for multi-criteria problem in [54], we showed how approximation algorithms for 2BP provide approximation results also for 2BP-DD.

To solve the problem in practice, we proposed a sequential value correction

Table 4.10: SVC-DD* solutions of 2OBP-DD on I_R and due-date type C.

Name	# items	$\bar{R}_1$	G_A	*#sol*	CPU	Name	# items	$\bar{R}_1$	G_A	*#sol*	CPU
r_1	11	0.00	0.00	1	< 0.005	r_{15}	84	0.00	1.40	1	1.84
r_2	14	0.21	16.71	1	0.27	r_{16}	86	0.08	4.74	1	3.78
r_3	24	0.56	39.10	1	0.30	r_{17}	87	0.08	9.42	1	3.19
r_4	27	0.28	25.28	1	0.42	r_{18}	98	0.00	1.40	1	2.36
r_5	28	0.00	0.00	1	0.38	r_{19}	119	0.28	25.46	2	3.92
r_6	30	0.02	3.86	1	0.30	r_{20}	141	0.10	12.58	1	4.89
r_7	33	0.13	13.15	1	0.49	r_{21}	164	0.09	10.94	1	6.03
r_8	42	0.02	2.97	1	0.58	r_{22}	186	0.01	3.28	1	7.78
r_9	49	0.08	8.84	1	1.08	r_{23}	215	0.01	7.44	1	8.19
r_{10}	61	0.21	20.33	1	1.02	r_{24}	218	0.08	8.10	3	22.36
r_{11}	63	0.05	4.90	2	2.09	r_{25}	238	0.08	10.62	2	38.52
r_{12}	67	0.26	26.49	1	1.58	r_{26}	257	0.02	5.62	2	13.73
r_{13}	69	0.22	24.91	1	0.44	r_{27}	314	0.06	9.09	1	20.45
r_{14}	83	0.09	13.70	1	2.25	r_{28}	352	0.21	19.05	1	52.06

heuristic (SVC-DD) and used a vast set of benchmark instances to compare its performance with MXGA of [22] and CPMIP of [110]. Results show that SVC-DD largely outperforms both algorithms, achieving in general better primal-dual gaps in a much smaller CPU time. We further tested SVC-DD on a set of new and more challenging instances derived from real-world orders: also in this case our heuristic proved to achieve good results in very reasonable computational time. Finally, we analyzed our results under a multi-objective perspective and gave details about the structure of the heuristic frontier built by SVC-DD, inferring also some features of the Pareto-optimal sets.

Acknowledgements

We are grateful to Julia A. Bennell that kindly provided us the instances used in [22].

Chapter 5

The one-dimensional bin packing with variable pattern processing time

[1]

Integrated problems between packing and scheduling want to capture the features of operations within real manufacturing and logistic systems, translating practical aspects into objectives and constraints that are added to the mathematical core of the problems. The choice of the level of compliance with real systems is generally made by evaluating a trade-off between several elements, among which the impact of practical aspects on the decision process. Neglecting key features with high impact could potentially affect the implementability of solutions, so that the entire decision process could be compromised and nothing came of it.

A crucial aspect that is often disregarded by academics in the context of C&P is the effect of processing time of packing (cutting) operations performed in the implementation of patterns. Indeed, while is common in literature to assume the packing (cutting) time of a pattern as a constant, in real systems it could count a number of different variables that make even finding a suitable definition an issue.

This chapter want to be one of the first attempts to study the impact of patterns with variable processing time in the context of a one-dimensional BP, focusing on mathematical formulations that extends the typical models for BP with scheduling features.

[1]A preliminary version of this chapter appears in: A. Pizzuti, W. Wu, F. Marinelli, M. Yagiura and Y. Hu: A pattern-based reformulation for the one-dimensional bin packing with variable pattern processing time. In Proceedings of the Scheduling Symposium 2018 - SS2018, pages 89-94., Otaru (JPN)

5.1 Introduction

In the one-dimensional bin packing problem (1BP) we want to find a packing of a given set $I = \{1, \dots, n\}$ of items with positive integer size $\{l_1, \dots, l_n\}$ into bins of identical integer size l, such that all items are completely contained in the bins without overlapping and the number N of used bins is minimized. Due to its relevance both in theory and practice, 1BP is considered one of the basic and main C&P problem and a rich literature was dedicated to its study [135].

Moreover, bin packing problems are connected to scheduling problems in which limitations on resources are taken into account, such as the resource constrained scheduling problem [60]: indeed, the bin size represents the availability of resource, whereas each item is intended as a job with a specific resource consumption.

Motivated by real applications in which the cost of time required by packing operations are more relevant than the cost of wasted space, an increasing number of papers are recently investigating integrated problems between bin packing and scheduling, see [12, 22, 113]. However, it is commonly assumed that the time spent for processing each bin is constant, and to the best of our knowledge only few papers explicitly deal with a processing time dependent by the selected patterns [88, 39, 44]. Nevertheless, such dependency cannot be neglected in many real applications, e.g., cutting and loading operations, without losing an important aspect of the solution costs.

The 1BP with VARIABLE PATTERN PROCESSING TIME (1BP-VPT) was firstly formalized in [88] to study the effects of pattern-dependent completion times: constant *set-up* and *packing* times s and t are required to initialize the packing of each bin and to process each item, respectively. Moreover, each item is provided with a due date d_i that expresses the time by which it should be processed. It can be useful to think a 1BP-VPT solution in terms of both a sequence of patterns and a sequence of items: the former gives the order in which the bins are processed, whereas the latter represents the absolute position of each item in the whole packing solution. The completion time C_i of the item i is then defined by introducing the following notation: let b_i be the position of the bin that contains the item i in the schedule of patterns, and q_i be the absolute position of the item i in the whole schedule of the packed items. Then $C_i = sb_i + tq_i$ and the lateness of the i-th item is given by $L_i = C_i - d_i$. The 1BP-VPT objective function requires the minimization of a convex combination of the packing and scheduling terms, formally defined as $\Omega = \alpha_1 N + \alpha_2 L_{\max}$, where $L_{\max} = \max_{i \in I}\{0, L_i\}$ and $\alpha_1, \alpha_2 \in \mathbb{R}^+$ such that $\alpha_1 + \alpha_2 = 1$. Actually α_1 and α_2 express the relative importance of material usage and delay penalty, respectively. Thus, their ratio should be selected accordingly to the specific

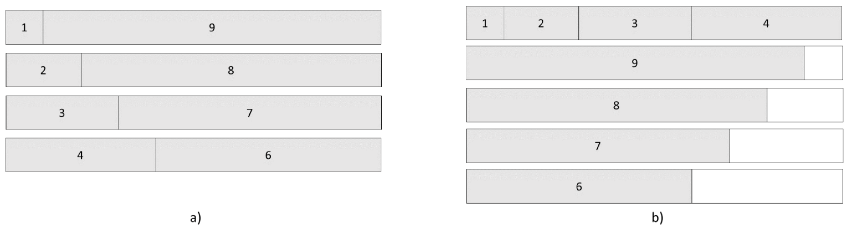

Figure 5.1: a) optimal for $t = 0$ and sub-optimal for $t = 1$; b) optimal for $t = 1$ and sub-optimal for $t = 0$

application in which the 1BP-VPT arises.

Note that optimal solutions of 1BP-VPT are inherently different from those obtained by considering constant processing times of bins. Indeed, the following example highlights that a solution which is optimal for $t = 0$, can be non-optimal for $t > 0$, i.e., when bin processing time is not constant, and vice-versa. Let $I = \{1, 2 \ldots, 8\}$ be the items set with lengths $\{l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8\} = \{1, 2, 3, 4, 6, 7, 8, 9\}$ and due-dates $\{d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8\} = \{1, 2, 3, 4, 8, 7, 6, 5\}$. Let moreover assume $l = 10$, $s = 1$ and $\alpha_1 = \alpha_2 = 0.5$, with $t = 0$ in first place. Fig. 5.1 shows two solutions in which the sequence of bins (items) proceeds from top (left) and labels refer to the item lengths. It is easy to see that solution a) with $N = 4$, $L_{\max} = 0$ and $\Omega = 2$ is optimal, whereas solution b) with $N = 5$, $L_{\max} = 0$ and $\Omega = 2.5$ is not. On the contrary, for $t = 1$, solution a) is no longer optimal since it has $N = 4$, $L_{\max} = 7$ defined by item 4 and $\Omega = 5.5$, whereas solution b) with $N = 5$, $L_{\max} = 5$ defined by item 5 and $\Omega = 5$ becomes optimal.

In this case the optimality changes to the looseness of the due-dates: for $t = 0$, the number of bins is critical, while the lateness are not binding; for $t = 1$, the relevance of the lateness increases over the number of bins weight and the use of an additional bin is required to redefine the items ordering and reduce the lateness value. Generally speaking, the introduction of item processing times modifies the balance among the objective function terms in such a way that optimal solutions for $t = 0$ significantly differ from the truly optimal value of Ω. Therefore, formulations that neglect the item processing times may lead to sub-optimal solutions with meaningful optimality gaps.

The 1BP-VPT is related to the well-known Single Batch Processing Machine Scheduling Problem (SBM). In SBM jobs must be processed on a batch processing machine, which is able to handle a number of simultaneous jobs up to a fixed capacity; the minimization of a function of the completion times is commonly considered. A typical SBM application can be found in the metalworking industries, where burn-in operations in semiconductor manufacturing must be scheduled, see [78] and [125]. The relationship with 1BP-VPT is indeed straightforward when job sizes are non-identical and by assuming batches (jobs) as patterns (items). Nevertheless, SBM differs by 1BP-VPT since in the former all the jobs in the same batch are generally released at the same time, whereas in the latter the definition of C_i implies that the item i becomes available as soon as it is processed. Moreover, in SBM the objective function usually includes only scheduling terms, without the explicit minimization of the number of processed batches. This term is directly minimized when the minimization of the makespan is required, case that corresponds to the minimization of the number of setups for 1BP-VPT.

In this chapter we introduce a novel pattern-based mixed integer linear programming formulations (R_{VPT}) for the 1BP-VPT and present a dynamic programming algorithm to exactly solve the corresponding quadratic pricing problem. Computational results show the improvement of R_{VPT} in terms of continuous bound and CPU time with respect to the continuous relaxation of the formulation by [88].

5.2 Problem reformulation

The 1BP-VPT has been formulated in [88] by extending the well-known *assignment formulation* [74] for 1BP. Let J be the set of available bins, with $|J| = m$. For each $i \in I$ and $j \in J$, the assignment of items to bins is expressed by $u_{ij} \in \{0,1\}$, i.e., $u_{ij} = 1$ if and only if the item i is packed within the j-th bin; $w_j \in \{0,1\}$ indicates the application of the j-th bin, i.e., if the j-th bin is used then $w_j = 1$. Variables $r_{ih} \in \{0,1\}$ are used to define the relative position between the items: $r_{ih} = 1$ if and only if item h is packed before item i. Finally, variables $b_i \in \mathbb{N} \setminus \{0\}$ and $q_i \in \mathbb{N} \setminus \{0\}$ indicate the position of the bin containing the item i in the sequence of used bins, and the position of the item i in the sequence of packed items, respectively.
The integer program (M_{VPT}) reads as:

$$\min \Omega = \alpha_1 \sum_{j \in J} w_j + \alpha_2 L_{\max} \tag{5.1}$$

$$\sum_{j \in J} u_{ij} = 1 \qquad \forall i \in I \tag{5.2}$$

$$\sum_{i \in I} l_i u_{ij} \leq W w_j \qquad \forall j \in J \tag{5.3}$$

$$q_i = \sum_{j \in J} j u_{ij} \qquad \forall i \in I \tag{5.4}$$

$$sb_i + tq_i - d_i \leq L_{\max} \qquad \forall i \in I \tag{5.5}$$

$\forall i, h \in I, i \neq h$

$$q_i - q_h + 1 \leq nr_{ih} \tag{5.6}$$

$$q_h - q_i + 1 \leq n(1 - r_{ih}) \tag{5.7}$$

$$b_i - b_h \leq (m-1)r_{ih} \tag{5.8}$$

$$b_h - b_i \leq (m-1)(1 - r_{ih}) \tag{5.9}$$

$$u_{ij}, w_j \in \{0,1\} \qquad \forall i \in I, j \in J \tag{5.10}$$

$$b_i, q_i \in \mathbb{N} \setminus \{0\} \qquad \forall i \in I \tag{5.11}$$

$$r_{ih} \in \{0,1\} \qquad \forall i, h \in I, i \neq h \tag{5.12}$$

$$L_{\max} \geq 0. \tag{5.13}$$

Constraints (5.2) and (5.3) are those of the 1BP assignment formulation: (5.2) ensure the demand fulfillment, and (5.3) set the bin variables w_j and model the packing constraints. Due to (5.2), the i-th equation of (5.4) expresses the position of the bin that contains item i, whereas constraints (5.5) bound from below the maximum lateness. Finally, the sequences of patterns and items are modelled by disjunctive constraints (5.6)–(5.9): when $r_{ih} = 1$ (5.6) and (5.8) become redundant while (5.7) and (5.9) force item h to be processed before i and to be packed in the same bin containing item i or in one of the previous. Analogously, the reverse relative position between i and h is ensured by (5.6) and (5.8) when $r_{ih} = 0$.

M_{VPT} can be reformulated by integer decomposition, thus giving rise to an alternative pattern-based integer program for 1BP-VPT that generalizes the well-known Gilmore-Gomory formulation for 1BP [62]. A packing *pattern* $p \in P$ is the incidence vector $\mathbf{a}^p \in \{0,1\}^n$ of a subset of items that satisfy the knapsack constraint, i.e., such that $\sum_{i \in I} l_i a_i^p \leq l$. For convenience, we indicate with $i \in p$ the case in which $a_i^p = 1$. Although the sequence of items within a single pattern affects the completion times and possibly the scheduling term of 1BP-VPT, patterns are defined regardless the item permutations. Indeed, due to the optimality of the Jackson's rule for the $L_{\max}$ minimization problem [25], we can assume that the items within a single pattern are sequenced according to the EDD (*Early Due Dates*) rule. For each $j \in J$, binary vectors $\mathbf{u}_j$ can be expressed as an integer convex combination of patterns:

$$\mathbf{u}_j = \sum_{p \in P} x_j^p \mathbf{a}^p \qquad \sum_{p \in P} x_j^p = 1 \qquad x_j^p \in \mathbb{Z}^+.$$

The relevant i-th component of $\mathbf{u}_j$ is then given by $u_{ij} = \sum_{p\in P} x_j^p a_i^p = \sum_{p\in P_i} x_j^p$, where $P_i \subseteq P$ consists in the set of patterns containing item i. Thus, the reformulation is defined on decision variables $\boldsymbol{x}$, where $x_j^p = 1$ if the j-th bin in the sequence is packed according to pattern p, and 0 otherwise.

Given any pattern p, let $\tau^p = t\sum_{i\in I} a_i^p$ be the total packing time of a bin, and $\tau_i^p = t\sum_{k\le i} a_k^p$ be the packing time of a bin up to the processing of the item i. The extended pattern-based reformulation for 1BP-VPT follows (R_{VPT}):

$$\min \Omega = \alpha_1 \sum_{j\in J}\sum_{p\in P} x_j^p + \alpha_2 L_{\max} \tag{5.14}$$

$$\sum_{j\in J}\sum_{p\in P_i} x_j^p \ge 1 \qquad \forall i \in I \tag{5.15}$$

$$\sum_{p\in P} x_j^p \le 1 \qquad \forall j \in J \tag{5.16}$$

$$\Psi_{j-1} + \sum_{p\in P} \tau^p x_j^p = \Psi_j \qquad \forall j \in J \tag{5.17}$$

$$\Psi_{j-1} + \sum_{p\in P_i} (sj + \tau_i^p + n_j t) x_j^p \le L_{\max} + d_i + n_j t \qquad \forall i \in I, \forall j \in J \tag{5.18}$$

$$x_j^p \in \{0,1\} \qquad \forall p \in P, \forall j \in J \tag{5.19}$$

$$\Psi_0 = 0, L_{\max} \ge 0. \tag{5.20}$$

Constraints (5.15) require, as in the pattern-based reformulation for 1BP, that all items must be packed. Constraints (5.16) impose that at most one pattern p is assigned to each available bin j. Equation (5.17) defines the completion time Ψ_j of the bin j in terms of its processing time τ^p and the completion time of the previous bin. Constraint (5.18) bounds $L_{\max}$ from below: let n_j be an upper bound to the number of items that fit within the first j bins; for each pattern p containing a given item i, either $x_j^p = 0$ or $x_j^p = 1$. In the former case, the constraint is always satisfied and hence redundant. In the latter case, i.e., when the item i is packed in the j-th bin according to the pattern p, the constraint becomes $\Psi_{j-1} + sj + \tau_i^p \le L_{\max} + d_i$, where the left term correctly defines the completion time of the item i (sj is the total set-up time up to the j-th bin).

We remark that any optimal solution exactly satisfies the requirements though constraint (5.15) admits overproduction. We also remark that R_{VPT} correctly models the problem for any dimension, provided that the definition of patterns is accordingly adapted, and that the EDD rule adopted to define the sequence of items within patters remains optimal for arbitrary processing times. Moreover, R_{VPT} can be modified to handle some alternative completion time functions, e.g. total sum of tardiness or total number of tardy jobs. This can be easily done by updating (5.14) and properly adapting constraints

(5.18).

As pointed out in [12], R_{VPT} can be tightened by replacing the variable $L_{\max}$ with $L_{LB}^n + \tilde{L}_{\max}$, where $\tilde{L}_{\max} \geq 0$ and L_{LB}^n is any valid lower bound for the maximum lateness. A lower bound L_{LB}^n can be obtained by generalizing the bound proposed by [22]: assuming the items ordered by EDD, let N_{LB}^i be any valid lower bound on the number of bins required for the first i items in I. Then, a valid lower bound for $L_{\max}$ is given by:

$$L_{LB}^n = \max_{1 \leq i \leq n} \{0, sz_i^{LB} + it - d_i\}.$$

The result can be further improved by adapting the framework discussed in §4.5.
We address the constraints resulting by exploiting L_{LB}^n as ($\tilde{18}$) and assume in the following that (5.18) is replaced by ($\tilde{18}$) in R_{VPT}.

Due to the exponential number of variables x_j^p, a column generation (CG) procedure is required to solve R_{VPT} [83].

Let us call R_{VPT}^{res} the continuous relaxation of R_{VPT} restricted to patterns $p \in P_{res} \subseteq P$, namely *restricted master problem*. Let moreover $(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\delta}}, \bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\gamma}})$ be an optimal solution to the restricted dual problem, where $\boldsymbol{\mu} \in \mathbb{R}_n^-, \boldsymbol{\delta} \in \mathbb{R}_m^-, \boldsymbol{\pi} \in \mathbb{R}_m$ and $\boldsymbol{\gamma} \in \mathbb{R}_{n \times m}^-$ are the dual variables corresponding to the constraints (5.15)–(5.18) of R_{VPT}^{res}.

The restricted dual problem results as follows:

$$\begin{aligned}
\max \quad & \sum_{i \in I} \mu_i + \sum_{j \in J} \delta_j + \sum_{j \in J} \sum_{i \in I} (L_{LB}^n + n_j t + d_i) \gamma_i^j \\
\text{s.t.} \quad & \pi_{j+1} - \pi_j + \sum_{i \in I} \gamma_i^{j+1} = 0 && \forall j \in J \\
& -\sum_{j \in J} \sum_{i \in I} \gamma_i^j \leq \alpha_2 \\
& \sum_{i \in p} \mu_i + \delta_j + \sum_{i \in p} (sj + \tau_i^p + n_j t) \gamma_i^j + \tau^p \pi_t \leq \alpha_1 && \forall p \in P_{res}, \forall j \in J \\
& \mu_i \geq 0 && \forall i \in I \\
& \delta_j \leq 0 && \forall j \in J \\
& \gamma_i^j \leq 0 && \forall i \in I, \forall j \in J.
\end{aligned}$$

Let $(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\delta}}, \bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\gamma}})$ be an optimal solution of the restricted dual problem. The pricing problem relative to the profitable new pattern for the j-th bin looks for:

$$\min\left\{\alpha_1 - \left[\sum_{i\in p}\bar{\mu}_i + \bar{\delta}_j + \sum_{i\in p}(sj + \tau_i^p + n_j t)\bar{\gamma}_i^j + \tau^p\bar{\pi}_j\right]\right\} =$$
$$\alpha_1 - \bar{\delta}_j - \max\left\{\sum_{i\in p}\bar{\mu}_j + \sum_{i\in p}(sj + \tau_i^p + n_j t)\bar{\gamma}_i^j + \tau^p\bar{\pi}_j\right\} =$$
$$\alpha_1 - \bar{\delta}_j - P_j(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\gamma}})$$

holds. Thus, let $(p_1, \ldots, p_n)$ be the 0-1 variable vector relative to the pattern for the current j-th pricing. The pricing objective function $P_j(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\gamma}})$ can then be rewritten as:

$$\begin{aligned}
&\max \sum_{i\in p}\bar{\mu}_i + \sum_{i\in p}(sj + \tau_i^p + n_j t)\bar{\gamma}_i^j + \tau^p\bar{\pi}_j \\
&= \sum_{i\in I}\bar{\mu}_i p_i + \sum_{i\in p}(sj + t\sum_{h\le i} p_h + n_j t)\bar{\gamma}_i^j + t\bar{\pi}_j\sum_{i\in I} p_i \\
&= \sum_{i\in I}\left[\bar{\mu}_i + (sj + n_j t)\bar{\gamma}_i^j + t\bar{\pi}_j\right] p_i + t\sum_{i\in I}\sum_{h\le i}\bar{\gamma}_i^j p_h p_i.
\end{aligned}$$

For convenience we define:

$$\theta_{ij} = \bar{\mu}_i + (sj + n_j t)\bar{\gamma}_i^j + t\bar{\pi}_j, \tag{5.21}$$

and the j-th pricing problem $Pr(j)$ can be formalized as the following 0-1 quadratic knapsack (QKP) [108]:

$$\begin{aligned}
[Pr(j)] : \max\ & \sum_{i\in I}\theta_{ij}p_i + t\sum_{i\in I}\sum_{h\le i}\bar{\gamma}_i^j p_h p_i \\
\text{s.t. } & \sum_{i\in I} l_i p_i \le l \\
& p_i \in \{0,1\} \quad \forall i \in I.
\end{aligned}$$

5.3 Dynamic Programming for $Pr(j)$

It is known that QKP is NP-hard in the strong sense and it is unlikely that it can be solved in pseudo-polynomial time [57]. However, a dynamic programming (DP) approach with such running time can be devised for $Pr(j)$ due to the special structure of the matrix Q^j of the quadratic terms. Indeed, for the classical QKP extending DP algorithm designed for 0-1 knapsack problem does not preserve the Bellman principle of optimality [57]; in $Pr(j)$ instead each Q^j is a lower triangular matrix and each state of the recursion can be fully described by the previous one within a forward approach.

