Filipe Daniel Alves Brandão

Cutting & Packing Problems: General Arc-flow Formulation with Graph Compression

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto

September 22, 2017

Filipe Daniel Alves Brandão

Cutting & Packing Problems: General Arc-flow Formulation with Graph Compression

Tese submetida à Faculdade de Ciências da Universidade do Porto
para obtenção do grau de Doutor em Ciência de Computadores
orientada pelo Professor João Pedro Pedroso

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
September 22, 2017

Esta tese foi financiada por uma bolsa de doutoramento da Fundação para a Ciência e a Tecnologia (referência SFRH/BD/91538/2012) no âmbito do programa POPH, financiado pelo Fundo Social Europeu e pelo Governo Português.

Resumo

Apresentamos um método exato, baseado num modelo de fluxos com restrições de procura, para a resolução de problemas de corte e empacotamento — incluindo variantes com restrições múltiplas — através da representação de todos os padrões validos num grafo bastante compacto. O nosso método inclui um algoritmo de compressão de grafos que geralmente reduz substancialmente o tamanho dos grafos sem enfraquecer o modelo.

A nossa formulação é equivalente à formulação de Gilmore e Gomory, tendo portanto uma relaxação linear bastante forte. No entanto, em vez de usar geração de colunas num processo iterativo, o nosso método constrói um grafo onde caminhos representam cada padrão de empacotamento válido.

O mesmo método, sem qualquer parametrização, foi utilizado para resolver uma grande variedade de instâncias de vários tipos de problemas de corte e empacotamento através de reduções a problemas de empacotamento de vetores. Nesta tese lidamos com *vector packing*, *bin packing*, *cutting stock*, *bin packing* com restrições de cardinalidade, *cutting stock* com limite no número de facas, *bin packing* com conflitos, entre muitos outros.

O conjunto de aplicações não se limita a reduções a problemas de empacotamento de vetores. O método proposto fornece uma maneira simples de representar todos os padrões válidos para instâncias de corte e empacotamento em modelos de programação inteira. Assim sendo, podemos usar vários modelos de fluxos num único modelo para modelar, por exemplo, variantes de múltiplos estágios. Dada a flexibilidade do método proposto, introduzimos duas novas variantes de problema: o problema de empacotamento de vetores multi-estágio e o problema de empacotamento de vetores generalizado. O nosso método lidou facilmente com estas duas novas variantes dado que tira total partido das heurísticas e dos geradores de planos de corte incluídos nos programas de resolução de modelos de programação inteira de última geração; isto é particularmente importante quando os modelos de fluxos são usados como parte de modelos mais complexos.

Um dos resultados desta tese é um projeto de código aberto chamado `VPSolver` que está atualmente a ser usado não só por outros investigadores mas também na indústria. A formulação de fluxos com compressão de grafos está a ser usada atualmente por uma grande multinacional no setor de rotulagem e empacotamento em mais de 60 locais em todo o mundo para resolver diariamente centenas de instâncias de corte com diversas restrições especificas do setor.

Palavras-chave: Corte; Empacotamento; Formulação de Fluxos; Compressão de Grafos.

Abstract

We present an exact method, based on an arc-flow formulation with side constraints, for solving bin packing and cutting stock problems — including multi-constraint variants — by simply representing all the patterns in a very compact graph. Our method includes a graph compression algorithm that usually reduces the size of the underlying graph substantially without weakening the model.

Our formulation is equivalent to Gilmore and Gomory's, thus providing a very strong linear relaxation. However, instead of using column-generation in an iterative process, the method constructs a graph where paths from the source to the target node represent every valid packing pattern.

The same method, without any problem-specific parameterization, was used to solve a large variety of instances from several cutting and packing problems through reductions to multiple-choice vector packing. In this thesis, we deal with vector packing, bin packing, cutting stock, cardinality constrained bin packing, cutting stock with cutting knife limitation, bin packing with conflicts, and many other problems.

The set of applications is not limited to reductions to multiple-choice vector packing problems. The proposed method provides a simple way to represent every feasible pattern for any cutting or packing instance in an integer programming model. Therefore, we can use multiple arc-flow models in the same model in order to model, for instance, multi-stage variants. Given the flexibility of the proposed method, we introduced two new problem variants: the multi-stage vector packing problem and the generalized vector packing problem. Both variants were easily tackled by our method since it takes full advantage of the heuristics and cutting plane generators that come with state-of-the-art MIP solvers. This is particularly important when arc-flow models are used as part of more complex models that may benefit substantially from the cuts generated by MIP solvers.

One of the outcomes of this thesis is an open-source project called `VPSolver`, which is currently being used not only by other researchers but also in the industry. The arc-flow formulation with graph compression is currently being used by a large multinational company in the labeling & packaging sector in over 60 sites worldwide to solve daily hundreds of cutting stock instances with several industry-specific constraints.

Keywords: Bin Packing; Cutting Stock; Vector Packing; Arc-flow Formulation; One-cut Formulation; Graph Compression.

Contents

Chapter 1

Introduction

There are problems whose solution can be computed in polynomial time. These problems are considered efficiently solvable, or tractable. On the other hand, there are provably intractable problems, e.g., undecidable or non-deterministic intractable (see, e.g., Garey and Johnson 1979). However, most of the apparently intractable problems encountered in practice are decidable and can be solved in polynomial time using a non-deterministic computer model that has the ability to do an unbounded number of independent computational sequences in parallel. There is neither a proof that verifies the apparent intractability of these problems nor an efficient algorithm to solve them. Problems that can be solved in polynomial time using a non-deterministic computer model are in the complexity class NP (non-deterministic polynomial time). The efficiently solvable problems belong to the class P (deterministic polynomial time), which is contained in NP. The NP class contains many important problems, the hardest of which are called NP-complete problems. A problem is NP-complete if it is in NP and any other NP problem can be reduced to it in polynomial time. It is not known whether every problem in NP can be quickly solved — this is called the P = NP conjecture. However, if any single NP-complete problem can be solved in polynomial time, then every problem in NP can also be solved in polynomial time. An optimization problem consists of finding the best solution from all the solutions satisfying all the problem's constraints, while the corresponding decision problem consists of finding a feasible solution not worse than a given value. When a decision version of a combinatorial optimization problem is NP-complete, the optimization version is NP-hard (non-deterministic polynomial-time hard). NP-hard problems are at least as hard as any problem in NP.

The bin packing problem (BPP) is a combinatorial NP-hard problem (see, e.g., Garey and Johnson 1979) in which objects of different volumes must be packed into a finite number of bins, each with capacity W, in a way that minimizes the number of bins used. In fact, the BPP is strongly NP-hard (see, e.g., Garey and Johnson 1978) since it remains so even when all of its numerical parameters are bounded by a polynomial in the length of the input. Therefore, the BPP cannot even be solved in pseudo-polynomial time unless P = NP. Besides being strongly NP-hard, the BPP is also hard to approximate within $3/2 - \varepsilon$ (see, e.g, Simchi-Levi 1994). If such approximation exists, one could partition n

non-negative numbers into two sets with the same sum in polynomial time. This problem — called the number partition problem — could be reduced to a bin packing problem with bins of capacity equal to half of the sum of all the numbers. Any approximation better than $3/2 - \varepsilon$ of the optimal value could be used to find a perfect partition, corresponding to a packing in $\lfloor 2(3/2 - \varepsilon) \rfloor = 2$ bins. However, the number partition problem is known to be NP-hard.

There are many variants of this problem and they have many applications, such as filling up containers, placing computer files with specified sizes into memory blocks of fixed size, loading trucks with volume or weight capacity limits, among others.

The BPP can be seen as a special case of the cutting stock problem (CSP). In this problem there is a number of rolls of stock material, such as paper rolls or sheet metal, of fixed width waiting to be cut for satisfying demand of different customers, who want pieces of various widths. Rolls must be cut in such a way that waste is minimized. Note that, in the paper industry, solving this problem to optimality can be economically significant; a small improvement in reducing waste can have a huge impact in yearly savings.

There are many similarities between the BPP and the CSP. However, in the CSP, the items of equal size (which are usually ordered in large quantities) are grouped into orders with a required level of demand, while in the BPP the demand for a given size is usually close to one. According to Wäscher et al. (2007), cutting stock problems are characterized by a weakly heterogeneous assortment of small items, in contrast with bin packing problems, which are characterized by a strongly heterogeneous assortment of small items.

The p-dimensional vector bin packing problem (VBP), also called general assignment problem by some authors, is a generalization of bin packing with multiple constraints (see, e.g, Garey et al. 1976). In this problem, one is required to pack n items of m different types, represented by p-dimensional vectors, into as few bins as possible. In practice, this problem models, for example, static resource allocation problems where the minimum number of servers with known capacities is used to satisfy a set of services with known demands.

Finally, the multiple-choice vector bin packing problem (MVBP) is a variant of the VBP in which bins have several types (i.e., sizes and costs) and items have several incarnations (i.e., will take one of several possible sizes); this occurs typically in situations where one of several incompatible decisions has to be made (see, e.g., Patt-Shamir and Rawitz 2012).

1.1 Goals and Contributions

Wolsey (1977) proposed for the first time arc-flow formulations for cutting and packing problems including multi-constraint variants. Despite not presenting computational results, some properties that suggested computational advantages of such formulations were presented. We revisit this paper in detail focusing mainly on the derivation of models from dynamic programming recursions.

In this thesis, we present a very flexible arc-flow formulation for modeling and solving a large variety of cutting and packing problems by simply representing all the patterns in a very compact graph; the key component of our method is a graph construction and compression algorithm, which is presented in Section 4.2.3.

In Brandão (2012), a graph compression algorithm was used to solve one and two-dimensional cutting and packing problems with state-of-the-art performance. The major limitation of this algorithm was the fact that the initial graph had to be built before being able to compress it. In Brandão and Pedroso (2016), one of the outcomes of this thesis, the idea was generalized and a new graph construction and compression algorithm was introduced. This new algorithm allowed modeling and solving instances with hundreds of dimensions by building compressed graphs on the fly; the main limitation now is the combination of large capacities and long patterns, which may still be difficult to represent in a compact way in the compressed graph.

Among applications which can be modeled through reductions to multiple-choice vector packing there are bin packing, cutting stock, cardinality constrained bin packing, cutting stock with cutting knife limitation, bin packing with conflicts, and many other problems. In addition to these applications, in this thesis we introduce two new problem variants: multi-stage vector packing and generalized vector packing.

In the two-dimensional rectangular cutting stock problem, the objective is to cut a set of rectangular items from identical rectangular plates in such a way that the number of plates used is minimized. In the multi-stage variant, the items are obtained by performing a series of guillotine cuts; alternating between horizontal and vertical guillotine cuts in each stage. The proposed method handles variants with multiple plate types, rotation of items, any number of stages, and multiple constraints per stage (e.g., a cardinality constraint limiting the number of cuts that can be performed in each stage). We call this generalization multi-stage vector packing.

Baldi (2013) introduced the generalized bin packing problem. In this problem, given a set of items characterized by volume and profit, and a set of bins with given volumes and costs, one aims to select the subset of profitable items and appropriate bins to optimize

the objective function which combines the cost of the used bins and the profit derived by the selected items. This problem generalizes bin packing with variable size and cost, and the knapsack problem, among others. Based on the same principle, we introduce and tackle the generalized vector packing problem.

Another outcome of this thesis is an open-source project called `VPSolver`,[1] which is currently being used not only by other researchers but also in the industry. This software tackles every cutting and packing application presented in this thesis, and can be easily adapted to handle a large variety of industry-specific constraints that are not handled by any other method in the literature. This flexibility and effectives is due to the possibility of using arc-flow models as part of more complex models, while taking full advantage of the heuristics and cutting plane generators that come with state-of-the-art MIP solvers.

1.2 Outline

The outline of this thesis is as follows. Chapter 2 presents previous work on mathematical programming models for cutting and packing problems, such as Gilmore-Gomory's formulation, Valério de Carvalho's arc-flow formulation, and Dyckhoff's one-cut formulation. Chapter 3 studies the relation between dynamic programming and arc-flow/one-cut models. Chapter 4 presents the general arc-flow formulation with graph compression, which allows us to model and solve a large variety of cutting and packing problems. Chapters 5 and 6 analyze the performance of the general arc-flow formulation with graph compression on a large variety of applications. Finally, Chapter 7 presents some conclusions.

[1] `http://vpsolver.fdabrandao.pt` or `https://github.com/fdabrandao/vpsolver`

Chapter 2

Previous work on exact methods

In this chapter, we will give account of previous approaches with exact methods to bin packing and related problems. Valério de Carvalho (2002) provides an excellent survey on integer programming models for bin packing and cutting stock problems. Another recent and exhaustive survey for one-dimensional problems is presented in Delorme et al. (2016), but many of the included methods are not general enough to tackle most of the applications presented in this thesis. Here we will reuse some material from Brandão (2012), while keeping the details to the bare minimum, and presenting only the most relevant and flexible models.

The bin packing problem (BPP) and the cutting stock problem (CSP) are special cases of the vector packing problem (VBP). In this chapter, unless otherwise specified, instances for all the problems will be represented using vector packing notation as follows: p is the number of dimensions; m is the number of different item types; d_i is the demand for items of type i; and, for each dimension k, w_i^k is the weight of item i and W^k is the bin capacity. For the sake of simplicity the dimension may be omitted in the one-dimensional case.

Section 2.1 presents Martello and Toth's formulation for the BPP, and its generalization to the p-dimensional vector packing problem. Section 2.2 presents Kantorovich's formulation for the CSP. Section 2.3 presents Gilmore-Gomory's formulation, which can be derived from Kantorovich's formulation by applying Dantzig-Wolfe decomposition. Finally, Sections 2.4 and 2.5 present Valério de Carvalho's arc-flow formulation and Dyckhoff's one-cut formulation, respectively; both formulations are equivalent to Gilmore-Gomory's formulation and have a pseudo-polynomial number of variables and constraints.

2.1 Martello and Toth's formulation

Martello and Toth (1990) developed a branch-and-bound algorithm for the BPP based

on the following mathematical programming formulation:

$$\min \quad \sum_{j=1}^{J} y_j \tag{2.1.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{J} x_{ij} = 1, \qquad i = 1, \ldots, n, \tag{2.1.2}$$

$$\sum_{i=1}^{n} w_i x_{ij} \leq W y_j, \qquad j = 1, \ldots, J, \tag{2.1.3}$$

$$y_j \in \{0, 1\}, \qquad j = 1, \ldots, J, \tag{2.1.4}$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, \ldots, n,\ j = 1, \ldots, J, \tag{2.1.5}$$

where J is a known upper bound to the number of bins needed, n is the number of items, w_i is the weight of item i, W is the bin capacity, and the variables are:

$$y_j = \begin{cases} 1 & \text{if bin } j \text{ is used,} \\ 0 & \text{otherwise;} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{if item } i \text{ is assigned to bin } j, \\ 0 & \text{otherwise.} \end{cases}$$

Martello and Toth (1990) proved that the lower bound for the linear relaxation of this model, which is equal to the minimum amount of space that is necessary to accommodate all the items if they could be divided, can be very weak for instances with large waste.

Property 1 (linear relaxation)**.** *The lower bound provided by the linear relaxation of the model (2.1.1)–(2.1.4) is equal to* $\lceil \sum_{i=1}^{n} w_i/W \rceil$.

Property 2 (worst case)**.** *In the worst case, as* W *increases, when all the items have a size* $w_i = \lfloor W/2 + 1 \rfloor$*, the lower bound approaches* $1/2$ *of the optimal solution.*

Proof. If $w_i = \lfloor W/2+1 \rfloor$ then $\sum_{i=1}^{n} w_i = n\lfloor W/2+1 \rfloor \leq nW/2+n$. Therefore $\lceil \sum_{i=1}^{n} w_i/W \rceil \leq \lceil (nW/2 + n)/W \rceil = \lceil n/2 + n/W \rceil$. As W increases, this lower bound approaches $\lceil n/2 \rceil$ while the optimal solution is n. □

Caprara (1998) analyzed properties of the following generalization of Martello and Toth's

formulation to the p-dimensional vector packing problem:

$$\min \quad \sum_{j=1}^{J} y_j \tag{2.1.6}$$

$$\text{s.t.} \quad \sum_{j=1}^{J} x_{ij} = 1, \qquad i = 1, \ldots, n, \tag{2.1.7}$$

$$\sum_{i=1}^{n} w_i^k x_{ij} \leq W^k y_j, \qquad j = 1, \ldots, J, \ k = 1, \ldots, p, \tag{2.1.8}$$

$$x_{ij} \leq y_j, \qquad i = 1, \ldots, n, \ j = 1, \ldots, J, \tag{2.1.9}$$

$$y_j \in \{0, 1\}, \qquad j = 1, \ldots, n, \tag{2.1.10}$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, \ldots, n, \ j = 1, \ldots, J. \tag{2.1.11}$$

Property 3 (linear relaxation)**.** *The lower bound provided by the linear relaxation of the model (2.1.6)–(2.1.11) is equal to* $\max\{\sum_{i=1}^{n} w_i^k / W^k : k = 1, \ldots, p\}$.
Property 4 (worst case)**.** *The worst case performance ratio of the lower bound provided by model (2.1.6)–(2.1.11) is* $1/(p+1)$.

These properties are a huge drawback of this type of models, as good quality lower bounds are vital in branch-and-bound procedures. Another drawback is due to the symmetry of the solution space, which makes assignment-based models very inefficient in practice.

2.2 Kantorovich's formulation

The BPP and the CSP are equivalent, in the sense that from the solution of one we can derive the solution of the other; however, the BPP takes a list of items as input, while the CSP takes a list of different item sizes and the corresponding demands. The size of the input for the BPP can be exponentially larger than the input for the CSP. Therefore, a polynomial-size formulation for the BPP is not necessarily polynomial-size for the CSP.

Kantorovich (1960) introduced the following mathematical programming formulation for the CSP, where the objective is to minimize the number of rolls used to cut all the items

demanded:

$$\min \quad \sum_{j=1}^{J} y_j \tag{2.2.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{J} x_{ij} \geq d_i, \qquad i = 1, \ldots, m, \tag{2.2.2}$$

$$\sum_{i=1}^{m} w_i x_{ij} \leq W y_j, \qquad j = 1, \ldots, J, \tag{2.2.3}$$

$$y_j \in \{0, 1\}, \qquad j = 1, \ldots, J, \tag{2.2.4}$$

$$x_{ij} \geq 0, \text{ integer}, \qquad i = 1, \ldots, m, \ j = 1, \ldots, J, \tag{2.2.5}$$

where J is a known upper bound to the number of rolls needed, m is the number of different item sizes, w_i and d_i are the weight and demand of item i, and W is the roll size. The variables are y_j, which is 1 if roll j is used and 0 otherwise, and x_{ij}, the number of times item i is cut in the roll j.

In the worst case, the lower bound provided by this model approaches 1/2 of the optimal solution, since its lower bound is the same as the one provided by Martello and Toth's formulation.

Dantzig-Wolfe decomposition is a method for solving linear programming problems with a special structure (see, e.g., Dantzig and Wolfe 1960). It is a powerful tool that can be used to obtain models for integer and combinatorial optimization problems with stronger linear relaxations. Vance (1998) applied a Dantzig-Wolfe decomposition to model (2.2.1)–(2.2.5), keeping constraints (2.2.1), (2.2.2) in the master problem, and the subproblem being defined by the integer solutions to the knapsack constraints (2.2.3). Vance showed that when all the rolls have the same size, the reformulated model is equivalent to the classical Gilmore-Gomory's model.

2.3 Gilmore-Gomory's formulation

Gilmore and Gomory (1961) proposed the following model for the CSP. A combination of orders in the width of the roll is called a cutting pattern. Let column vectors $a^j = (a_1^j, \ldots, a_m^j)^\top$ represent all possible cutting patterns j. The element a_i^j represents the number of pieces of size w_i obtained in cutting pattern j. Let x_j be a decision variable that designates the number of rolls to be cut according to cutting pattern j. The CSP

can be modeled in terms of these variables as follows:

$$\min \quad \sum_{j \in J} x_j \tag{2.3.1}$$

$$\text{s.t.} \quad \sum_{j \in J} a_i^j x_j \geq d_i, \qquad i = 1, \ldots, m, \tag{2.3.2}$$

$$x_j \geq 0, \text{ integer}, \qquad j \in J, \tag{2.3.3}$$

where J is the set of valid cutting patterns that satisfy:

$$\sum_{i=1}^{m} a_i^j w_i \leq W \text{ and } a_i^j \geq 0, \text{ integer}. \tag{2.3.4}$$

Since constraints (2.3.4) just accept non-negative integer linear combinations of items, the search space of the continuous relaxation is reduced and the lower bound provided is stronger when compared with Kantorovich's formulation.

It may be impractical to enumerate all the columns in the previous formulation, as their number may be very large, even for moderately sized problems. To tackle this problem, Gilmore and Gomory (1963) proposed column generation.

Let the linear relaxation of model (2.3.1)–(2.3.3) for a $J' \subseteq J$ be the restricted master problem. At each iteration of the column generation process, a subproblem is solved and a column (pattern) is introduced in the restricted master problem if its reduced cost is strictly less than zero. The subproblem, which is a knapsack problem, is the following:

$$\min \quad 1 - \sum_{i=1}^{m} c_i a_i \tag{2.3.5}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} w_i a_i \leq W \tag{2.3.6}$$

$$a_i \geq 0, \text{ integer}, \qquad i = 1, \ldots, m, \tag{2.3.7}$$

where c_i is the shadow price of the demand constraint of item i obtained from the solution of the restricted master problem, and $a = (a_1, \ldots, a_m)^\top$ is a cutting pattern whose reduced cost is given by the objective function.

2.3.1 Integrality gap

There have been several studies (see, e.g., Scheithauer and Terno 1995, Scheithauer and Terno 1997) about the integrality gap of Gilmore-Gomory's model. For a cutting stock instance E, let $z^*_{lp}(E)$ be optimum value of the linear relaxation of Gilmore-Gomory's formulation, and $z^*_{ip}(E)$ be the optimal integer solution.

Definition 1 (Integer Property)**.** *A linear integer optimization problem P has the integer property (IP) if*

$$z^*_{ip}(E) = z^*_{lp}(E) \text{ for every instance } E \in P$$

Definition 2 (Integer Round-Up Property)**.** *A linear integer optimization problem P has the integer round-up property (IRUP) if*

$$z^*_{ip}(E) = \lceil z^*_{lp}(E) \rceil \text{ for every instance } E \in P$$

Definition 3 (Modified Integer Round-Up Property)**.** *A linear integer optimization problem P has the modified integer round-up property (MIRUP) if*

$$z^*_{ip}(E) \leq \lceil z^*_{lp}(E) \rceil + 1 \text{ for every instance } E \in P$$

Rietz et al. (2002a) describe families of instances of the one-dimensional cutting stock problem without the integer round-up property. One of the families is the so-called divisible case, where every item size w_i is a factor of the bin capacity W, which was firstly proposed by Nica (1994). Consider an instance belonging to the divisible case family with $W = 396$, items of sizes $w_1 = 132$, $w_2 = 99$, $w_3 = 36$, and demands $b_1 = 2$, $b_2 = 3$, $b_3 = 6$. The linear relaxation of the Gilmore-Gomory's formulation for this instance is $1.9621\ldots$ and the optimal solution is 3. This and other examples of such instances are presented in Scheithauer and Terno (1995) and Scheithauer and Terno (1997).

Gau (1994) presents an instance with a gap of 1.0666. The largest gap known so far is 7/6 and it was found by Rietz et al. (2002b). Scheithauer and Terno (1997) conjecture that the general one-dimensional cutting stock problem has the modified integer round-up property (MIRUP). Moreover, instances for the BPP and the CSP usually have the integer round-up property (IRUP).

Additionally, good solutions are usually found when rounding the linear programming (LP) solution. Rounding up the fractional variables of the LP solution of Gilmore and Gomory's model guarantees a heuristic solution of value at most $z^*_{lp}(E)+m$, since we need to round up at most one variable for each different item size in order to obtain a valid integer solution. Wäscher and Gau (1996) present more elaborate rounding heuristics

for Gilmore and Gomory's model that usually lead to the optimal solution in cutting stock instances. Note that these rounding heuristics usually work well in cutting stock instances where the demands are large, but they may have a poor performance in bin packing instances where the values of variables are often a fraction of unity.

2.4 Valério de Carvalho's arc-flow formulation

Wolsey (1977) proposed for the first time arc-flow formulations derived from dynamic programming recursions for modeling cutting and packing problems. Despite not presenting computational results, some properties that suggested computational advantages of such formulations were presented. For solving cutting and packing problems, computational experiments with arc-flow formulations were only performed much later by Valério de Carvalho (1999), where an arc-flow formulation was used as a basis to produce a branch-and-price algorithm.

Valério de Carvalho's arc-flow formulation has a set of flow conservation constraints and a set of demand constraints to ensure that the demand of every item is satisfied. The corresponding path-flow formulation is equivalent to the classical Gilmore-Gomory's formulation. Gilmore and Gomory's model provides a very strong linear relaxation, but it is potentially exponential in the number of variables with respect to the input size; Valério de Carvalho's model is usually much smaller, being pseudo-polynomial in terms of decision variables and constraints.

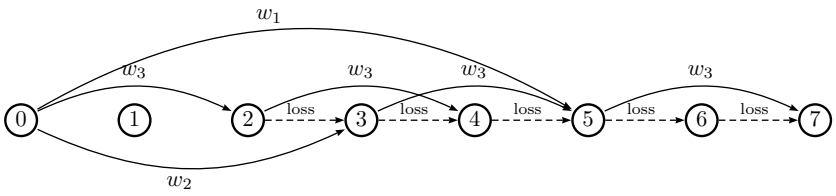
In the one-dimensional case, the problem of determining a valid solution to a single bin can be modeled as the problem of finding a path in a directed acyclic graph $G = (V, A)$ with $V = \{0, 1, 2, \ldots, W\}$ and $A = \{(i, j) \ : \ j - i = w_t, \text{ for } t = 1, \ldots, m \text{ and } 0 \leq i < j \leq W\}$, meaning that there exists an arc between two vertices i and $j > i$ if there are items of size $w_t = j - i$. The number of vertices and arcs are bounded by $\mathcal{O}(W)$ and $\mathcal{O}(mW)$, respectively. Additional arcs $(k, k + 1)$, for $k = w_{\min}, \ldots, W - 1$, with $w_{\min}$ being the minimum item size, are included for representing unoccupied portions of the bin.

In order to reduce the symmetry of the solution space and the size of the model, Valério de Carvalho introduced some rules. The idea is to consider only a subset of arcs from A. If we search for a solution in which the items are ordered by decreasing values of weight, only paths in which items appear according to this order must be considered.
Criterion 1. *An arc $(k, k + w_i)$ of size w_i can only leave a node $k > 0$ if there is another arc $(k - w_j, k)$ of size $w_j \geq w_i$ entering k; any arc can leave node $k = 0$.*
Criterion 2. *All the loss arcs $(k, k + 1)$ can be removed for $k < w_{\min}$ (recall that $w_{\min}$ is the smallest item).*

Criterion 3. *Given any node k that is the head of an arc of size w_j or $k = 0$, the only valid arcs for size w_i ($w_i < w_j$) are those that start at nodes $k + sw_i$, for $s = 0, 1, 2, \ldots, b_i - 1$, with $k + (s + 1)w_i \leq W$, where b_i is the demand of items of size w_i.*

Example 1. *Figure 2.4.1 shows the graph associated with an instance with capacity $W = 7$ and items of sizes 5, 3, 2 with demands 3, 1, 2, respectively.*

Figure 2.4.1: Graph corresponding to Example 1.



The BPP and the CSP are thus equivalently formulated as that of determining the minimum flow between vertex 0 and vertex W, with additional constraints enforcing the sum of the flows in the arcs for each item type to be greater than or equal to the corresponding demand. Consider decision variables x_{ij} (associated with arcs (i, j) defined above) corresponding to the number of items of size $j - i$ placed in any bin at a distance of i units from the beginning of the bin. A variable z, representing the number of bins required, aggregates the flow in the graph, and can be seen as a feedback arc from vertex W to vertex 0. The model is as follows:

$$\min \quad z \tag{2.4.1}$$

$$\text{s.t.} \quad \sum_{(i,j)\in A: j=k} x_{ij} - \sum_{(i,j)\in A: i=k} x_{ij} = \begin{cases} -z & \text{if } k = 0, \\ z & \text{if } k = W, \\ 0 & \text{for } k = 1, \ldots, W-1, \end{cases} \tag{2.4.2}$$

$$\sum_{(i,j)\in A: j=i+w_k} x_{ij} \geq d_k, \qquad i = 1, \ldots, m, \tag{2.4.3}$$

$$x_{ij} \geq 0, \text{ integer}, \qquad (i, j) \in A. \tag{2.4.4}$$

In Valério de Carvalho (2002), this model was generalized for variable-sized bin packing by using a feedback arcs for each bin size.

2.5 Dyckhoff's one-cut formulation

Dyckhoff (1981) proposed a formulation, equivalent to Gilmore-Gomory's formulation, in

which each decision variable corresponds to a single cutting operation performed on a piece of some size, which produces two pieces of smaller sizes. This is referred to as a one-cut. In this section, we use cutting stock terminology as this formulation was initially proposed to model the cutting stock problem.

Let $D = \{w_1, \ldots, w_m\}$ be the set of all order sizes, and let S be the set of sizes of stock material available ($S = \{W\}$ in the standard cutting stock problem). For the sake of simplicity, Dyckhoff (1981) assumes that all order sizes are different from standard sizes (i.e., $S \cap D = \varnothing$).

Let R be the set of residual sizes larger than the smallest order and that can be produced by one-cuts. Let d_l be the demand for pieces of size l; $d_l = 0$ for all $l \notin D$. Let $A_l = \{k \in S \cup R : k > l\}$ be the set of sizes that can be used to cut pieces of size $l \in D$; $A_l = \varnothing$ for all $l \notin D$. Let $B_l = \{k \in D : k + l \in S \cup R\}$ be the set of sizes that can produce a piece of size l as a residual size (i.e., if we cut a piece of size $k + l$ to produce a order of size $k \in D$, we produce a residual size l). Let $C_l = \{k \in D : k < l\}$ be the set of smaller sizes that can be produced using pieces of size l.

Let y_{kl} be the number of pieces of size k that are divided into a section of size $l \in D$ and a section of size $l - k \in R$. We model the standard cutting stock problem as follows:

$$\min \quad \sum_{l \in D} y_{Wl} \tag{2.5.1}$$

$$\text{s.t.} \quad \sum_{k \in A_l} y_{kl} + \sum_{k \in B_l} y_{k+l,l} \geq \sum_{k \in C_l} y_{lk} + d_l, \qquad l \in (D \cup R) \setminus S, \tag{2.5.2}$$

$$y_{kl} \geq 0, \text{ integer}, \qquad k \in S \cup R, l \in D. \tag{2.5.3}$$

In this model, we minimize the number of stock material of size W cut, while satisfying balance constraints. In the left hand side of constraint (2.5.2), we have the number of pieces of size l generated as an order ($\sum_{k \in A_l} y_{kl}$) plus the number of pieces generated as a residual size ($\sum_{k \in B_l} y_{k+l,l}$); in the right side, we have the number of pieces of size l used to produce smaller orders ($\sum_{k \in C_l} y_{lk}$) plus the number of pieces used to satisfy demand (d_l). Note that there are no balance constraints for standard sizes since they are available in unlimited quantities.