In this section we describe a dynamic programming algorithm $\mathcal{A}_{\mathcal{DP}}$ to solve $Pr(j)$. We formalize it for the more general case in which the scheduling term in the objective function is not limited to be $L_{\max}$, and its correctness holds as far as the structure of $Pr(j)$ remains unchanged, e.g. when the total sum of tardiness or the total number of tardy jobs are considered.

Let $(1, \dots, n)$ be sorted in non-decreasing order by $\bar{\gamma}_i^j$, and $f_j(h, i, k)$ be the optimal value of the j-th pricing under condition that:

1. the number of selected items is h;

2. the item candidates are $\{1, \dots, i\}$;

3. available integer knapsack capacity is $k \leq l$.

Then, the recurrence formula reads as:

$$\begin{aligned} f_j(h, i, k) &= \max\{f_j(h, i-1, k), \\ & f_j(h-1, i-1, k-l_i) + \theta_i^j + t \cdot h\bar{\gamma}_i^j\} \end{aligned} \tag{5.22}$$

where the boundary conditions are set as:

$$f_j(h, 0, k) = \begin{cases} 0 & h = 0 \\ -\infty & \text{otherwise.} \end{cases} \tag{5.23}$$

The optimal value of $Pr(j)$ is then given by:

$$P_j(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\pi}}, \bar{\gamma}) = \max_{h \in I} f_j(h, n, W). \tag{5.24}$$

The time complexity of $\mathcal{A}_{\mathcal{DP}}$ is $O(n^2 W)$. However, in practice the running time can be widely reduced by limiting the recursion to $h \leq n_1$, where n_1 represents any valid upper bound on the number of items that fits within a single bin.

Before proving the correctness of $\mathcal{A}_{\mathcal{DP}}$, we introduce the following lemma:

Lemma 5.3.1. *There exists an optimal solution to $Pr(j)$ in non-decreasing order of $\bar{\gamma}_i^j$.*

Proof. Let $\mathbf{p}^*$ be an arbitrary optimal solution for $Pr(j)$. If there exists a pair (h, i) of indices in $\mathbf{p}^*$ such that $h < i$ and $\bar{\gamma}_h^j > \bar{\gamma}_i^j$, a solution with no-worse objective function value can obtained by swapping the index of h and i. Clearly, the swapping operation does not affect the feasibility of $Pr(j)$. We can iterate the swapping procedure until we obtain an optimal solution in non-decreasing order of $\bar{\gamma}_i^j$. □

Now we proof the correctness of the proposed $\mathcal{A}_{\mathcal{DP}}$.

Theorem 5.3.1. *$\mathcal{A}_{\mathcal{DP}}$ is correct.*

Proof. Firstly, we observe that Lemma 5.3.1 holds true for all the subproblems corresponding to f_j.

The correctness of boundary condition is trivial. Let us assume all the $f_j(h,i,k)$ optimal for the corresponding subproblems $\forall h \leq h', \forall i < i'$ and $\forall k \leq k'$. For $f_j(h',i',k')$, if item i' is not chosen, then the optimal value is $f_j(h',i'-1,k')$; if i' is instead selected, it generates a profit equal to $\theta_{i'}^j + t \cdot h'\bar{\gamma}_{i'}^j$ under an arrangement of indices compliant with Lemma 5.3.1. The optimal value when item i' is selected is then $f_j(h'-1,i'-1,k'-l_{i'}) + \theta_{i'j} + t \cdot h'\bar{\gamma}_{i'}^j$. Therefore, (5.24) provides an optimal solution for the pricing problem $Pr(j)$. □

For all the cases in which the optimal schedule of items in each pattern is given by EDD, any pattern p with a different arrangement results in a sub-optimal column. Indeed, any solution of R_{VPT} in which such patterns are selected cannot have a strictly better optimal value, as the different arrangement can at most degrade the value of $L_{\max}$. We can then fix to EDD the order of $(1,\dots,n)$ and solve $Pr(j)$ by means of $\mathcal{A}_{\mathcal{DP}}$, while shrinking the space of research of potential patterns. This hopefully helps the convergence of the CG procedure, while controlling the size of R_{VPT}^{res} by limiting the number of redundant columns.

5.4 Computational results

Computational experiments were performed on a Intel® Core(TM) i7-5500U 2.40 GHz with 8Gb RAM. The CG procedure was coded in C++ and the formulations were solved by IBM® CPLEX® 12.5.0.0.

We compared the quality of the continuous relaxations of R_{VPT} and M_{VPT} on three group of instances (A,B,C), each one generated from fifty-three non-IRUP bin packing benchmark instances [43] with $n \in [20,200]$ and $W = 1000$. Integer due dates were randomly picked within $[s+t, \lceil \beta(sN_{LB_c} + tn) \rceil]$, where N_{LB_c} is the continuous lower bound for the 1BP [91] and $\beta \in \{0.6, 0.8, 1\}$ is used to typify A, B and C. We set $s = 1$ and $t = 0.2$, and consider two different scenarios with $\alpha_1 = 0.5$ and $\alpha_1 = 0.2$ respectively. For the 1BP-VPT, $|J|$ cannot be limited to the value of N of an arbitrary heuristic solution, as the minimization of Ω could require additional bins to reduce $L_{\max}$; hence, in all the experiments we trivially assumed $|J| = n$. The CG was initialized with solutions obtained by a First Fit heuristic for 1BP under EDD indexing and an SVC heuristic similar to the one exploited in [12]. The value of L^n_{LB} was computed accordingly to §5.2 and fairly embedded in both formulations.

We noted in preliminary experiments that often the dual vector $\bar{\gamma}$ results null and the actual number of quadratic knapsacks solved is extremely limited, i.e. between 2.1% for group A to 9.7% for group C. Thus, we improved the efficiency of the CG by relying on $\mathcal{A}_{\mathcal{DP}}$ only when $\bar{\gamma}$ is relevant, while using techniques for the classical 0-1 knapsack problem otherwise [80].

Table 5.1: Mean values of $\Omega_{\%}$ and Δ_{time} grouped by instance size n for $\alpha_1 = 0.5$.

		A		B		C	
n	# inst.	$\Omega\%$	Δ_t	$\Omega\%$	Δ_t	$\Omega\%$	Δ_t
20	1	42.3	0.1	43.1	0.1	53.6	0.1
40	3	42.4	0.2	54.4	0.2	67.2	0.1
60	9	42.6	0.4	55.2	0.2	61.9	0.3
80	3	41.8	3.8	53.2	10.3	55.1	2.5
100	4	43.5	0.7	55.3	-0.1	64.1	0.4
120	10	45.2	-1.7	57.5	-3.3	64.4	-4.2
140	3	46.8	-14.0	59.8	-30.3	58.0	-18.3
160	7	47.1	-30.2	59.1	-55.1	59.0	-36.4
180	5	46.8	-54.7	57.2	-89.6	58.8	-48.5
200	8	46.1	-52.6	58.4	-63.4	61.0	-49.7

Table 5.1 shows the results for $\alpha_1 = 0.5$. The first two columns report the size n of the instances and the number of instances with the same n value, respectively. Then, for each of the instance group, the mean percentage gap of the bound $\Omega_{\%}$ and the mean absolute CPU time difference Δ_t are reported. For all the instances, R_{VPT} outperforms M_{VPT} for the quality of the dual bound, as indicated by the positive values of $\Omega_{\%}$. The gap is exclusively related to the improvement of N in the objective function, as the value of L^n_{LB} is always tight for both formulations. Hence, $\Omega_{\%}$ increases from group A to group C as the due dates get loose (since L^n_{LB} becomes smaller), and also because N grows in absolute value. Regarding the CPU time, the CG runs in comparable time for up to 120 items, whereas it becomes widely faster for larger n as indicated by the values of Δ_t; the total time required to solve the continuous relaxation of R_{VPT} on all the groups of instances is 1853.7 seconds, while M_{VPT} needed 5211.8 seconds. The total number of pricing subproblems solved ranges between 33823 for group A to 38165 for group C. An improvement can clearly be reached by setting appropriate bounds on $|J|$ to contain both the size of R^{res}_{VPT} and the number of pricing subproblems. Moreover, we remark that the current CG implementation lacks of early termination and stabilization techniques that can meaningfully enhance the convergence.

We do not explicitly report the results for $\alpha_1 = 0.2$ as the values of $\tilde{L}_{\max}$ and N remain the same. It follows that the values of $\Omega_{\%}$ are reduced by the loss of weight of the packing component. In terms of efficiency, only a non-relevant increase of effort occurs to solve the instances as the CPU time rises up by 2.8% and the number of pricing solved is 2.6% larger.

We analyzed the integrality gap reached by using the bound value Ω provided by the CG and the best primal solution computed by the First Fit heuristic and the SVC algorithm. Table 5.2 reports the average gaps for both $\alpha_1 = 0.5$ and $\alpha_1 = 0.2$.

Table 5.2: Integrality gaps with respect to the best primal solution.

		$\alpha_1 = 0.5$			$\alpha_1 = 0.2$		
n	# inst.	A	B	C	A	B	C
20	1	14.3	21.7	14.3	17.2	27.0	21.3
40	3	11.0	19.7	14.3	15.0	30.6	27.1
60	9	11.7	16.6	19.2	14.8	26.1	40.5
80	3	10.6	10.5	19.4	12.2	15.8	41.9
100	4	10.2	15.5	16.5	12.2	25.7	39.0
120	10	11.8	13.4	16.5	14.3	20.6	38.7
140	3	10.7	12.8	17.3	13.2	19.8	37.3
160	7	9.5	13.6	16.1	12.0	21.9	35.4
180	5	9.7	12.6	17.5	12.2	19.6	43.8
200	8	9.4	11.1	15.0	11.8	17.6	33.0

Generally speaking, the integrality gap is non-negligible in all instances, resulting on the mean from 10.9% for group A, 14.8% for group B and 16.6% for group C with $\alpha_1 = 0.5$. When $\alpha_1 = 0.2$ is addressed, the gaps further increase to 13.5% for group A, 22.5% for group B and 35.8% for group C. Such increment however can be ascribed to the growth of the weight of the scheduling term in the convex combination.

Finally, we studied the quality of the incumbent solution provided by R^{res}_{VPT} after the CG termination, with integrality constraints restored and all the columns generated by the CG procedure. For any group of instance and tested value of α_1, no meaningful improvement was achieved in terms of primal bound value. Solving the MILP exactly required 4.4 second on the mean for $\alpha_1 = 0.5$ and 3.0 seconds on the mean for $\alpha_1 = 0.2$.

5.5 Conclusion and future development

In this chapter a new pattern-based extended reformulation R_{VPT} has been presented to solve the 1BP-VPT problem. Moreover, a dynamic programming algorithm $\mathcal{A}_{\mathcal{DP}}$ has been described to solve the quadratic knapsack problem, that arises not only when $L_{\max}$ is chosen as scheduling term. The computational experiments show that the dual bound provided by R_{VPT} outperforms the one obtained by M_{VPT}, and the gain in terms of CPU time becomes widely positive for large size of the instances.

Looking at the solution of the integer 1BP-VPT, the integrality gap is significant in all the cases and a potential full branch-and-price scheme should embed smart strategies and useful upper bounds on $|J|$ to reach optimal integer solutions in an efficient way. Also, we are interested in extending the problem to multiple-dimension CSP, since the structure of patterns is more complex and the issue of variable processing time become even more relevant.

Chapter 6

LP-based dual bounds for the maximum quasi-clique problem

Finding cohesive structures is a matter of interest in a wide span of fields, such as biology, genetics, telecommunication and so forth. The mathematical object that models perfectly aggregated substructures on graphs is the *clique*, in which each node is pairwise adjacent to each other. Allowing a restrained grade of imperfection, i.e. renouncing to some edges that connect nodes in the clique, brings to the class of *clique relaxations* that provide in many cases even better representations of cohesive structures w.r.t. cliques.

In this chapter, we focus on a density-based relaxation of the clique, namely the *γ-quasi-clique*. Finding the γ-quasi-clique of maximum order basically consists in a structure packing problem, since each clique on a graph has a one-to-one correspondence to an independent set on the complementary graph, see §1. We face this problem by computing fast dual bounds by means of CG procedure based on an integer Dantzig-Wolfe reformulation. Then, we discuss the issue of solution connectivity and propose a sufficient condition to test the connectivity of γ-quasi-clique.

6.1 Introduction

A clique is a complete graph, i.e., a graph with an edge for any pair of vertices, and it is one of the basic combinatorial structures in graph theory. The MAXIMUM CLIQUE PROBLEM (MCP) consists of finding an induced *clique* of maximum order in a simple and undirected graph G [104]. Solutions of the MCP are meaningful, at least in principle, for a wide range of applications, e.g., social network analysis, coding theory, telecommunication and genetics. In fact, cliques express an ideal aggregation measure and are representative when it is interesting to evaluate the degree of interaction between entities. However, the search for a complete structure like a clique often prevents the discovery of similarly interesting dense subgraphs. Furthermore, graphs derived from real-world applications are generated from incomplete data and are often acquired through error-prone processes. For these reasons, several clique relaxations have been defined and the corresponding MAXIMUM RELAXED-CLIQUE PROBLEMS (MRCP-s) have been investigated [106]. Clique relaxations can be classified according to the number of relaxed properties: first-order relaxations are defined by slackening a single clique-typifying property related, for instance, to the degree (*k-core*, *k-plex*), the distance (*k-clique*, *k-club*), the density (*γ-quasi-clique*, *k-defective clique*) and the connectivity (*k-block*, *k-bundle*) [64]. Higher-order relaxations can also be considered by relaxing more than one properties at the same time, see [106].

Among discrete optimization problems deriving from clique relaxations, there is a pair of $\mathcal{NP}$-hard reciprocal problems: the MAXIMUM QUASI-CLIQUE PROBLEM (γ-QCP) [107] and the K-DENSEST SUBGRAPH PROBLEM (KDSP). The former asks for the maximum-order induced subgraph with edge density of at least γ of a simple undirected graph G, whereas the latter calls for the densest subgraph of G of order k. Another problem closely related to the γ-QCP is the MAXIMUM DEGREE-BASED γ-QCP [105]. A *degree-based γ-quasi-clique* is a subgraph $H = (Q, E_Q)$ of a graph G induced by the set of vertices Q such that the degree of any vertex of H is at least $\gamma(|Q| - 1)$. It is easy to see that any degree-based γ-quasi-clique is a γ-quasi-clique but not vice-versa.

The γ-QCP is both theoretically and computationally difficult. Critical aspects (among others) lie in the lack of the *hereditary property* and in the existence of disconnected optimal quasi-cliques. Recall that a property $\mathcal{P}$ of graphs is an infinite class of graphs which is closed under isomorphism and it is *hereditary* if every induced subgraph of every member of $\mathcal{P}$ is also in $\mathcal{P}$ [26]. This implies that the inclusionwise maximality of the vertex-set of a graph satisfying a hereditary property can be tested in polynomial time. However, it is well known that γ-quasi-cliques are only *quasi-hereditary*. In fact, to obtain an induced γ-quasi-clique K from a γ-quasi-clique $H = (Q, E)$ it is sufficient to remove a vertex v from H with $\deg(v) < \frac{2\cdot|E|}{|Q|}$ but this makes not straightforward the maximality check of non-trivial γ-quasi-cliques.

The connectivity role in γ-QCP is also not completely settled. Allowing disconnected γ-quasi-cliques clearly extends the solution space but the computational consequences have not been investigated. Note that, differently from other clique relaxations (k-plex, k-defective clique, and k-bundle) for which disconnected subgraphs can be optimal solutions only if sufficiently small, the size of a disconnected γ-QCP optimal solution is generally not limited from above [64]. However, many real applications implicitly ask to find dense connected subgraphs in order to properly capture the relations between elements of clusters, and a solution comprised of more than one connected component misses this aspect and may become less meaningful for the application (the presence of disconnected optimal solutions could be even more frequent in both the vertex-weighted and edge-weighted version of γ-QCP). Therefore, the quasi-clique definition (and similar) should include the connectivity property to some extent. Section §6.4 is devoted to discuss such issues.

Several heuristic approaches were proposed to solve the γ-QCP, or problems with a slightly different definition of γ-quasi-clique [24, 30, 79]. In [1] the authors describe a greedy randomized adaptive search procedure (GRASP) to detect maximal quasi-cliques in massive sparse graphs, where the local search phase exploits the concept of vertex potential to move up to local optima; Tsourakakis et al. [123] present two heuristics, one based on iterated elimination of the vertex with the smallest degree, and the other performing a local search looking for a sequence of induced subgraphs with non-decreasing value of density.

To the best of our knowledge, the reference exact methods for γ-QCP are the MIP-based approach presented in [130] and the combinatorial branch-and bound algorithm described in [102]; for KDSP instead, the state-of-the-art exact method is the enumeration scheme with dual bounds computed via semidefinite programming proposed in [76].

In this chapter we develop earlier ideas originally formulated in [89]. We firstly review the main combinatorial and LP-based dual bounds for γ-QCP available in the literature. Then, we propose an integer reformulation $[D_\gamma]$ of γ-QCP and a surrogate relaxation $[D^S_\gamma]$ of $[D_\gamma]$ that provides dual bounds as good as those computed by the linear relaxation of $[D_\gamma]$.

The surrogate relaxation uses a number of constraints linear in the number of vertices of the graph and therefore it can be exploited for computing dual bounds on large and dense graphs. Then, we present a new sufficient condition for obtaining connected γ-quasi-cliques that dominates the previous result reported in the literature.

The outline of the chapter is as follows: in §6.2 the γ-QCP is formalized, then mixed integer linear programming formulations (MILPs) and combinatorial dual bounds from the literature are reviewed; a Dantzig-Wolfe [45] reformulation is presented in §6.3; in §6.4 a new sufficient condition for solutions connectivity is given; finally, computational results are reported in §6.5 while conclusions are captured in §6.6.

6.2 Problem definition, MILP formulations and bounds

The γ-QCP can be formalized as follow. Let $H = (Q, E_Q) = G[Q]$ be the subgraph of $G = (V, E)$ induced by the set of vertices $Q \subseteq V$. Given $\gamma \in (0, 1]$, an optimal solution of γ-QCP is an induced subgraph H of maximum order $|Q^*|$ and with a number of edges $|E_{Q^*}| \geq \gamma \cdot \frac{|Q^*|(|Q^*|-1)}{2}$.

6.2.1 MILP formulations

Veremyev et al. [130] propose four MILP formulations for γ-QCP, the tightest of which, reported in the following, consists of $O(|V| + |E|)$ variables and $O(|E|)$ constraints. Let x_i, $i \in V$, and z_e, $e \in E$, be binary variables with $x_i = 1$ iff $i \in Q$, and $z_e = 1$ iff $e \in E_Q$. Moreover, let y_k, $k \in K = \{k_L, \dots, k_U\}$, be the binary variable with $y_k = 1$ if H is of order k. The formulation reads as:

$$[C_\gamma]: \qquad |Q^*| = \max \sum_{i \in V} x_i \tag{6.1}$$

$$z_e \leq x_i, \quad z_e \leq x_j \qquad \forall e = \{i,j\} \in E \tag{6.2}$$

$$\sum_{i \in V} x_i \leq \sum_{k \in K} k y_k \tag{6.3}$$

$$\sum_{k \in K} y_k = 1 \tag{6.4}$$

$$\gamma \sum_{k \in K} \frac{k(k-1)}{2} y_k \leq \sum_{e \in E} z_e \tag{6.5}$$

$$x_i, z_e, y_k \in \{0,1\} \qquad \forall i \in V, \forall e \in E, \forall k \in K. \tag{6.6}$$

Edge $e = \{i,j\}$ belongs to the γ-quasi-clique H if (and only if) vertex i and j are both in Q, see constraints (6.2). The order k of H is defined by constraints (6.3) and (6.4), and constraint (6.5) bounds from below the density of H by γ.

$[C_\gamma]$ can be easily modified to model other density-based clique relaxations: a formulation for the maximum *s-defective* clique problem can be obtained by replacing constraint (6.5) with

$$\sum_{k \in K} \frac{k(k-1)}{2} y_k \leq \sum_{e \in E} z_e + s$$

whereas a formulation for the maximum degree-based γ-QCP results by replacing (6.5) with the set of constraints

$$\gamma\Big(\sum_{k \in K} k y_k - 1\Big) \leq \sum_{e=\{i,j\} \in E} z_e + \gamma(k_U - 1)(1 - x_i) \qquad \forall i \in V.$$

The size of $[C_\gamma]$ grows with the density of G whereas an alternative MILP, originally presented in [107], does not. Such a formulation is obtained by linearizing the quadratic constraint that models the density threshold condition $|E_{Q^*}| \geq \gamma \cdot \frac{|Q^*|(|Q^*|-1)}{2}$. Namely, by introducing an additional variable w_i, for each $i \in V$, such that:

$$w_i = \gamma x_i x_i + \sum_{j \in V} (a_{ij} - \gamma) x_i x_j$$

where a_{ij} is equal to one if $\{i, j\} \in E$ and zero otherwise, the γ-QCP can be formulated as follows:

$$\begin{aligned}
[P_\gamma]: \qquad & |Q^*| = \max \sum_{i \in V} x_i && && (6.7)\\
& \sum_{i \in V} w_i \geq 0 && && (6.8)\\
& w_i \leq u_i x_i, \quad w_i \geq l_i x_i && \forall i \in V && (6.9)\\
& w_i \leq \gamma x_i + \sum_{j \in V} (a_{ij} - \gamma) x_j - l_i(1 - x_i) && \forall i \in V && (6.10)\\
& w_i \geq \gamma x_i + \sum_{j \in V} (a_{ij} - \gamma) x_j - u_i(1 - x_i) && \forall i \in V && (6.11)\\
& x_i \in \{0, 1\}, w_i \in \mathbb{R} && \forall i \in V && (6.12)
\end{aligned}$$

where u_i and l_i respectively are upper and lower bounds on the value of w_i obtained by setting:

$$u_i = (1 - \gamma) \sum_{j \in V} a_{ij}, \quad l_i = -(n - 1 - \sum_{j \in V} a_{ij})\gamma \quad \forall i \in V.$$

Formulation $[C_\gamma]$ rapidly grows due to its $O(|E|)$ constraints and therefore it is suitable for computing dual bounds only on graphs sparse enough. On the contrary, the number of variables and constraints of $[P_\gamma]$ grow linearly with $|V|$ and does not depend on graph density, but the bound provided by the linear relaxation of $[P_\gamma]$ is weaker than the one obtained by $[C_\gamma]$ (see Section 6.5).

A smaller formulation on dense graphs can be derived by looking at *complementary* γ-quasi-cliques, i.e., induced subgraphs $G[Q]$ with a density that does not exceed $\gamma \in [0, 1)$ [27]. Any γ-quasi-clique on G then corresponds to a complementary $(1 - \gamma)$-quasi-clique on the complement graph $\bar{G} = (V, \bar{E})$, both induced by the subset of vertices Q. Hence, an optimal solution of γ-QCP can be achieved by solving to optimality the maximum complementary $(1 - \gamma)$-quasi-clique problem by means of the integer program $[\bar{C}_\gamma]$ consisting of (5.1), (5.4), (6.4), (5.6) plus the following constraints:

$$x_i + x_j - 1 \leq z_e \qquad \forall e = \{i,j\} \notin E \tag{6.13}$$

$$\sum_{e \notin E} z_e \leq (1-\gamma) \sum_{k \in K} \frac{k(k-1)}{2} y_k. \tag{6.14}$$

The formulation is a straightforward adaptation of $[C_\gamma]$; constraints (6.13) is expressed for each edge in $\bar{E}$ and ensures the presence of an edge $e = \{i,j\}$ within the complementary $(1-\gamma)$-quasi-clique if (and only if) both endpoints i and j are selected. Finally, inequality (6.14) models the density restriction of a complementary $(1-\gamma)$-quasi-clique. Formulation $[\bar{C}_\gamma]$ uses $O(|V| + |\bar{E}|)$ variables and $O(|\bar{E}|)$ constraints.

6.2.2 Primal and dual bounds

A lower bound k_L to $|Q^*|$ is given by the order of any clique of G. On the other hand, a basic upper bound to $|Q^*|$ is in [107]:

$$k_U = \left\lfloor \frac{1}{2} + \frac{1}{2}\sqrt{1 + \frac{8|E|}{\gamma}} \right\rfloor.$$

A better upper bound $\bar{k}_U$ can be obtained in $O(|V| \log |V|)$ as follows, see also [102]. Any γ-quasi-clique $H = (Q, E_Q)$ fulfils by definition $|Q|(|Q|-1) \leq 2\frac{|E_Q|}{\gamma}$. Moreover, $|E_Q| = \sum_{i \in Q} \frac{d_i^H}{2} \leq \sum_{i \in Q} \frac{\min\{|Q|-1, d_i\}}{2}$, where d_i is the degree of i in G and d_i^H is the degree of i in H. Therefore:

$$|Q|(|Q|-1) \leq \frac{1}{\gamma} \sum_{i \in Q} \min\{|Q|-1, d_i\} \leq \frac{1}{\gamma} \sum_{i=1}^{|Q|} \min\{|Q|-1, d_i\}$$

where the last inequality holds if vertices of G are sorted by non-increasing degrees, i.e., $d_i \geq d_j$ for $i < j$. It easy to see that the largest integer Q that satisfies the above inequality is a valid upper bound $\bar{k}_U$ for $|Q^*|$, and that $\bar{k}_U < k_U$ for the $|Q^*| < |V|$.

Let $\omega(G)$ be the clique number of graph G, i.e. the order of the maximum clique of G. If $1 - \frac{1}{\omega(G)} < \gamma$, an alternative upper bound k_U^ω on $|Q^*|$ has been defined in [107] as:

$$k_U^\omega = \left\lfloor \frac{\omega(G)\gamma}{1 - \omega(G) + \omega(G)\gamma} \right\rfloor \geq |Q^*|. \tag{6.15}$$

Unlike the aforementioned bounds, k_U^ω does not depend directly on the order and the size of G. Figure 6.1 shows the value of k_U^ω with respect to $\omega(G)$ and γ. Although particularly useful when applied to sparse graphs, in which $\omega(G)$ value is limited, it rapidly becomes poor on dense graphs. For instance let us consider a graph G with $|E| = 20,000$, more than 230 vertices, and $\omega(G) = 31$. Given $\gamma = 0.97$, it results $k_U^\omega = 429$ and $k_U = 203$. Moreover, the efficacy of k_U^ω diminishes as γ gets smaller; for instance, k_U^ω with $\gamma = 0.9$ is only defined up to $\omega(G) \leq 9$. Computing $\omega(G)$ can be extremely time consuming on sufficiently large graphs, so that a reasonable choice can be to substitute $\omega(G)$ with a suitable upper bound $\omega_U(G)$ within (6.15). However, condition $1 - \frac{1}{\omega_U(G)} < \gamma$ must hold, limiting the applicability of the substitution, and the resulting bound degrades as $\omega_U(G)$ loses tightness.

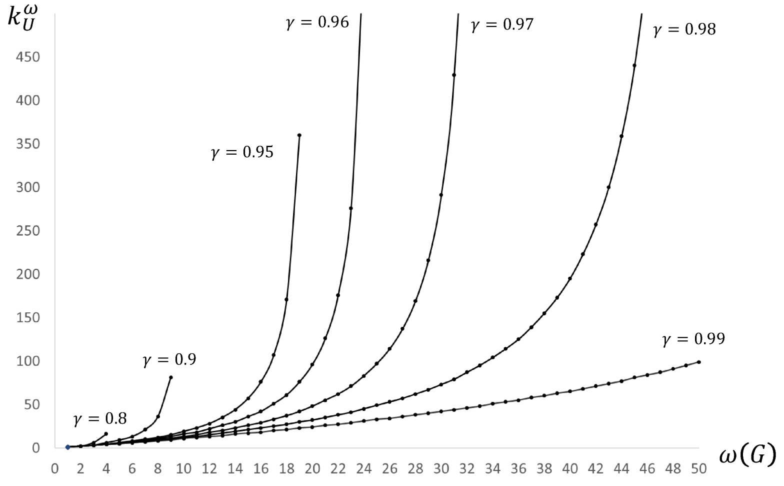

Figure 6.1: k_U^ω for several γ values and $\omega(G) \leq 50$

6.3 Star-based reformulation

Model $[C_\gamma]$ can be reformulated by integer decomposition [45]. Let d_i be the degree of vertex $i \in V$, $N(i)$ the set of neighbours of i, and $S(i)$ the set of incidence edges of i that, for the sake of conciseness, we call the *star* of i. The *star constraint*

$$\sum_{e \in S(i)} z_e \leq (k_U - 1)x_i \tag{6.16}$$

is a valid inequality for the convex hull of the integer solutions to $[C_\gamma]$.

Namely, the set of integer points that non-trivially satisfy (6.16) corresponds to the collection $\mathcal{S}_i = \{S_{i1}, S_{i2}, \dots\}$ of all the non-empty partial stars of i containing less than k_U edges. Let $\mathcal{S}_i^j \subset \mathcal{S}_i$ be the collection of the partial stars including the edge $\{i,j\} \in E$. Then for any $e = \{i,j\} \in E$, the variables x_i and z_e of $[C_\gamma]$ can be rewritten as

$$x_i = \sum_{h=1}^{|\mathcal{S}_i|} \lambda_{ih} \qquad z_e = \sum_{h=1}^{|\mathcal{S}_i^j|} \lambda_{ih} \qquad \sum_{h=1}^{|\mathcal{S}_i|} \lambda_{ih} \le 1 \qquad \lambda_{ih} \in \{0,1\},$$

where variable λ_{ih} is equal to 1 if the partial star S_{ih} is selected, and 0 otherwise.

The resulting MIP formulation reads as follows:

$$[D_\gamma]: \qquad |Q^*| = \max \sum_{i \in V} \sum_{h=1}^{|\mathcal{S}_i|} \lambda_{ih} \tag{6.17}$$

$$\sum_{h=1}^{|\mathcal{S}_i|} \lambda_{ih} \le 1 \qquad \forall i \in V \tag{6.18}$$

$$\sum_{h=1}^{|\mathcal{S}_i^j|} \lambda_{ih} - \sum_{h=1}^{|\mathcal{S}_j^i|} \lambda_{jh} = 0 \qquad \forall e = \{i,j\} \in E \tag{6.19}$$

$$\sum_{i \in V} \sum_{h=1}^{|\mathcal{S}_i|} \lambda_{ih} \le \sum_{k \in K} k y_k \tag{6.20}$$

$$\sum_{k \in K} y_k = 1 \tag{6.21}$$

$$\sum_{k \in K} \left\lceil \frac{\gamma k(k-1)}{2} \right\rceil y_k - \frac{1}{2} \sum_{i \in V} \sum_{h=1}^{|\mathcal{S}_i|} |S_{ih}| \lambda_{ih} \le 0 \tag{6.22}$$

$$\lambda_{ih} \in \{0,1\} \qquad \forall i \in V, h \in \{1, \dots, |\mathcal{S}_i|\} \tag{6.23}$$

$$y_k \in \{0,1\} \qquad \forall k \in K. \tag{6.24}$$

Constraint (6.18) requires that at most one partial star can be selected for each vertex. Constraints (6.19) impose a consistent selection of partial stars, that is, if a partial star S_{ih} is chosen and S_{ih} contains the edge $\{i,j\}$, then a partial star S_{jp} including the edge $\{j,i\}$ must be selected too. To this purpose, the coefficient of the variable λ_{ih} is $+1$ for edges $\{i,j\} \in S_{ih}$ with $i < j$ and -1 otherwise. Constraints (6.20), (6.21) and (6.22) directly derive from $[C_\gamma]$

Figure 6.2 shows the coefficients matrix of constraints (6.18) and (6.19) for the *kite graph* (the kite has $|V| = 5$ and $|E| = 6$, see [29]). Gray columns describe the optimal solution for $\gamma = 0.8$, corresponding to the clique induced by vertices $\{1,2,3\}$. The clique is composed by overlapping partial stars S_{11}, S_{21} and S_{31}, where edges are weighted $\frac{1}{2}$ in density constraint (6.22).

Figure 6.2: Star reformulation on a *kite graph*, $|V| = 5$, $|E| = 6$

		S_1						S_2						S_3						S_4	S_5				
V	1	**1**	1	1	1	1	1																	$\leq$	1
	2							**1**	1	1	1	1	1											$\leq$	1
	3													**1**	1	1	1	1	1					$\leq$	1
	4																			1				$\leq$	1
	5																				1	1	1	$\leq$	1
E	$\{1,2\}$	**1**	1		1			**–1**	–1		–1													=	0
	$\{1,3\}$	**1**		1		1								**–1**	–1		–1							=	0
	$\{1,4\}$		1	1			1													–1				=	0
	$\{2,3\}$							**1**		1		1		**–1**		–1		–1						=	0
	$\{2,5\}$								1	1			1								–1	–1		=	0
	$\{3,5\}$														1	1			1		–1		–1	=	0

Since each star inequality (6.16), together with the variable bound constraints, defines an integral polyhedron, the continuous relaxations of $[C_\gamma]$ plus star inequalities and the integer reformulation $[D_\gamma]$ have the same optimal value. Therefore, the dual bound provided by $[D_\gamma]$ is at least as tight the one provided by $[C_\gamma]$, where possible improvements arise from star constraints (6.16) or the rounding up of the y_k's coefficients in (6.22). For instance, on a set of sparse graphs taken from [130] the effect of star constraints appears extremely limited, with a mean percentage improvement of 0.4% for $\gamma = 0.9$. When edge-weighted graphs are taken into account [115], the problem can be generalized to the weighted γ-QCP. In this case, the polyhedron described by each corresponding weighted star constraint plus variable bound constraints becomes fractional and $[D_\gamma]$ is an integer reformulation providing tighter dual bounds.

Model $[D_\gamma]$ consists of an exponential number of variables in $|E|$ and therefore the solution of its continuous relaxation requires a column generation approach. The search for useful columns (i.e., partial stars) requires to solve for each vertex i the following pricing problem. Let $\sigma_i, \delta, \psi \in \mathbb{R}^+$ and $\pi_e \in \mathbb{R}$ be the values of dual variables associated to constraints (5.10),(5.12),(5.15) and (5.11), respectively, and let w_e be a binary variable equal 1 if edge e belongs to the partial star (0 otherwise). The reduced cost of the most profitable partial star of vertex i is:

$$\bar{c}(i) = -\sigma_i - \delta + \max \sum_{e \in S(i)} (\frac{1}{2}\psi + \Gamma(e)\pi_e)w_e \qquad (6.25)$$

with $\Gamma(e = \{i,j\}) = -1$ if $i < j$ and $\Gamma(e) = 1$ otherwise. The pricing problem search for the set of at most $(k_U - 1)$ edges that maximize $\bar{c}(i)$ (6.25). Such a set can be obtained simply by ranking the edges of $S(i)$ by non decreasing values of $(\frac{1}{2}\psi + \Gamma(e)\pi_e)$.
Formulations $[C_\gamma]$ and $[D_\gamma]$ are not suitable for solving the γ-QCP on dense graphs due their $O(|E|)$ constraints, and even the continuous relaxation can be difficult to solve for moderate size instances. However, at the cost of a small loss in the dual bounds quality, a surrogate relaxation $[D^S_\gamma]$ of $[D_\gamma]$ can be considered by replacing constraints (5.11) with the following ones:

$$\sum_{h=1}^{|\mathcal{S}_i|} |S_{ih}|\lambda_{ih} - \sum_{j \in N(i)} \sum_{h=1}^{|\mathcal{S}^i_j|} \lambda_{jh} = 0 \qquad \forall i \in V. \tag{6.26}$$

Constraint (6.26) are obtained, for each vertex i, by summing up all constraints in (5.11) for $e = \{i, N(i)\}$. Hence, $[D^S_\gamma]$ has $O(|V|)$ constraints. Now, let $\theta_i \in \mathbb{R}$ be the values of the dual variable related to (6.26). The pricing problem for $[D^S_\gamma]$ associated to vertex i can be easily derived from (6.25) by properly adapting it into

$$\bar{c}(i) = 1 - \sigma_i - \delta + \max \sum_{e \in S(i)} (\frac{1}{2}\psi - \theta_i + \theta_j) w_e. \tag{6.27}$$

6.4 Quasi-clique connectivity

In several real-world applications, finding cohesive clusters is naturally referred to the identification of single connected components on graphs [64]. In [59] a thorough discussion of approaches for the community detection problem is presented and the connectivity is assumed as a required property.

Such assumption is often reasonable as disjointed clusters actually represent clusters whose mutual interaction can be assumed irrelevant with respect to the aggregation properties of interest. By contrast, solutions composed of multiple connected components are suitable or even characterizing for alternative problems, such as *clique relaxation packing problems* or *maximal clique relaxation enumeration problems*, which arise by generalizing the corresponding optimization problems originally defined on cliques [32, 2].
Optimal k-core, k-defective and γ-quasi cliques can be disconnected and therefore the corresponding MRCPs should explicitly require the connectivity condition. One can argue that connectivity can be ensured by considering alternative clique relaxations, such as k-club (an induced subgraph of diameter at most k, see [3, 95]) or k-block (an induced subgraph whose minimum *vertex cut* is at least k, see [38, 64]), for which optimal solutions are always connected.

Generally speaking however, the purpose of k-clubs and k-blocks is to guarantee respectively a given degree of reachability and robustness, whereas density based relaxations, such as k-defective and γ-quasi-cliques, are useful to deal with noisy and missing data. Moreover, although a γ-quasi-clique can be composed by several very dense disconnected subgraphs, typically the significant γ threshold ($\gamma > 0.5$) results in solutions made of a single large connected component along with other small ones. On the other hand, it is easy to see that a γ-quasi-clique can be composed by k connected components of the same order only if $\gamma < 1/k$. This also implies that cut-edges of a connected solution generally link a large component to small ones.

We suppose that the lack of an explicit request of connectivity within formulations of MRCP can be attributed to the easiness of modeling it in mathematical programming terms. Indeed, several solution approaches can be conceived that take advantage of connectivity. For instance, in a branch-and-bound the selection of the branching variable can be done assuming the connectivity of the chased solution.

Ensuring the connectivity within MILP formulations can be done in several ways, typically by recalling variables and constraints used to model connectivity in Hamiltonian or shortest path problems [121]. Indeed, a large body of research exists on this topic, see for example [31] and the references therein, as well as the recent polyhedral study of the *connected subgraph polytope* [132].

As example, connectivity can be guaranteed for γ-QCP (and similarly for other MRCPs) by introducing variables $c_i \in \{0,1\}$ for each vertex $i \in V$, where c_i is set to 1 iff vertex i is selected as source, and flow variables $f_{ij} \in \mathbb{R}$ for each edge $\{i,j\} \in E$. Optimal connected γ-quasi-cliques can be found by solving a MILP formulation that adds to the formulation $[C_\gamma]$ in Section 6.2 the following constraints:

$$\sum_{i \in V} c_i = 1 \tag{6.28}$$

$$c_i \leq x_i \qquad \forall i \in V \tag{6.29}$$

$$\sum_{h \in V} x_h - 1 - k_u(1 - c_i) \leq \sum_{j \in N(i): i<j} f_{ij} - \sum_{j \in N(i): j<i} f_{ji} \qquad \forall i \in V \tag{6.30}$$

$$\sum_{h \in V} x_h - 1 + k_u(1 - c_i) \geq \sum_{j \in N(i): i<j} f_{ij} - \sum_{j \in N(i): j<i} f_{ji} \qquad \forall i \in V \tag{6.31}$$

$$-k_u(1 + c_i - x_i) - 1 \leq \sum_{j \in N(i): i<j} f_{ij} - \sum_{j \in N(i): j<i} f_{ji} \qquad \forall i \in V \tag{6.32}$$

$$k_u(1 + c_i - x_i) - 1 \geq \sum_{j \in N(i): i<j} f_{ij} - \sum_{j \in N(i): j<i} f_{ji} \qquad \forall i \in V \tag{6.33}$$

$$-(k_u - 1)z_e \leq f_{ij} \leq (k_u - 1)z_e \quad \forall e = \{i,j\} \in E \tag{6.34}$$
$$c_i \in \{0,1\} \quad \forall i \in V. \tag{6.35}$$

By means of (6.28) and (6.29), one vertex is selected as source node among the ones belonging to the γ-quasi-clique. Constraints (6.30)-(6.33) resemble single-flow constraints for TSP adapted on undirected graphs: (6.30) and (6.31) enforces that exactly $|Q^*| - 1$ units of flow leave the source, whereas (6.32) and (6.33) ensures that a single unit of flow is absorbed by any other vertices of the γ-quasi-clique. Finally, (6.34) and (6.35) set the bounds for f_{ij} variables and c_i respectively, where f_{ij} results zero if $e = \{i,j\} \notin E_Q$. On the whole, this method requires the addition of $O(|V| + |E|)$ variables and constraints to $[C_\gamma]$.