Model (2.5.1)–(2.5.3) is a simplified version of the original one-cut model which was proposed to model variable sized cutting stock problems. The only difference to the

original version is the objective:

$$\min \quad \sum_{l \in S} c_l \left(\sum_{k \in C_l} y_{lk} - \sum_{k \in B_l} y_{k+l,l} \right) \tag{2.5.4}$$

where c_l is the cost of standard size l. In the original objective, the number of pieces resulting from residual sizes is subtracted from the number of pieces used in order to obtain the number of stock pieces used.

This model has the disadvantage that there is more symmetry than in Gilmore-Gomory's model, but it has a pseudo-polynomial number of variables and constraints.

In model (2.5.1)–(2.5.3), it does not matter if a piece of a given size l was produced to satisfy an order ($\sum_{k \in A_l} y_{kl}$), or as a residual size ($\sum_{k \in B_l} y_{k+l,l}$). If we choose to make this distinction, the problem can be modeled as follows:

$$\min \quad \sum_{l \in D} y_{Wl} \tag{2.5.5}$$

$$\text{s.t.} \quad \sum_{k \in A'_l} y_{kl} \geq d_l, \qquad l \in D, \tag{2.5.6}$$

$$\sum_{k \in B_l} y_{k+l,l} \geq \sum_{k \in C'_l} y_{lk}, \qquad l \in R \setminus S, \tag{2.5.7}$$

$$y_{kl} \geq 0, \text{ integer}, \qquad k \in S \cup R, l \in D, \tag{2.5.8}$$

where $A'_l = A_l \cup \{l\}$ if $l \in D$, $A'_l = A_l$ otherwise, and $C'_l = C_l \cup \{l\}$ if $l \in D$, $C'_l = C_l$ otherwise. Model (2.5.1)–(2.5.3) can be derived from model (2.5.5)–(2.5.8) by eliminating variables y_{ll} for all $l \in D$.

Chapter 3

Deriving models from dynamic programming recursions

Various combinatorial optimization problems under the form $\max\{cx : x \in Q \subseteq \mathbb{R}^n\}$, where Q is a convex polytope for the feasible solution set of the problem, can be represented as discrete dynamic programming problems, or network problems. We refer to this type of problems as (P_0), and the main requirement is the existence of discrete dynamic programming representation of Q that can be transformed into a linear programming problem. Wolsey (1977) shows that such representations lead naturally to a characterization of the valid inequalities for the feasible solution sets Q of such problems. In particular, he obtains polytopes Γ of valid inequalities having the facets of the convex hull of Q among their extreme points. Moreover, he shows that cutting and packing problems, with P_0 as the underlying pattern feasibility problem, have natural network representations, which are duals of problems over Γ. In this chapter, we show how these natural network representations can be derived and how they lead to arc-flow and one-cut models.

Section 3.1 presents some preliminary concepts. Sections 3.2 and 3.3 analyze two types of polytopes Q, 0–1 monotone polytopes and integer monotone polytopes, respectively. Section 3.4 shows how the results from previous sections can be generalized to other problems. Finally, Section 3.5 presents some conclusions.

3.1 Preliminaries

Wolsey (1977) shows that the dynamic programming characterization of problems P_0 provides useful information to two related problems:

- P_1: Find a polytope Γ such that $(\pi; \pi_0) \in \Gamma$ if and only if $\pi x \leq \pi_0$ is a non-trivial valid inequality for Q;
- P_2: The covering problem $\min\{\mathbf{1} \cdot y : By \geq b, y \geq 0\}$ where the columns of B are vectors representing the feasible points of Q, and b is a non-negative integer vector.

Note that this is essentially the linear relaxation of Gilmore-Gomory's formulation for the cutting stock problem.

Definition 4 (Valid inequality)**.** *The inequality:*

$$\sum_{j=1}^{n} \pi_j x_j \leq \pi_0$$

is said to be a valid inequality, denoted $(\pi; \pi_0)$*, for* Q *if every feasible point in* Q *satisfies the inequality. A valid inequality is tight if there is some point of* Q *for which equality holds. A valid inequality is a facet of* Q *if there exist* n *affinely independent points of* Q *satisfying it with equality.*

Proposition 1. $\pi B \leq \pi_0 \mathbf{1}$ *if and only if* $(\pi; \pi_0) \in \Gamma$.

Proof. Let B^i be the i-th column of B, i.e., the i-th feasible point of Q.

$$\begin{aligned}
\pi B \leq \pi_0 \mathbf{1} &\Leftrightarrow \pi B^i \leq \pi_0 \text{ for all } i \\
&\Leftrightarrow \sum_{j=1}^{n} \pi_j B_j^i \leq \pi_0 \text{ for all } i \\
&\Leftrightarrow \sum_{j=1}^{n} \pi_j x_j \leq \pi_0 \text{ for every feasible point } x \in Q \\
&\Leftrightarrow (\pi; \pi_0) \in \Gamma
\end{aligned}$$

□

There are the following relationships between the problems P_0, P_1, and P_2:

- $(\mathrm{P}_1, \mathrm{P}_2)$: by duality and Proposition 1

$$\begin{aligned}
\min\{\mathbf{1} \cdot y : By \geq b, y \geq 0\} &= \max\{b\pi : \pi B \leq \mathbf{1}, \pi \geq 0\} \\
&= \max\{b\pi : \pi B \leq \pi_0 \mathbf{1}, \pi_0 \leq 1, \pi \geq 0\} \\
&= \max\{b\pi : (\pi; \pi_0) \in \Gamma, \pi_0 \leq 1, \pi \geq 0\}
\end{aligned}$$

- $(\mathrm{P}_0, \mathrm{P}_1)$: the problem P_0, $\max\{\pi x : x \in Q\}$, can be formulated as a DP recursion or as a network flow problem. The set of valid inequalities Γ can be obtained by constraining $(\pi; \pi_0)$ to be dual-feasible for the network problem. Moreover, the constraints of Γ can be derived directly from the DP recursion.
- $(\mathrm{P}_0, \mathrm{P}_2)$: The dual of $\max\{b\pi : (\pi; \pi_0) \in \Gamma, \pi_0 \leq 1, \pi \geq 0\}$, which is $\min\{\mathbf{1} \cdot y : By \geq b, y \geq 0\}$, gives a representation of the covering problem on the network

associated with P_0. Note that, unlike P_0, this problem is not totally unimodular due to capacity constraints involving several arcs simultaneously.

The proposed representation of Γ has two apparent advantages over other suggested representations: simplicity, and a limited number $\mathcal{O}(n\beta)$ of constraints and variables where β is the number of DP states.

Definition 5 (Monotone set)**.** *A set of non-negative integer vectors Y is monotone if for any non-negative integer vector x' such that $x' \leq x$, and $x \in Y$, then $x' \in Y$.*

In the next sections we shall only be concerned with valid inequalities with $\pi_0 \neq 0$. If Q is monotone, this only excludes the trivial inequalities $x_j \geq 0$.

3.2 0–1 monotone polytopes

Consider the 0–1 monotone set Q:

$$\sum_{j=1}^{n} w_j x_j \leq W, x_j \in \{0,1\}, \text{ for } j = 1, \ldots, n,$$

where $\{w_j \in \mathbb{Z}_+^p : j = 1, \ldots, n\}$ is a set of weight vectors, and $W \in \mathbb{Z}_+^p$ is a capacity vector; both being p-dimensional non-negative integer column vectors.

This monotone set Q corresponds to a multi-constraint binary knapsack solution set, where column vectors w_j are the weight of item j in each dimension, and W is the capacity vector. The problem P_0 over Q, $\max\{\pi x : x \in Q \subseteq \mathbb{R}^n\} = \max\{\pi x : \sum_{j=1}^{n} w_j x_j \leq W, x_j \in \{0,1\}\})$, where π_j is the value of item j, is a multi-constraint binary knapsack problem that can be represented as a discrete dynamic programming problem as follows.

Let

$$Q_r(\lambda) = \left\{ x : \sum_{j=1}^{r} w_j x_j \leq \lambda, x_j \in \{0,1\} \right\},$$

$$G_r(\lambda) = \max \left\{ \sum_{j=1}^{r} \pi_j x_j : x \in Q_r(\lambda) \right\},$$

where $0 \leq r \leq n$, $\lambda \in \mathbb{Z}_+^p$, and $\lambda \leq W$. $Q_r(\lambda)$ is the subset of Q considering only the first r items and capacity λ. $G_r(\lambda)$ is the maximum profit considering only the first r items and capacity λ.

Note that $Q = Q_n(W)$ and that $G_r(\lambda) = \max(G_{r-1}(\lambda), G_{r-1}(\lambda - w_r) + \pi_r)$, where $G_0(\lambda) =$

0 for $0 \leq \lambda \leq W$, and any expression containing undefined terms is ignored. More formally:

$$G_r(\lambda) = \begin{cases} \max(G_{r-1}(\lambda - w_r) + \pi_r, G_{r-1}(\lambda)) & r = 2, \ldots, n, w_r \leq \lambda, \\ G_{r-1}(\lambda) & r = 2, \ldots, n, w_r > \lambda, \\ \max(\pi_1, 0) & r = 1, w_1 \leq \lambda, \\ 0 & r = 1, w_1 > \lambda. \end{cases} \tag{3.2.1}$$

Note that $\pi x \leq \pi_0$ is a valid inequality for Q if and only if $\pi_0 \geq G_n(W)$. It is tight if $\pi_0 = G_n(W)$. Recall that $G_n(W) = \max\{\pi x : x \in Q\}$.

Let us write P_0, $\max\{\pi x : x \in Q\}$, as a network flow problem with nodes (r, λ), for $0 \leq r \leq n$, $0 \leq \lambda \leq W$, edges $((r-1, \lambda - w_r), (r, \lambda))$ with flows $\xi(\lambda)$, and $((r-1, \lambda), (r, \lambda))$ with flows $\eta_r(\lambda)$, for $r = 1, \ldots, n$, $0 \leq \lambda \leq W$, if both endpoints of the edge are legitimate nodes.

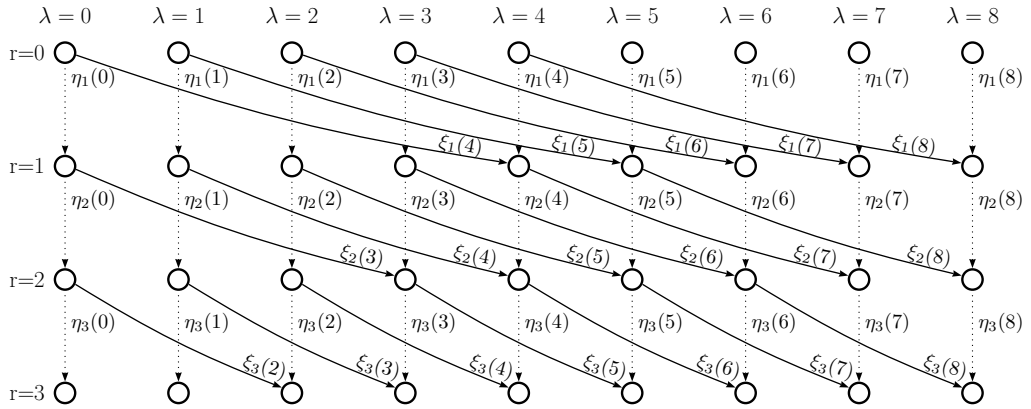
Proposition 2. *Problem P_0 is equivalent to the totally unimodular flow problem FP_0:*

$$\begin{aligned} G_n(W) = \max \quad & \sum_{\lambda} \sum_{r} \pi_r \xi_r(\lambda) \\ s.t. \quad & \xi_r(\lambda) + \eta_r(\lambda) - \xi_{r+1}(\lambda + w_{r+1}) - \eta_{r+1}(\lambda) = 0, \\ & \hookrightarrow \quad r = 1, \ldots, n-1, \quad 0 \leq \lambda \leq W, \\ & \xi_n(\lambda) + \eta_n(\lambda) = 0, \\ & \hookrightarrow \quad 0 \leq \lambda < W, \\ & \xi_n(W) + \eta_n(W) = 1, \\ & \xi_r(\lambda), \eta_r(\lambda) \geq 0, \\ & \hookrightarrow \quad r = 1, \ldots, n, \quad 0 \leq \lambda \leq W. \end{aligned}$$

Proof. Wolsey (1977) proves this equivalence as follows. Note that, with his choice of notation, $x \in Q$ with $wx = W - \mu$, where μ corresponds to unused space, corresponds to a path in the network from $(0, \mu)$ to (n, W) or a feasible solution of FP_0. Hence, $G_n(W) \leq \sum_\lambda \sum_r \pi_r \eta_r(\lambda)$. Conversely, the linear program has an optimal solution which is integer. It therefore corresponds to a path from $(0, \mu)$ to (n, W), and to an $x \in Q$ with $wx = W - \mu$. Hence $\sum_\lambda \sum_r \pi_r \eta_r(\lambda) \leq G_n(W)$. □

Figure 3.2.1 shows a small example ($n = 3$, $p = 1$) for a 0–1 knapsack instance with capacity $W = 8$ and items of sizes $w_1 = 4$, $w_2 = 3$, $w_3 = 2$. The arcs are labeled with the corresponding variables in the arc-flow model FP_0.

Figure 3.2.1: Arc-flow model for a 0–1 knapsack instance.



Theorem 1. $(\pi;\pi_0)$ *is a valid inequality for* Q *if and only if there exist values* $\theta_r(\lambda)$, $r = 1,\dots,n$, $0 \le \lambda \le W$ *such that* $(\pi;\theta_n(W)) \in \Gamma$ *with* $\pi_0 = \theta_n(W)$ *where:*

$$\begin{aligned}\Gamma &= \{(\pi;\theta_n(W)) : \theta_r(\lambda) - \theta_{r-1}(\lambda - w_r) - \pi_r \ge 0,\ \theta_r(\lambda) - \theta_{r-1}(\lambda) \ge 0, \forall r, \forall\lambda\}\\ &= \{(\pi;\theta_n(W)) : \theta_r(\lambda) \ge \theta_{r-1}(\lambda - w_r) + \pi_r,\ \theta_r(\lambda) \ge \theta_{r-1}(\lambda), \forall r, \forall\lambda\}\\ &= \{(\pi;\theta_n(W)) : \theta_r(\lambda) \ge \max(\theta_{r-1}(\lambda - w_r) + \pi_r, \theta_{r-1}(\lambda)), \forall r, \forall\lambda\}\end{aligned}$$

where again undefined terms vanish. More formally:

$$\Gamma = \left\{ (\pi;\theta_n(W)) : \begin{array}{ll} \theta_r(\lambda) \ge \max(\theta_{r-1}(\lambda - w_r) + \pi_r, \theta_{r-1}(\lambda)), & r = 2,\dots,n,\\ & w_r \le \lambda \le W,\\ \theta_r(\lambda) \ge \theta_{r-1}(\lambda), & r = 2,\dots,n,\\ & w_r > \lambda,\\ \theta_r(\lambda) \ge \max(\pi_1, 0), & r = 1, w_1 \le \lambda,\\ \theta_r(\lambda) \ge 0, & r = 1, w_1 > \lambda \end{array} \right\} \tag{3.2.2}$$

Proof. Wolsey (1977) proves this theorem as follows. $(\pi;\theta_n(W))$ is a valid inequality for Q if and only if $\theta_n(W) \ge G_n(W)$. Taking the dual of FP_0, he obtains $\min\{\theta_n(W) : (\pi;\theta_n(W)) \in \Gamma\} = G_n(W)$, and hence if $(\pi;\theta_n(W)) \in \Gamma$ with $\pi_0 = \theta_n(W)$, then $(\pi;\pi_0)$ is valid for Q. The converse is immediate taking $\theta_r(\lambda) = G_r(\lambda)$ as defined in (3.2.1).

□

Proposition 3. Γ *can be obtained directly by replacing* $G_r(\lambda)$ *by* $\theta_r(\lambda)$ *in the DP recursion.*

Proof. Recall that $\pi x \le \pi_0$ is a valid inequality for Q if and only if $\pi_0 \ge G_n(W)$. It is

tight if $\pi_0 = G_n(W)$. We define the set Γ' of all tight valid inequalities of Q as follows:

$$\Gamma' = \{(\pi; G_n(W)) : \pi \in \mathbb{R}^n\}$$

$$\text{where } G_r(\lambda) = \begin{cases} \max(G_{r-1}(\lambda - w_r) + \pi_r, G_{r-1}(\lambda)) & r = 2, \ldots, n, w_r \leq \lambda \leq W, \\ G_{r-1}(\lambda) & r = 2, \ldots, n, w_r > \lambda, \\ \max(\pi_1, 0) & r = 1, w_1 \leq \lambda, \\ 0 & r = 1, w_1 > \lambda. \end{cases} \tag{3.2.3}$$

Replacing $G_r(\lambda)$ by $\theta_r(\lambda)$ in (3.2.3), we obtain the following:

$$\Gamma' = \left\{ (\pi; \theta_n(W)) : \begin{array}{ll} \theta_r(\lambda) = \max(\theta_{r-1}(\lambda - w_r) + \pi_r, \theta_{r-1}(\lambda)), & r = 2, \ldots, n, \\ & w_r \leq \lambda \leq W, \\ \theta_r(\lambda) = \theta_{r-1}(\lambda), & r = 2, \ldots, n, \\ & w_r > \lambda, \\ \theta_r(\lambda) = \max(\pi_1, 0), & r = 1, w_1 \leq \lambda, \\ \theta_r(\lambda) = 0, & r = 1, w_1 > \lambda \end{array} \right\}$$

and $\Gamma = \{(\pi; \pi_0) : (\pi, \pi_0') \in \Gamma', \pi_0 \geq \pi_0'\}$, in (3.2.2), is the set of all valid inequalities of Q. □

Note that when we are solving a knapsack problem using dynamic programming, we are essentially computing $\theta_r(\lambda)$ values. The solution extraction procedure usually consists of going from state to state checking whether item r is used in the optimal path (if $\theta_r(\lambda) = \theta_{r-1}(\lambda - w_r) + \pi_r$), or not (if $\theta_r(\lambda) = \theta_{r-1}(\lambda)$). This procedure essentially computes a primal solution from the values of the dual.

Theorem 2. *If $\sum_{j=1}^{n} \pi_j x_j \leq \pi_0$ is a non-trivial facet of Q, then (π, π_0) is an extreme point of Γ with $\pi_0 = G_n(W)$.*

Proof. Wolsey (1977) proves this theorem as follows. Suppose that the implication is not true. Then

$$(\pi, G_n(W)) = \alpha(\pi^1, \theta_n^1(W)) + (1 - \alpha)(\pi^2, \theta_n^2(W)), 0 < \alpha < 1$$

Case (a). $\pi^i \neq \pi$, $i = 1, 2$. This contradicts the fact that a non-trivial facet of conv(Q) is extreme among the valid inequalities for Q. Case (b). $\pi^1 = \pi^2 = \pi$. Then as $(\pi^i, \theta_n^i(W)) \in \Gamma$, $\theta_n^i(W) \geq G_n(W)$, $i = 1, 2$. However, $\alpha\pi_0^1 + (1 - \alpha)\pi_0^2 = G_n(W)$, and hence $\pi_0^i = G_n(W)$, $i = 1, 2$, contradicting the hypothesis that $(\pi^1, \theta_n^1(W))$ and $(\pi^2, \theta_n^2(W))$ are distinct. Therefore, the extreme points of Γ, restricted to the variables $(\pi, \theta_n(W))$, include the non-trivial facets of Q. □

Remark 1. *Wolsey (1977) notes that the polytope* Γ *is very easy to describe and has at most* $(n+1)\beta$ *variables and* $2n\beta$ *constraints where* $\beta = \prod_{i=1}^{p}(W^i + 1)$.

Consider now the covering problem P_2, and in particular its representation in the form $\max\{b\pi : (\pi;\pi_0) \in \Gamma, \pi_0 \leq 1, \pi \geq 0\}$. Taking its dual we obtain the network flow problem:

$$
\begin{aligned}
\min \quad & z_0 \\
\text{s.t.} \quad & \xi_r(\lambda) + \eta_r(\lambda) - \xi_{r+1}(\lambda + w_{r+1}) - \eta_{r+1}(\lambda) = 0, \\
& \hookrightarrow \qquad r = 1, \ldots, n-1, \quad 0 \leq \lambda \leq W, \\
& \xi_n(\lambda) + \eta_n(\lambda) = 0, \\
& \hookrightarrow \qquad 0 \leq \lambda < W, \\
& \xi_n(W) + \eta_n(W) = z_0, \\
& \sum_{\lambda} \eta_r(\lambda) \geq b_r, \\
& \hookrightarrow \qquad r = 1, \ldots, n, \\
& z_0 \geq 0, \\
& \xi_r(\lambda), \eta_r(\lambda) \geq 0, \\
& \hookrightarrow \qquad r = 1, \ldots, n, \quad 0 \leq \lambda \leq W.
\end{aligned}
$$

This problem involves the same network as in FP_0 where the first three constraint sets represent flow feasibility constraints, but the last "covering" set of constraints imposes a minimum aggregate flow over certain subsets of arcs, and destroys the property of total unimodularity.

3.2.1 Examples

Example 2. *Consider the following 0–1 knapsack problem* (P_0) *with* $n = 2$ *and* $p = 1$*:*

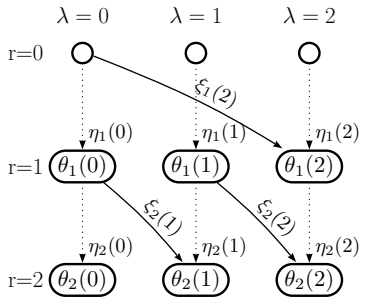
$$
\begin{aligned}
\max \quad & \pi_1 x_1 + \pi_2 x_2 \\
s.t. \quad & 2x_1 + x_2 \leq 2, \\
& x_1, x_2 \in \{0, 1\}.
\end{aligned}
$$

Problem P_0 can be solved using the following dynamic programming recursion:

$$\begin{aligned}
G_0(0) &= 0 \\
G_0(1) &= 0 \\
G_0(2) &= 0 \\
G_1(0) &= \max\{G_0(0)\} \\
G_1(1) &= \max\{G_0(1)\} \\
G_1(2) &= \max\{G_0(0) + \pi_1, G_0(2)\} \\
G_2(0) &= \max\{G_1(0)\} \\
G_2(1) &= \max\{G_1(0) + \pi_2, G_1(1)\} \\
G_2(2) &= \max\{G_1(1) + \pi_2, G_1(2)\}
\end{aligned}$$

Figure 3.2.2 shows the arc-flow model for Example 2 that can be derived from the dynamic programming recursion. Nodes can be seen as states and arcs as recursive calls for subproblems.

Figure 3.2.2: Arc-flow model for Example 2.



The following totally unimodular flow problem FP_0 is equivalent to P_0:

$$\begin{aligned}
\max \quad & \pi_1\xi_1(2) + \pi_2\xi_2(1) + \pi_2\xi_2(2) \\
\text{s.t.} \quad & \eta_1(0) - \xi_2(1) - \eta_2(0) = 0, \quad (\text{dual: } \theta_1(0)) \\
& \eta_1(1) - \xi_2(2) - \eta_2(1) = 0, \quad (\text{dual: } \theta_1(1)) \\
& \xi_1(2) + \eta_1(2) - \eta_2(2) = 0, \quad (\text{dual: } \theta_1(2)) \\
& \eta_2(0) = 0, \quad (\text{dual: } \theta_2(0)) \\
& \xi_2(1) + \eta_2(1) = 0, \quad (\text{dual: } \theta_2(1)) \\
& \xi_2(2) + \eta_2(2) = 1, \quad (\text{dual: } \theta_2(2)) \\
& \xi_1(2), \xi_2(1), \xi_2(2) \geq 0, \\
& \eta_1(0), \eta_1(1), \eta_1(2), \eta_2(0), \eta_2(1), \eta_2(2) \geq 0.
\end{aligned}$$

And its dual, $\min\{\theta_2(2) : (\pi;\theta_2(2)) \in \Gamma\}$, is the following:

$$\begin{array}{rll}
\min & \theta_2(2) & \\
\text{s.t.} & \theta_1(0) \geq 0, & (\text{dual: } \eta_1(0)) \\
& \theta_1(1) \geq 0, & (\text{dual: } \eta_1(1)) \\
& \theta_1(2) \geq 0, & (\text{dual: } \eta_1(2)) \\
& -\theta_1(0)+\theta_2(0) \geq 0, & (\text{dual: } \eta_2(0)) \\
& -\theta_1(1)+\theta_2(1) \geq 0, & (\text{dual: } \eta_2(1)) \\
& -\theta_1(2)+\theta_2(2) \geq 0, & (\text{dual: } \eta_2(2)) \\
& \theta_1(2) \geq \pi_1, & (\text{dual: } \xi_1(2)) \\
& -\theta_1(0)+\theta_2(1) \geq \pi_2, & (\text{dual: } \xi_2(1)) \\
& -\theta_1(1)+\theta_2(2) \geq \pi_2, & (\text{dual: } \xi_2(2)) \\
& \theta_1(0),\theta_1(1),\theta_1(2),\theta_2(0),\theta_2(1),\theta_2(2) \in \mathbb{R}. &
\end{array}$$

The polytope Γ of valid inequalities for P_0 is the projection of the dual-space of FP_0 onto $(\pi;\theta_2(2))$, i.e., $\Gamma = \{(\pi;\theta_2(2)) : \theta_1(0) \geq 0, \theta_1(1) \geq 0, \theta_1(2) \geq 0, -\theta_1(0)+\theta_2(0) \geq 0, -\theta_1(1)+\theta_2(1) \geq 0, -\theta_1(2)+\theta_2(2) \geq 0, \theta_1(2) \geq \pi_1, -\theta_1(0)+\theta_2(1) \geq \pi_2, -\theta_1(1)+\theta_2(2) \geq \pi_2, \theta_r(\lambda) \in \mathbb{R}, \pi \in \mathbb{R}^2\}$.

With $\pi_1 = 3$, $\pi_2 = 4$, the optimal solution of FP_0 is $\eta_1(1) = 1$, $\xi_2(2) = 1$ (with all other variables at zero) and corresponds to the path $(r = 0, \lambda = 1), (r = 1, \lambda = 1), (r = 2, \lambda = 2)$. The optimal solution of its dual is $\theta_1(2) = 3$, $\theta_2(1) = 4$, $\theta_2(2) = 4$ (with all other variables at zero). The optimal solution objective is 4.

The matrix B of feasible solutions to the problem P_0, where each column represents a feasible point of Q, is the following:

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\pi B \leq \pi_0 \mathbf{1} \Leftrightarrow \sum_{j=1}^{n} \pi_j B_j^i \leq \pi_0 \text{ for all } i \Leftrightarrow \begin{cases} \pi_1 \leq \pi_0 \\ \pi_2 \leq \pi_0 \end{cases}$$

Another possible definition for the polytope Γ, with one constraint for each feasible solution, is the following: $\Gamma = \{(\pi;\pi_0) : \pi_1 \leq \pi_0, \pi_2 \leq \pi_0, \pi_0 \in \mathbb{R}, \pi \in \mathbb{R}^2\}$. In this

example the polytope is easier to describe in this way, since there are just two feasible solutions. Although it is efficient for instances with very few feasible solutions, this description can rarely be used in practice directly due to the large number of feasible solutions.

The covering problem P_2, $\min\{\mathbf{1} \cdot y : By \geq b, y \geq 0\}$, which is equal by duality to $\max\{b\pi : (\pi; \pi_0) \in \Gamma, \pi_0 \leq 1, \pi \geq 0\}$, can be modeled by the dual of $\max\{b\pi : (\pi; \theta_2(2)) \in \Gamma, \theta_2(2) \leq 1, \pi \geq 0\}$, with Γ obtained from the DP recursion, as follows:

$$
\begin{aligned}
\min \quad & z_0 \\
\text{s.t.} \quad & \eta_1(0) - \eta_2(0) - \xi_2(1) = 0, \\
& \eta_1(1) - \eta_2(1) - \xi_2(2) = 0, \\
& \eta_1(2) - \eta_2(2) - \xi_1(2) = 0, \\
& \eta_2(0) = 0, \\
& \eta_2(1) + \xi_2(1) = 0, \\
& \eta_2(2) + \xi_2(2) = z_0, \\
& \xi_1(0) \geq b_1, \\
& \xi_2(0) + \xi_2(1) \geq b_2, \\
& \xi_1(0), \xi_2(0), \xi_2(1) \geq 0, \\
& \eta_1(0), \eta_1(1), \eta_1(2), \eta_2(0), \eta_2(1), \eta_2(2) \geq 0.
\end{aligned}
$$

Note that this last model corresponds to a cutting stock problem with binary patterns. The rolls have length 2 (the capacity of the knapsack constraint), and the items of sizes 2, 1 (the weights in the knapsack constraint) have demands b_1, b_2.

3.3 Integer monotone polytopes

Consider the integer monotone set Q:

$$\sum_{j=1}^{n} w_j x_j \leq W, x_j \geq 0 \text{ and integer, for } j = 1, \ldots, n,$$

where $\{w_j \in \mathbb{Z}_+^p : j = 1, \ldots, n\}$ is a set of weight vectors, and $W \in \mathbb{Z}_+^p$ is a capacity vector; both being p-dimensional non-negative integer column vectors.

This monotone set Q corresponds to a multi-constraint integer knapsack solution set,

where column vectors w_j are the weight of each item j in each dimension, and W is the capacity vector. The problem P_0 over Q, $\max\{\pi x : x \in Q \subseteq \mathbb{R}^n\} = \max\{\pi x : \sum_{j=1}^n w_j x_j \leq W, x_j \in \mathbb{Z}_+\}$, where π_j is the value of item j, is a multi-constraint integer knapsack problem that can be represented as a discrete dynamic programming problem as follows.

Defining $G(\lambda) = \max\{\sum_{j=1}^n \pi_j x_j : \sum_{j=1}^n w_j x_j \leq \lambda, x_j \geq 0 \text{ and integer}\}$, we obtain from dynamic programming the recursion:

$$G(\lambda) = \max\left(0, \max_{j=1,\ldots,n}\{G(\lambda - w_j) + \pi_j : w_j \leq \lambda\}\right)$$

where $\lambda \in \mathbb{Z}_+^p$, $\lambda \leq W$, and undefined terms are ignored. $G(\lambda)$ is the maximum profit with capacity λ.