In [106] the sufficient conditions that ensures the connectivity of optimal solutions are depicted for the main first-order clique relaxations. In particular, a solution of the γ-QCP with $|Q|$ vertices is connected if

$$\left\lceil \gamma \binom{|Q|}{2} - \binom{|Q|-1}{2} \right\rceil \geq 1 \tag{6.36}$$

holds. Figure (6.3) reports the upper bound on $|Q|$ set by inequality (6.36) with respect to γ.

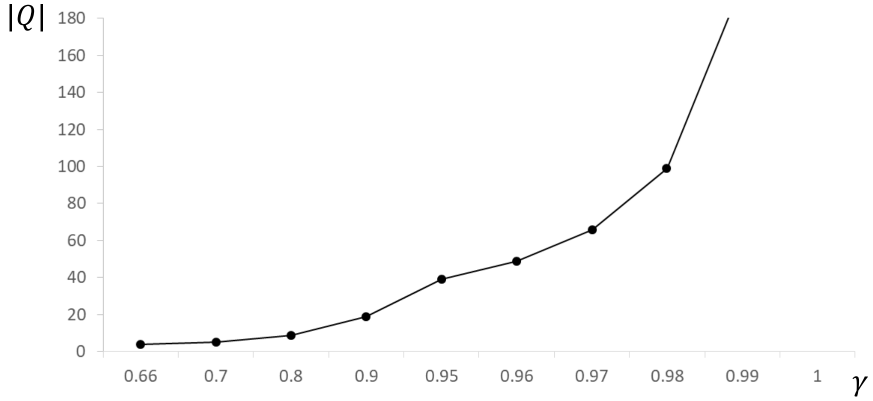

Figure 6.3: Upper bound on $|Q|$ values set by (6.36)

Firstly, condition is ineffective for $\gamma < 0.64$. Then, the values of $|Q|$ required are small even for reasonable values of γ. For instance, if $\gamma = 0.9$, the maximum γ-quasi clique must contain at most 19 vertices to verify the condition and this value is not meaningful for graphs of practical interest.

In the following we present results related to the structure of γ-quasi-cliques based on the existence of small connected components. Recall that $G[Q]$ refers to the subgraph (Q, E_Q) of G induced by the subset of vertices $Q \subseteq V$.

Lemma 6.4.1. *Let G be connected and $G[Q]$ be a γ-quasi-clique formed by μ connected components with at least one component having at most three vertices. Then, there exists a γ-quasi-clique of the same order and no smaller size with at most $\mu - 1$ connected components.*

Proof. Let X be a connected component of $G[Q]$ with at most 3 vertices, and $P = p_1 - \ldots - p_q$ be the longest simple path connecting X to some other connected component of $G[Q]$, say Y, with p_1 vertex of X and p_q vertex of Y (P always exists since G is connected). If $q \leq 4$, (part of) X can be replaced by (part of) P thus reducing the number of connected components of $G[Q]$ by one, while the number of edges (vertices) of $G[Q]$ does not decrease (increase). If $q > 4$, there are at least 3 vertices $\{u, v, w\}$ in $G \backslash G[Q]$ that can be connected to some other connected component of $G[Q]$ (since G is connected). Hence the vertices of X can be replaced by u, v, w without losing any edge and thus obtaining a γ-quasi-clique with at most $\mu - 1$ connected components.

□

The above lemma can be used to get the following result related to the maximum number of connected components of a γ-quasi-clique:

Lemma 6.4.2. *Let $G[Q]$ be a γ-quasi-clique with μ connected components and such that*

$$|E_Q| > \binom{|Q| - 4(\mu - 1)}{2} + 6(\mu - 1) \tag{6.37}$$

holds. If G is connected, then a γ-quasi-clique with $|Q|$ vertices and at most $\mu - 1$ connected components can be easily obtained by $G[Q]$.

Proof. Inequality (6.37) forces $G[Q]$ to have a number of edges that strictly exceeds the total number of edges incident on $|Q|$ vertices disjointed into a clique Y of order $|Q| - 4(\mu - 1)$ and $\mu - 1$ cliques of 4 vertices each one (connected components with less than 4 vertices can be excluded by Lemma 6.4.1). For the properties of the binomial coefficient, removing vertices of Y (up to a minimum order of 4) to enlarge the other $\mu - 1$ cliques cannot increase the total number of edges of the μ cliques. Hence, the right-hand side of (6.37) defines an upper bound on the number of edges for any induced subgraph of G with $|Q|$ vertices divided into μ (or more) connected components. Therefore, if $|E_Q|$ is greater than such bound there exists at least a connected component in $G[Q]$ with at most three vertices. The thesis follows by Lemma 6.4.1. □

A sufficient condition for the connectivity of γ-QCP solutions can be obtained by setting $\mu = 2$ in Lemma 6.4.2.

Proposition 9. *If G is connected and $|E_Q| \geq 17 + \frac{|Q|}{2}(|Q| - 9)$ then either $G[Q]$ is a connected γ-quasi-clique, or a connected γ-quasi-clique with $|Q|$ vertices can be easily obtained from $G[Q]$.*

Furthermore, the following dominance result holds:

Proposition 10. *The sufficient condition reported in Proposition 9 dominates condition (6.36) for any $|Q| \geq 5$.*

Proof. Let $G[Q]$ be a γ-quasi-clique such that (6.36) is valid. By definition $|E_Q| \geq \gamma\binom{|Q|}{2}$ holds. Then, by inequality (6.36) it follows

$$\left\lceil |E_Q| - \binom{|Q|-1}{2} \right\rceil = |E_Q| - \binom{|Q|-1}{2} \geq 1$$

where the second member is given by the difference of two integer terms. The statement then derives by algebraic manipulations.

$\square$

To show that the two conditions are not equivalent, we provide a simple example. Let $|Q| = 10$, $|E_Q| = 36$ and $\gamma = 0.8$. Clearly, $G[Q]$ is a γ-quasi-clique. It is easy to see that condition (6.36) is not satisfied, whereas Proposition 9 proves that $G[Q]$ is connected.

Note that Proposition 9 cannot explicitly detect the connectivity of γ-QCP solutions made of at most 4 vertices. Nevertheless, Proposition 6.4.1 implicitly ensures the connectivity of all γ-quasi-cliques with at most 7 vertices on connected graphs. Indeed, any partition of these vertices into two (or more) disjointed subsets would have (at least) a connected component made by at most 3 vertices. Hence, Proposition 9 integrated with Lemma 6.4.1 dominates (6.36) for any value of $|Q|$.

Finally, even if we contextualized the discussion to the γ-QCP, any result independent by γ can be straightforwardly inherited for analyzing the connectivity of solutions related to other optimization problems defined on graph, such as problems within the family of MRCP for which solutions can be disconnected (e.g., k-core, k-bundle).

6.5 Computational results

We carried out an extensive experimental campaign that mainly aim to study the quality of combinatorial and LP based dual bounds as well as the effectiveness of the sufficient conditions to recognize connected γ-quasi-cliques.

The column generation has been coded in C++ and all the linear programs solved by IBM® CPLEX® 12.5.0.0 on a Intel® Core i7-7500U 2.70 GHz machine with 16Gb RAM. A CPU time limit of 7200 seconds has been used for all the experiments, and all the integer programs have been set with $K = \{k_L = 2, \dots, \bar{k}_U\}$ (see (5.18)). Solutions obtained by the greedy vertex elimination procedure of [123] have been used to measure the optimality gap (when optimal solutions were not available), and to evaluate the effectiveness of connectivity conditions.

Experiments have multiple purposes: in §6.5.1 the quality of the formulations $[C_\gamma]$, $[\bar{C}_\gamma]$, $[D^S_\gamma]$ and $[P_\gamma]$ has been evaluated with respect to the graph density d and parameter γ. In §6.5.2 the combinatorial dual bounds k_U and $\bar{k}_U$ have been compared to each other, whereas CPU times and tightness of the dual bounds provided by $[D^S_\gamma]$ have been reported in §6.5.3. Finally, the effectiveness of the sufficient connectivity conditions is discussed in §6.5.4.

Detailed numerical results are listed in the Appendix. In the following we illustrate the experiments by means of *performance profiles* [53]: given a performance indicator β of two algorithms and/or programs a and b, e.g. the optimality gap or the CPU running time, the performance profile of a plots the fraction of the number of instances (the ordinate) for which the ratio $\beta_a / \min\{\beta_a, \beta_b\}$ is less than or equal to a given threshold (the abscissa). For the sake of readability, the abscissa axis is in logarhitmic scale in all the following charts.

6.5.1 LP-based dual bounds: sensitivity analysis

We perform a sensitivity analysis of models in sections 6.2 and 6.3 by using six values of $\gamma = \{0.5, 0.6, 0.7, 0.8, 0.9, 0.95\}$ and 80 Erdös-Rényi uniform random graphs [56] with $|V| = 50$. The graphs are grouped into 4 classes R_p of 20 instances each, where $p \in \{0.2, 0.4, 0.6, 0.8\}$ indicates the mean density d, e.g., $R_{0.2}$ is the set of 20 random graphs whose mean density is $d = 0.2$. Let $|Q^*|$ be the optimal (or the best) integer solution provided by CPLEX and Q^U_β the dual bound computed by means of the formulation β, with $\beta \in \{[C_\gamma], [D^S_\gamma], [\bar{C}_\gamma], [P_\gamma]\}$. The performance is evaluated in terms of the percentage optimality gap

$$OG_\beta = 100 \cdot \frac{Q^U_\beta - |Q^*|}{Q^U_\beta}. \tag{6.38}$$

Generally speaking, the optimality gaps often reach high values and the dual bounds are quite weak (see Table 6.1 in the Appendix for details). However, the best one always dominates $\bar{k}_U$. The gaps decrease as the density of the graphs gets larger or the γ diminishes. In any case, the bound provided by $[P_\gamma]$ is always dominated by either $Q^U_{[C_\gamma]}$ or $Q^U_{[\bar{C}_\gamma]}$.

For each class of graphs and value of γ, Figure 6.4 shows the best percentage gap between $OG_{[C_\gamma]}$ and $OG_{[\bar{C}_\gamma]}$: white bars indicate that $OG_{[C_\gamma]}$ is better than $OG_{[\bar{C}_\gamma]}$, black bars otherwise.

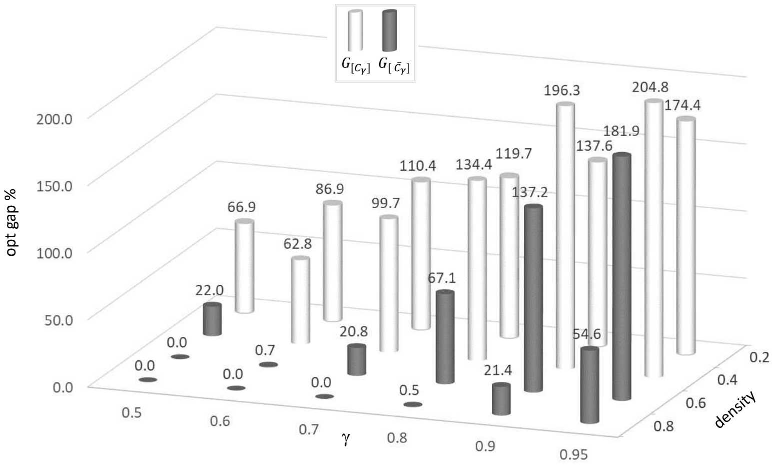

Figure 6.4: $\min\{OG_{[C_\gamma]}, OG_{[\bar{C}_\gamma]}\}$ for different γ and graph densities

$[C_\gamma]$ achieves the best results for $d \leq 0.4$ but one case ($\gamma = 0.5$ on class $R_{0.4}$). For larger densities, $[\bar{C}_\gamma]$ performs better in all the non-trivial cases, i.e., when the optimal solution corresponds to the trivial upper bound $|V|$. Formulations $[C_\gamma]$ and $[\bar{C}_\gamma]$ therefore looks to be complementary with respect to the graph density: $[C_\gamma]$ is suitable for computing dual bounds on sparse graphs, whereas $[\bar{C}_\gamma]$ is convenient on dense graphs. As a consequence of this combined behaviour random graphs with $d \leq 0.5$ appear the hardest to solve.

Looking at the formulation $[D^S_\gamma]$, the dual bound $Q^U_{[D^S_\gamma]}$ still continues to be slightly tighter than $Q^U_{[C_\gamma]}$ (see Table 6.1) though $[D^S_\gamma]$ is a surrogate relaxation of $[D_\gamma]$. On the other hand, the computational burden to solve the linear relaxation of $[D_\gamma]$ becomes much higher as the graph density increases, making $[D_\gamma]$ poorly competitive. As a final remark, the computation of all the dual bounds always required a negligible CPU time, given the small order of the considered random graphs.

6.5.2 Combinatorial dual bound comparison

The dual bounds k_U and $\bar{k}_U$ has been compared to each other on two groups of instances: the 16 benchmark sparse graphs used in [130], and the 64 benchmark DIMACS instances [114] for $\gamma = \{0.5, 0.7, 0.8, 0.9\}$ (see Table 6.2 and Table 6.3 in the Appendix). The graphs in the former set are very sparse (average density $d = 0.008$) and represent real-world networks in the fields of social networks, biology, telecommunications and transportation. Those in the latter set are denser graphs (average density $d = 0.621$) often used as benchmarks for clique problems.

The percentage optimality gap (6.38) has been computed by means of the best known lower bound $|Q^l|$, i.e., the maximum between an optimal solution value (if available) and the heuristic solution value.

Numerical results are reported in the Appendix, Tables 6.4 and 6.5. The performance profile on the quality of the two bounds is depicted in Figure 6.5. The cumulative distributions show that (*i*) $\bar{k}_U$ always dominates k_U (as expected), (*ii*) the weakness of k_U is more pronounced on sparse graphs for which the ratio $OG_{k_U}/OG_{\bar{k}_U}$ is always ≥ 2 and reaches peaks of $2^{5.9}$, and (*iii*) $OG_{\bar{k}_U}$ improves OG_{k_U} of at least 50% in roughly the 80% of DIMACS graphs.

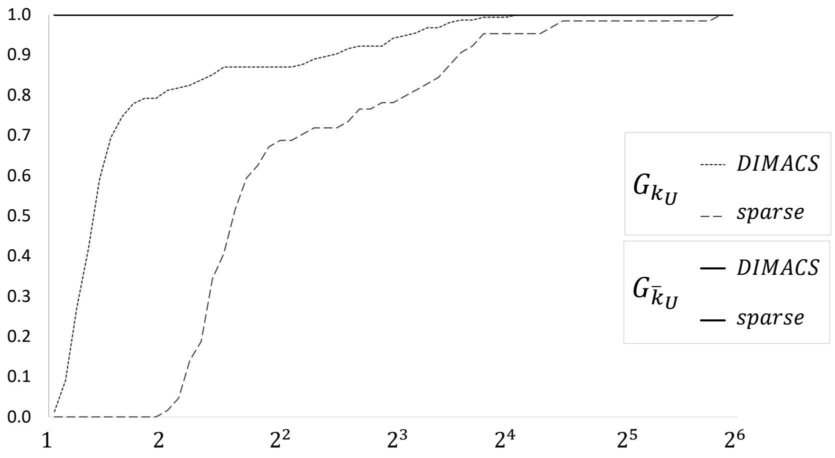

Figure 6.5: Performance profile of gaps OG_{k_U} and $OG_{\bar{k}_U}$

Besides trivial-instances and three additional cases with $\gamma = 0.5$, in which both OG_{k_U} and $OG_{\bar{k}_U}$ are $\leq 1.02\%$, the average optimality gaps always result considerably large, in particular for sparse graphs, and grow for increasing values of γ. The mean ratio $OG_{k_U}/OG_{\bar{k}_U}$ decreases as γ increases on DIMACS instances (from 3.97 for $\gamma = 0.5$ to 1.83 for $\gamma = 0.9$) and exhibits an opposite behaviour on sparse graphs (from 4.43 for $\gamma = 0.5$ to 8.96 for $\gamma = 0.9$).

As remarked in §6.2.2, the clique-based bound k_U^ω defined by (6.15) rapidly becomes weak on dense graphs. Indeed, it can be exploited only in two DIMACS instances, namely D_{35} and D_{38} with $\gamma = 0.9$, provided that we assume the best known lower bound $\omega_L(G)$ on the clique number $\omega(G)$ as the optimal clique number [114]. Under this hypothesis, the quality of k_U^ω appears largely better than $\bar{k}_U$, with an absolute reduction of 72 vertices in D_{35} and 104 vertices in D_{38}, respectively. Nevertheless, the optimality gap still remains large, i.e. 350% for the former instance and 710% for the latter. We point out that the bound k_U^ω is defined up to $\omega(G) = 9$ for $\gamma = 0.9$, and we used $\omega_L(G) = 8$ for D_{35} and $\omega_L(G) = 9$ for D_{35} that are very close to the upper limit. In all other DIMACS instances, the value of $\omega_L(G)$ is already sufficiently large to make k_U^ω unusable.

6.5.3 LP-based dual bound comparison

On the same set of instances (Table 6.2 and Table 6.3) we compared the linear relaxation of integer programs $[D_\gamma^S]$, $[C_\gamma]$ and $[\bar{C}_\gamma]$ for $\gamma = \{0.5, 0.7, 0.8, 0.9\}$. Numerical results on gaps and CPU times are reported in Tables 6.6 and 6.7 (sparse graphs) and Tables 6.8 and 6.11 (DIMACS graphs).

Sparse graphs Figures 6.6 and 6.7 depict the performance profiles of percentage optimality gaps and running times, respectively. The dual bound provided by $[C_\gamma]$ is not strictly dominated for all the values of γ. However, the quality of $[D_\gamma^S]$ is comparable to that of $[C_\gamma]$ since the mean ratio between gaps is 1.08, and in the 73.44% of the cases the bounds coincide. Moreover, $Q^U_{[D_\gamma^S]}$ improves on average the combinatorial bound $\bar{k}_U$ by 134.15% ($\gamma = 0.5$), 140.34% ($\gamma = 0.7$), 139.43% ($\gamma = 0.8$) and 138.63% ($\gamma = 0.9$).

We do not report the performance of $[P_\gamma]$ and $[\bar{C}_\gamma]$ because the former provides a dual bound always dominated by the other formulations, whereas the latter is either not able to provide a bound within the time limit (given its $O(|\bar{E}|)$ constraints) or $Q^U_{[\bar{C}_\gamma]}$ is dominated by both $Q^U_{[C_\gamma]}$ and $Q^U_{[D_\gamma^S]}$.

The computation of $Q^U_{[D_\gamma^S]}$ is faster than that of $Q^U_{[C_\gamma]}$ (up to 20%) in about the 60% of the 64 cases, whereas $Q^U_{[C_\gamma]}$ is obtained roughly 3.5 time faster in about the 6% of the cases: the whole CPU times to get the bounds for all the 64 cases are 1919.27 seconds for the former and 2062.90 seconds for the latter, with an overall gap of 7.48%. Varying the value of γ seems to not affect the computational time significantly and there is no evidence of correlation between the two measures.

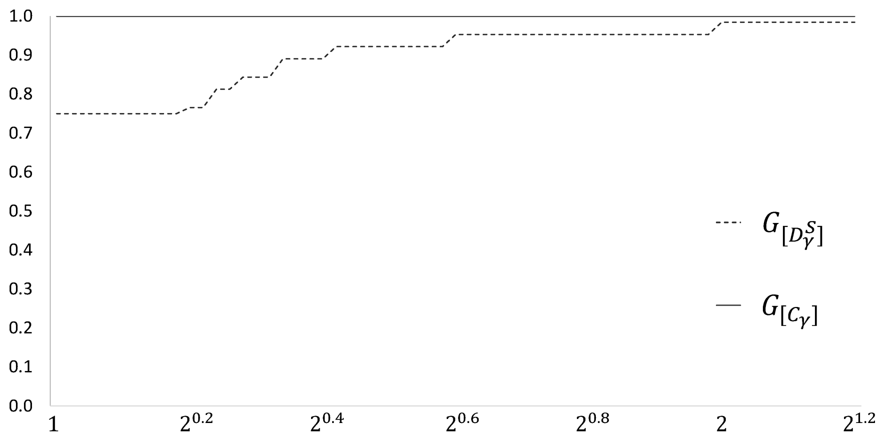

Figure 6.6: Performance profile of gaps $OG_{[D_\gamma^S]}$ and $OG_{[C_\gamma]}$ on sparse graphs.

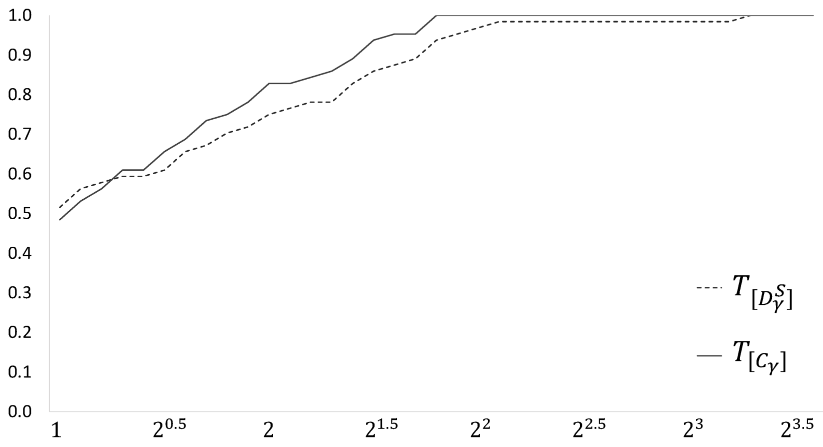

Figure 6.7: Performance profile of CPU running times for computing $Q^U_{[D_\gamma^S]}$ and $Q^U_{[C_\gamma]}$ on sparse graphs.

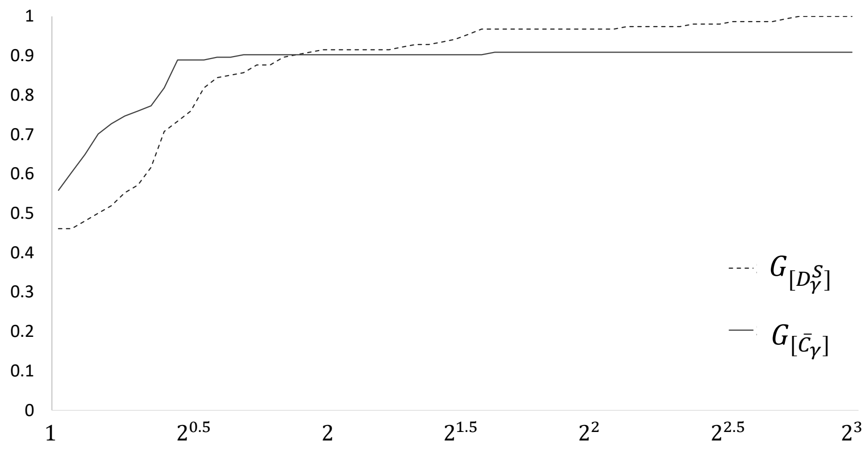

Figure 6.8: Performance profile of gaps $OG_{[D^S_\gamma]}$ and $OG_{[\bar{C}_\gamma]}$ on DIMACS graphs.

Dense graphs Figures 6.8 and 6.9 show the performance profiles (percentage optimality gaps and running times, respectively) of the formulations $[D^S_\gamma]$ and $[\bar{C}_\gamma]$. $Q^U_{[\bar{C}_\gamma]}$ is better than $Q^U_{[D^S_\gamma]}$ in 90.25% of the cases providing a smaller gap up to 86%. However, $[\bar{C}_\gamma]$ is not able to produce a valid bound within the time limit in 9.10% of the cases. In particular, on D_1-D_{12} instances $Q^U_{[\bar{C}_\gamma]}$ is, on average, 61.76% tighter than $Q^U_{[D^S_\gamma]}$ for $\gamma = 0.8$ and 79.79% for $\gamma = 0.9$, whereas it is 48.86% tighter for $\gamma = 0.9$ on the nontrivial instances of the subclass D_{51}-D_{60}. Instead, $Q^U_{[D^S_\gamma]}$ is generally better in groups D_{13}-D_{19} and D_{35}-D_{49}: in the former, $[D^S_\gamma]$ dominates $[\bar{C}_\gamma]$ for larger values of γ, whereas in the latter $[D^S_\gamma]$ is worse only for instances with high density value (d above 0.74) where however $[\bar{C}_\gamma]$ is not able to give a valid bound in 12 cases.

We do not report the performance profile of formulation $[C_\gamma]$ because when the solution of the continuous relaxation of $[C_\gamma]$ does not reach the time limit, the dual bounds $Q^U_{[C_\gamma]}$ and $Q^U_{[D^S_\gamma]}$ are always very close to each other. Indeed, $[C_\gamma]$ is not able to provide a valid bound in 31 of the 256 cases and, for the remaining instances, $[D^S_\gamma]$ is only slightly better with a 0.12% of mean gap. As a final remark on the quality of bounds, $Q^U_{[D^S_\gamma]}$ improves $\bar{k}_U$ by 7.85%, 7.74%, 5.79% and 6.37% for $\gamma = 0.5, 0.7, 0.8$ and 0.9, respectively.

The CPU time required for computing $Q^U_{[D^S_\gamma]}$ is always much smaller than that needed for obtaining $Q^U_{[\bar{C}_\gamma]}$: in 50% of the cases up to 16 times, in 8% of the cases up to 3 orders of magnitude. In particular, the column generation

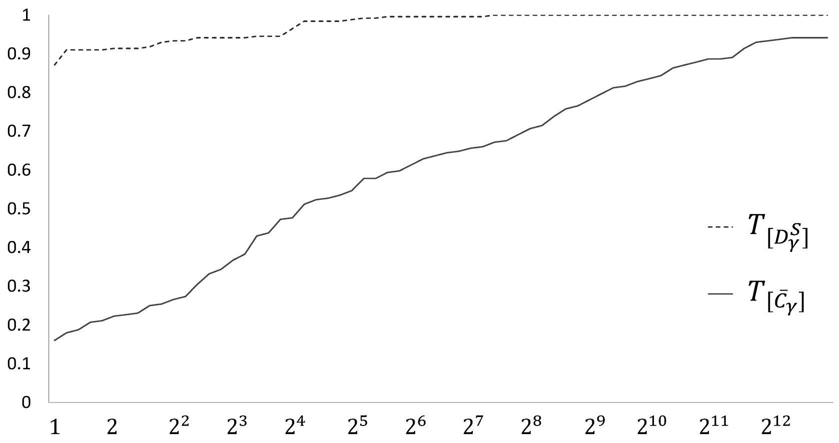

Figure 6.9: Performance profile of CPU running times for computing $Q^U_{[D^S_\gamma]}$ and $Q^U_{[\bar{C}_\gamma]}$ on DIMACS graphs.

procedure ran for 208.42 seconds, 0.81 seconds on average, to compute the $Q^U_{[D^S_\gamma]}$ bound of all the instances; CPLEX reached the CPU time limit 31 and 15 times to compute respectively $Q^U_{[C_\gamma]}$ and $Q^U_{[\bar{C}_\gamma]}$, and required 424.66 and 290.69 seconds on average to get the bound for the remaining instances. On the base of the cases in which time limit was reached, it appears that DIMACS instances become harder as γ increases. Looking more in detail, D_{26}-D_{29} is the only subgroups in which the time spent by $[\bar{C}_\gamma]$ has been roughly the same as $[D^S_\gamma]$ and the bound is better. For several cases of D_1-D_{12} and D_{51}-D_{60}, as example, the better dual bound provided by $[\bar{C}_\gamma]$ has been obtained by spending much more time.

Generally speaking, the above results show that $[\bar{C}_\gamma]$ is advantageous for computing bounds on quite dense graphs ($d > 0.4$), with a density threshold which slightly increases as γ grows, e.g., on instances D_{14} and D_{17}, $Q^U_{[\bar{C}_\gamma]}$ is the best bound for $\gamma = 0.5$ and then becomes dominated for greater values of γ. In any case, the employment of $[\bar{C}_\gamma]$ requires a significant amount of CPU time that rapidly increases with the size of the graph, appearing much less scalable if compared to $[D^S_\gamma]$.

6.5.4 Effectiveness of connectivity conditions

The γ-quasi-cliques found by the primal heuristic on the set of uniform random graphs are all connected. Inequality (6.36) certifies the connectivity of the solutions in 184 of 480 cases. Proposition 9 verifies additional 107 solutions, for a total of 291 positive occurrences. The numerical results show that both the sufficient conditions are weaker for small values of γ and large density of graphs. This seems reasonable because the order $|Q|$ of γ-quasi-cliques usually get larger on instances with such features, thus making sufficient conditions poorly effective. Indeed, Proposition 9 recognizes that the computed solution is connected only on the 28.13% of DIMACS instances, though the solutions obtained for this set of graphs are all connected for any value of γ. In particular, inequality (6.36) is fulfilled by 17 solutions, whereas Proposition 9 is verified in other 55 additional cases. On the other hand, 15 of the 64 solutions computed for the sparse graph instances are disconnected (actually, four of them can easily be made connected since they are composed by only two connected components, one of which consisting of at most 3 vertices) and Proposition 9 is able to recognize the 67.92% of the 53 connected (or easily connectable) γ-quasi-cliques found. In particular, 15 connected solutions fulfill the sufficient condition (6.36) and other 21 additional cases are certified by Proposition 9, generally for higher values of γ; e.g., all primal solutions are proved connected for $\gamma = 0.9$.

6.6 Conclusions and perspectives

In this chapter a new MIP reformulation $[D_\gamma]$ for the γ-QCP, obtained by decomposing star inequalities, has been presented. The bound provided by $[D_\gamma]$ is as good as that computed by the tightest formulation $[C_\gamma]$ reported in the literature and experiments show that also the surrogate relaxation $[D^S_\gamma]$ roughly provide the same bounds. However, $[D^S_\gamma]$ seems to be more scalable since the column generation procedure becomes much faster as the density of graphs increases. On dense graphs the best dual bound is provided by formulation $[\bar{C}_\gamma]$, that models the γ-QCP by recalling the concept of complementary $(1-\gamma)$-quasi-clique. Nevertheless, $[\bar{C}_\gamma]$ is computationally demanding for sufficiently large graphs and solving $[D^S_\gamma]$ by column generation remains a valid alternative, considering that the required CPU time is extremely limited.

Furthermore, the importance of taking into account the connectivity of MCRP solutions has been discussed. We presented a new sufficient condition to verify the connectivity of γ-quasi-cliques and, more in general, of solutions of graph optimization problems that lack of an explicit constraint of connectivity. For the γ-QCP, our result dominates the previous one reported in the literature and tests showed that it is also quite effective in practice. Indeed, it was able to prove the connectivity of primal solutions in the 49.87% of the total cases, whereas the benchmark condition was limited to the 27.00% of the instances.

Computational experiments highlighted that the γ-QCP is a challenging problem in practice and for several instances the integrality gap is very wide. As future work, firstly we aim to enhance time performance of $[D_\gamma]$ by exploring a dynamic strategy based on lazy constraints to lighten formulation, where edge-flow constraints (5.11) are checked on the fly to ensure feasibility. Then, we are interested at the implementation of polyhedral cuts (e.g., generalized neighborhood, matching, forest) to tighten formulations $[D_\gamma]$ and $[D^S_\gamma]$, embedding such inequalities within a dynamic framework. On this ground, we look at the design and implementation of a full *branch-and-cut-and-price* procedure able to solve challenging instances of γ-QCP to optimality.

6.7 Appendix

In this section we report the detailed numerical results discussed in Section 6.5. Table 6.1 lists the percentage optimality gaps obtained with the linear relaxations of the integer programs. Each entry is the average computed on the elements of one of the four considered classes of random graphs. Bold numbers indicate whenever a formulation strictly dominates all the others. Attributes of sparse and dense graphs (order, size and density) are indicated in Tables 6.2 and 6.3, respectively.

Tables 6.4 and 6.5 show the percentage optimality gaps OG_{k_U} and $OG_{\bar{k}_U}$ between the value $|Q^l|$ provided by the greedy vertex elimination heuristic and the dual bounds k_U and $\bar{k}_U$ respectively. A "-" mark indicates an instance for which the dual bounds are trivially equal to $|V|$ and the primal solution is optimal.

Percentage optimality gaps and CPU running times for solving the continuous relaxations of integer programs $[D^S_\gamma]$, $[C_\gamma]$ and $[\bar{C}_\gamma]$ are reported in Tables 6.6 and 6.7 (sparse graphs), and Tables 6.8-6.11 (DIMACS dense graphs); bold values refer to strictly better gaps or CPU times.

Table 6.1: percentage optimality gaps on random graphs

	$R_{0.2}$				$R_{0.4}$			
γ	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)	$OG_{[\bar{C}_\gamma]}$ (%)	$OG_{[P_\gamma]}$ (%)	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)	$OG_{[\bar{C}_\gamma]}$ (%)	$OG_{[P_\gamma]}$ (%)
0.5	66.91	**66.05**	97.50	141.38	26.63	26.60	**22.01**	26.90
0.6	86.92	**86.22**	123.92	203.69	62.82	**62.79**	63.60	66.61
0.7	110.36	**109.35**	157.11	279.57	99.66	**99.62**	110.46	112.51
0.8	119.65	**118.93**	169.10	335.80	134.41	**134.17**	154.58	160.82
0.9	137.61	**135.98**	193.50	415.98	196.31	**194.72**	222.65	245.88
0.95	174.43	**171.86**	232.25	518.15	204.80	**201.08**	230.78	264.36
	$R_{0.6}$				$R_{0.8}$			
0.5	0.00	0.00	0.0	0.00	0.00	0.00	0.0	0.00
0.6	1.75	1.75	**0.65**	1.82	0.00	0.00	0.0	0.00
0.7	38.73	38.73	**20.79**	29.61	0.00	0.00	0.0	0.00
0.8	97.55	97.53	**67.13**	75.14	1.28	1.28	**0.45**	1.40
0.9	183.67	182.66	**137.23**	143.51	61.76	61.70	**21.35**	33.71
0.95	238.74	234.95	**181.89**	186.83	124.47	123.26	**54.56**	65.72

Table 6.2: Attributes of sparse graphs

ID	Name	$\|V\|$	$\|E\|$	d
I_1	USAir97	332	2126	0.0387
I_2	Harvard500	500	2043	0.0164
I_3	Email	1133	5451	0.0085
I_4	Homer	561	1628	0.0104
I_5	SmallW	396	994	0.0127
I_6	Erdos971	472	1314	0.0118
I_7	Netscience	1589	2742	0.0022
I_8	C.Elegans	453	2025	0.0198
I_9	Erdos02	6927	8472	0.0004
I_{10}	Geom	7343	11898	0.0004
I_{11}	ca-HepTh	9877	25973	0.0005
I_{12}	ca-GrQc	5242	14484	0.0011
I_{13}	AS-735	7716	12572	0.0004
I_{14}	PGPgiantcompo	10680	24316	0.0004
I_{15}	EVA	8497	6711	0.0002
I_{16}	California	9664	15969	0.0003

Table 6.3: Attributes of DIMACS graphs

ID	Name	$\|V\|$	$\|E\|$	d	ID	Name	$\|V\|$	$\|E\|$	d
D_1	brock200-1	200	14834	0.75	D_{33}	MANN-a45	1035	533115	1.00
D_2	brock200-2	200	9876	0.50	D_{34}	MANN-a9	45	918	0.93
D_3	brock200-3	200	12048	0.61	D_{35}	p-hat300-1	300	10933	0.24
D_4	brock200-4	200	13089	0.66	D_{36}	p-hat300-2	300	21928	0.49
D_5	brock400-1	400	59723	0.75	D_{37}	p-hat300-3	300	33390	0.74
D_6	brock400-2	400	59786	0.75	D_{38}	p-hat500-1	500	31569	0.25
D_7	brock400-3	400	59681	0.75	D_{39}	p-hat500-2	500	62946	0.50
D_8	brock400-4	400	59765	0.75	D_{40}	p-hat500-3	500	93800	0.75
D_9	brock800-1	800	207505	0.65	D_{41}	p-hat700-1	700	60999	0.25
D_{10}	brock800-2	800	208166	0.65	D_{42}	p-hat700-2	700	121728	0.50
D_{11}	brock800-3	800	207333	0.65	D_{43}	p-hat700-3	700	183010	0.75
D_{12}	brock800-4	800	207643	0.65	D_{44}	p-hat1000-1	1000	122253	0.24
D_{13}	c-fat200-1	200	1534	0.08	D_{45}	p-hat1000-2	1000	244799	0.49
D_{14}	c-fat200-2	200	3235	0.16	D_{46}	p-hat1000-3	1000	371746	0.74
D_{15}	c-fat200-5	200	8473	0.43	D_{47}	p-hat1500-1	1500	284923	0.25
D_{16}	c-fat500-1	500	4459	0.04	D_{48}	p-hat1500-2	1500	568960	0.51
D_{17}	c-fat500-10	500	46627	0.37	D_{49}	p-hat1500-3	1500	847244	0.75
D_{18}	c-fat500-2	500	9139	0.07	D_{50}	san1000	1000	250500	0.50
D_{19}	c-fat500-5	500	23191	0.19	D_{51}	san200-0.7-1	200	13930	0.70
D_{20}	hamming10-2	1024	518656	0.99	D_{52}	san200-0.7-2	200	13930	0.70
D_{21}	hamming10-4	1024	434176	0.83	D_{53}	san200-0.9-1	200	17910	0.90
D_{22}	hamming6-2	64	1824	0.90	D_{54}	san200-0.9-2	200	17910	0.90
D_{23}	hamming6-4	64	704	0.35	D_{55}	san200-0.9-3	200	17910	0.90
D_{24}	hamming8-2	256	31616	0.97	D_{56}	san400-0.5-1	400	39900	0.50
D_{25}	hamming8-4	256	20864	0.64	D_{57}	san400-0.7-1	400	55860	0.70
D_{26}	johnson16-2-4	120	5460	0.76	D_{58}	san400-0.7-2	400	55860	0.70
D_{27}	johnson32-2-4	496	107880	0.88	D_{59}	san400-0.7-3	400	55860	0.70
D_{28}	johnson8-2-4	28	210	0.56	D_{60}	san400-0.9-1	400	71820	0.90
D_{29}	johnson8-4-4	70	1855	0.77	D_{61}	sanr200-0.7	200	13868	0.70
D_{30}	keller4	171	9435	0.65	D_{62}	sanr200-0.9	200	17863	0.90
D_{31}	keller5	776	225990	0.75	D_{63}	sanr400-0.5	400	39984	0.50
D_{32}	MANN-a27	378	70551	0.99	D_{64}	sanr400-0.7	400	55869	0.70

Table 6.4: percentage optimality gaps obtained with the dual bounds k_U and $\bar{k}_U$

	$\gamma = 0.5$		$\gamma = 0.7$		$\gamma = 0.8$		$\gamma = 0.9$	
ID	OG_{k_U} (%)	$OG_{\bar{k}_U}$ (%)	OG_{k_U} (%)	$OG_{\bar{k}_U}$ (%)	OG_{k_U} (%)	$OG_{\bar{k}_U}$ (%)	OG_{k_U} (%)	$OG_{\bar{k}_U}$ (%)
D_1	-	-	-	-	73.87	68.47	333.33	297.62
D_2	1.53	1.02	354.05	294.59	582.61	460.87	1133.33	858.33
D_3	-	-	129.63	116.05	370.27	316.22	680.95	557.14
D_4	-	-	47.33	44.27	262.00	232.00	677.27	577.27
D_5	-	-	-	-	109.78	103.80	613.73	556.86
D_6	-	-	-	-	116.20	109.50	574.07	520.37
D_7	-	-	-	-	112.09	106.04	586.79	532.08
D_8	-	-	-	-	109.19	102.70	574.07	520.37
D_9	-	-	150.81	142.35	800.00	717.50	2090.32	1780.65
D_{10}	-	-	139.44	131.99	768.67	691.57	2093.55	1790.32
D_{11}	-	-	154.97	146.36	767.47	687.95	2090.32	1780.65
D_{12}	-	-	154.13	145.87	847.37	761.84	2241.38	1913.79
D_{13}	160.00	16.67	247.37	31.58	287.50	37.50	346.15	46.15
D_{14}	96.55	15.52	152.63	26.32	181.25	31.25	226.92	46.15
D_{15}	24.32	15.54	62.50	28.13	84.81	35.44	110.77	47.69
D_{16}	282.86	17.14	413.64	31.82	488.89	44.44	566.67	53.33
D_{17}	33.33	15.74	74.64	28.23	97.11	35.84	128.37	48.23
D_{18}	189.39	16.67	285.71	30.95	331.43	37.14	393.10	48.28
D_{19}	85.98	15.24	142.45	27.36	177.01	36.78	219.72	49.30
D_{20}	-	-	-	-	-	-	-	-
D_{21}	-	-	-	-	-	-	84.59	77.26
D_{22}	-	-	-	-	-	-	-	-
D_{23}	65.63	40.63	462.50	300.00	740.00	460.00	900.00	525.00
D_{24}	-	-	-	-	-	-	-	-
D_{25}	-	-	57.42	50.32	221.13	187.32	923.81	766.67
D_{26}	-	-	-	-	244.12	235.29	1000.00	920.00
D_{27}	-	-	-	-	-	-	590.14	581.69
D_{28}	-	-	212.50	175.00	360.00	280.00	450.00	325.00
D_{29}	-	-	-	-	100.00	97.06	540.00	490.00
D_{30}	-	-	41.38	37.07	227.66	200.00	752.94	641.18
D_{31}	-	-	-	-	59.32	54.66	516.52	466.96
D_{32}	-	-	-	-	-	-	-	-
D_{33}	-	-	-	-	-	-	-	-
D_{34}	-	-	-	-	-	-	-	-
D_{35}	231.75	182.54	785.00	580.00	1169.23	830.77	1850.00	1250.00
D_{36}	1.02	0.68	42.86	33.14	105.26	84.21	268.33	211.67
D_{37}	-	-	-	-	21.94	18.99	122.95	108.20
D_{38}	277.66	223.40	1150.00	862.50	1552.94	1111.76	2550.00	1750.00
D_{39}	-	-	37.22	28.48	88.15	70.14	240.00	190.91
D_{40}	-	-	-	-	17.76	15.33	107.73	94.09
D_{41}	329.57	266.96	1337.93	1003.45	2343.75	1681.25	3988.89	2744.44
D_{42}	0.29	0.14	37.21	28.37	91.67	73.26	220.99	173.46
D_{43}	-	-	-	-	20.07	17.41	115.54	101.01
D_{44}	402.88	326.62	1746.88	1306.25	2810.53	2010.53	4636.36	3172.73
D_{45}	1.02	0.61	42.91	33.16	103.12	82.60	284.38	226.04
D_{46}	-	-	-	-	23.12	20.05	131.30	115.52
D_{47}	396.74	324.19	2212.82	1674.36	3736.36	2704.55	5585.71	3864.29
D_{48}	-	-	36.22	27.67	85.83	68.22	231.56	183.48
D_{49}	-	-	-	-	18.29	15.85	111.08	97.54