From the dynamic programming recursion $G(\lambda)$, Wolsey (1977) derives the following totally unimodular flow problem:

$$\begin{aligned} G(W) = \max \quad & \sum_{\lambda}\sum_{j} \pi_j \xi_{\lambda - w_j, \lambda} & \\ \text{s.t.} \quad & -\sum_{j} \xi_{0,w_j} & \leq 0, \\ & \sum_{j} \xi_{\lambda - w_j, \lambda} - \sum_{j} \xi_{\lambda, \lambda + w_j} \leq 0, & \quad 0 < \lambda \in \mathbb{Z}_+^p < W, \\ & \sum_{j} \xi_{W - w_j, W} & \leq 1, \\ & \xi_{\lambda - w_j, \lambda} \geq 0. & \end{aligned}$$

Lemma 1. *$(\pi; \pi_0)$ is a valid inequality for Q if and only if there exist values $\theta(\lambda)$, $0 \leq \lambda \leq W$, $\lambda \in \mathbb{Z}_+^p$ such that $(\pi; \theta(W)) \in \Gamma$ with $\pi_0 = \theta(W)$ where:*

$$\begin{aligned} \Gamma &= \{(\pi; \theta(W)) : \theta(\lambda) - \theta(\lambda - w_j) \geq \pi_j, \theta(\lambda) \geq 0, \ \forall j, \forall \lambda\} \\ &= \{(\pi; \theta(W)) : \theta(\lambda) \geq \max(0, \max_{j=1}^n \{\theta(\lambda - w_j) + \pi_j : w_j \leq \lambda\}), \ 0 \leq \lambda \in \mathbb{Z}_+^p \leq W\} \end{aligned} \tag{3.3.1}$$

Proof. $(\pi; \theta_n(W))$ is a valid inequality for Q if and only if $\theta(W) \geq G(W)$. Taking the dual of flow model above we obtain $\min\{\theta(W) : (\pi; \theta(W)) \in \Gamma\} = G(W)$, and hence if $(\pi; \theta(W)) \in \Gamma$ with $\pi_0 = \theta(W)$, then $(\pi; \pi_0)$ is valid for Q. The converse is immediate taking $\theta(\lambda) = G(\lambda)$. □

Once again the polytope Γ has been derived directly from the dynamic programming

recursion. Moreover, the covering problem P_2 can now be modeled as follows (R_1):

$$\max\{b\pi : (\pi; \theta(W)) \in \Gamma, \theta(W) \leq 1, \pi \geq 0\} \tag{3.3.2}$$

Wolsey (1977) notes that this model has $n + \beta$ variables and $n\beta$ constraints, where $\beta = \prod_{i=1}^{p}(W^i + 1)$, and hence in its applications in the cutting stock problem it has far fewer constraints than the Gilmore-Gomory's formulation in the worst case.

The dual of (3.3.2) is the following network flow problem (R_2):

$$\begin{aligned}
\min \quad & z_0 && && (3.3.3)\\
\text{s.t.} \quad & + \sum_j \xi_{0,w_j} && \geq 0, && (3.3.4)\\
& - \sum_j \xi_{\lambda-w_j,\lambda} + \sum_j \xi_{\lambda,\lambda+w_j} && \geq 0, \quad 0 < \lambda \in \mathbb{Z}_+^p < W, && (3.3.5)\\
& - \sum_j \xi_{W-w_j,W} + z_0 && \geq 0, && (3.3.6)\\
& \sum_\lambda \xi_{\lambda-w_j,\lambda} && \geq b_j, \quad j = 1, \ldots, n, && (3.3.7)\\
& \xi_{\lambda-w_j,\lambda} \geq 0, z_0 \geq 0. && && (3.3.8)
\end{aligned}$$

This network flow problem with demand constraints, where $\xi_{\lambda-w_j,\lambda}$ is the number of cutting patterns with a piece of size w_j cut between $\lambda - w_j$ and λ, models vector packing problems.

Please note that, in model (3.3.3)–(3.3.8), (3.3.4) is redundant and the variable z_0 can be eliminated. By doing so, we obtain the following model (R_3):

$$\begin{aligned}
\min \quad & \sum_j \xi_{W-w_j,W} && && (3.3.9)\\
\text{s.t.} \quad & \sum_\lambda \xi_{\lambda-w_j,\lambda} && \geq b_j, \quad j = 1, 2, \ldots, n, && (3.3.10)\\
& \sum_j \xi_{\lambda,\lambda+w_j} && \geq \sum_j \xi_{\lambda-w_j,\lambda}, \quad 0 < \lambda \in \mathbb{Z}_+^p < W, && (3.3.11)\\
& \xi_{\lambda-w_j,\lambda} \geq 0. && && (3.3.12)
\end{aligned}$$

In the cutting stock problem, variables $\xi_{\lambda-w_j,\lambda}$ can be interpreted as the number of times a piece of size λ is cut into a piece of size w_j and into a residual piece of size $\lambda - w_j$. For $p = 1$, model (3.3.9)–(3.3.12) corresponds to the one-cut model (2.5.5)–(2.5.8).

Wolsey (1977) shows that several other alternative representations of Γ could also be used

and that column generation can also be applied to the network flow models.

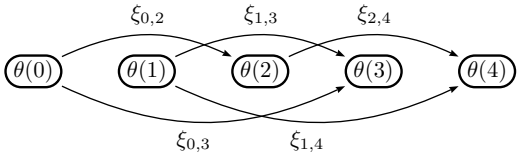
3.3.1 Examples

3.3.1.1 Primal/dual

Consider a cutting stock instance with capacity $W = 4$ and two item types, one of size $w_1 = 3$ with demand b_1, and another of size $w_2 = 2$ and demand b_2. This instance can be modeled using model (3.3.3)–(3.3.8) as follows:

$$\begin{aligned}
\min \quad & z_0 && && (3.3.13)\\
\text{s.t.} \quad & \xi_{0,3} + \xi_{0,2} && \geq 0, \quad \text{(dual: } \theta(0)\text{)} && (3.3.14)\\
& \xi_{1,4} + \xi_{1,3} && \geq 0, \quad \text{(dual: } \theta(1)\text{)} && (3.3.15)\\
& -\xi_{0,2} + \xi_{2,4} && \geq 0, \quad \text{(dual: } \theta(2)\text{)} && (3.3.16)\\
& -\xi_{0,3} - \xi_{1,3} && \geq 0, \quad \text{(dual: } \theta(3)\text{)} && (3.3.17)\\
& -\xi_{1,4} - \xi_{2,4} + z_0 && \geq 0, \quad \text{(dual: } \theta(4)\text{)} && (3.3.18)\\
& \xi_{0,3} + \xi_{1,4} && \geq b_1, \quad \text{(dual: } \pi_1\text{)} && (3.3.19)\\
& \xi_{0,2} + \xi_{1,3} + \xi_{2,4} && \geq b_2, \quad \text{(dual: } \pi_2\text{)} && (3.3.20)\\
& \xi_{0,3}, \xi_{1,4}, \xi_{0,2}, \xi_{1,3}, \xi_{2,4}, z_0 \geq 0. && && (3.3.21)
\end{aligned}$$

Figure 3.3.1: Graph associated with model (3.3.13)–(3.3.21).



The polytope Γ of valid inequalities for the underlying knapsack problem is the following: $\Gamma = \{(\pi; \theta(4)) : \theta(3) - \theta(0) \geq \pi_1, \theta(4) - \theta(1) \geq \pi_1, \theta(2) - \theta(0) \geq \pi_2, \theta(3) - \theta(1) \geq \pi_2, \theta(4) - \theta(2) \geq \pi_2, \theta(i) \in \mathbb{R}, \pi \in \mathbb{R}^2\}$. It can be easily derived from the dynamic programming recursion in Figure 3.3.1.

The dual of model (3.3.13)–(3.3.21), $\max\{b\pi : (\pi; \theta(W)) \in \Gamma, \theta(W) \leq 1, \pi \geq 0\}$, is the

following:

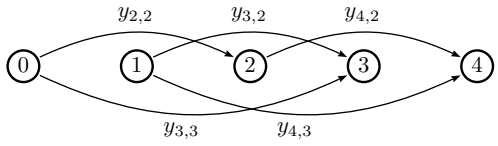
$$\begin{aligned}
\max \quad & b_1\pi_1 + b_2\pi_2 && (3.3.22)\\
\text{s.t.} \quad & \theta(0) - \theta(3) + \pi_1 \le 0, \quad (\text{dual: } \xi_{0,3}) && (3.3.23)\\
& \theta(1) - \theta(4) + \pi_1 \le 0, \quad (\text{dual: } \xi_{1,4}) && (3.3.24)\\
& \theta(0) - \theta(2) + \pi_2 \le 0, \quad (\text{dual: } \xi_{0,2}) && (3.3.25)\\
& \theta(1) - \theta(3) + \pi_2 \le 0, \quad (\text{dual: } \xi_{1,3}) && (3.3.26)\\
& \theta(2) - \theta(4) + \pi_2 \le 0, \quad (\text{dual: } \xi_{2,4}) && (3.3.27)\\
& \theta(4) \le 1, \quad (\text{dual: } z_0) && (3.3.28)\\
& \theta(0), \theta(1), \theta(2), \theta(3), \theta(4), \pi_0, \pi_1 \ge 0. && (3.3.29)
\end{aligned}$$

3.3.1.2 Dyckhoff's one-cut model

The following model, using the one-cut formulation (2.5.5)–(2.5.8), is exactly model (3.3.13)–(3.3.21) with different variable names and without redundant constraints:

$$\begin{aligned}
\min \quad & y_{4,2} + y_{4,3} && (3.3.30)\\
\text{s.t.} \quad & 0 \ge y_{3,3} + y_{3,2}, && (3.3.31)\\
& y_{4,2} \ge y_{2,2}, && (3.3.32)\\
& y_{4,3} + y_{3,2} \ge 0, && (3.3.33)\\
& y_{3,3} + y_{4,3} \ge b_1, && (3.3.34)\\
& y_{2,2} + y_{3,2} + y_{4,2} \ge b_2, && (3.3.35)\\
& y_{3,3}, y_{4,3}, y_{2,2}, y_{3,2}, y_{4,2} \ge 0. && (3.3.36)
\end{aligned}$$

Figure 3.3.2: Graph associated with model (3.3.30)–(3.3.36).



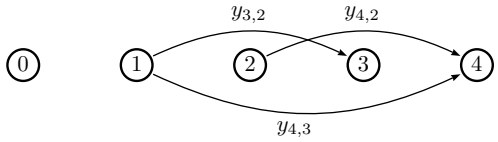
The following model, using Dyckhoff's one-cut formulation (2.5.1)–(2.5.3), is the result of eliminating the variables $y_{2,2}$ and $y_{3,3}$ from model (3.3.30)–(3.3.36):

$$\begin{aligned}
\min \quad & y_{4,3} + y_{4,2} && (3.3.37)\\
\text{s.t.} \quad & y_{4,3} \ge y_{3,2} + b_1, && (3.3.38)
\end{aligned}$$

$$y_{3,2} + y_{4,2} + y_{4,2} \geq b_2, \tag{3.3.39}$$

$$y_{4,3}, y_{3,2}, y_{4,2} \geq 0. \tag{3.3.40}$$

Figure 3.3.3: Graph associated with model (3.3.37)–(3.3.40).



3.3.1.3 Gilmore-Gomory's model

The following model is the result of adding slack variables to model (3.3.30)–(3.3.36):

$$\min \quad y_{4,2} + y_{4,3} \tag{3.3.41}$$

$$\text{s.t.} \quad 0 = y_{3,3} + y_{3,2} + s_3, \tag{3.3.42}$$

$$y_{4,2} = y_{2,2} + s_2, \tag{3.3.43}$$

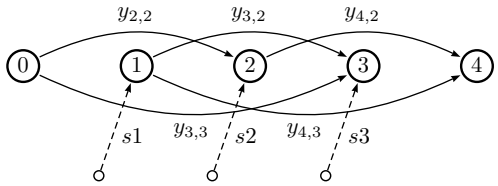
$$y_{4,3} + y_{3,2} = s_1, \tag{3.3.44}$$

$$y_{3,3} + y_{4,3} \geq b_1, \tag{3.3.45}$$

$$y_{2,2} + y_{3,2} + y_{4,2} \geq b_2, \tag{3.3.46}$$

$$y_{3,3}, y_{4,3}, y_{2,2}, y_{3,2}, y_{4,2}, s_1, s_2, s_3 \geq 0. \tag{3.3.47}$$

Figure 3.3.4: Graph associated with model (3.3.41)–(3.3.47).



The arcs s_1, s_2, and s_3 are the slack variables.

In model (3.3.41)–(3.3.47), variables $y_{3,3}$, $y_{3,2}$, and s_3 are always 0 because of $0 = y_{3,3} + y_{3,2} + s_3$ and hence they can be removed. By removing these variables, we obtain the following model:

$$\min \quad y_{4,2} + y_{4,3} \tag{3.3.48}$$

$$\text{s.t.} \quad y_{4,2} = y_{2,2} + s_2, \tag{3.3.49}$$

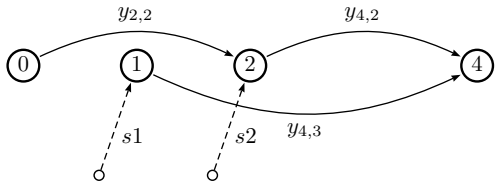
$$y_{4,3} + y_{3,2} = s_1, \tag{3.3.50}$$

$$y_{4,3} \geq b_1, \tag{3.3.51}$$

$$y_{2,2} + y_{4,2} \geq b_2, \tag{3.3.52}$$

$$y_{4,3}, y_{2,2}, y_{4,2}, s_1, s_2 \geq 0. \tag{3.3.53}$$

Figure 3.3.5: Graph associated with model (3.3.48)–(3.3.53).



For any node with flow conservation, the following procedure can be used to remove it. Let I_v and O_v be the sets of arcs entering and leaving the node v, respectively. For every pair of arcs $((u,v),(v,w)) \in I_v \times O_v$, we create a new arc (u,w) connecting directly u to w and covering the items previously covered by (u,v) and (v,w). By applying this method to the model (3.3.48)–(3.3.53), represented in Figure 3.3.5, the following model is obtained after removing nodes 1 and 2:

$$\min \quad z_1 + z_2 + z_3 \tag{3.3.54}$$

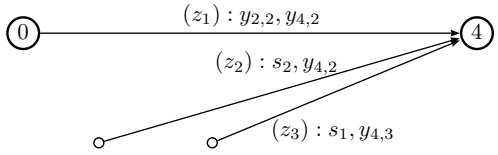
$$\text{s.t.} \quad z_3 \geq b_1, \tag{3.3.55}$$

$$2z_1 + z_2 \geq b_2, \tag{3.3.56}$$

$$z_1, z_2, z_3 \geq 0. \tag{3.3.57}$$

Note that model (3.3.54)–(3.3.57), represented in Figure 3.3.6, is a Gilmore-Gomory's model. By applying this node elimination procedure to arc-flow models derived from dynamic programming recursions, we will always obtain equivalent Gilmore-Gomory's models. Note that arc-flow models are essentially the result of breaking the patterns from a Gilmore-Gomory's model into pieces that cover individual items. By removing nodes we are "gluing" the pieces, and when there are no more nodes to remove we have complete patterns again.

Figure 3.3.6: Graph associated with model (3.3.54)–(3.3.57).

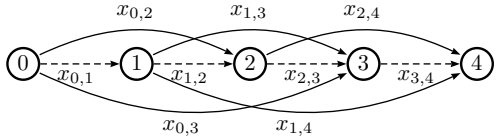


3.3.1.4 Valério de Carvalho's model

The following model, using Valério de Carvalho's arc-flow formulation (2.4.1)–(2.4.4), is essentially model (3.3.13)–(3.3.21) with different variable names and additional variables for slack that transform the balance constraints into flow conservation constraints:

$$\begin{aligned}
\min \quad & z_0 && (3.3.58)\\
\text{s.t.} \quad & z_0 - x_{0,3} - x_{0,2} - x_{0,1} = 0, && (3.3.59)\\
& x_{0,1} - x_{1,4} - x_{1,3} - x_{1,2} = 0, && (3.3.60)\\
& x_{0,2} + x_{1,2} - x_{2,4} - x_{2,3} = 0, && (3.3.61)\\
& x_{0,3} + x_{1,3} + x_{2,3} - x_{3,4} = 0, && (3.3.62)\\
& x_{1,4} + x_{2,4} + x_{3,4} - z_0 = 0, && (3.3.63)\\
& x_{0,3} + x_{1,4} \geq b_1, && (3.3.64)\\
& x_{0,2} + x_{1,3} + x_{2,4} \geq b_2, && (3.3.65)\\
& x_{0,1}, x_{1,2}, x_{2,3}, x_{3,4}, z_0 \geq 0, && (3.3.66)\\
& x_{0,3}, x_{1,4}, x_{0,2}, x_{1,3}, x_{2,4} \geq 0. && (3.3.67)
\end{aligned}$$

Figure 3.3.7: Graph associated with model (3.3.58)–(3.3.67).



3.4 Generalization to multiple capacity limits

For the multi-constraint integer knapsack problem, we obtained in the previous section the recursion:

$$G(\lambda) = \max\left(0, \max_{j=1,\ldots,n}\{G(\lambda - w_j) + \pi_j : w_j \leq \lambda\}\right)$$

where $\lambda \in \mathbb{Z}^p_+$, $\lambda \leq W$, and undefined terms are ignored. $G(\lambda)$ is the maximum profit with capacity λ.

Now suppose there are q types of knapsacks with different capacities W_t and costs C_t. Given that $G(\lambda)$ is the maximum profit under capacity λ, the problem of maximizing profit considering the different knapsack types can be solved as $\max\{G(W_t) - C_t : t = 1, \ldots, q\}$.

From the dynamic programming recursion $G(\lambda)$, we can model $\max\{G(W_t) - C_t : t = 1, \ldots, q\}$ as the following totally unimodular flow problem:

$$\begin{aligned}
\max \quad & \sum_{\lambda}\sum_{j} \pi_j \xi_{\lambda - w_j, \lambda} - \sum_{t=1}^{q} C_t z_t & & \\
\text{s.t.} \quad & -\sum_{j} \xi_{0,w_j} & \leq 0, & \\
& \sum_{j} \xi_{W_t - w_j, \lambda} - \sum_{j} \xi_{W_t, W_t + w_j} - z_t \leq 0, & & t = 1, \ldots, q, \\
& \sum_{j} \xi_{\lambda - w_j, \lambda} - \sum_{j} \xi_{\lambda, \lambda + w_j} & \leq 0, & \lambda \neq W_t, \\
& \sum_{t=1}^{q} z_t & \leq 1, & \\
& \xi_{\lambda - w_j, \lambda}, z_t \geq 0. & &
\end{aligned}$$

And its dual is the following:

$$\begin{aligned}
\min \quad & \delta & & \\
\text{s.t.} \quad & \theta(\lambda) - \theta(\lambda - w_j) \geq \pi_j, & & \lambda \in \mathbb{Z}^p_+ \geq w_j, \\
& \delta - \theta(W_t) & \geq -C_t, & t = 1, \ldots, q, \\
& \theta(\lambda), \delta \geq 0. & &
\end{aligned}$$

Now consider the covering problem $\min\{\sum_t (C_t \sum_j y^t_j) : \sum_t B^t y^t \geq b, y \geq 0\}$ where the

columns of B^t are vectors representing the feasible solutions under capacity W_t, and b is a non-negative integer vector. Note that:

$$\begin{aligned}
&\min\left\{\sum_t (C_t \sum_j y_j^t) : \sum_t B^t y^t \geq b, y \geq 0\right\} \\
=&\max\left\{b\pi : \pi B^t \leq C_t \mathbf{1}, t = 1, \ldots, q, \pi \geq 0\right\} \\
=&\max\left\{b\pi : \pi B^t \leq \pi_0^t \mathbf{1}, \pi_0^t \leq C_t, t = 1, \ldots, q, \pi \geq 0\right\} \\
=&\max\left\{b\pi : (\pi; \pi_0^t) \in \Gamma^t, \pi_0^t \leq C_t, t = 1, \ldots, q, \pi \geq 0\right\}
\end{aligned}$$

where $\Gamma^t = \{(\pi; \pi_0^t) : \pi B^t \leq \pi_0^t \mathbf{1}, \pi \geq 0\}$ is a polytope of valid inequalities for the integer monotone set:

$$Q^t = \left\{x \in \mathbb{Z}_+^n : \sum_{j=1}^n w_j x_j \leq W_t\right\}$$

A polytope Γ^t, for each knapsack type t, can also be easily derived directly from the dynamic programming recursion:

$$\begin{aligned}
\Gamma^t &= \{(\pi; \theta(W_t)) : \theta(\lambda) - \theta(\lambda - w_j) \geq \pi_j, \theta(\lambda) \geq 0, \ \forall j, \forall \lambda\} \\
&= \{(\pi; \theta(W_t)) : \theta(\lambda) \geq \max(0, \max_{j=1}^n \{\theta(\lambda - w_j) + \pi_j : w_j \leq \lambda\}), \ 0 \leq \lambda \in \mathbb{Z}_+^p \leq W_t\}
\end{aligned}$$

The problem $\max\{b\pi : (\pi; \pi_0^t) \in \Gamma^t, \pi_0^t \leq C_t, t = 1, \ldots, q, \pi \geq 0\}$, with Γ^t derived from the dynamic programming recursion, is the following:

$$\begin{aligned}
\max \quad & b\pi \\
\text{s.t.} \quad & \theta(\lambda - w_j) + \theta(\lambda) + \pi_j \leq 0, \quad \lambda \in \mathbb{Z}_+^p \geq w_j, \\
& \theta(W_t) \leq C_t, \quad t = 1, \ldots, q, \\
& \theta(\lambda), \pi \geq 0.
\end{aligned}$$

Its dual is the following network flow problem:

$$\min \quad \sum_{t=1}^q C_t z_t \tag{3.4.1}$$

$$\text{s.t.} \quad \sum_j \xi_{0,w_j} \geq 0, \tag{3.4.2}$$

$$-\sum_j \xi_{W_t - w_j, \lambda} + \sum_j \xi_{W_t, W_t + w_j} + z_t \geq 0, \quad t = 1, \ldots, q, \tag{3.4.3}$$

$$-\sum_j \xi_{\lambda - w_j, \lambda} + \sum_j \xi_{\lambda, \lambda + w_j} \geq 0, \quad \lambda \neq W_t, \tag{3.4.4}$$

$$\sum_{\lambda} \xi_{\lambda - w_j, \lambda} \geq b_j, \quad j = 1, \ldots, n, \tag{3.4.5}$$

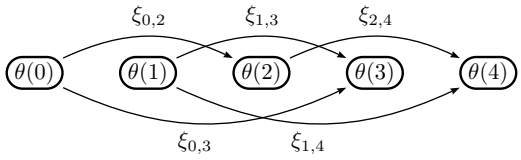
$$\xi_{\lambda - w_j, \lambda}, z_t \geq 0. \tag{3.4.6}$$

3.4.1 Examples

Consider a variable-sized cutting stock instance with two item types of sizes $w_1 = 3$ and $w_2 = 2$, and two capacity limits $W_1 = 2$ and $W_2 = 4$. This instance can be modeled using model (3.4.1)–(3.4.6) as follows:

$$\begin{aligned}
\min \quad & C_1 z_1 + C_2 z_2 && && (3.4.7)\\
\text{s.t.} \quad & \xi_{0,3} + \xi_{0,2} && \geq 0, \quad \text{(dual: } \theta(0)\text{)} && (3.4.8)\\
& \xi_{1,4} + \xi_{1,3} && \geq 0, \quad \text{(dual: } \theta(1)\text{)} && (3.4.9)\\
& -\xi_{0,2} + \xi_{2,4} + z_1 && \geq 0, \quad \text{(dual: } \theta(2)\text{)} && (3.4.10)\\
& -\xi_{0,3} - \xi_{1,3} && \geq 0 \quad \text{(dual: } \theta(3)\text{)} && (3.4.11)\\
& -\xi_{1,4} - \xi_{2,4} + z_2 && \geq 0, \quad \text{(dual: } \theta(4)\text{)} && (3.4.12)\\
& \xi_{0,3} + \xi_{1,4} && \geq b_1, \quad \text{(dual: } \pi_1\text{)} && (3.4.13)\\
& \xi_{0,2} + \xi_{1,3} + \xi_{2,4} && \geq b_2, \quad \text{(dual: } \pi_2\text{)} && (3.4.14)\\
& \xi_{0,3}, \xi_{1,4}, \xi_{0,2}, \xi_{1,3}, \xi_{2,4}, z_1, z_2 \geq 0. && && (3.4.15)
\end{aligned}$$

Figure 3.4.1: Graph associated with model (3.4.7)–(3.4.15).



The dual of model (3.4.7)–(3.4.15) is the following:

$$\begin{aligned}
\max \quad & b_1 \pi_1 + b_2 \pi_2 && \\
\text{s.t.} \quad & \theta(0) - \theta(3) + \pi_1 \leq 0, && \text{(dual: } \xi_{0,3}\text{)}\\
& \theta(1) - \theta(4) + \pi_1 \leq 0, && \text{(dual: } \xi_{1,4}\text{)}\\
& \theta(0) - \theta(2) + \pi_2 \leq 0, && \text{(dual: } \xi_{0,2}\text{)}\\
& \theta(1) - \theta(3) + \pi_2 \leq 0, && \text{(dual: } \xi_{1,3}\text{)}
\end{aligned}$$

$$\begin{aligned}
\theta(2) - \theta(4) + \pi_2 &\leq 0, && \text{(dual: } \xi_{2,4}\text{)} \\
\theta(2) &\leq C_1, && \text{(dual: } z_1\text{)} \\
\theta(4) &\leq C_2, && \text{(dual: } z_2\text{)} \\
\theta(0), \theta(1), \theta(2), \theta(3), \theta(4), \pi_0, \pi_1 &\geq 0.
\end{aligned}$$

3.5 Conclusions

In this chapter, we demonstrated how network flow models equivalent to Gilmore-Gomory's model can be derived from dynamic programming recursions. Moreover, we showed that arc-flow and one-cut models are essentially the same with the addition or elimination of some variables. Both arc-flow and one-cut models can be derived from dynamic programming recursions in a similar fashion.

We also showed that arc-flow/one-cut models are essentially the result of breaking the patterns from a Gilmore-Gomory's model into pieces that cover individual items, and that we can "glue" the pieces by removing nodes/break-points iteratively until an equivalent Gilmore-Gomory's model with complete patterns is obtained.

There have been attempts to reduce the size of arc-flow and one-cut models based, for instance, on symmetry reduction rules. However, even for one-dimensional problems, symmetry reduction rules have not been enough to tackle real world instances without resorting to branch-and-price techniques that have also been proposed in the literature for each of the models.

Chapter 4

General arc-flow formulation with graph compression

The method presented in this chapter allows solving several cutting and packing problems through reductions to multiple-choice vector packing, which is a variant of the vector packing problem in which bins have several types (i.e., sizes and costs) and items have several incarnations (i.e., will take one of several possible sizes). The reductions are made by defining vectors of capacities, a matrix of weights, and a vector of demands. Our method builds very strong integer programming models that can usually be easily solved using any state-of-the-art mixed integer programming (MIP) solver. Computational results obtained with many benchmark test datasets show a large advantage of our method with respect to traditional ones in several applications. Extensive computational results for a variety of problems are presented in Chapters 5 and 6.

In this chapter, instances will be presented as follows. Let J be the set of item types (each item type has one or more item incarnations), and $I = \{1, \dots, m\}$ the set of item incarnations. For each item type j, let d_j be the demand, and I_j the index set of incarnations i of items of type j. For each item incarnation i of items of type j, let w_i be its weight vector, and $b_i = d_j$ be the demand of the corresponding item type. For a given sorting criterion $\preceq$, which must be defined upfront, item incarnations $i \in I$ are labeled in such way that $i_1 \leq i_2$ if and only if $w_{i_1} \preceq w_{i_2}$. For the sake of simplicity, we define 0 as an item incarnation with weight zero in every dimension; this artificial item is used to label loss arcs (i.e., arcs that correspond to unused space). For each bin type $t \in \{1, \dots, q\}$, let W_t be its capacity vector, and C_t be its cost.

The remainder of this chapter is organized as follows. Section 4.1 presents the General arc-flow formulation. Section 4.2 presents the graph construction and compression techniques that are used to obtain smaller models. Section 4.2.3 presents a one-step graph construction and compression algorithm, which is the algorithm that we use in practice to build the models in the multiple-choice vector packing solver `VPSolver`.[1]

[1] `http://vpsolver.fdabrandao.pt` or `https://github.com/fdabrandao/vpsolver`

4.1 General arc-flow formulation

Given a directed multi-graph $G = (V, A)$ containing every valid packing pattern for each bin of type t represented as a path from the source S to the target T_t, and loss arcs connecting each target to the source, the following arc-flow formulation can be used to model the corresponding multiple-choice vector packing problem:

$$\min \quad \sum_{t=1}^{q} C_t f^0_{\mathrm{T}_t\mathrm{S}} \tag{4.1.1}$$

$$\text{s.t.} \quad \sum_{(u,v,i)\in A: v=k} f^i_{uv} \quad - \sum_{(u,v,i)\in A: u=k} f^i_{uv} = 0, \quad k \in V, \tag{4.1.2}$$

$$\sum_{(u,v,i)\in A: i\in I_j} f^i_{uv} \geq d_j, \qquad j \in J \setminus S, \tag{4.1.3}$$

$$\sum_{(u,v,i)\in A: i\in I_j} f^i_{uv} = d_j, \qquad j \in S, \tag{4.1.4}$$

$$f^i_{uv} \leq b_i, \qquad (u,v,i) \in A : i \neq 0, \tag{4.1.5}$$

$$f^i_{uv} \geq 0, \text{ integer}, \qquad (u,v,i) \in A, \tag{4.1.6}$$

where V is the set of vertices; A is the set of arcs, where each arc has three components (u, v, i) corresponding to an arc between nodes u and v associated with the item incarnation i; arcs (u, v, i) with $i = 0$ are the loss arcs; f^i_{uv} is the amount of flow along the arc (u, v, i); and $S \subseteq J$ is a subset of item types for which, for efficiency purposes, we decide that demands are required to be satisfied exactly. For having tighter constraints, one may set $S = \{j \in J : d_j = 1\}$ (we have done this in our experiments); but the optimum for $S = \emptyset$ is the same.

In Valério de Carvalho's model, a variable x_{ij} contributes to an item of weight $j - i$. In our model, a variable f_{uvi} contributes to items of type i; the label of u and v may have no direct relation to the item's weight. This new model is more general; Valério de Carvalho's model is a sub-case, where an arc between nodes u and v can only contribute to the demand of an item of weight $v - u$. As in Valério de Carvalho's model, each arc can only contribute to an item, but the new model has several differences with respect to that formulation:

- nodes are more general (e.g., they can encompass multiple dimensions);
- there may be more than one arc between two vertices (multigraph);
- demands in general may be satisfied with excess but for some items they are required to be satisfied exactly (this allows, for example, the MIP solver to take advantage of special ordered sets of type 1 when requiring the demands of items with demand

one to be satisfied exactly);

- arcs have capacities (i.e., upper bounds) equal to the total demand of the associated item (which allows reducing the search space by excluding many feasible solutions that would exceed the demand);
- arc lengths are not tied to the corresponding item weight (i.e., $(u, v, i) \in A$ even if $v - u \neq w_i$).

We will rely on the following flow decomposition theorem of Ahuja et al. (1993) to prove the equivalence to the classical Gilmore-Gomory model.

Theorem 3 (flow decomposition theorem)**.** *Every path- and cycle-flow has a unique representation as non-negative arc-flows. Conversely, every non-negative arc-flow x can be represented as a path- and cycle-flow (though not necessarily uniquely) with the following two properties:*

(a) Every directed path with positive flow connects a deficit node to an excess node.

(b) At most $n + m$ paths and cycles have nonzero flow; out of these, at most m cycles have nonzero flow.

Corollary 4.1.1 (flow decomposition for circulations)**.** *Any non-negative feasible circulation flow can be decomposed into the sum of flows around directed cycles.*

Corollary 4.1.1 follows directly from Theorem 3.

One of the properties of model (4.1.1)–(4.1.6) is the following.

Property 5 (equivalence to the classical Gilmore-Gomory model)**.** *For a graph with all valid packing patterns represented as paths from* S *to* T_t*, for $t = 1, \ldots, q$, model (4.1.1)–(4.1.6) is equivalent to the classical Gilmore-Gomory model with the same patterns as those obtained from paths in the graph.*

Proof. Extending Valério de Carvalho's proof (see, e.g., Valério de Carvalho 2002), we apply Dantzig-Wolfe decomposition to model (4.1.1)–(4.1.6) keeping (4.1.1), (4.1.3), (4.1.4), and (4.1.5) in the master problem and (4.1.2) and (4.1.6) in the subproblem. As the subproblem is a totally unimodular flow model whose solutions can be decomposed into cycles (each including one feedback arc associated with a bin type), only valid packing patterns are generated. Hence, we can substitute (4.1.2) and (4.1.6) by the patterns and obtain the classical model. From this equivalence follows that lower bounds provided by both models are the same when the same set of patterns is considered. The equality constraints (4.1.4) and the upper bound on variable values (4.1.5) have no effect on the lower bounds, since we are not excluding any valid optimal solution; only solutions that satisfy the demand of some items with excess are excluded, and for these there are equivalent solutions that do not exceed the demand (recall that every valid packing

pattern is represented in the graph). □

4.1.1 Model variants

Model (4.1.7)–(4.1.10) is a simplified version of model (4.1.1)–(4.1.6):

$$\min \quad \sum_{t=1}^{q} C_t f^0_{\mathrm{T}_t\mathrm{S}} \tag{4.1.7}$$

$$\text{s.t.} \quad \sum_{(u,v,i)\in A: v=k} f^i_{uv} - \sum_{(u,v,i)\in A: u=k} f^i_{uv} = 0, \quad k \in V, \tag{4.1.8}$$

$$\sum_{(u,v,i)\in A: i\in I_j} f^i_{uv} \geq d_j, \qquad j \in J, \tag{4.1.9}$$

$$f^i_{uv} \geq 0, \text{ integer}, \qquad (u,v,i) \in A, \tag{4.1.10}$$

By requiring the demands of some items to be satisfied exactly and by introducing upper bounds on variable values in model (4.1.1)–(4.1.6), it may become harder to find feasible solutions for the model. Even though the simplified model (4.1.7)–(4.1.10) may sometimes be a better option, our experiments presented in Chapter 5 show that our choices regarding these two aspects work very well on standard vector packing problems.

Model (4.1.11)–(4.1.14) is the one-cut version of model (4.1.7)–(4.1.10), which results from the elimination of variables $f^0_{\mathrm{T}_t\mathrm{S}}$:

$$\min \quad \sum_{t=1}^{q} C_t \left(\sum_{(u,v,i)\in A: v=\mathrm{T}_t} f^i_{uv} - \sum_{(u,v,i)\in A: u=\mathrm{T}_t} f^i_{uv} \right) \tag{4.1.11}$$

$$\text{s.t.} \quad \sum_{(u,v,i)\in A: u=k} f^i_{uv} \geq \sum_{(u,v,i)\in A: v=k} f^i_{uv}, \qquad k \in V \setminus \{\mathrm{S}, \mathrm{T}_1, \ldots, \mathrm{T}_q\}, \tag{4.1.12}$$

$$\sum_{(u,v,i)\in A: i\in I_j} f^i_{uv} \geq d_j, \qquad j \in J, \tag{4.1.13}$$

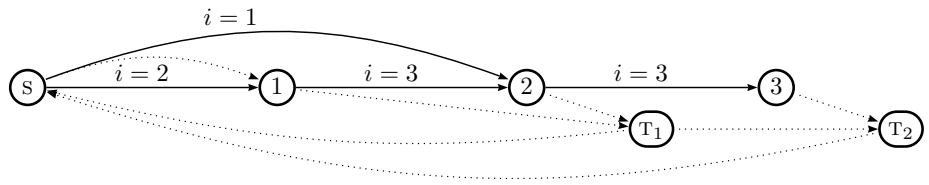
$$f^i_{uv} \geq 0, \text{ integer}, \qquad (u,v,i) \in A : v \neq \mathrm{S}, \tag{4.1.14}$$

4.1.2 Example

Consider a multiple-choice vector packing instance with two bin types ($q = 2$). Bins of type 1 have capacity $W_1 = (75, 50)$ and cost $C_1 = 2$. Bins of type 2 have capacity $W_2 = (100, 75)$ and cost $C_2 = 3$. There are two item types and three item incarnations; The first item type has demand $d_1 = 2$ and one incarnation, $I_1 = \{1\}$, with weight

$w_1 = (75, 50)$. The second item type has demand $d_2 = 1$ and two incarnations, $I_2 = \{2, 3\}$, with weights $w_2 = (40, 15)$, and $w_3 = (25, 25)$. Figure 4.1.1 shows a graph containing every valid packing pattern for this instance represented as paths from S to T_1 and T_2; paths from S to T_1 represent every valid packing pattern using bins of type 1, and paths from S to T_2 represent every valid packing pattern using bins of type 2.

Figure 4.1.1: Arc-flow graph for multiple-choice vector packing.



i	w_i^1	w_i^2	b_i
1	75	50	2
2	40	15	1
3	25	25	1

$W_1 = (75, 50)$
$W_2 = (100, 75)$

Paths from S to T_t represent every valid pattern for bins of type t, for each t.

The arc-flow graph in Figure 4.1.1 can be used with model (4.1.1)–(4.1.6), model (4.1.7)–(4.1.10), or model (4.1.11)–(4.1.14). After having the solution of the integer optimization model, we use a straightforward flow decomposition algorithm to extract a vector packing solution.

4.2 Graph construction and compression algorithms

As we have shown in Chapter 3, Valério de Carvalho's graph can be seen as the dynamic programming search space of the underlying one-dimensional knapsack problem. The vertices of the graph can be seen as states and, in order to model multi-constraint knapsack problems, we just need to add more information to them. In the one-dimensional case, arcs associated with items of weight w_i^1 lie between vertices (a) and $(a + w_i^1)$. In the multi-dimensional case, arcs associated with items of weight $(w_i^1, w_i^2, \ldots, w_i^p)$ lie between vertices $(a^1, a^2, \ldots, a^p)$ and $(a^1 + w_i^1, a^2 + w_i^2, \ldots, a^p + w_i^p)$.

4.2.1 Straightforward graph construction algorithm

Definition 6 (Order)**.** *Item incarnations are sorted in decreasing order by the sum of normalized weights ($\alpha_i = \sum_{k=1}^{p} w_i^k / \max\{W_t^k : t = 1, \ldots, q\}$), using decreasing lexicographical order in case of a tie.*

The source vertex (S) is labeled with 0 in every dimension. Arcs associated with items of weight $(w_i^1, w_i^2, \ldots, w_i^p)$ are created between vertices $(a^1, a^2, \ldots, a^p)$ and $(a^1 + w_i^1, a^2 +$

$w_i^2, \ldots, a^p + w_i^p$). Since, in order to remove symmetry, we should only consider paths that respect a fixed order, there may be an arc with tail in a node only if it is either the source node or the head of an arc associated with a previous item incarnation (according to the order of Definition 6). Our algorithm to construct the graph relies on this rule. Initially, there is only the source node. For each item incarnation, we insert in the graph arcs associated with it starting from all previously existing nodes. After processing an item incarnation, we add to the graph the set of new nodes that appeared as heads of new arcs. This process is repeated for every item incarnation and at the end we just need to connect every node, except the source, to the targets corresponding to bin types in which patterns ending at that node fit. Using this algorithm, the graph can be constructed in pseudo-polynomial time $\mathcal{O}(|V|m)$, where $|V|$ is the number of vertices in the graph. Figure 4.2.1 shows a small example of the construction of a graph using this method.

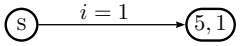
The graph construction process is summarized in Algorithm 1. In this algorithm, V and A are the current sets of vertices and arcs in the graph. This algorithm produces a graph that contains all the valid packing patterns represented as paths from S to T_t, for $t = 1, \ldots, q$. We build the graph item incarnation by item incarnation (line 6) since for each item incarnation we can only have an arc starting from a node $u \neq \mathrm{S}$ if there is an arc for a previously considered incarnation entering u. Initially, we have only the source node. For each item incarnation, we insert in the graph arcs associated with it starting from all previously existing nodes (line 8). After processing each item incarnation we add the new nodes to the set of vertices (in line 16). Using line 13 we avoid processing the same arc twice, since if its tail is already in the list of vertices, either it has already been processed, or it will be processed in a different iteration of line 8. Therefore, we process each node at most m times and each arc once. The complexity of the algorithm is therefore $\mathcal{O}(|V|m)$. This algorithm outputs a graph that contains every valid packing pattern (respecting the order) represented as a path from S to T_t, for $t = 1, \ldots, q$.

Figure 4.2.1: Graph construction example.

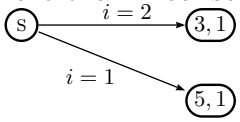
Consider a two-dimensional instance with only one bin type with capacity (7,3), and three item incarnations of sizes (5,1), (3,1), (2,1) and demands 3, 1, 2, respectively. This instance corresponds to adding cardinality limit 3 to Example 1 from Chapter 2.

a) We start with a graph with only the source node:

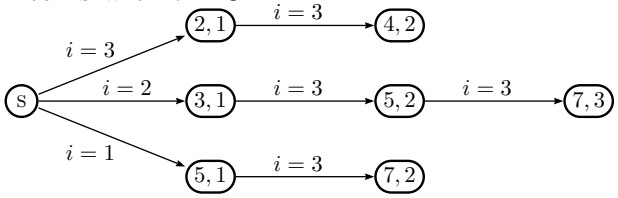
b) At the first iteration, we add arcs associated with the first item incarnation, which has weight (5,1). There is space only for one item with this weight.



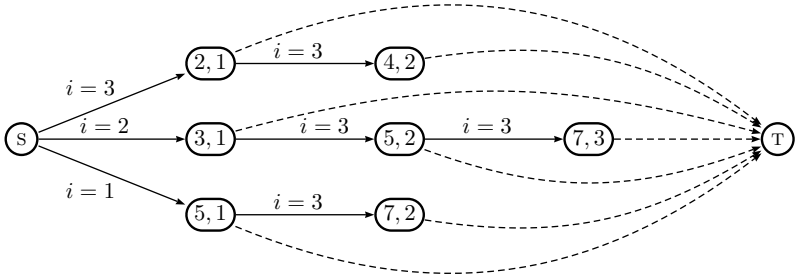
c) At the second iteration, we add arcs for the second item incarnation, which has weight (3,1). The demand does not allow us to form paths with more than 1 consecutive arc with $i = 2$.



d) At the third iteration, we add the arcs associated with the third item incarnation, which has weight (2,1). Since the demand for this item is 2, we can add paths (starting from previously existing nodes) with at most 2 items with $i = 3$.



e) Finally, we add the loss arcs connecting each node, except the source, to the target. The resulting graph corresponds to the model to be solved by a general-purpose mixed-integer optimization solver.



Algorithm 1: Graph construction algorithm

input : m - number of item incarnations; w - weights; b - demands; q - number of bin types; W - capacity vectors

output: V - set of vertices; A - set of arcs; S - source; T - targets

function `buildGraph`(m, w, b, q, W):
- $\text{S} \leftarrow (0, \ldots, 0)$
- $\text{T} \leftarrow \langle \text{“T}_t\text{”} \rangle_{t=1,\ldots,q}$
- $V \leftarrow \{\text{S}\}$ *// we start with the source node only*
- $A \leftarrow \{\ \}$
- **for** $i = 1$ **to** m **do** *// for each item incarnation*
  - $T \leftarrow \{\ \}$
  - **foreach** $u \in V$ **do** *// for each node already in V*
    - **for** $k \leftarrow 1$ **to** b_i **do**
      - $v \leftarrow (u^1 + w_i^1, \ldots, u^p + w_i^p)$
      - **if** $v > W_t$ **for every** $t = 1, \ldots, q$ **then break** *// check if it fits in some bin type*
      - $A \leftarrow A \cup \{(u, v, i)\}$
      - **if** $v \in V$ **then break** *// to avoid repeating work*
      - $T \leftarrow T \cup \{v\}$
      - $u \leftarrow v$
  - $V \leftarrow V \cup T$
- $A \leftarrow A \cup \{(u, \text{T}_t, 0) \mid u \in V \setminus \{\text{S}\}, t = 1, \ldots, q, u \leq W_t\}$ *// connect internal nodes to the targets*
- $V \leftarrow V \cup \{\text{T}_1, \ldots, \text{T}_q\}$
- **return** $(G = (V, A), \text{S}, \text{T})$

Let $W_{\max} = \langle \max\{W_t^k : t = 1, \ldots, q\} \rangle_{k=1,\ldots,p}$. In the one-dimensional case, the number of vertices and arcs in the arc-flow formulation is bounded by $\mathcal{O}(W_{\max})$ and $\mathcal{O}(mW_{\max})$, respectively, and thus graphs are usually reasonably small. However, in the multi-dimensional case, the number of vertices and arcs are bounded by $\mathcal{O}(\prod_{k=1}^{p}(W_{\max}^k+1))$ and $\mathcal{O}(m\prod_{k=1}^{p}(W_{\max}^k+1))$, respectively. Despite the possible intractability indicated by these bounds, the graph compression method that we present in Section 4.2.2 usually leads to reasonably small graphs, even for very hard instances with hundreds of dimensions.

4.2.2 Graph compression

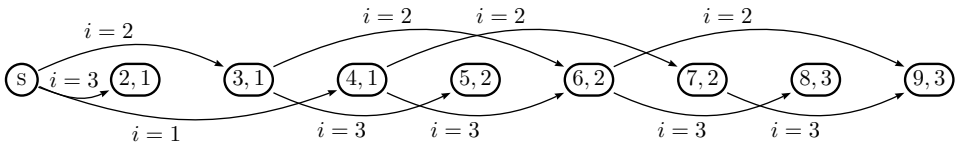
In Section 4.2.1, we presented an algorithm to build arc-flow graphs, which we will refer to as Step-1 graphs. In this section, we will present a three-step graph compression method whose first step, which builds Step-2 graphs, consists of breaking the symmetry and is presented in Section 4.2.2.1. The two remaining compression steps, which build Step-3 and Step-4 graphs, are presented in Section 4.2.2.2. In Section 4.2.2.3, we show that this graph compression technique can also be applied to constraints other than simple capacity

constraints. Finally, an alternative and more efficient graph construction and compression algorithm based on the same concepts is presented in Section 4.2.3.

4.2.2.1 Symmetry breaking algorithm

Let us consider an instance with one bin type of capacity $W = (9, 3)$, and item incarnations of sizes (4,1), (3,1), (2,1) with demands 1, 3, 1, respectively. Figure 4.2.2 shows the Step-1 graph produced by the graph construction algorithm of Section 4.2.1 without the final loss arcs. This graph contains symmetry. For instance, the paths (s,(4,1),i=1) ((4,1),(7,2),i=2) ((7,2),(9,3),i=3) and (s,(4,1),i=1) ((4,1),(6,2),i=3) ((6,2),(9,3),i=2) correspond to the same pattern with one item of each type, but the second one does not respect the order of Definition 6.

Figure 4.2.2: Initial graph/Step-1 graph (with symmetry).



Graph corresponding to an instance with bins of capacity $W = (9, 3)$, and items of sizes (4,1), (3,1), (2,1) with demands 1, 3, 1, respectively. For the sake of simplicity, loss arcs connecting internal nodes to the target were omitted in this figure.

An easy way to break symmetry is to divide the graph into levels, one level for each item type. We introduce in each node a new entry that indicates the level where it belongs. For example, for the two-dimensional case, nodes (a', b') are transformed into sets of nodes $\{(a', b', i'), (a', b', i''), \ldots\}$. Each set has at most one node per level; nodes in consecutive levels are connected by loss arcs. Arcs $((a', b'), (a'', b''), i)$ are transformed into arcs $((a', b', i), (a'', b'', i), i)$. In level i, we have only arcs associated with the item incarnation i. If we connect a node (a', b', i') to a node (a', b', i'') only in case $i' < i''$, we ensure that every path will respect the order (of Definition 6) and thus there is no symmetry. Recall that the initial graph must contain every valid packing pattern (respecting the order) represented as a path from s to t_t, for $t = 1, \ldots, q$.

Algorithm 2 shows how to compute the graph division by levels (Step-2 graph). It receives as input the Step-1 graph produced by Algorithm 1 and creates a Step-2 graph. The algorithm starts by transforming every arc (u, v, i) into $((u, i), (v, i), i)$, in lines 5-12. During this process, in line 10, the algorithm stores for each node u of the initial graph the set of new nodes (V'_u) in which the original node is subdivided. In lines 13-15, after processing all the arcs, the algorithm connects the nodes from different levels that were created from the same node. For each node u in the input graph, it sorts in lexicographical

order the list of nodes V'_u in which the original node was subdivided and it connects the pairs of nodes that appear consecutively in the list.

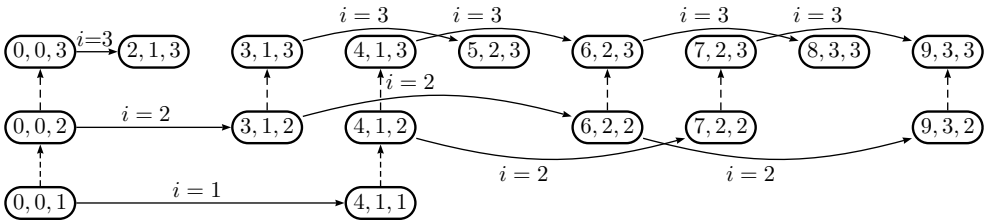
Algorithm 2: Break symmetry with levels

input : V - set of vertices; A - set of arcs; S - source; q - number of bin types; T - targets
output: V' - new set of vertices; A' - new set of arcs; S' - new source; T - targets

function `breakSymmetry`$(G = (V, A), \mathrm{S}, q, \mathrm{T})$:
  $A' \leftarrow \{\ \}$
  $V' \leftarrow \{\mathrm{T}_1, \ldots, \mathrm{T}_q\}$
  $V'_i \leftarrow \{\ \}$, **for all** i
  foreach $(u, v, i) \in A$ **do** *// for each arc in the original graph*
    if $v \notin \{\mathrm{T}_1, \ldots, \mathrm{T}_q\}$ **then**
      $u' \leftarrow (u, i)$; $v' \leftarrow (v, i)$ *// the labels of the new nodes*
      $A' \leftarrow A' \cup \{(u', v', i)\}$ *// add the arc to the new graph*
      $V' \leftarrow V' \cup \{u', v'\}$ *// add the new vertices to the new graph*
      $V'_u \leftarrow V'_u \cup \{u'\}$; $V'_v \leftarrow V'_v \cup \{v'\}$
    else
      $A' \leftarrow A' \cup \{((u, i), v, i)\}$ *// keep the connection to the target in the new graph*
  foreach $u \in V$ **do** *// connect the nodes from different levels associated with each original node u*
    $\mathsf{lst} \leftarrow \mathtt{sort}(V'_u)$ *// sort the set of nodes in lexicographical order*
    for $i = 2$ **to** $|\mathsf{lst}|$ **do** $A' \leftarrow A' \cup \{(\mathsf{lst}[i-1], \mathsf{lst}[i], 0)\}$ *// connect consecutive nodes*
  $\mathrm{S}' \leftarrow \min(V)$
  return $(G' = (V', A'), \mathrm{S}', \mathrm{T})$

Figure 4.2.3 shows the graph with levels (Step-2 graph) that results from applying this symmetry breaking method to the graph in Figure 4.2.2. Although there is no symmetry, there are still patterns that use some items more than their demand. To avoid this, other alternatives to break symmetry could be used; however this method is appropriate for the sake of simplicity and speed.

Figure 4.2.3: Graph with levels/Step-2 graph (without symmetry).



All the patterns respect the order since there are no arcs from higher levels to lower levels. Moreover, it is also easy to check that no valid pattern has been removed. In this graph, the source node is $\mathrm{S} = (0, 0, 1)$ since it is the only node without arcs incident to it. For the sake of simplicity, loss arcs connecting internal nodes to the target were omitted in this figure.

Property 6. *No valid pattern (respecting the order) is removed by breaking symmetry with levels as long as the original graph contains every valid packing pattern (respecting the order) for each bin type t represented as a path from* S *to target* T_t.

Proof. For every valid packing pattern (respecting the order) in the initial graph, there is a path in the Step-2 graph corresponding to the same pattern. Note that every path can be seen as a sequence of consecutive arcs. Let (v_1, v_2, i_1) and (v_2, v_3, i_2) be a pair of consecutive arcs in any valid packing pattern in the Step-1 graph. In the Step-2 graph, these arcs appear as $((v_1, i_1), (v_2, i_1), i_1)$ and $((v_2, i_2), (v_3, i_2), i_2)$. If $i_1 = i_2$, the pair of consecutive arcs appear connected at the same level. If not, and given that (v_2, i_1) and (v_2, i_2) were created from v_2, a set of loss arcs that connect nodes in different levels exists between them and hence there is, again, a sequence of arcs for that part of the pattern. □

4.2.2.2 Graph compression algorithms

Breaking symmetry completely usually leads to much larger graphs; besides some symmetry may be helpful as long as it leads to substantial reductions in the graph size. In this section we show how to reduce the graph size by taking advantage of common sub-patterns that can be represented by a single sub-graph. This method may introduce some symmetry, but it usually helps by dramatically reducing the graph size. This graph compression method is composed of three steps, the first of which was presented in Section 4.2.2.1.

In the graphs we have seen so far, a node label $(a^1, a^2, \ldots, a^p)$ means that, for every dimension k, every sub-pattern from the source to the node uses a^k space in that dimension. This means that a^k corresponds to the length of the longest path from the source to the node in dimension k. Similarly, the length of the longest path from a node to the targets in each dimension can also be used as an entry in the node label, and nodes with the same label can be combined into one single node. In the main compression step, given a graph $G = (V, A)$, a new graph $G' = (V', A')$ is constructed along these lines by creating a set of vertices $V' = \{\phi(v) \mid v \in V\}$ and a set of arcs $A' = \{(\phi(u), \phi(v), i) \mid (u, v, i) \in A, \phi(u) \neq \phi(v)\}$, where ϕ is the map between the original and new labels. This usually allows large reductions in the graph size. This reduction can be improved by breaking symmetry first (as described in the previous section), which allows us to consider only paths to the target respecting a specific order.

The main compression step is applied to the Step-2 graph. In the Step-3 graph, the longest paths to the targets in each dimension are used to relabel the internal nodes $(V \setminus \{\mathrm{S}, \mathrm{T}_1, \ldots, \mathrm{T}_q\})$, dropping the level dimension label of each node. Let $(\varphi^1(u), \varphi^2(u),$

$\ldots, \varphi^p(u))$ be the new label of node u in the Step-3 graph, where

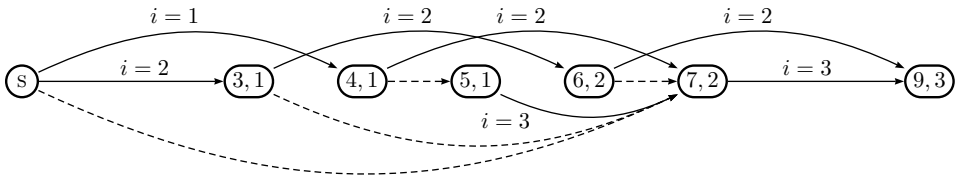
$$\varphi^k(u) = \begin{cases} W_t^k & \text{if } u = \mathrm{T}_t \text{ (base cases)}, \\ \min_{(u',v,i)\in A:u'=u}\{\varphi^k(v) - w_i^k\} & \text{otherwise.} \end{cases} \tag{4.2.1}$$

For the sake of simplicity, we define w_0^k for loss arcs as zero in every dimension, $w_0^0 = w_0^1 = \ldots = w_0^p = 0$. In the paths from S to T_t in the Step-2 graph usually there is slack in some dimension. In this process, we are moving this slack as much as possible to the beginning of the paths. The label in each dimension of every node u (except S) corresponds to the highest position (i.e., closest to all targets T_t) where the sub-patterns from u to any target T_t can start in each dimension without violating capacity constraints. By using these labels, we are allowing arcs to be longer than the items to which they are associated. We use dynamic programming to compute these labels in linear time in the graph size.

Algorithm 3 presents the main compression algorithm that creates the Step-3 graph. This algorithm receives as input the Step-2 graph produced by Algorithm 2. We use dynamic programming to compute the labels iteratively. A reverse topological order, which is needed for dynamic programing, is computed in line 2; the way the nodes are labeled in Step-2 graph allows us to obtain a reverse topological order just by sorting the nodes in decreasing lexicographical order. In line 3, we obtain an adjacency list of the graph (including loss arcs connecting every internal node to the target). In lines 4-6, we compute the new label of each node using dynamic programming. The Step-3 graph is produced in lines 9-10. The complexity of this algorithm is linear in the graph size for a fixed dimensions p.

Figure 4.2.4 shows the Step-3 graph that results from applying the main compression step to the graph of Figure 4.2.3. Even in this small instance, a few nodes and arcs were removed, comparing with the initial graph of Figure 4.2.2.

Figure 4.2.4: Step-3 graph (after the main compression step).



The Step-3 graph has 8 nodes and 17 arcs (considering also the final loss arcs connecting internal nodes to the target, which were omitted).

Finally, in the last compression step, a new graph is constructed once more. In order to try to reduce the graph size even more, we relabel the internal nodes once more using the

Algorithm 3: Main compression step

input : V - set of vertices; A - set of arcs; S - source; q - number of bin types; T - targets; w - weights; W - capacity vectors

output: V' - new set of vertices; A' - new set of arcs; S' - new source; T - targets

function `mainCompression`$(G = (V, A), \mathrm{S}, q, \mathrm{T}, w, W)$:

- order $\leftarrow$ `reverse`(`sort`$(V \setminus \{\mathrm{T}_1, \ldots, \mathrm{T}_q\})$) *// reverse topological order of the graph*
- $\mathsf{adj}_u \leftarrow \{(v, i) \mid (u, v, i) \in A\}$, **for all** $u \in$ order *// adjacency list of the graph*
- $\varphi(\mathrm{T}_t) \leftarrow (W_t^1, \ldots, W_t^p)$, **for** $t = 1, \ldots, q$ *// base cases*
- **foreach** $u \in$ order **do**
  - $\varphi(u) \leftarrow \left\langle \max\{\varphi^k(v) - w_i^k : (v, i) \in \mathsf{adj}_u\} \right\rangle_{k=1,\ldots,p}$ *// compute $\varphi(v)$*
- $\mathrm{S}' \leftarrow \varphi(\mathrm{S})$
- $\varphi(\mathrm{T}_t) \leftarrow \mathrm{T}_t$, **for** $t = 1, \ldots, q$
- $A' \leftarrow \{(\varphi(u), \varphi(v), i) \mid (u, v, i) \in A, \varphi(u) \neq \varphi(v)\}$ *// relabel the graph and remove self-loops*
- $V' \leftarrow \{u \mid (u, v, i) \in A'\} \cup \{v \mid (u, v, i) \in A'\}$ *// new set of vertices*
- **return** $(G' = (V', A'), \mathrm{S}', \mathrm{T})$

longest paths from the source in each dimension. Let $(\psi^1(v), \psi^2(v), \ldots, \psi^p(v))$ be the label of node v in the Step-4 graph, where

$$\psi^k(v) = \begin{cases} 0 & \text{if } v = \mathrm{S} \text{ (base case)}, \\ \max_{(u,v',i) \in A : v' = v} \{\psi^k(u) + w_i^k\} & \text{otherwise.} \end{cases} \tag{4.2.2}$$

Algorithm 4 presents the final compression step that relabels the graph using the longest paths from the source in each dimension. This algorithm uses dynamic programming to compute the labels, and it is very similar to Algorithm 3; the main difference is the use of the transpose graph, which results from reversing the orientation of the arcs in the input graph. The adjacency list of the transpose graph is required for dynamic programming since we need to know which arcs enter in each node. In this case, the reverse topological order of the transpose graph is simply the lexicographical order of the nodes. The complexity of this algorithm is also linear in the graph size.

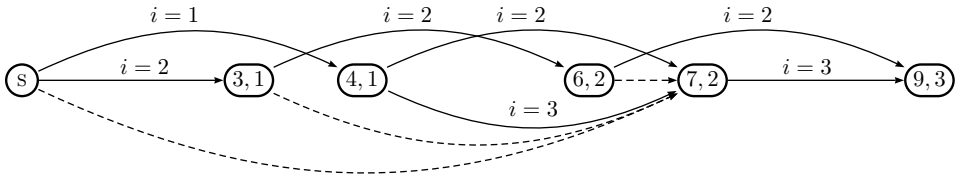
Algorithm 4: Final compression step

input : V - set of vertices; A - set of arcs; S - source; T - targets; w - weights
output: V' - new set of vertices; A' - new set of arcs; S′ - new source; T - targets

function `finalCompression`$(G = (V, A), \mathrm{S}, \mathrm{T}, w)$:
  $\mathsf{adj}_v^{\mathrm{T}} \leftarrow \{(u, i) \mid (u, v, i) \in A\}$, **for all** $v \in V$ *// adjacency list of the transpose graph*
  $\mathsf{order} \leftarrow \mathtt{sort}(V \setminus \{\mathrm{S}, \mathrm{T}_1, \ldots, \mathrm{T}_q\})$ *// reverse topological order of the transpose graph*
  $\psi(\mathrm{S}) \leftarrow \langle 0 \rangle_{k=1,\ldots,p}$ *// base case*
  $\psi(\mathrm{T}_t) \leftarrow \mathrm{T}_t$, **for all** $t = 1, \ldots, q$
  foreach $v \in \mathsf{order}$ **do** *// for each vertex in reverse topological order of the transpose graph*
    $\psi(v) \leftarrow \left\langle \max\{\psi^k(u) + w_i^k : (u, i) \in \mathsf{adj}_v^{\mathrm{T}}\} \right\rangle_{k=1,\ldots,p}$ *// compute $\psi(v)$*
  $\mathrm{S}' \leftarrow \psi(\mathrm{S})$
  $A' \leftarrow \{(\psi(u), \psi(v), i) \mid (u, v, i) \in A, \psi(u) \neq \psi(v)\}$ *// relabel the graph and remove self-loops*
  $V' \leftarrow \{u \mid (u, v, i) \in A'\} \cup \{v \mid (u, v, i) \in A'\}$ *// new set of vertices*
  return $(G' = (V', A'), \mathrm{S}', \mathrm{T})$

Figure 4.2.5 shows the Step-4 graph without the final loss arcs. This last compression step is not as important as the main compression step, but it nevertheless usually removes many nodes and arcs and is easy to compute.

Figure 4.2.5: Step-4 graph (after the last compression step).



The Step-4 graph has 7 nodes and 15 arcs (considering also the final loss arcs connecting internal nodes to the target, which were omitted). In this case, the only difference from the Step-3 graph is the node $(5, 1)$ that collapsed with the node $(4, 1)$. The initial Step-1 graph had 10 nodes and 18 arcs.