Table 6.5: percentage optimality gaps obtained with the dual bounds k_U and $\bar{k}_U$

	$\gamma = 0.5$		$\gamma = 0.7$		$\gamma = 0.8$		$\gamma = 0.9$	
ID	OG_{k_U} (%)	$OG_{\bar{k}_U}$ (%)	OG_{k_U} (%)	$OG_{\bar{k}_U}$ (%)	OG_{k_U} (%)	$OG_{\bar{k}_U}$ (%)	OG_{k_U} (%)	$OG_{\bar{k}_U}$ (%)
D_{50}	-	-	36.89	18.77	40.75	15.48	2094.12	1611.76
D_{51}	-	-	-	-	41.67	33.33	58.56	41.44
D_{52}	-	-	-	-	16.88	11.88	27.54	18.12
D_{53}	-	-	-	-	-	-	-	-
D_{54}	-	-	-	-	-	-	-	-
D_{55}	-	-	-	-	-	-	-	-
D_{56}	-	-	36.29	18.55	41.07	16.07	1319.05	1009.52
D_{57}	-	-	-	-	49.60	40.80	66.82	48.82
D_{58}	-	-	-	-	46.67	38.04	63.72	46.05
D_{59}	-	-	-	-	41.13	33.21	58.56	41.89
D_{60}	-	-	-	-	-	-	-	-
D_{61}	-	-	1.53	1.53	165.71	151.43	433.33	375.76
D_{62}	-	-	-	-	-	-	1.53	1.53
D_{63}	-	-	576.00	486.00	1164.00	932.00	2192.31	1669.23
D_{64}	-	-	-	-	252.83	232.08	802.56	707.69
I_1	37.31	14.93	59.18	22.45	69.77	23.26	97.14	31.43
I_2	143.24	40.54	192.31	42.31	195.83	33.33	191.30	21.74
I_3	492.00	184.00	792.86	278.57	800.00	261.54	746.15	223.08
I_4	145.45	69.70	277.78	138.89	326.67	153.33	328.57	128.57
I_5	125.00	57.14	211.76	94.12	257.14	107.14	327.27	127.27
I_6	217.39	95.65	369.23	169.23	470.00	210.00	575.00	250.00
I_7	238.71	22.58	270.83	20.83	277.27	13.64	271.43	4.76
I_8	200.00	83.33	322.22	127.78	373.33	140.00	458.33	158.33
I_9	922.22	361.11	1318.18	472.73	1522.22	522.22	1612.50	525.00
I_{10}	395.45	109.09	607.69	169.23	616.67	158.33	608.70	130.43
I_{11}	475.00	44.64	615.79	60.53	628.57	54.29	627.27	45.45
I_{12}	197.53	16.05	269.09	29.09	272.55	23.53	265.31	12.24
I_{13}	522.22	202.78	691.67	241.67	831.58	273.68	1013.33	313.33
I_{14}	327.40	53.42	438.78	71.43	448.89	64.44	452.38	54.76
I_{15}	1722.22	766.67	2660.00	1080.00	3150.00	1200.00	2950.00	1025.00
I_{16}	873.08	342.31	1683.33	616.67	1566.67	533.33	1466.67	450.00

Table 6.6: percentage optimality gaps on sparse graphs

	$\gamma = 0.5$			$\gamma = 0.7$			$\gamma = 0.8$			$\gamma = 0.9$		
ID	$\|Q^I\|$	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)	$\|Q^I\|$	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)	$\|Q^I\|$	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)	$\|Q^I\|$	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)
I_1	67	0.00	0.00	49	0.00	0.00	43	0.00	0.00	35	**8.57**	11.43
I_2	37	10.81	10.81	26	15.38	15.38	24	8.33	8.33	23	4.35	4.35
I_3	25	28	28	14	64.29	64.29	13	53.85	53.85	13	38.46	38.46
I_4	33	**3.03**	6.06	18	**33.33**	38.89	15	**40.00**	46.67	14	35.71	35.71
I_5	28	**3.57**	7.14	17	23.53	23.53	14	**28.57**	35.71	11	**45.45**	54.55
I_6	23	13.04	13.04	13	46.15	46.15	10	**60.00**	70.00	8	87.5	87.5
I_7	31	0.00	0.00	24	4.17	4.17	22	4.55	4.55	21	0.00	0.00
I_8	30	3.33	3.33	18	22.22	22.22	15	**26.67**	33.33	12	**41.67**	50.00
I_9	18	**11.11**	16.67	11	36.36	36.36	9	**44.44**	55.56	8	**37.50**	50.00
I_{10}	44	**0.00**	2.27	26	23.08	23.08	24	16.67	16.67	23	8.7	8.7
I_{11}	56	0.00	0.00	38	10.53	10.53	35	8.57	8.57	33	3.03	3.03
I_{12}	81	2.47	2.47	55	12.73	12.73	51	7.84	7.84	49	2.04	2.04
I_{13}	36	0.00	0.00	24	8.33	8.33	19	21.05	21.05	15	33.33	33.33
I_{14}	73	**2.74**	4.11	49	12.24	12.24	45	6.67	6.67	42	2.38	2.38
I_{15}	9	0.00	0.00	5	40.00	40.00	4	50.00	50.00	4	25.00	25.00
I_{16}	26	34.62	34.62	12	108.33	108.33	12	83.33	83.33	12	**58.33**	66.67

Table 6.7: CPU times on sparse graphs (in sec.)

	$\gamma = 0.5$		$\gamma = 0.7$		$\gamma = 0.8$		$\gamma = 0.9$	
ID	$T_{[C_\gamma]}$	$T_{[D^S_\gamma]}$	$T_{[C_\gamma]}$	$T_{[D^S_\gamma]}$	$T_{[C_\gamma]}$	$T_{[D^S_\gamma]}$	$T_{[C_\gamma]}$	$T_{[D^S_\gamma]}$
I_1	0.45	**0.39**	0.72	**0.36**	**0.83**	1.2	0.88	**0.27**
I_2	1.06	**0.31**	0.98	**0.64**	1.53	**0.52**	1.11	**0.33**
I_3	11.48	**8.41**	12.13	**6.55**	11.33	**8.28**	12.02	**6.22**
I_4	0.66	**0.64**	**0.59**	0.89	0.70	**0.61**	0.83	**0.55**
I_5	**0.09**	0.23	0.23	**0.22**	0.31	**0.14**	0.25	**0.09**
I_6	**0.17**	1.61	**0.30**	1.11	**0.22**	0.92	**0.17**	0.59
I_7	**0.97**	1.11	**0.95**	1.02	1.30	**0.77**	**1.00**	1.19
I_8	0.95	**0.59**	1.05	**0.56**	0.80	**0.56**	0.94	**0.36**
I_9	5.98	**2.28**	**6.38**	17.11	**6.94**	15.86	**6.59**	16.27
I_{10}	**11.17**	30.69	**10.11**	36.36	**9.69**	31.7	**9.55**	31.36
I_{11}	225.55	**210.61**	291.54	**180.00**	278.91	**253.8**	348.85	**263.52**
I_{12}	27.28	**22.39**	21.67	**20.84**	**18.88**	19.59	**21.95**	22.70
I_{13}	**8.55**	12.55	**7.49**	11.95	**7.47**	10.20	**7.72**	13.09
I_{14}	**62.99**	123.19	**56.25**	117.36	**65.65**	110.98	**60.97**	115.92
I_{15}	**1.75**	3.22	**1.17**	3.39	**1.19**	3.08	**1.11**	3.36
I_{16}	100.61	**43.09**	112.52	**42.09**	79.42	**40.89**	120.03	**42.58**

Table 6.8: percentage optimality gaps on DIMACS graphs

	$\gamma = 0.5$				$\gamma = 0.7$				$\gamma = 0.8$				$\gamma = 0.9$			
ID	$\lvert Q^I \rvert$	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)	$OG_{[\bar{C}_\gamma]}$ (%)	$\lvert Q^I \rvert$	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)	$OG_{[\bar{C}_\gamma]}$ (%)	$\lvert Q^I \rvert$	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)	$OG_{[\bar{C}_\gamma]}$ (%)	$\lvert Q^I \rvert$	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)	$OG_{[\bar{C}_\gamma]}$ (%)
D_1	200	0.00	0.00	0.00	200	0.00	0.00	0.00	111	67.95	67.95	**48.25**	42	294.81	294.80	**194.65**
D_2	196	1.29	1.02	**0.76**	37	283.99	284.00	**252.74**	23	441.07	441.07	**405.87**	12	822.77	822.68	**790.74**
D_3	200	0.00	0.00	0.00	81	113.71	113.71	**90.83**	37	309.72	309.72	**244.22**	21	542.22	542.18	**428.70**
D_4	200	0.00	0.00	0.00	131	43.50	43.50	**33.97**	50	229.22	229.22	**173.28**	22	565.60	565.58	**421.42**
D_5	400	0.00	0.00	0.00	400	0.00	0.00	0.00	184	103.41	103.41	**78.68**	51	552.54	552.53	**381.89**
D_6	400	0.00	0.00	0.00	400	0.00	0.00	0.00	179	109.31	109.31	**83.64**	54	516.93	516.92	**355.36**
D_7	400	0.00	0.00	0.00	400	0.00	0.00	0.00	182	105.50	105.50	**80.34**	53	527.47	527.46	**363.22**
D_8	400	0.00	0.00	0.00	400	0.00	0.00	0.00	185	102.45	102.45	**77.76**	54	516.72	516.71	**355.48**
D_9	800	0.00	0.00	0.00	307	141.72	141.72	**119.83**	80	-	711.82	**562.65**	31	-	1762.58	**1353.49**
D_{10}	800	0.00	0.00	0.00	322	131.20	131.20	**111.16**	83	-	684.96	**541.40**	31	-	1768.51	**1356.53**
D_{11}	800	0.00	0.00	0.00	302	145.52	145.52	**123.39**	83	-	681.82	**538.86**	31	-	1761.04	**1354.19**
D_{12}	800	0.00	0.00	0.00	303	145.08	145.08	**123.29**	76	-	755.11	**598.65**	29	-	1892.36	**1454.60**
D_{13}	30	11.31	**11.24**	16.67	19	26.58	26.58	31.58	16	32.73	**32.72**	37.5	13	46.15	**46.06**	46.15
D_{14}	58	13.27	**13.23**	15.52	38	24.19	24.19	26.32	32	29.46	29.46	31.25	26	42.09	**41.96**	46.15
D_{15}	148	15.18	15.17	**8.09**	96	27.12	27.12	24.34	79	35.33	35.33	35.44	65	46.37	**46.36**	47.69
D_{16}	35	13.13	**13.06**	17.14	22	29.56	29.56	31.82	18	39.56	39.56	44.44	15	49.56	**49.44**	53.33
D_{17}	324	15.44	15.44	**10.32**	209	27.96	27.96	28.23	173	35.34	35.34	35.84	141	47.68	47.68	48.23
D_{18}	66	12.80	**12.76**	16.67	42	27.24	27.24	30.95	35	33.99	33.99	37.14	29	44.15	**44.13**	48.28
D_{19}	164	13.74	**13.73**	15.24	106	25.95	25.95	27.36	87	34.43	34.43	36.78	71	46.58	**46.56**	49.30
D_{20}	1024	0.00	0.00	0.00	1024	0.00	0.00	0.00	1024	0.00	0.00	0.00	1024	0.00	0.00	0.00
D_{21}	1024	0.00	0.00	0.00	1024	0.00	0.00	0.00	1024	0.00	0.00	0.00	532	-	77.26	**31.66**
D_{22}	64	0.00	0.00	0.00	64	0.00	0.00	0.00	64	0.00	0.00	0.00	64	0.00	0.00	0.00
D_{23}	32	40.63	40.63	**36.63**	8	300.00	300.00	300.00	5	460.00	460.00	460.00	4	535.87	525.00	525.00
D_{24}	256	0.00	0.00	0.00	256	0.00	0.00	0.00	256	0.00	0.00	0.00	256	0.00	0.00	0.00
D_{25}	256	0.00	0.00	0.00	155	50.32	50.32	**32.77**	71	187.32	187.32	**131.24**	21	767.20	766.67	**575.90**
D_{26}	120	0.00	0.00	0.00	120	0.00	0.00	0.00	34	235.29	235.29	**195.61**	10	921.10	920.00	**631.37**
D_{27}	496	0.00	0.00	0.00	496	0.00	0.00	0.00	496	0.00	0.00	0.00	71	-	581.69	**484.44**
D_{28}	28	0.00	0.00	0.00	8	175.00	175.00	**136.27**	5	280.00	280.00	**228.57**	4	341.35	325.00	**273.60**
D_{29}	70	0.00	0.00	0.00	70	0.00	0.00	0.00	34	97.06	97.06	**74.93**	10	498.87	490.00	326.80
D_{30}	171	0.00	0.00	0.00	116	36.75	36.75	**26.70**	47	195.60	195.60	**148.38**	17	627.12	627.07	**481.76**
D_{31}	776	0.00	0.00	0.00	776	0.00	0.00	0.00	472	-	54.46	**36.62**	115	-	463.62	**320.96**
D_{32}	378	0.00	0.00	0.00	378	0.00	0.00	0.00	378	0.00	0.00	0.00	378	0.00	0.00	0.00
D_{33}	1035	0.00	0.00	0.00	1035	0.00	0.00	0.00	1035	0.00	0.00	0.00	1035	0.00	0.00	0.00
D_{34}	45	0.00	0.00	0.00	45	0.00	0.00	0.00	45	0.00	0.00	0.00	45	0.00	0.00	0.00

Table 6.9: percentage optimality gaps on DIMACS graphs

	$\gamma = 0.5$				$\gamma = 0.7$				$\gamma = 0.8$				$\gamma = 0.9$			
ID	$\lvert Q^l \rvert$	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)	$OG_{[\bar{C}_\gamma]}$ (%)	$\lvert Q^l \rvert$	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)	$OG_{[\bar{C}_\gamma]}$ (%)	$\lvert Q^l \rvert$	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)	$OG_{[\bar{C}_\gamma]}$ (%)	$\lvert Q^l \rvert$	$OG_{[C_\gamma]}$ (%)	$OG_{[D^S_\gamma]}$ (%)	$OG_{[\bar{C}_\gamma]}$ (%)
D_{35}	63	**134.29**	134.62	182.54	20	**428.52**	429.25	580.00	13	612.39	**612.23**	830.77	8	930.53	**928.99**	1250.00
D_{36}	293	**0.27**	0.30	0.48	175	**20.08**	20.18	24.44	114	**61.40**	61.53	67.02	60	172.78	**172.24**	179.95
D_{37}	300	0.00	0.00	0.00	300	0.00	0.00	0.00	237	17.83	17.83	**10.09**	122	103.55	103.31	**63.42**
D_{38}	94	**170.06**	170.27	223.40	24	**656.70**	657.55	862.50	17	**835.49**	835.71	1111.76	10	**1314.75**	1315.29	1750.00
D_{39}	500	0.00	0.00	0.00	309	**16.82**	16.92	20.19	211	**49.75**	49.87	53.04	110	155.43	**154.76**	156.72
D_{40}	500	0.00	0.00	0.00	500	0.00	0.00	0.00	411	14.36	14.36	**7.74**	220	89.95	**89.63**	53.59
D_{41}	115	**204.43**	204.66	266.96	29	**763.27**	763.94	1003.45	16	1269.89	**1269.82**	1681.25	9	**2066.02**	2066.68	2744.44
D_{42}	696	0.12	**0.11**	0.13	430	**15.84**	15.99	19.98	288	60.28	**51.58**	56.43	162	139.28	**138.95**	143.94
D_{43}	700	0.00	0.00	0.00	700	0.00	0.00	0.00	563	16.27	16.27	**9.11**	296	-	96.32	**58.76**
D_{44}	139	**252.98**	253.25	-	32	**996.08**	996.89	-	19	**1515.94**	1516.25	2010.53	11	**2382.06**	2382.23	3172.73
D_{45}	980	-	**0.11**	0.34	585	-	**19.91**	24.55	385	-	**59.44**	-	192	-	**183.47**	-
D_{46}	1000	0.00	0.00	0.00	1000	0.00	0.00	0.00	783	-	18.82	**10.79**	393	-	110.22	**69.30**
D_{47}	215	-	**254.42**	-	39	-	**1296.10**	-	22	-	**2064.86**	2704.55	14	-	**2924.86**	3864.29
D_{48}	1500	0.00	0.00	-	936	-	**16.03**	-	642	-	**48.03**	-	339	-	**148.42**	-
D_{49}	1500	0.00	0.00	0.00	1500	0.00	0.00	0.00	1230	-	**14.88**	-	650	-	**92.97**	-
D_{50}	1000	0.00	0.00	0.00	618	-	**15.97**	-	562	-	**11.61**	-	34	-	**1540.19**	-
D_{51}	200	0.00	0.00	0.00	200	0.00	0.00	0.00	132	32.67	32.67	**11.09**	111	40.34	40.34	**5.85**
D_{52}	200	0.00	0.00	0.00	200	0.00	0.00	0.00	160	9.45	9.45	**3.27**	138	12.88	12.87	**2.26**
D_{53}	200	0.00	0.00	0.00	200	0.00	0.00	0.00	200	0.00	0.00	0.00	200	0.00	0.00	0.00
D_{54}	200	0.00	0.00	0.00	200	0.00	0.00	0.00	200	0.00	0.00	0.00	200	0.00	0.00	0.00
D_{55}	200	0.00	0.00	0.00	200	0.00	0.00	0.00	200	0.00	0.00	0.00	200	0.00	0.00	0.00
D_{56}	400	0.00	0.00	0.00	248	15.32	15.32	**5.71**	224	11.77	11.77	**3.99**	21	960.31	960.30	**917.76**
D_{57}	400	0.00	0.00	0.00	400	0.00	0.00	0.00	250	40.05	40.05	**15.92**	211	47.55	47.55	**10.79**
D_{58}	400	0.00	0.00	0.00	400	0.00	0.00	0.00	255	37.30	37.30	**13.65**	215	44.81	44.80	**8.64**
D_{59}	400	0.00	0.00	0.00	400	0.00	0.00	0.00	265	32.12	32.12	**11.18**	222	40.24	40.24	**6.01**
D_{60}	400	0.00	0.00	0.00	400	0.00	0.00	0.00	400	0.00	0.00	0.00	400	0.00	0.00	0.00
D_{61}	200	0.00	0.00	0.00	196	1.53	1.53	**1.07**	70	149.07	149.07	**109.07**	33	369.96	369.88	**256.77**
D_{62}	200	0.00	0.00	0.00	200	0.00	0.00	0.00	200	0.00	0.00	0.00	196	1.53	1.53	**0.78**
D_{63}	400	0.00	0.00	0.00	50	473.19	473.19	**422.24**	25	903.60	903.60	**830.65**	13	1616.41	1616.24	**1544.51**
D_{64}	400	0.00	0.00	0.00	400	0.00	0.00	0.00	106	230.36	230.36	**175.33**	39	698.42	698.41	**501.94**

Table 6.10: CPU times on DIMACS graphs (in sec.)