Note that, in this case, the initial Step-1 graph had some symmetry and the final Step-4 graph does not contain any symmetry. Graph compression may increase symmetry in some situations, but in practice this is not a problem since it usually leads to large reductions in the graph size. Since we are dealing with a very small instance, the improvement obtained by compression is not as substantial as in large instances. For example, in the standard BPP instance HARD4 from Scholl et al. (1997) with 200 items of 198 different sizes and bins of capacity $100,000$, we obtained reductions of 97% in the number of vertices and 95% in the number of the arcs. The solution of the resulting model was found in a few seconds by a MIP solver. Without graph compression it would be much harder to solve this kind of instances using the arc-flow formulation.

Property 7 (Graph compression 1)**.** *Non-redundant patterns in the initial graph are not*

removed by graph compression.

Proof. Any pattern in the initial graph is represented by a path from S to a target T_t. Consider a path $\pi = v_1 v_2 \dots v_n$. Graph compression will reduce the graph size by relabeling nodes, collapsing nodes that receive the same label and removing self-loops (loss arcs). Therefore, the path $\pi' = \phi(v_1)\phi(v_2)\dots\phi(v_n)$, where ϕ is the map between the original and new labels, will represent exactly the same pattern as the one represented by the path π, possibly with some self-loops that do not affect the patterns. □

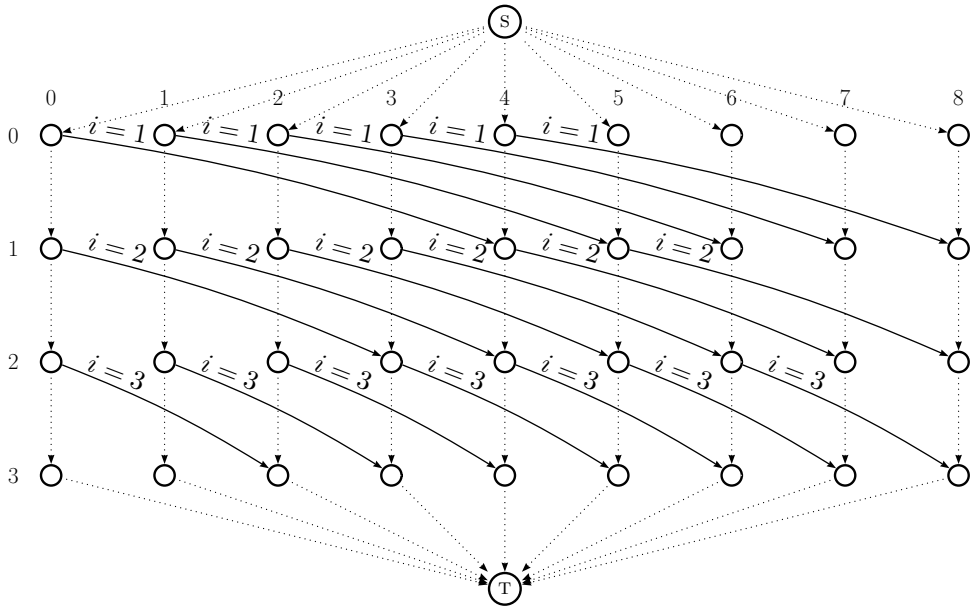
Property 8 (Graph compression 2)**.** *Graph compression will not introduce any invalid pattern.*

Proof. An invalid pattern consists of a set of items whose sum of weights exceeds the bin capacity in some dimension. A pattern is formed from a path in the graph and its total weight in each dimension is the sum of weights of the items in the path. The main compression step just relabels every node u (except targets T_t) in each dimension with the highest position (i.e., closest to the targets) where the sub-patterns from u to any T_t can start without violating capacity constraints. Each target node T_t is labeled with $(W_t^1, \dots, W_t^p)$ and no label will be smaller than $(0, \dots, 0)$, since all the patterns in the input graph are required to be valid. However, we could have invalid patterns, even with all the nodes in paths from S to T_t labeled between zero and $(W_t^1, \dots, W_t^p)$, if an arc had length smaller than the item it represents, but this is not possible. The label of every node u (except targets) is given by $(\varphi^1(u), \varphi^2(u), \dots, \varphi^p(u))$, where $\varphi^k(u) = \min\{\varphi^k(v) - w_i^k \mid (u', v, i) \in A,\ u' = u\}$; for every pair of nodes u and v such that there is an arc between u and v, the difference in each dimension between their labels is at least the weight of the item associated with the arc. Therefore, no invalid patterns are introduced. An analogous proof can be derived for the last compression step. □

4.2.2.3 Graph compression with binary patterns

Our graph compression technique can also be applied to constraints other than simple capacity constraints. For instance, it can be easily adapted to handle binary constraints. Cutting stock with binary patterns (0–1 CSP) is a cutting stock variant in which repetitions of items are not allowed in the same roll. Consider a 0–1 CSP instance with rolls of size $W = 8$ and items of sizes 4, 3, 2 with demands 3, 2, 5, respectively. Figure 3.2.1 shows the graph that can be derived from a straightforward dynamic programming recursion for the underlying 0–1 knapsack problem.

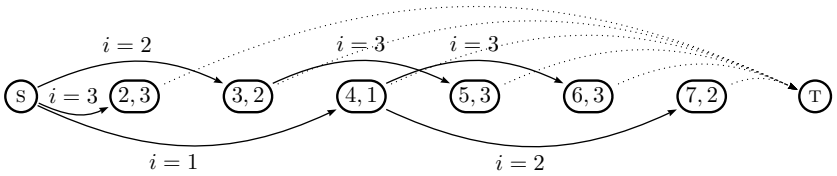
Figure 4.2.6: Graph derived from a straigthforward dynamic programming recursion.



Graph corresponding to a dynamic programming recursion for a 0-1 knapsack instance with capacity 8 and items of sizes 4, 3, 2.

Figure 4.2.7 shows a Step-1 graph which contains every valid packing pattern (respecting that order) for this instance represented as path from the source S to the target T. In this graph, a node label (a', b') means that every sub-pattern from the source to the node uses no more than a' space and contains no item with an index higher than b'.

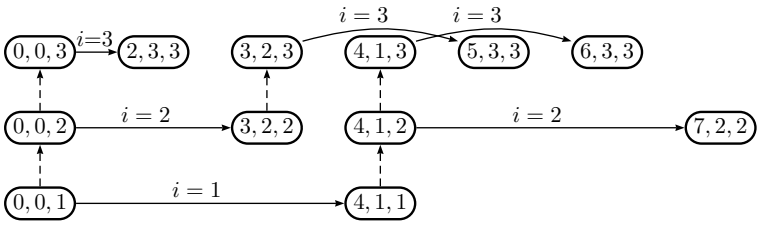
Figure 4.2.7: Initial graph/Step-1 graph.



Graph corresponding to a 0–1 CSP instance with bins of capacity $W = 8$ and items of sizes 4, 3, 2 with demands 3, 2, 5, respectively.

By breaking symmetry with levels, we obtain the following Step-2 graph:

Figure 4.2.8: Graph with levels/Step-2 graph.



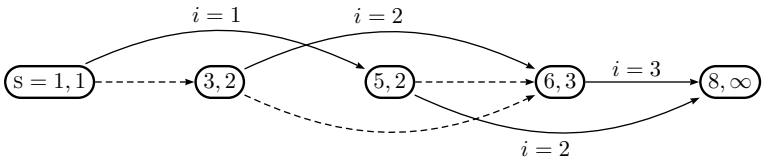
The Step-2 graph has 13 nodes and 22 arcs (considering also the final loss arcs connecting internal nodes to T, which were omitted). In this graph, S = (0, 0, 1).

In the main compression step, a new graph is constructed using the longest path to the target in each capacity dimension, and the smallest item index that appears in any path from the node to the target in the binary dimension. Let $(\varphi^1(u), \varphi^2(u), \ldots, \varphi^p(u), \sigma(u))$ be the new label of node u in the Step-3 graph, where $\varphi^k(u)$ is defined in (4.2.1) and

$$\sigma(u) = \begin{cases} \infty & \text{if } u = \mathrm{T}_t, \\ \min(\min_{(u',v,i)\in A:u'=u}\{\sigma(v)\}, \min_{(u',v,i\neq 0)\in A:u'=u}\{i\}) & \text{otherwise.} \end{cases} \tag{4.2.3}$$

Figure 4.2.9 shows the graph that results from applying the main compression step to the graph of Figure 4.2.8.

Figure 4.2.9: Step-3 graph (after the main compression step).



The Step-3 graph has 8 nodes and 17 arcs (considering also the final loss arcs connecting internal nodes to T, which were omitted).

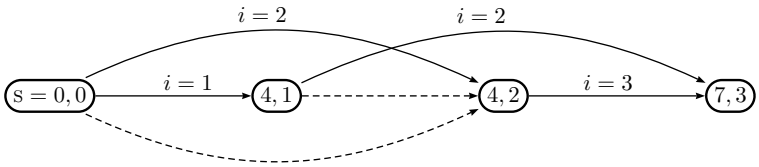
In the final compression step, a new graph is constructed once more, using in the capacity dimensions of the node label the longest path from the source to the current node, and in the binary dimension label the highest item index that appears in any path from the source to the current node. Let $(\psi^1(v), \psi^2(v), \ldots, \psi^p(v), \rho(v))$ be the label of node v in the Step-4 graph, where $\psi^k(u)$ is defined in (4.2.2) and

$$\rho(v) = \begin{cases} 0 & \text{if } v = \mathrm{S}, \\ \max_{(u,v',i)\in A:v'=v}\{\max(\rho(u), i)\} & \text{otherwise.} \end{cases} \tag{4.2.4}$$

Figure 4.2.10 shows the final Step-4 graph that results from applying the final compression

step to the graph of Figure 4.2.9.

Figure 4.2.10: Step-4 graph (after the final compression step).



The Step-4 graph has 5 nodes and 9 arcs (considering also the final loss arcs connecting internal nodes to t, which were omitted in this figure). The straightforward dynamic programming graph had 38 vertices and 63 arcs.

4.2.3 VPSolver's graph construction and compression algorithm

In this section, we present the algorithm that is used by the multiple-choice vector packing solver `VPSolver`, which solves multiple-choice vector packing problems using the general arc-flow formulation with graph compression presented in this chapter.

As we said in Section 4.2.1, in the p-dimensional case, the number of vertices and arcs is limited by $\mathcal{O}(\prod_{k=1}^{p}(W_{\max}^k + 1))$ and $\mathcal{O}(m\prod_{k=1}^{p}(W_{\max}^k + 1))$, respectively. On the other hand, our graph compression method usually leads to very high compression ratios and in some cases it may lead to final graphs hundreds of times smaller than the initial ones. Hence, the size of the initial graph can be the limiting factor and its construction should be avoided.

In practice, we build the Step-3 graph directly in order to avoid the construction of huge Step-1 and Step-2 graphs that may have many millions of vertices and arcs. Algorithm 5 uses dynamic programming to build the Step-3 graph recursively over the structure of the Step-2 graph (without building it). The basic idea for this algorithm comes from the fact that, in the main compression step, the label of any internal node ($\varphi(u) = \langle \min\{\varphi^k(v) - w_i^k : (u', v, i) \in A, u' = u\}\rangle_{k=1,\dots,p}$) only depends on the labels of the two nodes to which it is connected; a node in its level (line 16) and another in the level above (line 13). After directly building the Step-3 graph from the instance's data using this algorithm, we apply the final compression step (line 27) using Algorithm 4, and connect the internal nodes to the targets (line 28) using Algorithm 6. Since parallel arcs associated to the same item type but different incarnations are redundant, all redundant arcs are removed in line 29. In practice, this method allows obtaining arc-flow models even for large benchmark instances quickly.

The dynamic programming states are identified by the space already used (x), the current

Algorithm 5: VPSolver's graph construction and compression algorithm

input : m - number of item incarnations; w - weights; b - demands; q - number of bin types; W - capacity vectors

output: V - set of vertices; A - set of arcs; S - source; T - targets

1 **function** `buildGraph`(m, w, b, q, W):

2 $\quad \mathsf{dp}[x, i, c] \leftarrow \mathsf{NIL}$, **for all** x, i, c *// dynamic programming table*

3 $\quad$ **function** `lift`(x, i, c): *// auxiliary function: lift* dp *states solving knapsack problems in each dimension*

$\quad\quad$ **input** : x - used capacity; i - current item; c - number of repetitions

4 $\quad\quad$ **function** `highestPosition`(k, t):

5 $\quad\quad\quad$ **return** $\min \left\{ W_t^k - \sum_{j=i}^m w_j^k y_j : \begin{array}{l} \sum_{j=i}^m w_j^k y_j \le W_t^k - x^k, \\ y_i \le b_i - c,\ y_j \le b_j,\ j = i+1, \ldots, m, \\ y_j \ge 0,\ \text{integer},\ j = i, \ldots, m \end{array} \right\}$

6 $\quad\quad$ **return** $\langle \min\{\texttt{highestPosition}(k, t) : t = 1, \ldots, q, x \le W_t\} \rangle_{k=1,\ldots,p}$

7 $\quad (V, A) \leftarrow (\{\ \}, \{\ \})$

8 $\quad$ **function** $\varphi(x, i, c)$:

$\quad\quad$ **input** : x - used capacity; i - current item; c - number of repetitions

9 $\quad\quad x \leftarrow \texttt{lift}(x, i, c)$ *// lift x in order to reduce the number of* dp *states*

10 $\quad\quad$ **if** $\mathsf{dp}[x, i, c] \neq \mathsf{NIL}$ **then** *// avoid repeating work*

11 $\quad\quad\quad$ **return** $\mathsf{dp}[x, i, c]$

12 $\quad\quad u \leftarrow \left\langle \min\{W_t^k : t = 1, \ldots, q, x \le W_t\} \right\rangle_{k=1,\ldots,p}$ *// value of $\varphi(x, i, c)$ if nothing else fits*

13 $\quad\quad$ **if** $i < m$ **then** *// option 1: do not use the current item (go to the level above)*

14 $\quad\quad\quad up_x \leftarrow \varphi(x, i+1, 0)$

15 $\quad\quad\quad u \leftarrow up_x$ *// value of $\varphi(x, i, c)$ if no more items of the current type are introduced*

16 $\quad\quad$ **if** $c < b_i$ **and** $x + w_i \le W_t$ **for some** $t = 1, \ldots, q$ **then** *// option 2: use the current item*

17 $\quad\quad\quad v \leftarrow \varphi(x + w_i, i, c+1)$

18 $\quad\quad\quad u \leftarrow \left\langle \min(u^k, v^k - w_i^k) \right\rangle_{k=1,\ldots,p}$ *// update the value of $\varphi(x, i, c)$*

19 $\quad\quad\quad A \leftarrow A \cup \{(u, v, i)\}$ *// connect u to the node resulting from option 2*

20 $\quad\quad\quad V \leftarrow V \cup \{u, v\}$

21 $\quad\quad$ **if** $k < m$ **and** $u \neq up_x$ **then**

22 $\quad\quad\quad A \leftarrow A \cup \{(u, up_x, 0)\}$ *// connect u to the node resulting from option 1*

23 $\quad\quad\quad V \leftarrow V \cup \{up_x\}$

24 $\quad\quad \mathsf{dp}[x, i, c] \leftarrow u$

25 $\quad\quad$ **return** u *// returns $u = \varphi(x, i, c)$*

26 $\quad \text{S} \leftarrow \varphi(x = \langle 0 \rangle_{k=1,\ldots,p}, i = 1, c = 0)$ *// build the Step-3 graph*

27 $\quad (V, A, \text{S}) \leftarrow \texttt{finalCompression}(V, A, \text{S}, w)$ *// final compression step*

28 $\quad (V, A, \text{S}, \text{T}) \leftarrow \texttt{connectTargets}(V, A, \text{S}, q, W)$ *// connect the internal nodes to the targets*

29 $\quad A \leftarrow A \setminus \{(u, v, i_2) \in A : (u, v, i_1) \in A, i_1 \in I_j, i_2 \in I_j, i_2 > i_1\}$ *// remove redundant arcs*

30 $\quad$ **return** $(G = (V, A), \text{S}, \text{T})$

item incarnation (i) and the number of times it has already been used (c). In order to reduce the number of states, we lift (line 9) each state by solving (using again dynamic

programming though this is not explicit in the algorithm) knapsack/longest-path problems in each dimension considering the remaining items (line 4); we try to increase the space used in each dimension to its highest value considering the valid packing patterns for the remaining items. Note that all valid bin sizes must be considered in the lifting procedure.

When there are multiple valid targets for the same node, we need to connect the node to each of them, or take advantage of a transitive reduction, relying on an order as defined in Definition 7, to connect each node to as little targets as possible (as we do in Algorithm 6).

Definition 7. *A bin type t_1 of capacity W_{t_1} dominates a bin type t_2 of capacity W_{t_2}, $(t_1, W_{t_1}) \prec (t_2, W_{t_2})$ for short, if $W_{t_1} = W_{t_2}$ and $t_1 < t_2$, or $W_{t_1} \neq W_{t_2}$ and $W_{t_1} \leq W_{t_2}$.*

For instance, in the variable-sized bin packing problem, since there is just one dimension, the transitive reduction allows us to connect each internal node to just one target (i.e., it uses only 1 additional arc per node instead of q additional arcs per node).

Algorithm 6: Connect internal nodes to the targets

input : V - set of vertices; A - set of arcs; S - source; q - number of bin types; W - capacity vectors

output: V - set of vertices; A - set of arcs; S - source; T - targets

1 **function** `connectTargets`(V, A, S, q, W):
2 $\quad \mathrm{T} \leftarrow \langle \text{“}\mathrm{T}_t\text{”} \rangle_{t=1,\ldots,q}$
3 $\quad$ **foreach** $v \in V \setminus \{\mathrm{S}\}$ **do** *// for each internal node*
4 $\quad\quad \tau \leftarrow \{t : t = 1, \ldots, q, v \leq W_t\}$ *// valid bin types for vertex v*
5 $\quad\quad$ **foreach** $t \in \{1, \ldots, q\}$ **do**
6 $\quad\quad\quad$ **if** $t \in \tau$ **then**
7 $\quad\quad\quad\quad \tau \leftarrow \tau \setminus \{t' \in \tau : (t, W_t) \prec (t', W_{t'})\}$ *// remove dominated bin types*
8 $\quad\quad A \leftarrow A \cup \{(v, \mathrm{T}_t, 0) : t \in \tau\}$ *// connect v to non-dominated targets*
9 $\quad$ **foreach** $t \in \{1, \ldots, q\}$ **do** *// for each bin type*
10 $\quad\quad \tau \leftarrow \{t' = 1, \ldots, q : (t, W_t) \prec (t', W_{t'})\}$ *// dominated bin types*
11 $\quad\quad$ **foreach** $t' \in \{1, \ldots, q\}$ **do**
12 $\quad\quad\quad$ **if** $(t, W_t) \prec (t', W_{t'})$ **then**
13 $\quad\quad\quad\quad \tau \leftarrow \tau \setminus \{t'' \in \tau : (t', W_{t'}) \prec (t'', W_{t''})\}$ *// transitive reduction*
14 $\quad\quad A \leftarrow A \cup \{(\mathrm{T}_t, \mathrm{T}_{t'}, 0) : t' \in \tau\}$ *// connect T_t the targets it directly dominates*
15 $\quad V \leftarrow V \cup \{\mathrm{T}_t : t = 1, \ldots, q\}$
16 $\quad A \leftarrow A \cup \{(\mathrm{T}_t, \mathrm{S}, 0) : t = 1, \ldots, q\}$ *// add the feedback arcs*
17 $\quad$ **return** $(G = (V, A), \mathrm{S}, \mathrm{T})$

Chapter 5

Applications through reductions to vector packing

In Delorme et al. (2016), the arc-flow formulation with graph compression was compared against several other models and problem-specific algorithms on one-dimensional bin packing and cutting stock problems. The arc-flow formulation outperformed all other models and the only method with comparable performance was the problem-specific branch-and-cut-and-price algorithm of Belov and Scheithauer (2006). Note that branch-and-cut-and-price algorithms are usually much more complex than standard branch-and-price algorithms. In this chapter, we test the arc-flow formulation on a large variety of problems and datasets. The results in this chapter appear in Brandão and Pedroso (2016).

Results were obtained using a computer with a Quad-Core Intel Xeon at 2.66GHz, running Mac OS X 10.8.0, with 16 GBytes of memory (though only a few of the hardest instances required more than 4 GBytes of memory). The graph construction algorithm was implemented in `C++` (the source code is available online[1]), and the resulting MIP was solved using `Gurobi` 5.0.0, a state-of-the-art mixed integer programming solver. The parameters used in `Gurobi` were Threads = 1 (single thread), Presolve = 1 (conservative), Method = 2 (interior-point methods), MIPFocus = 1 (feasible solutions), Heuristics = 1, MIPGap = 0, MIPGapAbs = $1-10^{-5}$ and the remaining parameters were `Gurobi`'s default values. The use of interior-point methods at the root node considerably improves the time for solving the linear relaxation, compared to using the simplex algorithm. The branch-and-cut solver used in `Gurobi` uses a series of cuts; in our models, the most frequently used were Gomory, Zero half and MIR. Detailed results, log files and datasets are available online.[2]

[1] `https://github.com/fdabrandao/vpsolver` or `http://www.dcc.fc.up.pt/~fdabrandao/code`
[2] `http://www.dcc.fc.up.pt/~fdabrandao/research/vpsolver/results/`

5.1 p-dimensional vector packing

In the p-dimensional vector packing problem (VBP), the set S of valid patterns is defined as follows:

$$A = \begin{bmatrix} w_1^1 & \dots & w_m^1 \\ \vdots & & \vdots \\ w_1^p & \dots & w_m^p \end{bmatrix} \qquad L = \begin{bmatrix} W^1 \\ \vdots \\ W^p \end{bmatrix} \qquad S = \{x \in \mathbb{Z}_+^m : Ax \leq L\} \tag{5.1.1}$$

where A is the matrix of weights and L is the vector of capacities. The set S is the set of valid packing patterns that satisfy all the following knapsack constraints:

$$w_1^1 x_1 + w_2^1 x_2 + \ldots + w_m^1 x_m \leq W^1 \tag{5.1.2}$$
$$w_1^2 x_1 + w_2^2 x_2 + \ldots + w_m^2 x_m \leq W^2 \tag{5.1.3}$$
$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$
$$w_1^p x_1 + w_2^p x_2 + \ldots + w_m^p x_m \leq W^p \tag{5.1.5}$$
$$x_i \geq 0, \text{ integer}, \ i = 1, \ldots, m, \tag{5.1.6}$$

Two-constraint bin packing (2CBP) is a 2-dimensional vector packing problem. We used the arc-flow formulation to solve to optimality 330 of the 400 instances from the DEIS-OR's two-constraint bin packing test dataset,[3] which was proposed by Caprara and Toth (2001). Table 5.1 summarizes the results. This dataset has several sizes n for each class, each pair (class, size) having 10 instances.

Table 5.1: Results for 2-dimensional vector packing.

	$n \in \{24, 25\}$			$n \in \{50, 51\}$			$n \in \{99, 100\}$			$n \in \{200, 201\}$		
class	n^{ip}	t^{tot}	#open	n^{ip}	t^{tot}	#open	n^{ip}	t^{tot}	#open	n^{ip}	t^{tot}	#open
1	0.0	0.12	0	0.0	1.62	0	0.0	66.96	5	0.0	7,601.35	7
2	0.0	0.01	10	0.0	0.04	10	0.0	0.21	10	0.0	6.93	10
3	0.0	0.01	10	0.0	0.02	10	0.0	0.05	10	0.0	0.20	10
4	0.0	21.97	10	-	-	-	-	-	-	-	-	-
5	0.0	10.62	10	-	-	-	-	-	-	-	-	-
6	0.0	0.02	0	0.0	0.06	1	0.0	0.31	5	0.0	4.75	8
7	0.0	0.03	0	0.0	0.14	1	0.3	1.69	7	31.4	14.01	3
8	0.0	0.01	10	0.0	0.02	10	0.0	0.07	10	0.0	0.24	10
9	0.0	0.10	0	0.0	0.66	1	0.0	28.11	10	-	-	-
10	0.0	0.02	0	0.0	0.10	0	0.0	0.66	0	226.9	155.43	0

n - total number of items; n^{ip} - average number of nodes explored in the branch-and-cut procedure; t^{tot} - average run time in seconds; #open - number of previously open instances solved; "-" means that the subclass was not solved by our algorithm.

[3] `http://www.or.deis.unibo.it/research_pages/ORinstances/ORinstances.htm`

Note that among the instances presented in Table 5.1 there were 188 instances with no previously known optimum. The arc-flow formulation allowed the solution of 330 instances out of the 400 instances. The graph compression algorithm is remarkably effective on all of this subset of two-constraint bin packing instances. In many cases, graph compression allowed the removal of more than 90% of the vertices and arcs; without it, it would not be viable to solve many of these instances within a reasonable amount of time.

The classes 4 and 5 are hard for our arc-flow formulation due to a large number of items that fit in a single bin, which leads to a huge number of valid packing patterns that are difficult to represent in a compact way. Most of the instances that were not solved belong to these two classes. Note that, however, the type of instances with practical interest for exact methods usually does not involve long patterns since optimal or near-optimal solutions can usually be found with little effort using heuristics. The seven subclasses that we did not solve (appearing as "-" in Table 5.1) contain at least one instance that takes more than 12 hours to be solved exactly. The average run time in the 330 solved instances was 4 minutes, and none of these instances took longer than 5 hours to be solved exactly.

In order to test the behavior of the arc-flow formulation in instances with the same characteristics and more dimensions, we created 20-dimensional vector packing instances by combining the ten 2-dimensional vector packing instances of each subclass (class, size) into one instance. Table 5.2 summarizes the results.

Table 5.2: Results for 20-dimensional vector packing.

	$n \in \{24, 25\}$		$n \in \{50, 51\}$		$n \in \{99, 100\}$		$n \in \{200, 201\}$	
class	n^{ip}	t^{tot}	n^{ip}	t^{tot}	n^{ip}	t^{tot}	n^{ip}	t^{tot}
1	0	0.09	0	0.81	0	36.28	0	1,374.18
2	0	0.01	0	0.01	0	0.01	0	0.01
3	0	0.01	0	0.01	0	0.01	0	0.01
4	0	50.27	-	-	-	-	-	-
5	0	73.20	-	-	-	-	-	-
6	0	0.01	0	0.02	0	0.05	0	0.19
7	0	0.01	0	0.03	0	0.06	0	0.19
8	0	0.01	0	0.01	0	0.03	0	0.10
9	0	0.05	0	0.37	0	12.80	-	-
10	0	0.02	0	0.11	0	0.91	0	14.52

n - total number of items; n^{ip} - number of nodes explored in the branch-and-cut procedure; t^{tot} - run time in seconds; "-" means that the instance was not solved by our algorithm.

The arc-flow formulation allowed the solution of 33 out of the 40 instances. The same subclasses that were solved in the 2-dimensional case were also solved in the 20-dimensional case. Moreover, some 20-dimensional instances were easier to solve than the original 2-dimensional ones due to the reduction in the number of valid packing patterns that results

from the addition of constraints. Seven instances were not solved within a 12 hour time limit; these are instances in which the patterns are very long. The average run time for the remaining 33 solved instances was 48 seconds, and none of these instances took longer than 23 minutes to be solved exactly.

5.2 Bin packing and cutting stock

Standard bin packing (BPP) and cutting stock (CSP) are one-dimensional vector packing problems whose set S of valid patterns is defined as follows:

$$A = \begin{bmatrix} w_1 & \dots & w_m \end{bmatrix} \qquad L = \begin{bmatrix} W \end{bmatrix} \qquad S = \{x \in \mathbb{Z}_+^m : Ax \leq L\} \tag{5.2.1}$$

We used the arc-flow formulation to solve a large variety of bin packing and cutting stock test datasets. OR-Library[4] provides two bin packing test datasets that were proposed by Falkenauer (1996). The first dataset (BFLK_u) is composed of uniform instances, where items have randomly generated weights, and the second dataset (BFLK_t) is composed of triplets instances, where each bin is completely filled with three items in the optimal solution. Each of these is further divided into subclasses of varying sizes. Scholl et al. (1997) provides three datasets that are available online.[5] The first of them (Scholl C1) is composed of randomly generated instances whose expected number of items per bin is not larger than 3. The second test dataset (Scholl C2) is composed of instances whose expected average number of items per bin is 3, 5, 7, or 9. Finally, the third test dataset (Scholl C3) is composed of 10 difficult instances with a total number of items $n = 200$ and bins of capacity $W = 100,000$. Schoenfield (2002) provides a bin packing test dataset (Hard28) composed of 28 instances selected from a huge testing. Among these 28 instances, 5 are non-IRUP (see, e.g., Baum and Trotter Jr. 1981), so the integer programming solver has to use branch-and-cut to reduce the gap and prove optimality. The remaining instances are IRUP, yet very hard for heuristics. ESICUP[6] provides two test datasets collected from Schwerin and Wäscher (1997) and Wäscher and Gau (1996) (SCH/WAE and WAE/GAU, respectively). We generated cutting stock datasets (CFLK_u and CFLK_t) from Falkenauer's bin packing test datasets by multiplying the demand of each item by one million; note that, however, multiplying the demand of each item by one million has very little effect on the model size. Two additional large test datasets for cutting stock problems are available online.[7] The first dataset (Cutgen)

[4] http://people.brunel.ac.uk/~mastjjb/jeb/info.html
[5] http://www.wiwi.uni-jena.de/entscheidung/binpp/
[6] http://paginas.fe.up.pt/~esicup/
[7] http://www-sys.ist.osaka-u.ac.jp/~umetani/benchmark.html

is composed of 1800 randomly generated instances of 18 classes. These instances were generated using the problem generator proposed by Gau and Wäscher (1995). The second dataset (Fiber) was taken from a real application in a chemical fiber company in Japan. Finally, a cutting stock test dataset (1Dbar) was obtained from 1D-bar relaxations of the two-dimensional bin packing test dataset of Lodi et al. (1999). Tables 5.3 and 5.4 present some characteristics of these datasets.

Table 5.3: Demand characteristics of BPP/CSP datasets.

dataset	type	m^{avg}	m^{min}	m^{max}	n^{avg}	n^{min}	n^{max}	b^{avg}	b^{min}	b^{max}
BFLK_u	BPP	75.56	58	81	467.50	120	1,000	5.92	1	24
BFLK_t	BPP	117.61	46	203	232.50	60	501	1.74	1	18
Scholl C1	BPP	63.78	31	100	212.50	50	500	2.96	1	20
Scholl C2	BPP	97.58	28	350	212.50	50	500	2.31	1	22
Scholl C3	BPP	199.00	197	200	200.00	200	200	1.01	1	3
Hard28	BPP	161.75	136	189	181.43	160	200	1.12	1	3
SCH/WAE	BPP	45.09	39	49	110.00	100	120	2.44	1	10
WAE/GAU	BPP	49.65	33	64	129.41	57	239	2.60	1	38
CFLK_u	CSP	75.56	58	81	4.68E8	1.20E8	1.00E9	5.92E6	1.0E6	2.40E7
CFLK_t	CSP	117.61	46	203	2.33E8	6.00E7	5.01E8	1.74E6	1.0E6	1.80E7
Fiber	CSP	10.79	4	20	418.79	155	1,121	42.82	1	555
Cutgen	CSP	22.46	8	40	1,283.37	100	4,000	56.81	1	504
1Dbar	CSP	60.00	20	100	2,140.19	93	7,478	35.67	1	100

Average, minimum and maximum values for: m - number of different items; n - number of items; b - demand.

Table 5.4: Size characteristics of BPP/CSP datasets.

dataset	type	W^{avg}	W^{min}	W^{max}	w^{avg}	w^{min}	w^{max}	r^{avg}	r^{min}	r^{max}
BFLK_u	BPP	150.00	150	150	60.35	20	100	0.40	0.13	0.67
BFLK_t	BPP	1,000.00	1,000	1,000	333.33	250	499	0.33	0.25	0.50
Scholl C1	BPP	123.33	100	150	58.56	1	100	0.49	0.01	1.00
Scholl C2	BPP	1,000.00	1,000	1,000	195.44	13	627	0.20	0.01	0.63
Scholl C3	BPP	100,000.00	100,000	100,000	27,503.03	20,000	35,000	0.28	0.20	0.35
Hard28	BPP	1,000.00	1,000	1,000	386.73	1	800	0.39	0.00	0.80
SCH/WAE	BPP	1,000.00	1,000	1,000	174.81	150	200	0.17	0.15	0.20
WAE/GAU	BPP	10,000.00	10,000	10,000	1,577.48	2	7,332	0.16	0.00	0.73
CFLK_u	CSP	150.00	150	150	60.35	20	100	0.40	0.13	0.67
CFLK_t	CSP	1,000.00	1,000	1,000	333.33	250	499	0.33	0.25	0.50
Fiber	CSP	7,080.00	5,180	9,080	930.23	500	2,000	0.14	0.06	0.39
Cutgen	CSP	1,000.00	1,000	1,000	344.59	10	800	0.34	0.01	0.80
1Dbar	CSP	98.00	10	300	35.14	1	100	0.41	0.00	1.00

Average, minimum and maximum values for: W - capacity; w - item sizes; r - relative item sizes.

The results are summarized in Table 5.5. No instance took longer than 5 hours to be solved to optimality. The average run time for the 4,114 instances was 13 seconds; 97% of these instances were solved in less than 1 minute each. The run time is usually very dependent on the graph size, as we show in Section 5.8. The hardest BPP and CSP instances for our method usually combine very small items and very large capacities, which tend to lead to huge graphs, due to the number of patterns that need to be represented. The

performance of other methods such as that proposed in Valério de Carvalho (1999) are very dependent on the capacity limits; in our method, graph compression attenuates this problem. We were able to solve easily all the instances from the third Scholl's dataset, which have capacity $100,000$, since the items are not very small; in Valério de Carvalho's model there would be around $80,000$ constraints, while in our model there are around $2,000$ constraints.

Table 5.5: Results for the standard BPP/CSP.

dataset	type	#inst.	$\#v$	$\#a$	$\%v$	$\%a$	n^{ip}	t^{tot}
BFLK_u	BPP	80	107.16	2,620.26	19%	14%	1.81	0.34
BFLK_t	BPP	80	125.35	4,987.63	80%	75%	1.10	0.92
Scholl C1	BPP	720	72.32	1,309.33	35%	36%	0.00	0.15
Scholl C2	BPP	480	596.99	30,824.36	33%	37%	0.10	43.43
Scholl C3	BPP	10	1,810.20	80,180.10	96%	95%	0.00	12.17
Hard28	BPP	28	789.46	27,284.00	19%	26%	102.57	29.69
SCH/WAE	BPP	200	210.11	3,553.57	70%	70%	0.00	0.62
WAE/GAU	BPP	17	6,235.06	128,212.76	33%	37%	2.59	1,641.09
CFLK_u	CSP	80	108.69	2,657.41	17%	13%	0.00	0.36
CFLK_t	CSP	80	122.06	5,032.21	81%	75%	0.00	0.83
Fiber	CSP	39	253.38	1,631.79	73%	71%	0.00	0.29
Cutgen	CSP	1,800	339.85	4,204.77	58%	51%	0.06	1.98
1Dbar	CSP	500	87.25	2,111.17	10%	7%	0.00	0.42

#inst. - number of instances; $\#v, \#a$ - average number of vertices and arcs in the final arc-flow graph; $\%v, \%a$ - average percentage of vertices and arcs removed by the graph compression method. n^{ip} - average number of nodes explored in the branch-and-cut procedure; t^{tot} - average run time in seconds.

5.3 Cardinality constrained bin packing and cutting stock

One of the BPP variants is the cardinality constrained bin packing in which, in addition to the capacity constraint, the number of items per bin is also limited. Similarly, one of the variants of the CSP is cutting stock with cutting knife limitation in which there is a limit on the number of pieces that can be cut from each roll due to a limit on the number of knives. The BPP and the CSP with cardinality constraints can be seen as special cases of the 2-dimensional vector packing problem. The set S of valid packing patterns for these problems is defined as follows:

$$A = \begin{bmatrix} w_1 & \dots & w_m \\ 1 & \dots & 1 \end{bmatrix} \qquad L = \begin{bmatrix} W \\ C \end{bmatrix} \qquad S = \{x \in \mathbb{Z}_+^m : Ax \leq L\} \tag{5.3.1}$$

Cardinality constrained bin packing is strongly NP-hard for any cardinality larger than 2 (see, e.g., Epstein and van Stee 2011); for cardinality 2, the cardinality constrained bin

packing problem can be solved in polynomial time as a maximum non-bipartite matching problem in a graph where each item is represented by a node and every compatible pair of items is connected by an edge.

Using the arc-flow formulation, we solved every instance from the bin packing and cutting stock datasets with cardinalities between 2 and the minimum cardinality limit that allowed the optimum to be the same with or without cardinality constraints. Table 5.6 summarizes the results for each dataset. No instance took longer than 8 hours to be solved to optimality. The average run time for the 14,568 instances was 13 seconds; 99% of these instances were solved in less than 1 minute each.

Table 5.6: Results for the cardinality constrained BPP/CSP.

dataset	type	#inst.	$C^{\max}$	$\#v$	$\#a$	$\%v$	$\%a$	n^{ip}	t^{tot}
BFLK_u	BPP	160	3	53.04	789.81	79%	79%	0.00	0.11
BFLK_t	BPP	160	3	64.13	2,610.92	92%	86%	2.88	0.58
Scholl C1	BPP	1,189	4	42.49	451.39	80%	81%	0.00	0.06
Scholl C2	BPP	2,529	10	301.93	8,927.75	89%	90%	0.05	14.71
Scholl C3	BPP	30	4	624.70	27,181.63	99%	98%	0.00	5.73
Hard28	BPP	56	3	100.73	1,991.96	94%	94%	25.13	0.82
SCH/WAE	BPP	1,000	6	74.90	924.78	89%	90%	0.00	0.20
WAE/GAU	BPP	131	18	6,833.89	103,768.11	91%	90%	0.73	999.58
CFLK_u	CSP	160	3	52.85	792.15	79%	79%	0.04	0.10
CFLK_t	CSP	160	3	62.08	2,632.82	93%	86%	0.00	0.44
Fiber	CSP	279	12	83.23	525.35	94%	90%	0.00	0.07
Cutgen	CSP	7,299	18	253.15	2,459.78	93%	89%	0.06	1.26
1Dbar	CSP	1,415	8	59.81	965.96	82%	79%	0.05	0.25

#inst. - number of instances; $C^{\max}$ - maximum cardinality limit; $\#v, \#a$ - average number of vertices and arcs in the final arc-flow graph; $\%v, \%a$ - average percentage of vertices and arcs removed by the graph compression method. n^{ip} - average number of nodes explored in the branch-and-cut procedure; t^{tot} - average run time in seconds.

Graph compression reduces substantially the graph sizes and usually leads to graphs of size comparable to the size without cardinality constraints. In fact, there are instances in which cardinality constraints help to reduce the final graph size, thus leading to easier models. The arc-flow model allowed us to solve the cardinality constrained BPP/CSP as easily as the standard BPP/CSP. We are not aware of any good method in the literature for solving the cardinality constrained BPP/CSP in general.

5.4 Graph coloring

The graph coloring problem is a combinatorial NP-hard problem in which one has to assign a color to each vertex of a graph in such a way that no two adjacent vertices share the same color and by using the minimum number of colors (see, e.g., Garey and Johnson 1979).

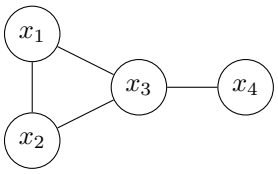
Graph coloring can be reduced to vector packing in several ways. Let variables x_i of constraints (5.1.2)–(5.1.5) represent whether or not vertex i appears in a given pattern (each pattern corresponds to a set of vertices that can share the same color). Considering each color as a bin and each vertex as an item with demand one, the following reductions are valid:

- Adjacency constraints: For each pair of adjacent vertices i and j, there is an adjacency constraint $x_i + x_j \leq 1$. Each adjacency constraint can be represented by a dimension k of capacity $W^k = 1$, with $w_i^k = w_j^k = 1$.
- Degree constraints: Let $\deg(i)$ and $\text{adj}(i)$ be the degree and the list of adjacent vertices of vertex i, respectively. For each vertex i, there is a constraint $\deg(i)x_i + \sum_{j \in \text{adj}(i)} x_j \leq \deg(i)$. Each constraint can be represented by a dimension k of capacity $W^k = \deg(i)$, with $w_i^k = \deg(i)$ and $w_j^k = 1$ for every $j \in \text{adj}(i)$.
- Clique constraints: For each clique C, there is a constraint $\sum_{i \in C} x_i \leq 1$. Each clique constraint can be represented by a dimension k of capacity $W^k = 1$, with $w_i^k = 1$ for every $i \in C$. The algorithm of Bron and Kerbosch (1973) can be used to decompose the graph into maximal cliques.

Using any of the three reductions above, a vector packing solution with z bins corresponds to a graph coloring solution with z colors. Different reductions result in different vector packing instances that can be harder or easier to solve. According to our experiments, it is usually a good idea to choose reductions that lead to vector packing instances with fewer dimensions. Figure 5.4.1 illustrates the three reductions defined above.

Figure 5.4.1: Graph coloring reductions to vector packing.

Graph coloring instance:



Adjacency constraints:

$$\begin{aligned} x_1+x_2 &\leq 1, \\ x_1 \quad +x_3 &\leq 1, \\ x_2+x_3 &\leq 1, \\ x_3+x_4 &\leq 1, \\ x_i \in \{0,1\},\ i = 1..4 & \end{aligned}$$

Degree constraints:

$$\begin{aligned} 2x_1+x_2+x_3 &\leq 2, \\ 2x_2+x_1+x_3 &\leq 2, \\ 3x_3+x_1+x_2+x_4 &\leq 3, \\ 1x_4+x_3 &\leq 1, \\ x_i \in \{0,1\},\ i = 1..4 & \end{aligned}$$

Clique constraints:

$$\begin{aligned} x_1+x_2+x_3 &\leq 1, \\ x_3+x_4 &\leq 1, \\ x_i \in \{0,1\},\ i = 1..4 & \end{aligned}$$

Note that, in graph coloring, the lengths of the patterns are usually very long, especially when the graphs are sparse. However, there are problems that can be reduced to graph coloring problems with reasonably short patterns, and thus it may be possible to solve them using the proposed arc-flow model.

Table 5.7 shows the results for a small subset of graph coloring instances from OR-Library. These instances correspond to queen graphs; given a $q \times q$ chessboard, there are q^2 nodes (one for each square of the board) which are connected by an edge if the corresponding squares are in the same row, column, or diagonal. If there is a solution

with at most q colors, then it is possible to place q sets of q queens on the board so that no two queens of the same set are in the same row, column, or diagonal. In these instances, q limits the length of the color patterns and hence the arc-flow formulation can be applied with success for small values of q. We reduced these instances to vector packing through degree constraints.

Table 5.7: Results for graph coloring.

name	n	e	p	z^*	z^{lp}	$\#v$	$\#a$	t^{pp}	t^{lp}	t^{ip}	n^{ip}	t^{tot}
queen5_5	25	320	25	5	5.00	95	378	0.01	0.00	0.01	0	0.02
queen6_6	36	580	36	7	7.00	367	1,502	0.04	0.02	0.03	0	0.09
queen7_7	49	952	49	7	7.00	1,559	6,571	0.23	0.11	0.12	0	0.46
queen8_8	64	1,456	64	9	8.44	7,799	34,280	1.69	1.78	5.41	0	8.88

n - number of vertices/items; e - number of edges; p - number of dimensions used to represent the constraints; z^* - optimal integer solution; z^{lp} - linear relaxation; $\#v, \#a$ - number of vertices and arcs in the final arc-flow graph; t^{pp} - time spent building the graph; t^{lp} - time spent in the linear relaxation of the root node; t^{ip} - time spent in the branch-and-cut procedure; n^{ip} - number of nodes explored in the branch-and-cut procedure; t^{tot} - total run time in seconds.

5.4.1 Timetabling (reduced to graph coloring)

The timetabling problem has several variants and applications in many areas (see, e.g., Abramson 1991 and Smith et al. 2003). In this section, we consider the class-teacher-venue problem in which one has to find a conflict-free timetable. Suppose there are c classes, t teachers, and v venues. Given the number of times each pair class-teacher must meet at each venue, we want to find a timetable with zero clashes. Classes, teachers and venues cannot be chosen twice for the same time period. These constraints are represented by dimensions α, γ and δ. For each class k, there is a dimension α^k of capacity 1. For each teacher k, there is a dimension γ^k of capacity 1. For each venue k, there is a dimension δ^k of capacity 1. Each requirement is a triplet (class c_i, teacher t_i, venue v_i) that we represent by an item i with weights $\alpha_i^{c_i} = \gamma_i^{t_i} = \delta_i^{v_i} = 1$; the demand of each item is the number of times c_i must meet t_i at v_i. In addition, there is a limit on the number of available time slots (that are represented by bins). This constraint is replaced by searching for a solution that minimizes the number of periods. This reduction can be seen as a graph coloring reduction to vector packing through clique constraints. The set S of valid patterns for each time period is defined as follows:

$$A = \begin{bmatrix} \alpha_1^1 & \dots & \alpha_m^1 \\ \vdots & & \vdots \\ \alpha_1^c & \dots & \alpha_m^c \\ \gamma_1^1 & \dots & \gamma_m^1 \\ \vdots & & \vdots \\ \gamma_1^t & \dots & \gamma_m^t \\ \delta_1^1 & \dots & \delta_m^1 \\ \vdots & & \vdots \\ \delta_1^v & \dots & \delta_m^v \end{bmatrix} \qquad L = \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \qquad S = \{x \in \mathbb{Z}_+^m : Ax \leq L\}$$

The arc-flow model was used to solve the "hard timetabling" instances from OR-Library[8]. The hard classification comes from the fact that these instances have been designed so that each class, teacher and venue is required for each time period. Each instance has a solution with zero clashes that uses no more than 30 periods (6 periods per day, 5 days per week). All the instances were solved quickly except hdtt8 that took several hours to be solved. Table 5.8 shows the results for this problem.

Table 5.8: Results for timetabling.

name	t	c	v	n	m	$\#v$	$\#a$	t^{pp}	t^{lp}	t^{ip}	n^{ip}	t^{tot}
hdtt4	4	4	4	120	59	148	906	0.03	0.02	0.03	0	0.07
hdtt5	5	5	5	150	88	644	4,221	0.14	0.15	0.29	0	0.58
hdtt6	6	6	6	180	125	2,719	19,566	0.95	2.62	12.51	0	16.08
hdtt7	7	7	7	210	154	11,140	82,725	5.59	47.62	1,697.31	16	1,750.52
hdtt8	8	8	8	240	197	43,397	368,072	32.46	2,329.86	53,983.17	15	56,345.49

t, c, v - number of teachers, classes and venues. n - number of items/requirements; m - number of different items/requirements; $\#v, \#a$ - number of vertices and arcs in the final arc-flow graph; t^{pp} - time spent building the graph; t^{lp} - time spent in the linear relaxation of the root node; t^{ip} - time spent in the branch-and-cut procedure; n^{ip} - number of nodes explored in the branch-and-cut procedure; t^{tot} - total run time in seconds.

5.5 Bin packing with conflicts

The bin packing problem with conflicts (BPPC) is one of the most important bin packing variants. This problem consists of the combination of bin packing with graph coloring. In addition to the capacity constraints, there are compatibility constraints. This problem can be solved as a vector packing problem with $c+1$ dimensions, where c is the number of dimensions used to model conflicts. The set S of valid packing patterns for this problem

[8] `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`

can be defined as follows:

$$A = \begin{bmatrix} w_1 & \dots & w_n \\ \alpha_1^1 & \dots & \alpha_n^1 \\ \vdots & & \vdots \\ \alpha_1^c & \dots & \alpha_n^c \end{bmatrix} \qquad L = \begin{bmatrix} W \\ \beta^1 \\ \vdots \\ \beta^c \end{bmatrix} \qquad S = \{x \in \mathbb{Z}_+^n : Ax \leq L\} \tag{5.5.1}$$

The conflicts (the last c rows of A and L) can be modeled using any of the graph coloring reductions to vector packing presented in Section 5.4. In our experiments, we used degree constraints for modeling conflicts in this problem.

In order to test the arc-flow formulation in the BPPC, we used the dataset proposed by Fernandes-Muritiba et al. (2010). This dataset was created from the bin packing dataset of Falkenauer (1996), adding conflict graphs with several densities. Tables 5.9 and 5.10 summarize the results for each class and density, respectively. Fernandes-Muritiba et al. (2010) solved some of these instances using a branch-and-price algorithm under a time limit of 10 hours. Sadykov and Vanderbeck (2013) solved all the instances within a one-hour time limit using a branch-and-price algorithm. As opposed to our method, both branch-and-price algorithms of Fernandes-Muritiba et al. (2010) and Sadykov and Vanderbeck (2013) included algorithms tailored for bin packing with conflicts, namely special purpose algorithms form solving the subproblem. The instances of class u1000 were the most difficult for our method. Note that instances of this class have 1000 items and, in the literature, instances with 200 items are already considered difficult even in the one-dimensional case. Nevertheless, no instance took longer than 50 minutes to be solved exactly. The average run time of our method in the 800 instances was 2 minutes and 80% of these instances were solved in less than 1 minute each. In this problem, due to the high number of dimensions, a large part of the run time is spent building the arc-flow graph.

Table 5.9: Results for the BPP with conflicts.

class	#inst.	n	d	$d^{\max}$	$\#v$	$\#a$	t^{pp}	t^{lp}	t^{ip}	n^{ip}	t^{tot}
u120	100	120	84.96	121	359.99	3,012.12	0.18	0.06	0.18	0.07	0.43
u250	100	250	175.21	251	1,167.75	9,393.41	2.21	0.37	1.29	0.00	3.86
u500	100	500	350.48	501	3,486.90	27,440.63	32.40	1.87	10.88	0.00	45.16
u1000	100	1,000	702.21	1001	11,316.90	88,323.32	643.68	13.24	104.79	0.00	761.72
t60	100	60	42.22	61	99.69	693.14	0.03	0.01	0.02	0.00	0.06
t120	100	120	84.01	121	298.36	2,627.20	0.20	0.05	0.27	3.70	0.52
t249	100	249	174.42	250	1,045.20	10,300.08	2.82	0.32	1.65	3.12	4.78
t501	100	501	352.26	502	3,796.35	36,809.01	53.90	2.56	17.50	0.00	73.95

Sadykov and Vanderbeck (2013) also propose harder instances, most of which we were not able to solve in less than 12 hours. In order to solve these instances, they used branch-and-bound to solve the subproblems, instead of the dynamic programming algorithm

Table 5.10: Results for the BPP with conflicts (grouped by density).

density	#inst.	d	$d^{\max}$	$\#v$	$\#a$	t^{pp}	t^{lp}	t^{ip}	n^{ip}	t^{tot}
0%	80	1.00	1	113.47	12,288.20	0.28	0.24	1.24	0.00	1.76
10%	80	69.76	222	814.02	20,034.35	18.35	0.62	4.01	0.00	22.98
20%	80	140.60	420	2,077.11	24,327.59	75.96	1.38	11.45	0.00	88.79
30%	80	211.31	632	3,540.24	29,554.71	161.19	2.77	50.21	8.61	214.18
40%	80	280.51	818	4,951.05	35,266.04	233.93	4.73	70.75	0.00	309.40
50%	80	350.02	1,001	5,862.62	39,442.55	227.11	6.55	17.02	0.00	250.68
60%	80	351.00	1,001	4,456.75	29,639.79	126.87	4.03	10.46	0.00	141.36
70%	80	351.00	1,001	3,020.04	19,390.99	56.31	1.95	4.06	0.00	62.32
80%	80	351.00	1,001	1,643.91	10,220.38	16.17	0.73	1.37	0.00	18.28
90%	80	351.00	1,001	484.70	3,084.05	3.10	0.09	0.16	0.00	3.35

#inst. - number of instances; n - number of items; d - average number of dimensions; $\#v, \#a$ - average number of vertices and arcs in the final arc-flow graph; t^{pp} - average time spent building the graph; t^{lp} - average time spent in the linear relaxation of the root node; t^{ip} - average time spent in the branch-and-cut procedure; n^{ip} - average number of nodes explored in the branch-and-cut procedure; t^{tot} - average run time in seconds.

that they used on the dataset of Fernandes-Muritiba et al. (2010). A possible reason for this is that the dynamic programming search space is difficult to represent in a compact form for these harder instances. A limitation of our method is the combination of large capacities and long patterns, which can be difficult to represent in a compact way. In practice, branch-and-price can overcome this problem by using a tailored algorithm in the sub-problem. This is not possible in our method; but when it generates reasonably small graphs, it usually outperforms more complex approaches, such as branch-and-price algorithms.

5.6 Cutting stock with binary patterns

Cutting stock with binary patterns (0–1 CSP) is a CSP variant in which items of each type may be cut at most once in each roll. In this problem, pieces are identified by their types and some types may have the same weight. This problem usually appears as bar and slice relaxations of orthogonal packing problems (see, e.g., Scheithauer 1999 and Belov et al. 2009). Cutting stock with binary patterns can be modeled as a vector packing problem with $m + 1$ dimensions. The binary constraints are introduced by m binary dimensions and the set S of valid packing patterns for this problem can be defined as follows:

$$A = \begin{bmatrix} w_1 & w_2 & \dots & w_m \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \qquad L = \begin{bmatrix} W \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \qquad S = \{x \in \mathbb{Z}_+^m : Ax \leq L\}$$

The arc-flow formulation was used to solve all the CSP instances with binary patterns. Table 5.11 summarizes the results for each dataset. The average run time in the 2,499 instances was 5 seconds and 98% of the instances were solved in less than one minute.

Table 5.11: Results for the 0–1 CSP.

dataset	#inst.	$m^{\max}$	$n^{\max}$	$W^{\max}$	$\#v$	$\#a$	t^{pp}	t^{lp}	t^{ip}	n^{ip}	t^{tot}
CFLK_u	80	81	1.00E9	150	727.86	3,586.51	0.23	0.15	0.50	0.14	0.88
CFLK_t	80	203	5.01E8	1,000	175.08	5,039.54	0.69	0.13	0.79	0.00	1.62
Fiber	39	20	1,121	9,080	104.33	357.54	0.02	0.01	0.01	0.00	0.03
Cutgen	1,800	40	4,000	1,000	1,001.79	3,320.18	0.23	1.22	3.71	0.00	5.16
1Dbar	500	100	7,478	300	889.21	3,095.53	0.58	0.36	2.44	0.12	3.39

#inst. - number of instances; $m^{\max}$ - maximum number of different items; $n^{\max}$ - maximum number of items; $W^{\max}$ - maximum bin capacity; $\#v, \#a$ - average number of vertices and arcs in the final arc-flow graph; t^{pp} - average time spent building the graph; t^{lp} - average time spent in the linear relaxation of the root node; t^{ip} - average time spent in the branch-and-cut procedure; n^{ip} - average number of nodes explored in the branch-and-cut procedure; t^{tot} - average run time in seconds.

The time required to construct the graphs could be improved by using the technique presented in Section 4.2.2.3 for modeling binary constraints using a single dimension instead of m binary dimensions.

5.7 Cutting stock with binary patterns and forbidden pairs

Cutting stock with binary patterns and forbidden pairs (0–1 CSPC) is a variant of cutting stock with binary patterns that also includes compatibility constraints. This problem usually appears as a relaxation of orthogonal packing problems (see, e.g., Belov et al. 2009). It can be modeled as a vector packing problem with $c + m + 1$ dimensions, where c is the number of dimensions used to model the conflicts. The set S of valid packing patterns for this problem can be defined as follows:

$$A = \begin{bmatrix} w_1 & w_2 & \dots & w_m \\ \alpha_1^1 & \alpha_2^1 & \dots & \alpha_m^1 \\ \vdots & \vdots & & \vdots \\ \alpha_1^c & \alpha_2^c & \dots & \alpha_m^c \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \qquad L = \begin{bmatrix} W \\ \beta^1 \\ \vdots \\ \beta^c \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \qquad S = \{x \in \mathbb{Z}_+^m : Ax \leq L\}$$

In this problem, we also used degree constraints to model the conflicts. Since these constraints already guarantee that the corresponding item cannot occur more than once in the same pattern, the binary constraints for items with conflicts are discarded. In order to test the behavior of the arc-flow model in this problem, we created a 0–1 CSPC dataset from the BPPC dataset of Fernandes-Muritiba et al. (2010). We assigned random values of demand between 1 and 100 to each item. Tables 5.12 and 5.13 summarize the results for each class and density, respectively. The average run time in the 800 instances was 6 minutes and 72% of these instances were solved in less than 1 minute. Some instances took almost 3 hours to be solved and 1 instance took almost 20 hours. Note that the graphs sizes for this type of problems can be really large due to the high number of different items and dimensions; in the class u1000, every instance has thousands of items of 1000 different types and 1001 dimensions.

Table 5.12: Results for the CSP with binary patterns and conflicts.

class	#inst.	m	$n^{\max}$	d	$d^{\max}$	$\#v$	$\#a$	t^{pp}	t^{lp}	t^{ip}	n^{ip}	t^{tot}
u120	100	120	6,989	121	121	724.72	3,763.07	0.37	0.12	0.33	0.00	0.82
u250	100	250	13,672	251	251	2,266.74	12,211.75	4.64	0.70	2.43	0.00	7.76
u500	100	500	26,568	501	501	6,409.75	37,970.82	72.01	4.04	21.92	0.00	97.98
u1000	100	1,000	52,492	1,001	1,001	19,171.67	126,689.21	1,518.38	28.63	1,198.08	48.58	2,745.09
t60	100	60	3,567	61	61	110.98	709.32	0.04	0.01	0.03	0.00	0.08
t120	100	120	6,856	121	121	322.66	2,637.34	0.32	0.05	0.20	0.44	0.57
t249	100	249	13,578	250	250	1,102.75	10,311.04	5.17	0.35	2.29	0.00	7.81
t501	100	501	27,933	502	502	3,873.08	36,974.09	113.25	2.66	60.12	14.84	176.03

Table 5.13: Results for the CSP with binary patterns and conflicts (grouped by density).

density	#inst.	d	$d^{\max}$	$\#v$	$\#a$	t^{pp}	t^{lp}	t^{ip}	n^{ip}	t^{tot}
0%	80	351	1,001	3,006.74	18,074.72	152.58	1.29	12.93	0.11	166.80
10%	80	351	1,001	4,855.32	33,330.04	413.73	5.15	42.34	0.55	461.22
20%	80	351	1,001	5,910.48	42,229.65	438.98	8.10	99.10	2.50	546.18
30%	80	351	1,001	6,627.90	47,384.56	389.95	10.85	281.94	18.26	682.74
40%	80	351	1,001	6,613.45	46,262.95	317.95	9.24	1,071.72	58.40	1,398.90
50%	80	351	1,001	5,858.65	39,466.18	226.37	5.31	53.56	0.00	285.24
60%	80	351	1,001	4,456.75	29,639.79	127.10	3.39	34.20	0.00	164.68
70%	80	351	1,001	3,020.04	19,390.99	56.67	1.65	9.60	0.00	67.92
80%	80	351	1,001	1,643.91	10,220.38	16.26	0.64	1.20	0.00	18.10
90%	80	351	1,001	484.70	3,084.05	3.12	0.09	0.17	0.00	3.38

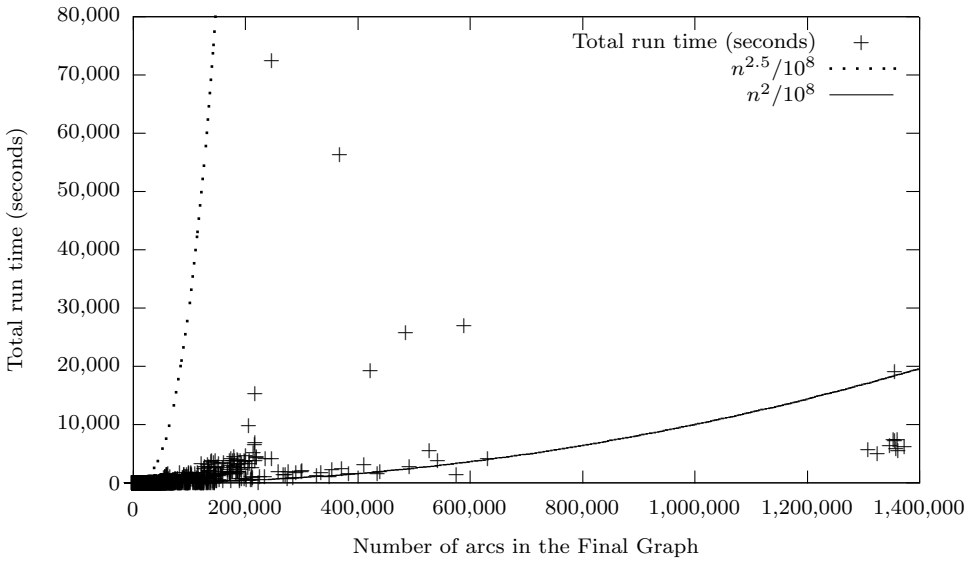
#inst. - number of instances; m - number of different items; $n^{\max}$ - maximum number of items; d - average number of dimensions; $\#v, \#a$ - average number of vertices and arcs in the final arc-flow graph; t^{lp} - average time spent in the linear relaxation of the root node; t^{ip} - average time spent in the branch-and-cut procedure; n^{ip} - average number of nodes explored in the branch-and-cut procedure; t^{tot} - average run time in seconds.

5.8 Run time analysis

Using the proposed method, we solved sequentially 23,153 benchmark instances in 9 days, spending 33 seconds per instance, on average. These benchmark instances belong to several different problems. The same method was used to tackle all the instances without any problem-specific adjustment. The linear relaxations are very strong in every problem we considered. The largest absolute gap (i.e., absolute difference between the optimal solution value and the value of the continuous relaxation of the model) we found in all the instances from benchmark test datasets was 1.0027.

For instances that could be solved by our method (all except 77 instances), Figure 5.8.1 shows the relation between the number of arcs in the final arc-flow graph and the total run time. The two curves $n^2/10^8$ and $n^{2.5}/10^8$ show an approximation of the run time (in seconds) for algorithms with complexities $\Theta(n^2)$ and $\Theta(n^{2.5})$, with very low constant factors. The large majority of the observed run times for our method appear between these two curves, and many of the observed run times are very close to the quadratic run time. The few instances that lead to run times far from quadratic are mainly instances where the number of items that fit in each bin is large (e.g., more than 10) and hence the total number of patterns is huge.