	$\gamma = 0.5$			$\gamma = 0.7$			$\gamma = 0.8$			$\gamma = 0.9$		
ID	$T_{[C_\gamma]}$	$T_{[D^S_\gamma]}$	$T_{[\bar{C}_\gamma]}$	$T_{[C_\gamma]}$	$T_{[D^S_\gamma]}$	$T_{[\bar{C}_\gamma]}$	$T_{[C_\gamma]}$	$T_{[D^S_\gamma]}$	$T_{[\bar{C}_\gamma]}$	$T_{[C_\gamma]}$	$T_{[D^S_\gamma]}$	$T_{[\bar{C}_\gamma]}$
D_1	3.35	$<$ **0.005**	0.03	0.39	$<$ **0.005**	0.03	14.71	**0.06**	0.09	28.92	**0.05**	0.48
D_2	21.51	**0.08**	0.11	3.26	**0.05**	9.78	6.86	**0.05**	10.11	11.22	**0.05**	11.03
D_3	2.57	**0.02**	0.08	4.54	**0.03**	0.61	14.24	**0.03**	0.56	19.06	**0.03**	1.03
D_4	3.31	**0.02**	0.06	4.01	**0.06**	0.11	13.63	**0.05**	0.37	20.98	**0.03**	0.42
D_5	21.33	**0.02**	0.28	3.53	**0.03**	0.33	398.93	**0.13**	24.80	655.91	**0.19**	58.59
D_6	34.20	**0.02**	0.31	3.53	**0.03**	0.30	366.06	**0.13**	24.21	264.52	**0.19**	67.08
D_7	32.31	**0.03**	0.30	3.53	**0.03**	0.31	369.46	**0.14**	22.68	1029.45	**0.20**	46.15
D_8	32.78	**0.03**	0.31	3.67	**0.02**	0.31	324.50	**0.13**	36.60	385.29	**0.20**	66.5
D_9	464.48	**0.09**	5.69	5172.18	**0.81**	2782.22	–	**0.89**	4565.12	–	**0.84**	3222.22
D_{10}	428.80	**0.08**	5.62	4560.08	**0.80**	2328.41	–	**0.86**	2987.82	–	**0.91**	4226.71
D_{11}	419.67	**0.08**	5.77	5342.64	**0.75**	2465.00	–	**1.03**	3075.39	–	**0.94**	2893.40
D_{12}	483.92	**0.08**	5.83	5085.10	**0.77**	967.21	–	**1.00**	3012.10	–	**1.02**	2930.90
D_{13}	1.44	**0.03**	0.05	0.22	**0.03**	0.09	0.64	**0.02**	0.08	1.76	**0.02**	0.09
D_{14}	6.24	**0.02**	0.09	0.84	**0.02**	0.09	7.52	**0.02**	0.05	6.02	**0.02**	0.08
D_{15}	37.10	**0.05**	5.97	5.41	**0.05**	11.39	22.18	**0.05**	3.37	37.47	**0.03**	0.14
D_{16}	10.59	**0.11**	0.51	1.44	**0.11**	0.58	1.86	**0.11**	1.01	6.15	**0.09**	0.66
D_{17}	684.63	**0.23**	706.45	128.34	**0.19**	179.45	748.23	**0.17**	1.50	530.97	**0.22**	1.22
D_{18}	39.30	**0.08**	0.73	7.04	**0.08**	0.56	28.67	**0.08**	0.56	30.89	**0.06**	0.50
D_{19}	103.20	**0.17**	1.29	14.68	**0.14**	0.94	344.14	**0.13**	0.84	121.57	**0.14**	0.58
D_{20}	907.80	0.17	**0.05**	144.86	0.16	**0.05**	432.11	0.16	**0.05**	1568.37	0.16	**0.05**
D_{21}	697.12	**0.13**	4.06	121.14	**0.13**	3.98	352.89	**0.13**	3.88	–	**2.03**	2400.82
D_{22}	0.55	0.02	$<$ **0.005**	0.11	$<$ 0.005	$<$ 0.005	0.84	0.02	0.02	1.15	0.02	0.02
D_{23}	0.09	0.02	0.02	0.03	$<$ **0.005**	0.09	0.03	**0.02**	0.16	0.09	$<$ **0.005**	0.16
D_{24}	6.79	0.02	**0.00**	1.03	0.02	0.02	5.01	0.02	0.02	12.23	0.02	$<$ **0.005**
D_{25}	6.10	**0.02**	0.14	10.08	**0.09**	6.19	5.80	**0.05**	16.65	45.52	**0.05**	20.76
D_{26}	8.72	**0.02**	0.02	1.44	$<$ **0.005**	0.02	18.83	0.02	0.02	23.48	0.02	0.02
D_{27}	35.24	**0.03**	0.17	8.53	**0.05**	0.19	32.81	**0.05**	0.16	–	**0.25**	0.45
D_{28}	0.02	0.02	$<$ **0.005**	$<$ 0.005	$<$ **0.005**	0.02	0.02	$<$ 0.005	$<$ 0.005	0.03	$<$ 0.005	$<$ 0.005
D_{29}	0.67	0.02	$<$ **0.005**	0.14	$<$ **0.005**	0.02	1.28	0.02	$<$ 0.005	2.03	$<$ 0.005	$<$ 0.005
D_{30}	4.51	**0.02**	0.03	7.29	0.05	0.05	14.74	**0.03**	0.17	36.07	**0.02**	0.28
D_{31}	576.38	**0.08**	2.76	38.19	**0.08**	2.79	–	**0.81**	710.62	–	**0.91**	1025.27
D_{32}	17.50	0.03	$<$ **0.005**	3.73	0.03	$<$ **0.005**	13.15	0.05	**0.02**	34.21	0.03	**0.02**
D_{33}	1091.77	0.16	**0.03**	162.02	0.16	**0.03**	655.49	0.16	$<$ **0.005**	1785.99	0.16	**0.02**
D_{34}	0.03	0.02	$<$ **0.005**	0.03	$<$ 0.005	$<$ 0.005	0.05	$<$ 0.005	$<$ 0.005	0.11	$<$ 0.005	$<$ 0.005

Table 6.11: CPU times on DIMACS graphs (in sec.)

	$\gamma = 0.5$			$\gamma = 0.7$			$\gamma = 0.8$			$\gamma = 0.9$		
ID	$T_{[C_\gamma]}$	$T_{[D^S_\gamma]}$	$T_{[\bar{C}_\gamma]}$	$T_{[C_\gamma]}$	$T_{[D^S_\gamma]}$	$T_{[\bar{C}_\gamma]}$	$T_{[C_\gamma]}$	$T_{[D^S_\gamma]}$	$T_{[\bar{C}_\gamma]}$	$T_{[C_\gamma]}$	$T_{[D^S_\gamma]}$	$T_{[\bar{C}_\gamma]}$
D_{35}	14.23	**0.19**	56.92	7.75	**0.17**	1.67	9.80	**0.11**	1.23	18.08	**0.11**	0.83
D_{36}	127.59	**0.42**	25.83	69.89	**0.16**	79.52	44.03	**0.14**	93.56	62.76	**0.13**	81.73
D_{37}	8.92	**0.02**	0.38	12.55	**0.02**	0.80	157.88	**0.09**	25.09	205.91	**0.13**	43.80
D_{38}	144.52	**0.59**	802.59	87.38	**0.69**	6.98	101.23	**0.41**	5.44	147.20	**0.41**	5.06
D_{39}	49.06	**0.03**	9.86	655.58	**0.53**	285.92	685.61	**0.59**	589.22	626.69	**0.55**	428.13
D_{40}	70.56	**0.05**	2.28	30.25	**0.03**	2.53	1891.25	**0.31**	27.94	2032.66	**0.24**	132.73
D_{41}	559.87	**1.72**	3522.42	160.02	**1.56**	21.70	143.17	**1.19**	18.89	527.61	**1.81**	15.17
D_{42}	879.42	**3.19**	808.44	2564.84	**1.39**	3341.45	5765.97	**1.50**	5142.14	3406.08	**1.86**	2095.45
D_{43}	218.23	**0.06**	7.03	75.52	**0.06**	7.48	6168.50	**0.80**	316.52	–	**0.83**	552.95
D_{44}	4819.48	**3.97**	–	930.89	**3.61**	–	3174.97	**2.70**	97.44	3121.17	**2.64**	81.22
D_{45}	–	**8.10**	80.14	–	**4.13**	3268.00	–	**3.67**	–	–	**4.06**	–
D_{46}	2071.77	**0.12**	18.44	156.02	**0.13**	18.53	–	**2.03**	908.89	–	**2.25**	2858.16
D_{47}	–	**21.86**	–	–	**19.49**	–	–	**11.80**	514.92	–	**10.64**	360.02
D_{48}	1927.84	**0.19**	–	–	**14.13**	–	–	**13.88**	–	–	**12.50**	–
D_{49}	332.20	**0.25**	88.16	430.66	**0.27**	85.75	–	**7.03**	–	–	**5.89**	–
D_{50}	501.26	**0.11**	24.51	–	**1.80**	–	–	**2.05**	–	–	**1.81**	–
D_{51}	3.18	**0.02**	0.05	3.01	**< 0.005**	0.05	19.75	**0.05**	0.14	32.28	**0.05**	0.27
D_{52}	2.45	**0.02**	0.06	3.20	**0.02**	0.05	14.46	**0.06**	0.08	21.08	**0.06**	0.08
D_{53}	3.45	< 0.005	< 0.005	3.17	0.02	0.02	3.79	0.02	**< 0.005**	8.25	**< 0.005**	0.02
D_{54}	3.60	0.02	0.02	3.68	**0.02**	0.03	5.09	0.02	0.02	8.28	0.02	0.02
D_{55}	3.88	**< 0.005**	0.02	3.87	**0.02**	0.02	5.23	**< 0.005**	0.02	7.41	0.02	**< 0.005**
D_{56}	25.44	**0.02**	0.97	205.10	**0.14**	230.12	267.29	**0.19**	315.14	225.80	**0.16**	282.57
D_{57}	25.55	**0.03**	0.41	40.48	**0.03**	0.34	665.64	**0.17**	63.74	2105.47	**0.16**	102.88
D_{58}	30.30	**0.03**	0.41	27.86	**0.02**	0.41	609.03	**0.14**	72.77	2513.60	**0.14**	100.31
D_{59}	26.27	**0.03**	0.44	31.56	**0.03**	0.39	714.61	**0.14**	69.36	2467.72	**0.14**	80.61
D_{60}	31.68	**0.03**	0.08	25.77	**0.02**	0.08	32.73	**0.02**	0.08	59.50	**0.03**	0.05
D_{61}	3.32	**< 0.005**	0.05	5.79	**0.05**	0.06	20.90	**0.05**	0.25	18.77	**0.03**	0.28
D_{62}	3.39	**0.02**	0.03	4.77	0.02	**< 0.005**	5.16	0.02	0.02	14.20	0.05	**< 0.005**
D_{63}	24.43	**0.02**	0.97	204.69	**0.16**	232.10	251.02	**0.14**	209.34	202.55	**0.16**	201.34
D_{64}	29.30	**0.03**	0.44	38.61	**0.02**	0.42	823.79	**0.22**	71.26	535.68	**0.14**	64.24

Chapter 7

Conclusion

In this thesis we discussed the application of column generation procedures and sequential value correction heuristics to geometric and structure packing problems. On the side geometric packing, we dealt with extensions of BPP and CSP in cases with additional features and criteria that relevantly affect the optimization of real manufacturing process. For the one-dimensional BPP, we present a novel Dantzig-Wolfe reformulation for minimizing a convex combination of number of employed bins and maximum lateness in presence of variable packing time. In the two-dimensional context, where MIP formulations are generally not suitable to solve large size instances, we applied SVC algorithms for a real multi-stock size CSP and a bi-objective BPP with delay minimization. Concerning the packing of graph substructures, we dealt with the γ-QCP problem by introducing a new integer reformulation based on star constraints. For all the topics, computational results were presented to analyze the performances of the proposed algorithms.

Pricing-based techniques appear as useful tool to provide primal and dual bounds for challenging problems. Achieving good bounds in limited time is an important requirement if one aim to solve the original integer problem taken into account. When the CG procedure performs effectively, a branch-and-price approach can be seen as a valuable option to compute optimal integer solutions and fast primal heuristics (e.g. SVC) appear useful (*i*) to warm and speed-up the CG with good columns, (*ii*) to enhance the exploration of the underlying branch-and-bound tree. The implementation of branch-and-price algorithms for problems considered in chapters 5 and 6 can be a natural development of the work done so far.

CG and SVC share a common ground by relying on shadow prices and pseudo-prices within their frameworks, which guide the solution of optimization subproblems. For C&P problems, exchanging the information carried by prices between the two algorithms (in particular from CG to SVC) can be an interesting point of investigation. In [18] the authors provide the starting value of the pseudo-prices by inheriting the values of dual variables of the optimal LP relaxation solution. This idea can be exploited also, as instance, when the pricing subproblems called by the CG are challenging and using the SVC can be a valuable way to generate columns and feed the restricted master problem, which at each solution will provide a new optimal dual vector to initialize the subsequent call of SVC. Linking CG and SVC implementations into alternative and original ways could be of interest.

More in general, the study of the evolution of price values in both methods can be a topic of investigation. The whole dynamic of dual variables exploited in the CG procedure depends on a number of factors that makes it hardly predictable and results in a behaviour that could be comparable to a chaotic one. In the SVC heuristic instead, the pseudo-prices dynamic is defined by the updating formulae, whose effects are generally related to the sequence of solutions built. The focus could be on the seek of a pattern (if exists) that describes the evolution of prices able to converge close to optimal solutions for the relaxed master problem in CG, and to optimal primal solutions in SVC. For this purpose, thinking about the use of machine learning approaches (e.g., neural networks) appears intriguing. Furthermore, integration between machine learning and SVC can be appealing for setting the parameters (or entire components) of pseudo-prices formulae by learning from the features of instances. In this way the performance of the heuristic can be improved by specifying the updating phase according to the characteristics of the instance to solve.

Bibliography

[1] J. Abello, M. Resende, and S. Sudarsky. “Massive Quasi-Clique Detection”. In: *LATIN 2002: Theoretical Informatics*. Ed. by Sergio Rajsbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 598–612.

[2] E.A. Akkoyunlu. “The Enumeration of Maximal Cliques for Large Graphs”. In: *SIAM Journal of Computing* 2.1 (1973), pp. 1 –6.

[3] M.T. Almeida and F.D. Carvalho. “Integer models and upper bounds for the 3-club problem”. In: *SIAM Journal of Computing* 60.3 (2012), pp. 155–166.

[4] A. Aloisio, C. Arbib, and F. Marinelli. “Cutting Stock with no Three Parts per Pattern: Worki-in-Process and Pattern Minimization”. In: *Discrete Optimization* 8 (2010), pp. 315–3323.

[5] A. Aloisio, C. Arbib, and F. Marinelli. “On LP relaxations for the pattern minimization problem”. In: *Networks* 57.3 (2011), pp. 247–253.

[6] C. Alves and J.M. Valério de Carvalho. “A branch-and-price-and-cut algorithm for the pattern minimization problem”. In: *RAIRO - Operations Research* 42 (4) (2008), pp. 435–453.

[7] C. Alves, F. Clautiaux, J. de Carvalho, and J. Rietz. *Dual-Feasible Functions for Integer Programming and Combinatorial Optimization: Basics, Extensions and Applications*. Springer International Publishing, 2016. ISBN: 978-3-319-27602-1.

[8] C. Arbib and F. Marinelli. “On cutting stock with due dates”. In: *Omega* 46 (2014), pp. 11 –20.

[9] C. Arbib, F. Marinelli, and P. Ventura. “One-dimensional cutting stock with a limited number of open stacks: Bounds and solutions from a new integer linear programming model”. In: *Omega* 46 (2014), pp. 11 –20.

[10] C. Arbib, F. Marinelli, and P. Ventura. “One-dimensional cutting stock with a limited number of open stacks: bounds and solutions from a new integer linear programming model”. In: *International Transactions in Operational Research* 23.1-2 (2016), pp. 47–63. URL: `http://onlinelibrary.wiley.com/doi/10.1111/itor.12134/full`.

[11] C. Arbib and M. Marinelli. "Integrating process optimization and inventory planning in cutting-stock with skiving option: an optimization model and its application". In: *European Journal of Operational Research* 163 (2005), pp. 617–630. URL: `https://doi.org/10.1016/j.ejor.2003.12.021`.

[12] C. Arbib and M. Marinelli. "Maximum lateness minimization in one-dimensional bin packing". In: *Omega* 68 (2017), pp. 76–84. URL: `https://doi.org/10.1016/j.omega.2016.06.003`.

[13] N. Bansal, A. Caprara, and M. Sviridenko. "Improved approximation algorithms for multidimensional bin packing problems". In: *FOCS: Proc. 47nd IEEE Symposium on Foundations of Computer Science*. 2006, pp. 697–708.

[14] N. Bansal, A. Caprara, k. Jansen, L. Prädel, and M. Sviridenko. "A structural lemma in 2-dimensional packing, and its implications on approximability". In: *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC 2009)*. 2009, pp. 77–86.

[15] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. "Column Generation for Solving Huge Integer Programs". In: *Operations Research* 46 (1998), pp. 316–329.

[16] S. Baum and L.E. Jr Trotter. "Integer Rounding for Polymatroid and Branching Optimization Problems". In: *SIAM Journal on Algebraic Discrete Methods* 2.4 (1981), pp. 416–425.

[17] G. Belov and G. Scheithauer. "A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting". In: *European Journal of Operational Research* 171.1 (2006), pp. 85–106. URL: `https://doi.org/10.1016/j.ejor.2004.08.036`.

[18] G. Belov and G. Scheithauer. "A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths". In: *European Journal of Operational Research* 141.2 (2002), pp. 274–294. URL: `https://doi.org/10.1016/S0377-2217(02)00125-X`.

[19] G. Belov and G. Scheithauer. "Setup and open-stacks minimization in one-dimensional stock cutting". In: *INFORMS Journal on Computing* 19.1 (2007), pp. 27–35. DOI: `10.1287/ijoc.1050.0132`.

[20] G. Belov, G. Scheithauer, and E.A. Mukhacheva. "One-dimensional heuristics adapted for two-dimensional rectangular strip packing". In: *Journal of Operational Research Society* 59.6 (2008), pp. 823–832. DOI: `10.1057/palgrave.jors.2602393`.

[21] H. Ben Amor, J. Desrosiers, and A. Frangioni. *Stabilization in column generation*. Tech. rep. G-2004-62. 2004.

[22] J.A. Bennel, L.S. Lee, and C.N. Potts. "A genetic algorithm for two-dimensional bin packing with due dates". In: *Int. J. of Production Economics* 145.2 (2013), pp. 547–560. URL: `http://dx.doi.org/10.1016/j.ijpe.2013.04.040`.

[23] J.O. Berkey and P.Y. Wang. "Two-dimensional finite bin-packing algorithms". In: *Journal of the Operational Research Society* 38 (1987), pp. 423–429.

[24] M. Bhattacharyya and S. Bandyopadhyay. "Mining the Largest Quasi-clique in Human Protein Interactome". In: *2009 International Conference on Adaptive and Intelligent Systems*. 2009, pp. 194–199.

[25] J. Blazewic, K.E. Ecker, E. Pesch, G. Schmidt, and J. Weglar. *Handbook on Scheduling: From Theory to Application*. Springer-Verlag Berlin Heidelberg, 2007. DOI: `10.1007/978-3-540-32220-7`.

[26] B. Bollobás and A.G. Thomason. "Hereditary and monotone properties of graphs". In: *The mathematics of Paul Erdös, II, Algorithms and Combinatorics 14*. Ed. by R.L. Graham and J. Něsetřil. Springer-Verlag, 1997, pp. 70–78.

[27] N. Bourgeois, A. Giannakos, G. Lucarelli, I. Milis, V.Th. Paschos, and O. Pottié. "The max quasi-independent set problem". In: *CSR 2010: Computer Science – Theory and Applications*. 2010, pp. 60–71.

[28] J.M. Bourjolly, G. Laporte, and H. Mercure. "A combinatorial column generation algorithm for the maximum stable set problem". In: *Operations Research Letters* 20.1 (1997), pp. 21–29.

[29] A. Brandstädt, V.B. Le, and J.P. Spinrad. "Graph Classes: A Survey". In: *SIAM* (1987), p. 18.

[30] M. Brunato, H.H. Hoss, and R. Battiti. "On Effectively Finding Maximal Quasi-cliques in Graphs". In: *Learning and Intelligent Optimization* (2008), pp. 41–55.

[31] Constantino M. Goycoolea M. Vielma J. P. Carvajal R. and Andrés Weintraub. "Imposing connectivity constraints in forest planning models". In: *Operations Research* 64 (2013), pp. 824–836.

[32] F. Chataigner, G. Manić, Y. Wakabayashi, and R. Yuster. "Approximation algorithms and hardness results for the clique packing problem". In: *Discrete Applied Mathematics* 157 (7 2009), pp. 1396–1406.

[33] Q. Chen, Y. Cui, and Y. Chen. "Sequential value correction heuristic for the two-dimensional cutting stock problem with three-staged homogenous patterns". In: *Optimization Methods and Software* 31.1 (2016), pp. 68–87. DOI: `10.1080/10556788.2015.1048860`.

[34] Y. Chen, X. Song, D. Ouelhadj, and Y. Cui. "A heuristic for the skiving and cutting stock problem in paper and plastic film industries". In: *International Transaction in Operational Research* 26 (2019), pp. 157–179. DOI: `DOI:10.1111/itor.12390`.

[35] C. Chu and J. Antonio. "Approximation Algorithms to Solve Real-Life Multicriteria Cutting Stock Problems". In: *Operations Research* 47.4 (1999), pp. 495–508. URL: `http://www.jstor.org/stable/223155`.

[36] F. Clautiaux, A. Jouglet, and J. El Hayek. "A new lower bound for the non-oriented two-dimensional bin-packing problem". In: *Operations Research Letters* 35.4 (2007), pp. 365–373.

[37] E.G.Jr. Coffman, M.R. Garey, D.S. Johnson, and R.E. Tarjan. "Performance bounds for level-oriented two-dimensional packing algorithms". In: *SIAM Journal on Computing* 9.4 (1980), pp. 808–826.

[38] D. Cornaz, F. Furini, M. Lacroix, E. Malaguti, A.R. Mahjoub, and Martin S. "The vertex *k*-cut problem". In: *Discrete Optimization* 31 (2019), pp. 8–28.

[39] Y. Cui and B. Huang. "Reducing the number of cuts in generating three-staged cutting patterns". In: *European J. of Operational Research* 218.2 (2012), pp. 358–365. URL: `http://dx.doi.org/10.1016/j.ejor.2011.10.047`.