Figure 5.8.1: Run time analysis (Gurobi).



Very good results were also obtained using non-commercial MIP solvers such as `COIN-OR`.[9] Note that the non-commercial solvers are usually inferior to commercial solvers and hence they may not be able to solve the large models as easily as `Gurobi`.[10]

[9] `http://www.coin-or.org/`
[10] `http://www.gurobi.com`

Chapter 6

General applications

In this chapter, we present the results obtained using the arc-flow formulation on more general applications which cannot be trivially reduced to vector packing instances. Results were obtained using a computer with a Quad-Core Intel Xeon at 2.66GHz, running Mac OS X 10.11.6, with 16 GBytes of memory. The graph construction algorithm was implemented in `C++`, the models were built using `Python 2.7`, and the resulting MIP was solved using `Gurobi` 7.0.2, a state-of-the-art mixed integer programming solver. The source code is available online.[1] The parameters used in `Gurobi` were Threads = 1 (single thread), Presolve = 1 (conservative), Method = 2 (interior-point methods), MIPGap = 0, MIPGapAbs = $1-10^{-5}$ and the remaining parameters were `Gurobi`'s default values.

6.1 Multi-stage vector packing problem

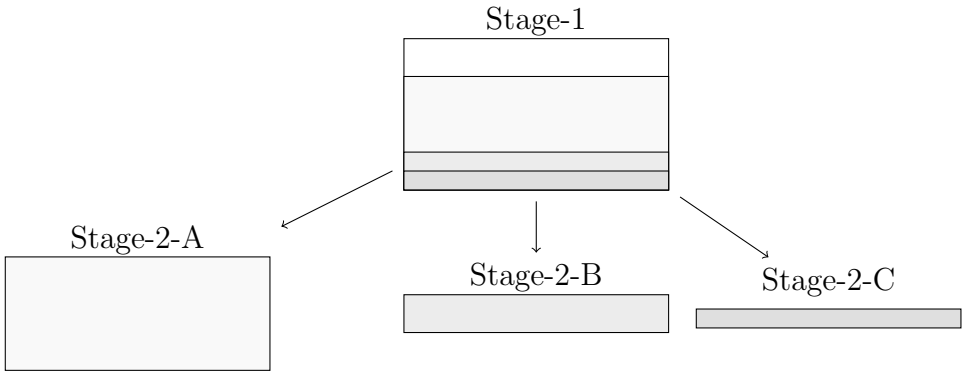
In the two-dimensional cutting stock problem, the objective is to cut a set of rectangular items from rectangular plates in such a way that the number of plates used is minimized. In the two-stage variant, the items are obtained by first performing a set of horizontal guillotine cuts, dividing the plate in strips, followed by a set of vertical cuts separating the items. In the three-stage variant, a third set of cuts is allowed: a set of vertical cuts is performed on the horizontal strips, produced in the first stage, producing vertical strips, which are then cut into items by a third set of horizontal cuts. These problems can be exact or non-exact; in the non-exact case an additional set of cuts is allowed to separate the items from waste, while in the exact case the items must be cut exactly by the guillotine cuts. In theory, the dimension of the strips in the first stage should be every possible integer linear combination of item heights, and the width of the strips in the second stage should be every possible integer linear combination of item widths. However, a restricted version of these problems is often considered instead; in this version, the strips height in the first stage are confined to the height of the items, and the strips width in the second stage are confined to the width of the items.

[1] `https://github.com/fdabrandao/vpsolver` or `http://www.dcc.fc.up.pt/~fdabrandao/code`

Macedo et al. (2010) presents an exact model for the two-dimensional cutting stock problem with two stages and guillotine constraints based on Valério de Carvalho's arc-flow formulation. They use an arc-flow graph for the first stage, and a series of arc-flow graphs for each stage-2 strip. We generalize this idea to multiple initial plates, any number of stages, and multiple constraints per stage (e.g., a cardinality constraint limiting the number of cuts that can be performed in each stage). We call this generalization multi-stage vector packing.

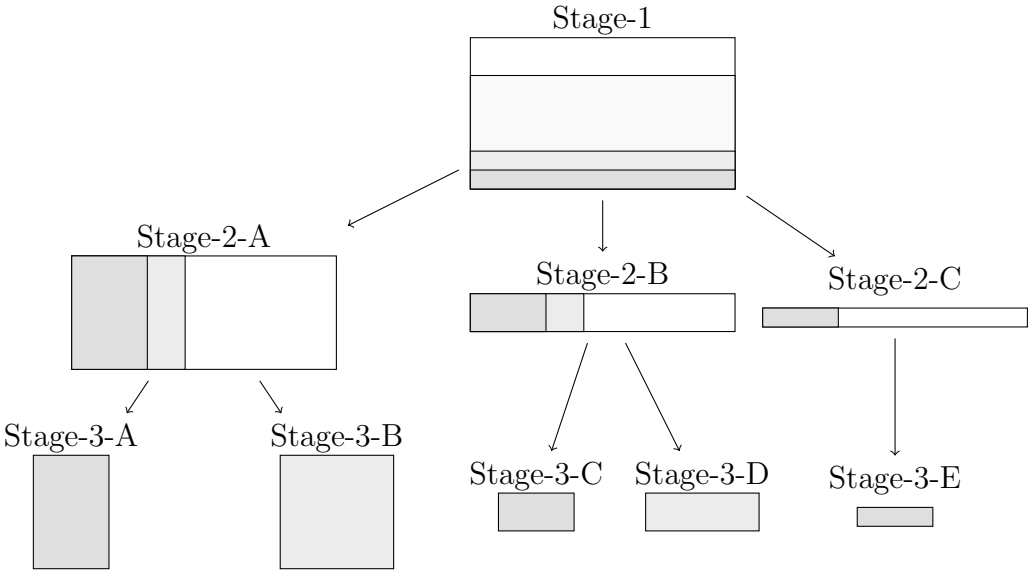
A multi-stage vector packing instance can be modeled as series of vector packing instances. Items produced in any non-final stage are plates in instances of the following stage. Figures 6.1.1, 6.1.2, and 6.1.3 illustrate how a series of cutting stock instances can be enumerated for a restricted three-stage non-exact cutting stock problem. In Figure 6.1.1, we enumerate horizontal cuts on stock plates to generate stage-2 strips; for each stock plate there is a cutting stock problem associated with capacity equal to its height. In Figure 6.1.2, we apply vertical cuts on stage-2 strips to produce stage-3 strips; for each stage-2 strip there is a cutting stock problem associated with capacity equal to its width. Finally, in Figure 6.1.3, we apply horizontal cuts on stage-3 strips to produce items; for each stage-3 strip there is a cutting stock problem associated with capacity equal to its height.

Figure 6.1.1: Instance enumeration (first stage).



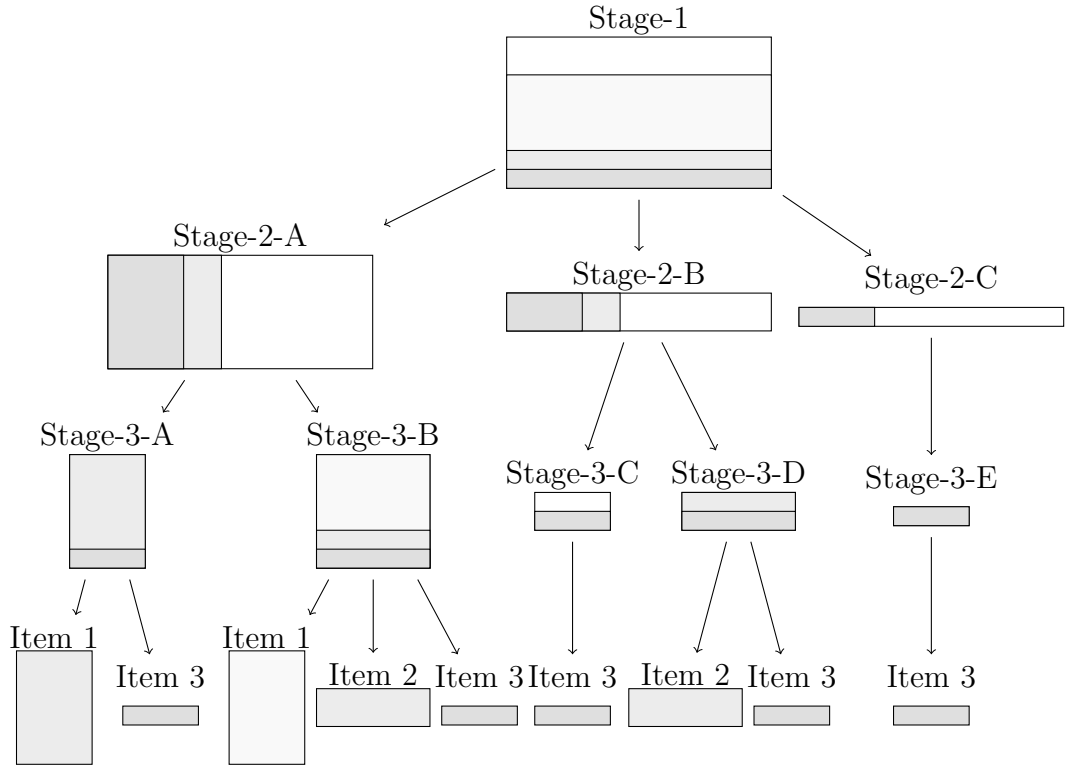
At the root of the tree, we have the stock plates that are going to be cut. There may exist multiple nodes at the first level if there are multiple plate sizes. We apply horizontal cuts on the plates at the first level to generate each of the stage-2 strips for the second level.

Figure 6.1.2: Instance enumeration (second stage).



We apply vertical cuts on stage-2 strips (second-level) to generate stage-3 strips (third-level).

Figure 6.1.3: Instance enumeration (third stage).



We apply horizontal cuts on stage-3 strips (third-level) to generate items. Note that this example is a non-exact three-stage problem. In an exact three-stage problem, the items must have the same width as the strip (i.e., items 1 and 3 could not be cut from Stage-3-B, and item 3 could not be cut from Stage-3-D).

Let I be the set of item types. We label each individual plate type with an index $s \in S$, $S \cap I = \emptyset$, which is the label used in the arc-flow graphs for items that correspond to the production of that plate type. Each stock plate $s_0 \in S_0 \subseteq S$ has a cost C_{s_0}.

Note that, in Figure 6.1.3, instances Stage-3-A, Stage-3-C, Stage-3-E are the same cutting stock instance but with different roll lengths. The same happens with Stage-3-B and Stage-3-D. When there are multiple cutting stock instances with the same set of items associated, we can group them into variable-sized cutting stock instances. Instead of building an arc-flow graph for every plate type (i.e., for every cutting stock problem), we group compatible instances into variable-sized cutting stock instances. Let Γ be the index set of variable-sized cutting stock instances. For each variable-sized cutting stock problem $\gamma \in \Gamma$, we build an arc-flow graph $G_\gamma = (V_\gamma, A_\gamma)$. For any plate type $s \in S$, the corresponding cutting stock problem is identified by its group $\gamma \in \Gamma$ and the type t corresponding to its size; $\Phi(s) = (\gamma, t)$ identifies the problem corresponding to each plate type $s \in S$.

Let decision variables f^i_{guv} be the amount of flow along the arc (u, v, i) of graph g; for each plate type s, $f_{\gamma \mathrm{T} t \mathrm{S}}$, with $(\gamma, t) = \Phi(s)$, is the number of plates of type s used. Given arc-flow graphs $G_\gamma = (V_\gamma, A_\gamma)$, for every $\gamma \in \Gamma$, we model the corresponding multi-stage cutting stock problem as follows:

$$\min \quad \sum_{\substack{s_0 \in S_0, \\ (\gamma,t)=\Phi(s_0)}} C_{s_0} f^0_{\gamma \mathrm{T} t \mathrm{S}} \tag{6.1.1}$$

$$\text{s.t.} \quad \sum_{(u,v,i) \in A_\gamma : v = k} f^i_{\gamma uv} - \sum_{(u,v,i) \in A_\gamma : u = k} f^i_{\gamma uv} = 0, \quad \gamma \in \Gamma, k \in V_\gamma, \tag{6.1.2}$$

$$\sum_{g \in \Gamma} \sum_{(u,v,i) \in A_g : i \in I_j} f^i_{guv} \geq d_j, \qquad j \in J, \tag{6.1.3}$$

$$\sum_{g \in \Gamma} \sum_{(u,v,i) \in A_g : i = s} f^i_{guv} \geq f^0_{\gamma \mathrm{T} t \mathrm{S}}, \qquad s \in S, (\gamma, t) = \Phi(s), \tag{6.1.4}$$

$$f^i_{\gamma uv} \geq 0, \text{ integer}, \qquad \gamma \in \Gamma, (u, v, i) \in A_\gamma. \tag{6.1.5}$$

Note that having multiple graphs connected with constraints of type (6.1.4) is equivalent to having a single graph in which each arc whose flow contributes to the feedback arc of another is replaced by a copy of the other graph.

Silva et al. (2010) proposed a extension of the one-cut model to tackle two- and three-stage two-dimensional cutting stock problems. In this extension each cut generates an item and two residual plates. They tested their method on instances from real-world applications and in two datasets adapted from the literature. Table 6.1 summarizes the instance characteristics for the two datasets adapted from the literature. Silva et al. (2010) solved

restricted versions of these problems. Note that, in their restricted version the item that defines the strip size must be in the strip, while our method is a little bit less restricted, since it just restricts the sizes of the strips to dimensions of items. Moreover, our method can be used to solve the non-restricted version of the problem, but that results in much larger models and usually does not have a significant impact in the solution.

Table 6.1: Two-dimensional instances.

Dataset 1

Instance	m	n	d_{avg}	W	H	$w_{\min}$	$h_{\min}$
2	10	23	2.3	40	70	9	7
3	19	62	3.3	40	70	9	11
A1	19	62	3.3	50	60	9	11
A2	20	53	2.7	60	60	12	14
A3	20	46	2.3	70	80	15	14
A4	19	35	1.8	90	70	9	11
A5	20	45	2.3	132	100	13	12
CHL1	30	63	2.1	132	100	13	12
CHL2	10	19	1.9	62	55	11	9
CHL5	10	18	1.8	20	20	1	2
CHL6	30	65	2.2	130	130	18	12
CHL7	34	75	2.2	130	130	19	18
CU1	25	82	3.3	100	125	20	28
CU2	34	90	2.6	150	175	31	35
CW1	25	67	2.7	125	105	25	21
CW2	35	63	1.8	145	165	34	34
CW3	40	96	2.4	267	207	59	45
Hchl2	34	75	2.2	130	130	19	18
Hchl3s	10	51	5.1	127	98	15	13
Hchl4s	10	32	3.2	127	98	15	13
Hchl6s	22	60	2.7	253	244	35	38
Hchl7s	40	90	2.3	263	241	33	38
Hchl8s	10	18	1.8	49	20	1	2
Hchl9	35	76	2.2	65	76	10	10
HH	5	18	3.6	127	98	18	13
OF1	10	23	2.3	70	40	9	4
OF2	10	24	2.4	70	40	13	4
STS2	30	78	2.6	55	85	10	10
STS4	20	50	2.5	99	99	14	14
W	19	62	3.3	70	40	11	9

Dataset 2

Instance	m	n	d_{avg}	W	H	$w_{\min}$	$h_{\min}$
ATP30	38	192	5.1	927	152	57	7
ATP31	51	258	5.1	856	964	44	50
ATP32	55	249	4.5	307	124	16	6
ATP33	44	224	5.1	241	983	15	52
ATP34	27	130	4.8	795	456	46	22
ATP35	29	153	5.3	960	649	50	34
ATP36	28	153	5.5	537	244	30	20
ATP37	43	222	5.2	440	881	23	51
ATP38	40	202	5.1	731	358	41	19
ATP39	33	163	4.9	538	501	28	48
ATP40	56	289	5.2	683	138	34	6
ATP41	36	177	4.9	837	367	43	32
ATP42	58	325	5.6	167	291	8	21
ATP43	49	259	5.3	362	917	19	46
ATP44	39	196	5	223	496	11	29
ATP45	33	156	4.7	188	578	9	49
ATP46	42	197	4.7	416	514	23	40
ATP47	43	204	4.7	392	554	25	32
ATP48	34	167	4.9	931	254	47	18
ATP49	25	119	4.8	759	449	42	23

m - number of item types; n - number of items; d_{avg} - average demand; W - plate width; H - plate height; $w_{\min}$ - smaller item width; $h_{\min}$ - smaller item height.

Table 6.2 presents the results for exact and non-exact two-stage problems. Table 6.3 presents the results for exact and non-exact three-stage problems. In these tables, we compare the solutions obtained with our method with the solutions obtained by Silva et al. (2010). Note that in our experiment the time limit for the solution of the model was set to 30 minutes, while in their experiment the time limit was set to 2 hours. Nevertheless, our method was able to find solutions at least as good in all instances except one, found better solutions for 14 instances instances, and found solutions for 4 unsolved instances.

In order to test the limits of our method, we also solved the same set of instances but with allowed rotation of items. Table 6.4 presents the results for exact and non-exact two-stage problems with allowed rotation of items. Table 6.5 presents the results for exact

and non-exact three-stage problems with allowed rotation of items.

Table 6.2: Results for exact and non-exact two-stage problems.

	2E								2NE							
Instance	#g	#vars	#cons	z^{lp}	z^{ip}	n^{ip}	t^{tot}	imp.	#g	#vars	#cons	z^{lp}	z^{ip}	n^{ip}	t^{tot}	imp.
2	8	166	80	2.26	3	0	2.94	0	8	327	107	1.82	2	0	2.96	0
3	20	309	136	23.50	24	0	7.17	0	20	714	206	22.50	23	0	7.33	0
A1	20	226	130	27.00	27	0	7.28	0	20	1,079	277	22.50	23	0	7.23	0
A2	19	187	129	14.50	15	0	6.89	0	19	1,621	333	11.61	12	0	7.06	0
A3	19	326	145	9.18	10	0	6.93	0	19	1,613	341	7.10	8	0	7.07	0
A4	20	290	143	7.58	8	0	7.15	0	20	3,856	659	4.20	5	0	8.14	0
A5	15	264	126	7.25	8	0	5.44	0	15	2,772	505	4.04	5	0	5.60	0
CHL1	23	530	197	10.07	11	0	8.27	0	23	10,813	1,225	5.31	6	506	34.01	0
CHL2	11	152	80	3.48	4	0	3.91	0	11	403	125	2.31	3	0	3.98	0
CHL5	10	115	67	4.10	5	0	3.57	0	10	213	86	3.30	4	0	3.56	0
CHL6	21	673	206	8.10	9	0	7.46	0	21	9,399	1,108	5.01	6	0	8.96	0
CHL7	21	813	226	7.84	9	27	7.91	0	21	10,073	1,051	5.27	6	0	9.55	0
CU1	21	271	163	14.48	15	0	7.53	0	21	1,94	351	11.25	12	0	8.59	0
CU2	29	432	202	19.51	20	0	10.39	0	29	4,591	684	14.05	15	0	10.76	0
CW1	21	293	156	12.19	13	0	7.41	0	21	2,315	423	9.16	10	0	7.78	0
CW2	28	426	201	16.38	17	0	9.88	0	28	3,076	563	11.80	12	282	14.83	0
CW3	35	586	248	21.77	22	0	12.76	0	35	5,349	820	15.35	16	0	13.20	0
HH	6	91	54	1.65	2	0	2.12	0	6	179	75	1.33	2	0	2.10	0
Hchl2	22	833	229	8.29	9	0	8.12	0	22	10,361	1,073	5.35	6	0	10.07	0
Hchl3s	9	262	112	3.20	4	0	3.16	0	9	1,199	304	2.78	3	0	3.63	0
Hchl4s	9	211	93	2.49	3	0	3.28	0	9	1,06	274	1.83	2	26	4.56	0
Hchl6s	19	533	187	6.60	7	0	7.00	0	19	4,726	709	4.38	5	0	9.61	0
Hchl7s	33	1,303	324	10.34	11	0	12.86	0	33	24,754	2,32	6.52	7	48	57.78	0
Hchl8s	10	125	72	2.90	3	0	3.54	0	10	560	167	1.39	2	0	3.68	0
Hchl9	27	696	225	13.30	14	0	9.78	0	27	8,08	904	9.62	10	32	16.92	0
OF1	11	95	72	5.00	5	0	3.98	0	11	369	129	3.18	4	0	3.90	0
OF2	9	97	64	4.92	6	0	3.17	0	9	265	94	3.87	5	0	3.47	0
STS2	25	659	204	16.88	17	0	8.86	0	25	4,643	640	11.65	12	0	10.33	0
STS4	13	278	122	5.69	6	0	4.72	0	13	2,784	485	4.62	5	0	5.54	0
W	18	132	117	-	31	0	6.36	0	18	1,886	405	23.39	24	0	6.79	0
ATP30	26	2,067	347	10.71	12	0	9.90	0	26	77,038	6,905	8.16	9	0	32.98	0
ATP31	46	13,249	1,017	18.00	19	0	41.73	0	46	220,646	18,21	13.33	15 (1)	14	1,830.19	0
ATP32	38	2,775	439	15.28	16	0	14.81	0	38	120,484	6,613	12.01	13	0	44.99	0
ATP33	43	9,874	886	17.69	18	0	17.83	0	43	63,325	4,961	11.91	13 (1)	1,185	1,814.70	0
ATP34	24	3,697	533	8.03	9	0	9.58	0	24	38,113	5,198	5.19	6	0	41.54	0
ATP35	28	5,794	657	9.58	10	0	16.67	0	28	38,044	5,243	7.24	8	0	60.36	0
ATP36	25	1,879	359	10.03	11	0	9.06	0	25	40,762	4,462	7.25	8	33	53.96	0
ATP37	41	9,491	865	15.13	16	0	25.62	0	41	103,125	8,54	10.97	12	35	306.53	0
ATP38	32	4,408	510	13.57	15	0	12.06	0	32	91,52	8,598	10.08	11	0	35.79	0
ATP39	31	2,992	458	15.09	16	0	11.36	0	31	39,041	4,645	10.92	12 (1)	3,555	1,808.77	0
ATP40	35	2,639	461	18.30	20	0	12.33	0	35	172,128	11,797	14.75	16 (1)	334	1,814.85	0
ATP41	31	2,804	441	15.23	16	0	10.88	0	31	61,843	7,126	11.22	12	0	13.82	0
ATP42	44	4,871	627	19.41	21	0	18.48	0	44	101,544	5,873	14.51	16 (1)	768	1,813.40	0
ATP43	47	11,478	1,013	17.47	18	0	35.32	0	47	146,607	9,882	11.97	14 (2)	59	1,815.30	0
ATP44	39	6,412	648	13.07	14	0	18.01	0	39	60,375	4,922	8.34	9	88	246.48	0
ATP45	31	4,38	575	10.13	11	0	13.86	0	31	31,162	3,099	7.37	8	297	123.17	0
ATP46	40	5,924	623	15.89	16	0	21.73	0	40	100,654	8,199	10.76	12 (1)	595	1,812.07	0
ATP47	41	5,191	623	17.38	18	0	18.18	0	41	89,286	7,053	12.07	13	27	160.67	0
ATP48	26	2,049	395	10.28	11	0	10.89	0	26	68,301	7,234	7.68	9 (1)	718	1,808.68	0
ATP49	26	3,405	470	7.45	8	0	10.39	0	26	40,398	5,154	4.85	6 (1)	2,013	1,807.04	-1

#g - number of graphs; #vars - number of variables; #cons - number of constraints; z^{lp} - lower bound; z^{ip} - best solution found (values in parentheses are absolute gaps if the solution is non-optimal); n^{ip} - number of nodes explored in the branch-and-cut procedure; t^{tot} - total run time in seconds (including model generation); imp. - solution improvement in relation to Silva et al. (2010).

Table 6.3: Results for exact and non-exact three-stage problems.

	3E								3NE							
Inst.	#g	#vars	#cons	z^{lp}	z^{ip}	n^{ip}	t^{tot}	imp.	#g	#vars	#cons	z^{lp}	z^{ip}	n^{ip}	t^{tot}	imp.
2	17	464	227	1.69	2	0	6.19	0	17	733	285	1.63	2	0	6.24	0
3	37	1,096	621	22.50	23	0	13.53	0	37	1,838	839	22.50	23	0	13.61	0
A1	37	1,466	694	22.50	23	0	13.47	0	37	2,208	912	22.50	23	0	13.59	0
A2	36	1,921	737	11.50	12	0	12.95	0	36	2,429	877	11.50	12	0	13.26	0
A3	33	1,78	687	7.07	8	0	12.06	0	33	2,169	783	7.05	8	0	12.24	0
A4	37	4,221	1,073	4.17	5	0	14.08	0	37	4,921	1,289	4.11	5	0	15.28	0
A5	30	2,619	813	3.87	4	396	13.37	0	30	3,228	949	3.83	4	576	24.60	0
CHL1	46	10,485	1,922	5.09	6	27	24.01	0	46	11,943	2,209	5.07	6	135	45.61	0
CHL2	20	525	252	2.29	3	0	7.52	0	20	641	286	2.27	3	0	7.43	0
CHL5	17	308	195	2.70	3	0	5.99	0	17	488	229	2.70	3	0	6.20	0
CHL6	44	9,061	1,769	4.91	6 (1)	24,27	1,816.06	0	44	10,319	2,031	4.91	5	83	30.27	0
CHL7	41	8,296	1,656	5.19	6	0	14.29	0	41	9,16	1,871	5.18	6	0	21.90	0
CU1	38	1,784	744	11.25	12	81	13.34	0	38	2,256	885	11.25	12	81	14.92	0
CU2	57	4,966	1,694	13.87	14	64	19.99	0	57	6,492	2,041	13.87	14	0	23.91	0
CW1	42	2,635	943	8.93	10	13	15.25	0	42	3,53	1,156	8.90	9	2,228	35.06	1
CW2	53	3,788	1,503	11.69	12	0	18.02	0	53	5,193	1,875	11.66	12	0	21.33	0
CW3	69	6,078	2,151	15.11	16	0	22.16	0	69	8,311	2,675	15.02	16	81	36.16	0
HH	11	248	127	1.27	2	0	3.51	0	11	308	146	1.23	2	0	3.90	0
Hchl2	42	8,3	1,693	5.28	6	0	15.37	0	42	9,458	1,929	5.26	6	81	27.75	0
Hchl3s	19	1,337	435	2.73	3	0	5.91	0	19	1,659	499	2.73	3	0	6.90	0
Hchl4s	19	1,221	410	1.79	2	0	6.22	0	19	1,544	477	1.78	2	0	6.81	0
Hchl6s	36	4,395	1,107	4.29	5	0	11.26	0	36	4,81	1,224	4.29	5	0	13.85	0
Hchl7s	62	20,129	3,416	6.45	7	1	114.27	0	62	24,806	4,006	6.42	7	220	153.78	0
Hchl8s	17	610	276	1.16	2	0	5.01	0	17	790	310	1.07	2	0	5.59	0
Hchl9	49	7,153	1,703	9.57	10	776	37.30	0	49	9,084	2,044	9.54	10	103	30.82	0
OF1	19	494	255	3.01	4	0	5.40	0	19	715	308	2.97	4	0	6.01	0
OF2	18	376	203	3.78	4	0	5.04	0	18	491	227	3.71	4	0	5.53	0
STS2	47	4,867	1,359	11.57	12	61	16.25	0	47	6,134	1,665	11.50	12	0	18.38	0
STS4	30	2,865	765	4.62	5	0	10.23	0	30	3,331	857	4.62	5	232	10.77	0
W	37	2,119	763	22.97	24	0	11.12	0	37	2,61	907	22.81	23	0	10.94	1
ATP30	62	75,408	8,068	7.96	9 (1)	2,286	1,821.58	0	62	82,203	9,018	7.89	9 (1)	784	1,821.98	∞
ATP31	94	220,703	20,495	13.24	15 (1)	33	1,829.89	∞	94	240,702	23,683	13.21	15 (1)	1	1,838.04	0
ATP32	80	100,464	8,558	11.89	13 (1)	644	1,823.78	1	80	112,206	9,812	11.84	13 (1)	77	1,823.17	1
ATP33	78	56,547	6,71	11.77	13 (1)	1,408	1,819.95	0	78	66,401	8,898	11.70	13 (1)	3,379	1,819.97	0
ATP34	49	35,619	5,921	5.14	6	0	24.00	0	49	41,588	6,988	5.12	6	0	24.01	0
ATP35	55	38,289	6,113	7.14	8	77	119.78	0	55	43,778	7,228	7.10	8	77	142.05	0
ATP36	50	38,408	5,207	7.14	8	0	27.12	0	50	39,9	5,599	7.14	8	0	48.10	0
ATP37	77	92,676	10,207	10.75	12 (1)	1,007	1,821.12	3	77	101,123	11,724	10.73	12 (1)	489	1,820.06	0
ATP38	70	92,687	10,022	9.88	11 (1)	568	1,819.60	1	70	105,071	11,971	9.82	11 (1)	95	1,818.58	0
ATP39	60	35,901	5,616	10.86	11	513	225.26	1	60	38,536	6,23	10.85	12 (1)	1,431	1,814.30	0
ATP40	84	168,271	13,774	14.64	16 (1)	197	1,825.03	∞	84	179,989	15,188	14.59	16 (1)	114	1,825.09	∞
ATP41	64	61,97	8,37	11.13	12	0	22.24	0	64	65,276	9,071	11.11	12	0	105.31	0
ATP42	82	79,515	8,014	14.40	16 (1)	1,69	1,822.50	1	82	92,959	10,003	14.37	16 (1)	2,052	1,822.28	0
ATP43	91	145,527	12,275	11.88	13 (1)	682	1,825.56	2	91	163,883	16,051	11.85	14 (2)	123	1,826.19	0
ATP44	72	56,109	6,412	8.26	9	0	68.99	0	72	62,645	7,618	8.22	9	1,863	778.14	1
ATP45	55	26,833	3,983	7.30	8	891	135.53	0	55	29,491	4,516	7.29	8	864	185.32	0
ATP46	77	100,395	9,89	10.71	12 (1)	1,753	1,819.34	0	77	109,308	11,475	10.67	12 (1)	2,885	1,819.70	0
ATP47	79	85,246	8,857	11.95	13 (1)	1,436	1,819.44	1	79	91,405	10,317	11.90	13 (1)	1,484	1,819.11	0
ATP48	59	68,723	8,19	7.56	9 (1)	6,932	1,814.87	0	59	72,307	8,998	7.53	8	405	525.43	1
ATP49	51	41,418	5,886	4.79	5	77	79.30	1	51	43,148	6,3	4.77	5	4,536	1,013.97	1

#g - number of graphs; #vars - number of variables; #cons - number of constraints; z^{lp} - lower bound; z^{ip} - best solution found (values in parentheses are absolute gaps if the solution is non-optimal); n^{ip} - number of nodes explored in the branch-and-cut procedure; t^{tot} - total run time in seconds (including model generation); imp. - solution improvement in relation to Silva et al. (2010) (∞ means that no feasible solution had been found in Silva et al. 2010 within the time limit).

Table 6.4: Results for exact and non-exact two-stage problems with rotations.