[40] Y. Cui, L. Yang, and Q. Chen. "Heuristic for the rectangular strip packing problem with rotation of items". In: *Computers and Operations Research* 40.4 (2008), pp. 1094–1099. URL: `https://doi.org/10.1016/0377-2217(91)90222-H`.

[41] Y. Cui, C. Zhong, and Y. Yao. "Pattern-set generation algorithm for the one-dimensional cutting stock problem with setup cost". In: *European Journal of Operational Research* 59.6 (2015), pp. 540–546. URL: `https://doi.org/10.1016/j.ejor.2014.12.015`.

[42] Y.P. Cui, Y. Cui, and T. Tang. "Sequential heuristic for the two-dimensional bin-packing problem". In: *European Journal of Operational Research* 240.1 (2015), pp. 43–53.

[43] Dresden Cutting and Packing Group (CaPaD). http://www.math.tu-dresden.de/ capad/.

[44] Wuttke D.A. and H.S. Heese. "Two-dimensional cutting stock problem with sequence dependent setup times". In: *European Journal of Operational Research* 265.1 (2018), pp. 303–315. URL: `https://doi.org/10.1016/j.ejor.2017.07.036`.

[45] G.B. Dantzig and p. Wolfe. "Decomposition principle for linear programs". In: *Operations Research* 8.1 (1960), pp. 101–111. URL: https://www.jstor.org/stable/167547.

[46] S. Das and D. Ghosh. "Binary knapsack problems with random budgets". In: *Journal of the Operational Research Society* 54.9 (2003), pp. 970–983. URL: https://doi.org/10.1057/palgrave.jors.2601596.

[47] M. Dell'Amico, S. Martello, and D. Vigo. "A lower bound for the non-oriented two-dimensional bin packing problem". In: *Discrete Applied Mathematics* 118 (2002), pp. 13–24.

[48] G. Desaulniers, J. Desrosiers, and M.M. Solomon. "Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems". In: *Essays and Surveys in Metaheuristics*. Ed. by C.C. Ribeiro and P. Hansen. Boston, MA, 2001, pp. 390–324.

[49] G. Desaulniers, J. Desrosiers, I. loachim, M.M. Solomon, F. Soumis, and D. Villeneuve. "A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems". In: *Fleet Management and Logistics*. Centre for Research on Transportation. Boston, MA: Springer, 1998. Chap. 3, pp. 57–93.

[50] M.J. Desrochers and F. Soumis. "A column generation approach to urban transit crew scheduling". In: *Transportation Science* 23 (1998), pp. 1–13.

[51] J Desrosiers and M.E. Lübbecke. "A Primer in Column Generation". In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M.M. Solomon. Boston, MA: Springer US, 2005, pp. 1–32. DOI: 10.1007/0-387-25486-2_1.

[52] P. Detti, A. Agnetis, and G. Ciaschetti. "Polynomial algorithms for a two-class multiprocessor scheduling problem in mobile telecommunications systems". In: *Journal of Scheduling* 8.3 (2005), pp. 255 –273.

[53] E. D. Dolan and J.J. Moré. "Benchmarking optimization software with performance profiles". In: *Mathematical programming* 91(2) (2002), pp. 201–213.

[54] M. Ehrgott. "Approximation algorithms for combinatorial multicriteria optimization problems". In: *International Transactions in Operational Research* 7 (2000), pp. 5–31.

[55] M. Ehrgott and M.M. Wiecek. *Multiobjective Programming*. Springer, New York, 2005, pp. 667–708.

[56] P. Erdös and A. Rényi. "On Random Graphs. I". In: *Publicationes Mathematicae* 6 (1959), pp. 290–297.

[57] F.D. Fomeni and A.N. Letchford. "A Dynamic Programming Heuristic for the Quadratic Knapsack Problem". In: *INFORMS Journal on Computing* 26.1 (2014), pp. 173–182. URL: https://doi.org/10.1287/ijoc.2013.0555.

[58] L.R. Ford and D.R. Fulkerson. "A suggested computation for maximal multicommodity network flows". In: *Management Science* 5 (1958), pp. 97–101.

[59] S. Fortunato. "Community detection in graphs". In: *Physics Report* 486(3–5) (2010), pp. 75–174.

[60] M.R. Garey, R.L. Graham, and D.S. Johnson. "Resource Constrained Scheduling as Generalized Bin Packing". In: *Journal of Combinatorial Theory (A)* 21.3 (1976), pp. 257–298. URL: https://doi.org/10.1016/0097-3165(76)90001-7.

[61] M.R. Garey and D.S. Johnson. "" Strong " NP-Completeness Results: Motivation, Examples, and Implications". In: *Journal of the ACM* 25.3 (1978), pp. 499–508. URL: http://doi.acm.org/10.1145/322077.322090.

[62] P.C. Gilmore and R.E. Gomory. "A Linear Programming Approach to the Cutting-Stock Problem". In: *Operations Research* 9.6 (1961), pp. 849–859. URL: https://doi.org/10.1287/opre.9.6.849.

[63] P.C. Gilmore and R.E. Gomory. "A Linear Programming Approach to the Cutting-Stock Problem – Part II". In: *Operations Research* 11 (1963), pp. 863–888. URL: https://doi.org/10.1287/opre.9.6.849.

[64] T. Gschwind, S. Irnich, F. Furini, and R. Wolfler Calvo. *Social network analysis and community detection by decomposing a graph into relaxed cliques*. Tech. rep. Johannes Gutenberg University Mainz, Mainz, Germany.

[65] R.W. Haessler. "Controlling cutting pattern changes in one-dimensional trim problems". In: *Operations Research* 23 (1975), pp. 483–493. URL: https://doi.org/10.1287/opre.23.3.483.

[66] R. Harren and R. Van Stee. "Packing Rectangles into 2OPT Bins Using Rotations". In: *Algorithm Theory – SWAT 2008. Lecture Notes in Computer Science*. Ed. by Gudmundsson J. 5124. Springer, 2008.

[67] J. Hemminki, T. Leipälä, and O. Nevalainen. "On-line packing with boxes of different sizes". In: *International Journal of Production Research* 36.8 (1998), pp. 2225–22245. URL: https://doi.org/10.1080/002075498192869.

[68] L.C. Hendry, K.K. Fok, and K.W. Shek. "A cutting stock and scheduling problem in the copper industry". In: *Journal of the Operational Research Society* 47.1 (1996), pp. 38–47. DOI: 10.1057/jors.1996.4.

[69] J.B. Hiriart-Urruty and C. Lemarechal. *Convex Analysis and Minimization Algorithms II: Advanced Theory and Bundle Methods*. Vol. 306. Springer-Verlag Berlin Heidelberg, 1993. DOI: 10.1007/978-3-662-06409-2.

[70] K. Jansen and L. Prädel. "New approximability results for two-dimensional bin packing". In: *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*. 153 (1). Springer, 2004, pp. 919–936.

[71] K. Jansen, L. Prädel, and U.M. Schwarz. "Two for One: Tight Approximation of 2D Bin Packing". In: *Algorithms and Data Structures. WADS 2009. Lecture Notes in Computer Science*. Ed. by F. Dehne, M. Gavrilova, JR. Sack, and C.D. Tóth. 5664. Springer, 2009.

[72] K. Jansen and R. Solis-Oba. "Rectangle packing with one-dimensional resource augmentation". In: *Discrete Optimization* 6 (2009), pp. 310–323.

[73] R.E. Johnson and E. Sadinlija. "A new model for complete solutions to one-dimensional cutting stock problems". In: *European Journal of Operational Research* 153 (1) (2004), pp. 176–183.

[74] L. Kantorovich. "Mathematical Methods of Organizing and Planning Production". In: *Management Science* 6.4 (1960), pp. 366–422.

[75] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Boston, MA: Springer US, 1972, pp. 85–103. ISBN: 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2_9.

[76] N. Krislock, J. Malick, and F. Roupin. "Computational results of a semidefinite branch-and-bound algorithm for k-cluster". In: *Computers and Operations Research* 66 (2016), pp. 153–159.

[77] J. Kupke. "Lösung von ganzzahligen Verschnittproblemen mit Branch-and-Price". Köln, Germany: Institut für Informatik, Universität zu Köln, 1998.

[78] C.Y. Lee, R. Uzsoy, and L.A. Martin-Vega. "Efficient Algorithms for Scheduling Semiconductor Burn-In Operations". In: *Operations Research* 40.4 (1992), pp. 764–775.

[79] G. Liu and L. Wong. "Effective Pruning Techniques for Mining Quasi-Clique". In: *ECML PKDD '08 Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases - Part II*. 2008, pp. 33–49.

[80] A. Lodi, S. Martello, and D. Vigo. "Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems". In: *INFORMS Journal on Computing* 11.4 (1999b), pp. 345–357.

[81] A. Lodi, S. Martello, and D. Vigo. "Recent advances on two-dimensional bin packing problems". In: *Discrete Applied Mathematics* 123 (2002), pp. 379–396.

[82] A. Lodi, S. Martello, M. Monaci, C. Cicconetti, L. Lenzini, E. Mingozzi, C. Elkund, and G. Moilanen. "Efficient two-dimensional packing algorithms for mobile WiMAX". In: *Management Science* 57 (2011), pp. 2130–2144.

[83] M. E. Lübbecke and J. Desrosiers. "Selected Topics in Column Generation". In: *Operations Research* 53 (2005), pp. 1007–1023.

[84] E. Malaguti, R.M. Durán, and P. Toth. "Approaches to real world two-dimensional cutting problems". In: *Omega* 47 (2014), pp. 99–115.

[85] O. Marcotte. "An instance of the cutting stock problem for which the rounding property does not hold". In: *Operations Research Letters* 4.5 (1986), pp. 239–243.

[86] O. Marcotte. "The cutting stock problem and integer rounding". In: *Mathematical Programming* 3.1 (1985), pp. 82–92.

[87] F. Marinelli and A. Pizzuti. "A Sequential Value Correction heuristic for a bi-objective two-dimensional bin-packing". In: *Electronic Notes in Discrete Mathematics* 64 (2018), pp. 25–34.

[88] F. Marinelli and A. Pizzuti. "Bin Packing Problems with Variable Pattern Processing Times: A Proof-of-concept". In: *Springer Proceedings in Mathematics & Statistics* 217 (2017), pp. 453–460. URL: `https://doi.org/10.1007/978-3-319-67308-0_46`.

[89] F. Marinelli, A. Pizzuti, and F. Rossi. "A star-based reformulation for the maximum quasi-clique problem". In: *Proceedings of the* 16^{th} *CTW on Graphs and Combinatorial Optimization*. 2018, pp. 118–121.

[90] R.E. Marsten. "The use of the boxstep method in discrete optimization". In: *Nondifferentiable Optimization*. Ed. by M.L. Balinski and P. Wolfe. Berlin, Heidelberg: Springer Berlin Heidelberg, 1975, pp. 127–144. URL: `https://doi.org/10.1007/BFb0120702`.

[91] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990. ISBN: 0-471-92420-2.

[92] K. Matsumoto, S. Umetani, and H. Nagamochi. "On the one-dimensional stock cutting problem in the paper tube industry". In: *Journal of Scheduling* 14 (2011), pp. 281–290. DOI: `10.1007/s10951-010-0164-2`.

[93] C. McDiarmid. "Pattern Minimisation in Cutting Stock Problems". In: *Discrete Applied Mathematics* 98 (1999), pp. 121–131.

[94] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. "Stabilized column generation". In: *Discrete Mathematics* 194.1 (1999), pp. 229 –237. URL: `https://doi.org/10.1016/S0012-365X(98)00213-1`.

[95] E. Moradi and B. Balasundaram. "Finding a maximum *k*-club using the *k*-clique formulation and canonical hypercube cuts". In: *Optimization Letters* 12(8) (2018), pp. 1947–1957.

[96] E.A. Mukhacheva and V.A. Zalgaller. "Linear programming cutting problems". In: *International Journal of Software Engineering and Knowledge Engineering* 3.4 (1993), pp. 463–476. URL: `https://www.worldscientific.com/doi/abs/10.1142/S0218194093000240`.

[97] E.A. Mukhacheva, G.N. Belov, V.M. Kartack, and A.S. Mukhacheva. "Linear one-dimensional cutting-packing problems: numerical experiments with the sequential value correction method (SVC) and a modified branch-and-bound (MBB)". In: *Pesquisa Operacional* 20.2 (2000), pp. 153–168. URL: `http://dx.doi.org/10.1590/S0101-74382000000200002`.

[98] A.T. Murray and R.L. Church. "Facets for node packing". In: *European Journal of Operational Research* 101.3 (1997), pp. 598–608. URL: `https://doi.org/10.1016/S0377-2217(96)00175-0`.

[99] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. New York, NY, USA: John Wiley & Sons, Inc., 1988.

[100] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., 1988. ISBN: 9780471828198. DOI: `10.1002/9781118627372`.

[101] S.L. Nonås and A. Thorstenson. "A combined cutting-stock and lot-sizing problem". In: *European Journal of Operations Research* 120.2 (2000), pp. 327–342. URL: `https://doi.org/10.1016/S0377-2217(99)00160-5`.

[102] F.M. Pajouh, Z. Miao, and B. Balasundaram. "A branch-and-bound approach for maximum quasi-cliques". In: *Annals of Operations Research* 216 (2014), pp. 145–161.

[103] C.H. Papadimitriou and M. Yannakakis. "On the approximability of trade-offs and optimal access of web sources". In: *Proceedings 41st Annual Symposium on Foundations of Computer Science* (2000), pp. 86–92.

[104] P.M. Pardalos and J. Xue. "The maximum clique problem". In: *Journal of Global Optimization. Optim.* 4.3 (1994), pp. 301–328.

[105] G. Pastukhov, A. Veremyev, V. Boginski, and Oleg A. Prokopyev. "On maximum degree-based γ-quasi-clique problem: Complexity and exact approaches". In: *Networks* 71.2 (2018), pp. 136–152.

[106] J. Pattilo, N. Youssef, and S. Butenko. "On clique relaxation models in network analysis". In: *European Journal of Operational Research* 226.1 (2013), pp. 9–18.

[107] J. Pattilo, A. Veremyev, S. Butenko, and V. Boginski. "On the maximum quasi-clique problem". In: *Discrete Applied Mathematics* 161 (2013), pp. 224–257.

[108] D. Pisinger. "The quadratic knapsack problem – a survey". In: *Discrete Applied Mathematics* 155.5 (2007), pp. 623–648. URL: `https://www.sciencedirect.com/science/article/pii/S0166218X06003878`.

[109] K.C. Poldi and M.N. Arenales. "Heuristics for the one-dimensional cutting stock problem with limited multiple stock lengths". In: *Computers and Operations Research* 36.6 (2009), pp. 2074–2081. DOI: `10.1016/j.cor.2008.07.001`.

[110] S. Polyakovskiy and R M'Hallah. "A hybrid feasibility constraints-guided search to the two-dimensional bin packing problem with due dates". In: *European Journal of Operational Research* 266 (2018), pp. 819–839.

[111] J. Puchinger, G. Raidl, and K. Gabriele. "Solving a Real-World Glass Cutting Problem". In: *Evolutionary Computation in Combinatorial Optimization, 4th European Conference, EvoCOP 2004.* 2004. DOI: `10.1007/978-3-540-24652-7_17`.

[112] X. Qi. "A note on worst-case performance of heuristics for maintenance scheduling problems". In: *Discrete Applied Mathematics* 155 (2007), pp. 416–422.

[113] H. Reinertsen and T.W.M. Vossen. "The one-dimensional cutting stock problem with due-dates". In: *European Journal of Operational Research* 201 (2010), pp. 701–711.

[114] R.A. Rossi and N.K. Ahmed. In: *The network Data Repository with Interactive Graph Analytics and Visualization*. 2015. URL: `http://networkrepository.com`.

[115] P. San Segundo, Coniglio S., F. Furini, and I. Ljubić. "A new branch-and-bound algorithm for the maximum edge-weighted clique problem". In: *European Journal of Operational Research* (2019). In press.

[116] A. Schrijver. *Theory of Linear and Integer Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1986. ISBN: 0-471-90854-1.

[117] SCM. *SCM Group*. https://www.scmgroup.com. Accessed: 2017-06-30. 2017.

[118] J.F. Shapiro. *Mathematical Programming, Structures and Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 1979.

[119] H. Stadtler. "A one-dimensional cutting stock problem in the aluminium industry and its solution". In: *European Journal of Operational Research* 44 (1990), pp. 209–223.

[120] A. Steinberg. "A strip-packing algorithm with absolute performance bound 2". In: *SIAM Journal on Computing* 26 (1997), pp. 401–409.

[121] L. Taccari. "Integer programming formulations for the elementary shortest path problem". In: *European Journal of Operational Research* 252 (2016), pp. 122–130.

[122] V. T'kind and J.C. Billaut. *Multicriteria Scheduling Theory, Models and Algorithms*. Springer, Berlin, Heidelberg, 2006.

[123] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and Tsiarli M. "Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees". In: *KDD '13 Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2013, pp. 104–112.

[124] S. Umetani, M. Yagiura, and T. Ibaraki. "One-dimensional cutting stock problem to minimize the number of different patterns". In: *European Journal of Operational Research* 146.2 (2003), pp. 388–402. URL: `http://dx.doi.org/10.1016/S0377-2217(02)00239-4`.

[125] R. Uzsoy. "Scheduling a single batch processing machine with non-identical job sizes". In: *International Journal of Production Research* 32.7 (1994), pp. 1615–1635. URL: `https://doi.org/10.1080/00207549408957026`.

[126] R. Vahrenkamp. "Random search in the one-dimensional cutting stock problem". In: *European Journal of Operational Research* 95.1 (1996), pp. 191–200. URL: https://doi.org/10.1016/0377-2217(95)00198-0.

[127] F. Vanderbeck. "Exact algorithm for minimizing the number of setups in the one-dimensional cutting stock problem". In: *Operations Research* 48 (6) (2000), pp. 915–926.

[128] F. Vanderbeck and L.A. Wolsey. "An exact algorithm for IP column generation". In: *Operations Research Letters* 19.4 (1996), pp. 151 –159. URL: https://doi.org/10.1016/0167-6377(96)00033-8.

[129] C.R. Varela R.and Vela, J. Puente, M. Sierra, and I. González-Rodríguez. "An effective solution for a real cutting stock problem in manufacturing plastic rolls". In: *Annals of Operations Research* 166.1 (2008), p. 125. URL: http://dx.doi.org/10.1007/s10479-008-0407-1.

[130] A. Veremyev, O.A. Prokopyev, S. Butenko, and E.L. Pasiliao. "Exact MIP-based approaches for finding maximum quasi-cliques and dense subgraphs". In: *Computational Optimization and Applications* 64 (2016), pp. 177–214.

[131] M.A. Verkhoturov and O.Y. Sergeyeva. "The sequential value correction method for the two-dimensional irregular cutting stock problem". In: *Pesquisa Operacional* 20.2 (2000), pp. 233–246. URL: http://dx.doi.org/10.1590/S0101-74382000000200007.

[132] Y. Wang, A. Buchanan, and S. Butenko. "On imposing connectivity constraints in integer programs". In: *Mathematical Programming* 166.1 (2017), pp. 241–271.

[133] G. Wäscher. "An LP-based approach to cutting stock problems with multiple objectives". In: *European Journal of Operational Research* 44.2 (1990), pp. 175–184. URL: https://doi.org/10.1016/0377-2217(90)90353-D.

[134] G. Wäscher and T. Gau. "Heuristics for the integer one-dimensional cutting stock problem: A computational study". In: *Operations Research Spectrum* 18.3 (1996), pp. 131–144.

[135] G. Wäscher, H. Haußner, and H. Schumann. "An improved typology of cutting and packing problems". In: *European Journal of Operational Research* 183.3 (2007), pp. 1109–1130. URL: https://doi.org/10.1016/j.ejor.2005.12.047.

[136] H.H. Yanasse. "On a pattern sequencing problem to minimize the maximum number of open stacks". In: *European Journal of Operational Research* 100 (1997), pp. 454–463. URL: https://doi.org/10.1016/S0377-2217(97)84107-0.

[137] H.H. Yanasse and M.J. Pinto Lamosa. "An integrated cutting stock and sequencing problem". In: *European Journal of Operational Research* 183 (3) (2007), pp. 1353–1370. URL: https://doi.org/10.1016/j.ejor.2005.09.054.

[138] H.H. Yanasse, A.S.I. Zinober, and R.G. Harris. "Two-dimensional Cutting Stock with Multiple Stock Sizes". In: *Journal of the Operational Research Society* 42.8 (1991), pp. 673–683. URL: http://dx.doi.org/10.1057/jors.1991.133.

[139] B.J. Yuen. "Heuristics for sequencing cutting patterns". In: *European Journal of Operational Research* 55.2 (1991), pp. 183–190. URL: https://doi.org/10.1016/0377-2217(91)90222-H.

[140] W. Zheng, P. Ren, P. Ge, Y. Qiu, and Z. Liu. "Hybrid heuristic algorithm for two-dimensional steel coil cutting problem". In: *Computers and Industrial Engineering* 62.3 (2012), pp. 829 –838. URL: https://doi.org/10.1016/j.cie.2011.12.012.

[141] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, and V.G. da Fonseca. "Performance assessment of multiobjective optimizers: an analysis and review". In: *IEEE Transactions on Evolutionary Computation* 7 (2003).