	2ER							2NER						
Instance	#g	#vars	#cons	z^{lp}	z^{ip}	n^{ip}	t^{tot}	#g	#vars	#cons	z^{lp}	z^{ip}	n^{ip}	t^{tot}
2	15	490	141	1.95	2	0	3.36	15	1,132	261	1.66	2	0	3.41
3	31	656	196	18.81	19	0	6.62	31	1,823	381	18.17	19	0	6.66
A1	31	521	202	18.06	19	0	6.97	31	2,982	556	16.56	17	0	7.13
A2	27	367	181	11.73	12	0	6.00	27	2,628	500	10.66	11	0	6.15
A3	26	569	196	7.45	8	0	5.77	26	3,396	575	6.66	7	27	6.58
A4	31	696	221	4.87	6	0	6.87	31	9,461	1,296	3.90	4	1,262	59.60
A5	24	649	216	4.29	5	0	5.29	24	9,388	1,264	3.69	4	0	9.24
CHL1	37	1,251	311	6.14	7	0	8.53	37	25,713	2,265	4.97	6	15	27.38
CHL2	15	262	113	2.66	3	0	3.30	15	840	219	2.18	3	0	3.47
CHL5	15	182	98	3.00	4	0	3.31	15	602	183	2.55	3	0	3.38
CHL6	36	1,741	332	5.33	6	0	7.91	36	24,158	2,161	4.84	5	27	30.40
CHL7	31	1,571	328	5.63	6	152	7.80	31	20,692	1,777	5.11	6	0	9.45
CU1	34	862	261	11.57	13	0	7.51	34	3,857	623	10.71	11	0	8.79
CU2	48	1,279	338	15.84	17	0	10.94	48	8,684	1,163	13.48	14	0	11.41
CW1	37	654	263	10.23	11	27	8.40	37	7,941	1,153	8.72	9	0	9.29
CW2	44	1,001	302	13.06	14	0	10.24	44	7,115	1,032	11.31	12	0	11.08
CW3	63	1,134	417	17.74	19	0	14.09	63	28,494	2,94	14.85	15	0	21.78
HH	11	231	99	1.64	2	0	2.48	11	572	189	1.23	2	0	2.77
Hchl2	32	1,578	327	5.74	6	80	7.96	32	21,093	1,792	5.20	6	0	9.64
Hchl3s	17	635	203	2.77	3	0	3.79	17	3,955	797	2.65	3	0	4.55
Hchl4s	17	567	175	2.11	3	0	3.73	17	3,655	746	1.72	2	229	6.18
Hchl6s	31	1,533	323	4.83	6	0	7.31	31	12,613	1,588	4.22	5	0	8.21
Hchl7s	48	3,105	474	7.24	8	0	11.75	48	54,962	4,292	6.37	7	0	92.62
Hchl8s	15	202	107	1.60	2	0	3.31	15	1,411	360	1.11	2	0	3.63
Hchl9	32	1,042	279	9.86	11	0	7.57	32	12,227	1,144	9.35	10	0	8.66
OF1	16	193	112	3.89	4	0	3.58	16	1,146	304	2.97	3	0	3.96
OF2	15	175	104	4.05	5	0	3.29	15	953	247	3.50	4	0	3.62
STS2	32	1,113	256	12.50	13	27	7.62	32	6,476	775	11.33	12	0	7.92
STS4	25	801	220	4.88	6	4	5.97	25	7,12	1,042	4.45	5	0	6.35
W	29	279	191	21.69	23	0	6.47	29	5,495	872	18.03	19	0	7.32
ATP30	35	2,263	424	10.55	11	0	8.22	35	158,037	12,748	8.15	9	0	34.11
ATP31	90	35,364	1,444	14.99	17	0	57.20	90	645,925	40,271	13.16	18 (4)	0	2,259.36
ATP32	69	3,842	799	13.38	15	12	18.58	69	402,063	15,947	11.79	13 (1)	0	1,832.56
ATP33	75	24,833	1,239	15.12	17	0	39.21	75	81,278	5,578	11.89	13 (1)	510	1,816.83
ATP34	48	5,979	732	6.78	8	0	13.24	48	213,904	19,915	5.10	6	0	747.25
ATP35	52	9,438	876	8.75	9	81	56.70	52	206,287	18,264	7.02	8	0	794.90
ATP36	47	3,242	580	8.24	10	0	11.28	47	159,535	12,68	7.12	8	0	762.91
ATP37	72	30,235	1,238	11.99	13	0	51.66	72	177,75	12,725	10.73	12 (1)	59	1,825.87
ATP38	66	6,35	803	11.90	13	0	17.26	66	484,848	30,747	9.80	14 (4)	0	1,845.84
ATP39	56	8,257	733	12.13	13	0	16.98	56	130,824	12,877	10.78	11	0	351.13
ATP40	57	3,299	680	17.65	19	0	13.67	57	444,142	25,159	14.71	17 (2)	0	1,836.43
ATP41	64	4,497	728	13.85	15	0	15.35	64	348,746	26,128	11.02	12	0	1,760.97
ATP42	67	11,001	844	15.37	16	60	32.57	67	142,3	8,062	14.41	15	85	1,425.56
ATP43	89	42,328	1,408	13.31	14	0	89.97	89	227,519	14,559	11.90	13 (1)	13	1,833.41
ATP44	68	18,188	908	9.56	10	0	51.24	68	95,941	7,169	8.26	9	22	449.05
ATP45	53	13,577	847	8.64	10	0	14.70	53	42,135	3,572	7.34	8	35	163.51
ATP46	66	14,47	877	12.00	13	0	31.92	66	205,943	14,83	10.65	12 (1)	11	1,824.17
ATP47	72	16,636	917	13.64	15	0	36.91	72	160,912	11,375	11.86	13 (1)	83	1,824.37
ATP48	52	3,053	600	10.25	11	0	15.26	52	300,328	23,623	7.61	9 (1)	2	1,829.35
ATP49	50	6,618	688	6.53	7	0	15.11	50	166,645	17,167	4.74	5	64	1,593.67

#g - number of graphs; #vars - number of variables; #cons - number of constraints; z^{lp} - lower bound; z^{ip} - best solution found (values in parentheses are absolute gaps if the solution is non-optimal); n^{ip} - number of nodes explored in the branch-and-cut procedure; t^{tot} - total run time in seconds (including model generation).

Table 6.5: Results for exact and non-exact three-stage problems with rotations.

	3ER							3NER						
Instance	#g	#vars	#cons	z^{lp}	z^{ip}	n^{ip}	t^{tot}	#g	#vars	#cons	z^{lp}	z^{ip}	n^{ip}	t^{tot}
2	29	1,751	633	1.58	2	0	6.58	29	2,453	779	1.58	2	0	6.72
3	59	3,469	1,597	18.12	19	0	13.39	59	5,812	2,1	18.12	19	0	13.70
A1	61	5,167	1,853	16.44	17	0	13.67	61	7,844	2,41	16.24	17	0	14.03
A2	53	4,677	1,637	10.66	11	0	12.07	53	6,232	2,021	10.66	11	0	12.38
A3	51	5,275	1,52	6.62	7	0	12.60	51	6,672	1,831	6.62	7	0	13.80
A4	61	15,396	2,68	3.86	4	297	39.41	61	17,966	3,24	3.85	4	81	48.07
A5	47	12,43	2,317	3.65	4	167	27.89	47	15,475	2,838	3.65	4	27	29.72
CHL1	73	35,857	4,498	4.95	6 (1)	3,855	1,817.38	73	42,618	5,403	4.94	6 (1)	1,68	1,817.65
CHL2	29	1,611	558	2.17	3	0	8.05	29	1,847	628	2.16	3	0	7.48
CHL5	29	1,045	498	2.52	3	0	7.79	29	1,728	631	2.50	3	0	7.69
CHL6	71	32,611	4,279	4.80	5	561	442.86	71	38,548	5,169	4.79	5	378	233.44
CHL7	61	22,409	3,427	5.09	6	0	28.84	61	25,627	3,923	5.08	6	0	20.58
CU1	67	6,656	2,299	10.64	11	27	20.89	67	10,43	2,937	10.61	11	108	31.76
CU2	95	14,11	4,108	13.31	14	0	30.25	95	20,381	5,251	13.26	14	0	29.72
CW1	73	13,65	3,006	8.67	9	187	40.11	73	17,117	3,714	8.64	9	72	47.83
CW2	87	12,954	3,932	11.25	12	0	24.16	87	17,699	4,947	11.20	12	0	25.78
CW3	125	51,14	8,197	14.70	15	0	90.70	125	62,816	10,183	14.65	15	270	409.32
HH	21	1,262	432	1.16	2	0	5.64	21	1,427	486	1.16	2	0	4.44
Hchl2	63	23,816	3,584	5.17	6	0	44.52	63	27,48	4,137	5.16	6	54	75.10
Hchl3s	33	6,9	1,397	2.61	3	0	10.88	33	7,87	1,594	2.61	3	54	14.79
Hchl4s	33	6,724	1,369	1.70	2	0	10.75	33	7,669	1,564	1.69	2	263	15.93
Hchl6s	61	21,844	3,448	4.20	5	0	19.18	61	23,381	3,792	4.20	5	0	19.24
Hchl7s	95	71,479	8,141	6.32	7	30	234.36	95	83,258	9,372	6.31	7	0	54.39
Hchl8s	29	2,585	750	1.01	2	0	7.24	29	3,268	883	1.00	2	0	7.42
Hchl9	63	13,217	2,75	9.27	10	0	17.10	63	19,185	3,414	9.25	10	0	16.14
OF1	32	2,046	684	2.92	3	0	7.36	32	2,738	839	2.90	3	71	8.08
OF2	30	1,665	594	3.33	4	0	6.45	30	2,11	690	3.27	4	0	6.69
STS2	63	8,831	2,376	11.29	12	0	15.75	63	13,41	3,027	11.20	12	0	15.75
STS4	49	9,875	2,036	4.42	5	0	13.67	49	11,36	2,354	4.42	5	0	15.73
W	59	9,217	2,065	17.98	19	0	12.99	59	11,27	2,549	17.97	18	4	14.29
ATP30	96	184,39	15,422	7.92	9 (1)	2	1,823.94	96	228,274	18,682	7.85	9 (1)	2	1,827.58
ATP31	179	1,068,104	53,477	13.13	16 (2)	0	1,917.35	179	1,183,407	67,1	13.12	15 (1)	0	1,891.07
ATP32	137	446,679	22,726	11.70	14 (2)	0	1,854.01	137	538,675	28,025	11.70	13 (1)	1	1,856.48
ATP33	128	99,232	10,655	11.67	13 (1)	623	1,834.69	128	158,164	18,773	11.63	13 (1)	66	1,835.27
ATP34	95	350,67	25,818	5.07	6	0	362.42	95	392,824	31,396	5.06	6	0	509.16
ATP35	103	363,849	29,683	6.99	9 (2)	0	1,839.47	103	393,264	34,585	6.98	9 (2)	0	1,842.68
ATP36	93	251,373	16,771	7.09	9 (1)	0	1,832.69	93	274,444	19,672	7.09	9 (1)	0	1,836.83
ATP37	143	241,141	20,02	10.67	12 (1)	4	1,843.22	143	348,374	30,873	10.64	12 (1)	0	1,846.54
ATP38	131	714,721	38,24	9.73	14 (4)	0	1,889.45	131	814,831	48,048	9.73	13 (3)	0	1,901.36
ATP39	111	241,249	18,107	10.74	12 (1)	3	1,837.16	111	259,461	21,301	10.73	12 (1)	5	1,835.25
ATP40	138	564,66	30,609	14.55	17 (2)	0	1,876.38	138	646,077	35,802	14.51	17 (2)	0	1,862.68
ATP41	127	596,615	36,981	10.99	14 (3)	0	1,870.43	127	651,877	43,692	10.97	14 (3)	0	1,874.68
ATP42	133	154,527	14,448	14.29	16 (1)	165	1,838.71	133	227,023	19,695	14.28	15	0	246.74
ATP43	177	304,551	24,924	11.82	13 (1)	0	1,857.84	177	527,231	43,439	11.79	14 (2)	0	1,882.98
ATP44	135	132,866	13,347	8.19	10 (1)	116	1,832.51	135	200,612	20,155	8.17	10 (1)	32	1,841.57
ATP45	100	49,531	6,711	7.25	8	0	85.47	100	85,145	10,643	7.23	8	243	774.97
ATP46	131	327,718	22,02	10.60	12 (1)	0	1,840.79	131	380,114	28,47	10.56	12 (1)	0	1,841.81
ATP47	143	253,822	19,254	11.80	13 (1)	25	1,839.96	143	295,934	25,297	11.78	13 (1)	0	1,840.03
ATP48	110	505,968	29,888	7.49	9 (1)	0	1,849.63	110	543,542	34,442	7.47	9 (1)	0	1,858.30
ATP49	99	318,604	22,945	4.71	6 (1)	0	1,836.32	99	345,992	27,118	4.70	6 (1)	0	1,836.93

#g - number of graphs; #vars - number of variables; #cons - number of constraints; z^{lp} - lower bound; z^{ip} - best solution found (values in parentheses are absolute gaps if the solution is non-optimal); n^{ip} - number of nodes explored in the branch-and-cut procedure; t^{tot} - total run time in seconds (including model generation).

6.2 Generalized vector packing problem

Baldi (2013) introduced the generalized bin packing problem. In this problem, given a set of items characterized by volume and profit, and a set of bins with given volumes and costs, one aims to select the subset of profitable items and appropriate bins to optimize the objective function which combines the cost of the used bins and the profit derived by the selected items. This problem generalizes the variable size and cost bin packing, knapsack, and many other problems. Based on the same principle, we generalize the vector packing problem.

Let J be the set of item types, and $I = \{1, \ldots, m\}$ the set of item incarnations. For each item type j, let I_j be the set of incarnations i of items of type j. For each bin type $t \in \{1, \ldots, q\}$, let W_t be its capacity vector, and C_t its cost. Let L_t be the minimum number of bins of type t that have to be used, and U_t the total number of bins of type t available. Let U be the maximum number of bins that can be used. For each item type j, let R_j be the set of required demands, and O_j be the set of optional demands. For each item of type j, let d_{jl} for $l \in \mathrm{R}_j$ be the compulsory demand with profit p_{jl} per unit, and d_{jl} for $l \in \mathrm{O}_j$ be the maximum optional demand with profit p_{jl} per unit. For each incarnation $i \in I_j$ of items of type j, the maximum number of occurrences is $b_i = \sum_{l \in \mathrm{R}_j} d_{jl} + \sum_{l \in \mathrm{O}_j} d_{jl}$. We model the generalized vector packing problem as follows:

$$\min \quad \sum_{t=1}^{q} C_t f^0_{\mathrm{T}_t\mathrm{S}} - \sum_{j \in J} \sum_{l \in \mathrm{R}_j} p_{jl} - \sum_{j \in J} \sum_{l \in \mathrm{O}_j} p_{jl}\delta_{jl} \tag{6.2.1}$$

$$\text{s.t.} \quad \sum_{(u,v,i) \in A : v = k} f^i_{uv} - \sum_{(u,v,i) \in A : u = k} f^i_{uv} = 0, \qquad k \in V, \tag{6.2.2}$$

$$\sum_{(u,v,i) \in A : i \in I_j} f^i_{uv} = \sum_{l \in \mathrm{R}_j} d_{jl} + \sum_{l \in \mathrm{O}_j} \delta_{jl}, \qquad j \in J, \tag{6.2.3}$$

$$L_t \le f^0_{\mathrm{T}_t\mathrm{S}} \le U_t, \qquad t \in \{1, \ldots, q\}, \tag{6.2.4}$$

$$\sum_{t=1}^{q} f^0_{\mathrm{T}_t\mathrm{S}} \le U, \tag{6.2.5}$$

$$0 \le \delta_{jl} \le d_{jl}, \text{ integer}, \qquad j \in J, l \in \mathrm{O}_j \tag{6.2.6}$$

$$f^i_{uv} \ge 0, \text{ integer}, \qquad (u,v,i) \in A, \tag{6.2.7}$$

where δ_{jl} is the quantity of optional demand $l \in \mathrm{O}_j$ that is satisfied.

This version of the vector packing problem generalizes many problems. For instance, if all demand is required, this problem is essentially the multiple-choice vector packing problem; but if $U = 1$, $q = 1$, and all demand is optional, we have a knapsack problem. Moreover, this model can also be easily adapted to multi-stage, and it is also possible

to combine several arc-flow models into a multi-period variant. One could easily model multi-period multi-stage generalized vector packing problems due to the flexibility of our method.

Baldi (2013) introduced a dataset for the generalized bin packing problem divided in 4 classes of instances. Each class is characterized as follows:

- Class 0: 300 variable-sized bin packing instances by Monaci (2002). This dataset is characterized as follows:
  - Number of items: 25, 50, 100, 200, or 500;
  - Item sizes: $[1, 100]$, $[20, 100]$, or $[50, 100]$;
  - Item profits: not defined, since all items are compulsory;
  - Number of bin types:
    - Three bin types with capacities 100, 120, and 150, respectively, and costs equal to the capacity.
    - Five bin types with capacities 60, 80, 100, 120, and 150, respectively, and costs equal to the capacity.
- Class 1: same instances of Class 0, but with all items non-compulsory and item profits generated according to the $p_i \in \lceil \mathcal{U}(0.5, 3)w_i \rceil$ uniform distribution, where w_i is the item size.
- Class 2: same instances of Class 0, but with all items non-compulsory and item profits generated according to the $p_i \in \lceil \mathcal{U}(0.5, 4)w_i \rceil$ uniform distribution, where w_i is the item size.
- Class 3: 5 sets of instances, with 0%, 25%, 50%, 75%, and 100% of compulsory items, generated from 12 large instances (500 items) selected from Class 1 and Class 2. Class 3 is composed by 60 instances (twelve for each percentage of compulsory items).

Baldi (2013) solved 721 out of 960 instances under a time limit of 1 hour using a problem-specific branch-and-price algorithm. Using the arc-flow formulation with graph compression, we solved 952 out of 960 instances under a time limit of 5 minutes. In the 8 instances that were not solved under the time limit, the largest relative and absolute gaps at the moment of termination were 0.01% and 2, respectively. Note that our method takes full advantage of the heuristics and cutting plane generators that come with state-of-the-art MIP solvers. This is particularly important when arc-flow models are used in the middle of more complex models that may benefit substantially from the cuts generated by MIP solvers. Table 6.6 presents the results for classes 0, 1, and 2. Table 6.7 presents the results for class 3.

Table 6.6: Generalized vector packing results for classes 0, 1, and 2.

Class	#types	#items	#vars	#cons	n^{ip}	t^{ip}	OPT
Class 0	3	25	257.37	61.67	4.67	0.05	30/30
		50	728.00	104.47	19.37	0.29	30/30
		100	1,550.57	143.23	41.07	0.93	30/30
		200	2,425.17	166.57	109.73	2.09	30/30
		500	2,916.40	176.80	189.03	4.32	30/30
	5	25	310.33	71.13	0.00	0.03	30/30
		50	792.73	109.87	4.73	0.13	30/30
		100	1,548.27	143.80	5.10	0.24	30/30
		200	2,481.07	169.00	84.30	2.08	30/30
		500	2,971.17	179.20	95.03	2.31	30/30
Class 1	3	25	282.37	61.67	6.53	0.05	30/30
		50	778.00	104.47	201.63	0.72	30/30
		100	1,650.57	143.23	241.50	3.87	30/30
		200	2,625.17	166.57	358.30	8.12	30/30
		500	3,416.40	176.80	1,755.60	20.33	29/30
	5	25	335.33	71.13	0.00	0.04	30/30
		50	842.73	109.87	352.87	1.49	30/30
		100	1,648.27	143.80	431.43	3.06	30/30
		200	2,681.07	169.00	170.43	4.08	30/30
		500	3,471.17	179.20	1,220.87	20.26	29/30
Class 2	3	25	282.37	61.67	15.07	0.06	30/30
		50	778.00	104.47	87.70	0.75	30/30
		100	1,650.57	143.23	1,184.00	10.30	30/30
		200	2,625.17	166.57	902.13	12.09	30/30
		500	3,416.40	176.80	4,704.57	42.93	28/30
	5	25	335.33	71.13	17.03	0.05	30/30
		50	842.73	109.87	38.53	0.61	30/30
		100	1,648.27	143.80	618.43	4.93	30/30
		200	2,681.07	169.00	1,210.10	15.48	29/30
		500	3,471.17	179.20	704.53	9.02	30/30

#types - number of bin types; #items - number of items; #vars - average number of variables; #cons - average number of constraints; n^{ip} - average number of nodes explored in the branch-and-cut procedure; t^{ip} - average run time in seconds; OPT - fraction of instances solved to optimality.

Table 6.7: Generalized vector packing results for class 3.

%Compulsory	#vars	#cons	n^{ip}	t^{ip}	OPT
0%	3,897.67	190.67	2,304.67	33.43	11/12
25%	3,772.67	190.67	275.58	5.03	12/12
50%	3,647.67	190.67	2,565.00	39.84	11/12
75%	3,522.67	190.67	2,972.25	44.56	11/12
100%	3,397.67	190.67	149.58	4.40	12/12

%Compulsory - percentage of compulsory items; #vars - average number of variables; #cons - average number of constraints; n^{ip} - average number of nodes explored in the branch-and-cut procedure; t^{ip} - average run time in seconds; OPT - fraction of instances solved to optimality.

Chapter 7

Conclusions

The general arc-flow formulation with graph compression proposed in this thesis proved to be a very powerful tool for solving several cutting and packing problems. The model is equivalent to Gilmore and Gomory's, thus providing a very strong linear relaxation. Nevertheless, it replaces column-generation by the generation of a graph able to represent one permutation for each valid packing pattern. These are implicitly enumerated through the construction of a compressed graph, which is proven to hold all the paths from the source to the target that are required for determining the optimum solution of the original problem.

Our method can be used for solving several problems through reductions to multiple-choice vector packing; examples include bin packing, cutting stock, and bin packing with conflicts. Without any problem-specific adjustment, we solved most of the known benchmark instances of these problems on a standard desktop computer, spending less than one minute per instance, on average. The linear relaxations are very strong for every problem we solved through reductions to vector packing: the largest absolute gap we found in all the instances solved was 1.0027.

The proposed graph compression algorithm is simple and proved to be very effective on a large variety of problems. The major limitation of our method is the combination of large capacities and long patterns, which can be difficult to represent in a compact way. Nevertheless, when the graphs generated are reasonably small, the proposed method usually outperforms more complex approaches such as branch-and-price algorithms. Moreover, the instances with practical interest for exact methods usually do not combine small items with large capacities, since heuristics are usually enough to find very good solutions with little effort for this type of instances.

In an independent study, available in the literature (Delorme et al., 2016), the arc-flow formulation with graph compression was compared against several other models and problem-specific algorithms on one-dimensional bin packing and cutting stock problems. The arc-flow formulation outperformed all other models and the only method with comparable performance was a problem-specific branch-and-cut-and-price algorithm. Note that branch-and-cut-and-price algorithms are usually much more complex than standard

branch-and-price algorithms.

The set of applications is not limited to reductions to multiple-choice vector packing problems. The proposed method provides a simple way to represent every feasible pattern for any cutting or packing instance in an integer programming model. Therefore, we can use multiple arc-flow models in the same model in order to model, for instance, multi-stage variants or even multi-period variants.

Due to the flexibility of the proposed method, we introduced two new problem variants: the multi-stage vector packing problem and the generalized vector packing problem. Both variants were easily tackled by the arc-flow formulation with graph compression. Note that our method takes full advantage of the heuristics and cutting plane generators that come with state-of-the-art MIP solvers. This is particularly important when arc-flow models are used as part of more complex models that may benefit substantially from the cuts generated by MIP solvers.

One of the outcomes of this thesis is an open-source project called `VPSolver`, which is currently being used not only by other researchers but also in the industry. The arc-flow formulation with graph compression is currently being used by a large multinational company in the labeling & packaging sector in over 60 sites worldwide to solve daily hundreds of cutting stock instances with several industry-specific constraints.

For many years pseudo-polynomial models such arc-flow formulations were not seen as realistic solution methods due to the large number of constraints and variables, and hence they were rarely used directly as MIP models. Nevertheless, over the years the computational power has increased substantially and the MIP solvers have been evolving too. This work shows that today this approach is not only viable, but also very effective on a large variety of applications. Moreover, as the computational power of MIP solvers improves, the performance of this method is expected to keep improving.

7.1 Future work

For the sake of brevity, we kept the algorithm as simple as possible, while still effective enough to tackle most of the instances with practical interest. It is possible to reduce the model sizes even further, and there are several methods to do so. One of these methods consists in identifying subgraphs occurring multiple times in the final arc-flow model. If a sub-graph occurs several times in an arc-flow graph, it can become an independent arc-flow graph, and all its occurrences can be replaced by a single arc.

By simply removing common occurrences of parallel arcs between pairs of nodes, we were

able to remove over 30% of the arcs in some instances, which, even though is residual when compared with the reductions obtained by the graph compression algorithm, may still correspond to the removal of over 600,000 integer variables in models with over 2 million variables.

The graph compression algorithm proposed in this thesis is effective mostly on the reduction of the number of nodes, hence additional techniques to reduce the number of arcs may improve the performance of the arc-flow formulation with graph compression on every application.

References

Abramson, D. (1991). Constructing school timetables using simulated annealing: Sequential and parallel algorithms. *Manage. Sci.*, 37(1):98–113. (Cited on page 77.)

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows - theory, algorithms and applications*. Prentice-Hall. (Cited on page 51.)

Baldi, M. M. (2013). *Generalized Bin Packing Problems*. PhD thesis, Politecnico di Torino. (Cited on pages 15, 94, and 95.)

Baum, S. and Trotter Jr., L. E. (1981). Integer rounding for polymatroid and branching optimization problems. *SIAM Journal on Algebraic Discrete Methods*, 2(4):416–425. (Cited on page 72.)

Belov, G., Kartak, V., Rohling, H., and Scheithauer, G. (2009). One-dimensional relaxations and LP bounds for orthogonal packing. *International Transactions in Operational Research*, 16(6):745–766. (Cited on pages 80 and 81.)

Belov, G. and Scheithauer, G. (2006). A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research*, 171:85–106. (Cited on page 69.)

Brandão, F. (2012). Bin Packing and Related Problems: Pattern-Based Approaches. Master's thesis, Faculdade de Ciências da Universidade do Porto, Portugal. (Cited on pages 15 and 17.)

Brandão, F. and Pedroso, J. P. (2016). Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, 69:56 – 67. (Cited on pages 15 and 69.)

Bron, C. and Kerbosch, J. (1973). Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577. (Cited on page 76.)

Caprara, A. (1998). Properties of some ILP formulations of a class of partitioning problems. *Discrete Appl. Math.*, 87:11–23. (Cited on page 18.)

Caprara, A. and Toth, P. (2001). Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Appl. Math.*, 111:231–262. (Cited on page 70.)

Dantzig, G. B. and Wolfe, P. (1960). Decomposition Principle for Linear Programs. *Operations Research*, 8(1):101–111. (Cited on page 20.)

Delorme, M., Iori, M., and Martello, S. (2016). Bin packing and cutting stock problems:

Mathematical models and exact algorithms. *European Journal of Operational Research*, 255:1–20. (Cited on pages 17, 69, and 97.)

Dyckhoff, H. (1981). A new linear programming approach to the cutting stock problem. *Operations Research*, 29:1092–1104. (Cited on pages 24 and 25.)

Epstein, L. and van Stee, R. (2011). Improved Results for a Memory Allocation Problem. *Theor. Comp. Sys.*, 48(1):79–92. (Cited on page 74.)

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30. (Cited on pages 72 and 79.)

Fernandes-Muritiba, A. E., Iori, M., Malaguti, E., and Toth, P. (2010). Algorithms for the bin packing problem with conflicts. *INFORMS Journal on Computing*, 22(3):401–415. (Cited on pages 79, 80, and 82.)

Garey, M. R., Graham, R. L., and Johnson, D. S. (1976). Resource Constrained Scheduling as Generalized Bin Packing. *J. Comb. Theory, Ser. A*, 21(3):257–298. (Cited on page 14.)

Garey, M. R. and Johnson, D. S. (1978). " Strong " NP-Completeness Results: Motivation, Examples, and Implications. *J. ACM*, 25(3):499–508. (Cited on page 13.)

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA. (Cited on pages 13 and 75.)

Gau, T. (1994). Counterexamples to the IRU property. *SICUP Bulletin 12*. (Cited on page 22.)

Gau, T. and Wäscher, G. (1995). CUTGEN1: A problem generator for the standard one-dimensional cutting stock problem. *European Journal of Operational Research*, 84(3):572 – 579. (Cited on page 73.)

Gilmore, P. and Gomory, R. (1963). A linear programming approach to the cutting stock problem–part II. *Operations Research*, 11:863–888. (Cited on page 21.)

Gilmore, P. C. and Gomory, R. E. (1961). A Linear Programming Approach to the Cutting-Stock Problem. *Operations Research*, 9:849–859. (Cited on page 20.)

Kantorovich, L. V. (1960). Mathematical methods of organising and planning production. *Management Science*, 6(4):366–422. (Cited on page 19.)

Lodi, A., Martello, S., and Vigo, D. (1999). Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11(4):345–357. (Cited on page 73.)

Macedo, R., Alves, C., and Valério de Carvalho, J. M. (2010). Arc-flow model for the two-dimensional guillotine cutting stock problem. *Computers & Operations Research*, 37(6):991 – 1001. (Cited on page 85.)

Martello, S. and Toth, P. (1990). *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA. (Cited on pages 17 and 18.)

Monaci, M. (2002). *Algorithms for packing and scheduling problems*. PhD thesis, Università di Bologna, Bologna, Italy. (Cited on page 95.)

Nica, V. (1994). General counterexample to the integer round-up property. Working paper, Department of Economic Cybernetics, Academy of Economic Studies, Bucharest. (Cited on page 22.)

Patt-Shamir, B. and Rawitz, D. (2012). Vector bin packing with multiple-choice. *Discrete Appl. Math.*, 160(10-11):1591–1600. (Cited on page 14.)

Rietz, J., Scheithauer, G., and Terno, J. (2002a). Families of non-IRUP instances of the one-dimensional cutting stock problem. *Discrete Applied Mathematics*, 121(1-3):229–245. (Cited on page 22.)

Rietz, J., Scheithauer, G., and Terno, J. (2002b). Tighter bounds for the gap and non-IRUP constructions in the one-dimensional cutting stock problem. *Optimization*, 6:927–963. (Cited on page 22.)

Sadykov, R. and Vanderbeck, F. (2013). Bin packing with conflicts: A generic branch-and-price algorithm. *INFORMS Journal on Computing*, 25(2):244–255. (Cited on page 79.)

Scheithauer, G. (1999). LP-based bounds for the container and multi-container loading problem. *International Transactions in Operational Research*, 6(2):199–213. (Cited on page 80.)

Scheithauer, G. and Terno, J. (1995). The Modified Integer Round-Up Property of the One-Dimensional Cutting Stock Problem. (Cited on page 22.)

Scheithauer, G. and Terno, J. (1997). Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. *Operations Research Letters*, 20:93–100. (Cited on page 22.)

Schoenfield, J. (2002). Fast, Exact Solution of Open Bin Packing Problems without Linear Programming. draft, US Army Space & Missile Defence Command, Huntsville, 20 Alabama. (Cited on page 72.)

Scholl, A., Klein, R., and Jürgens, C. (1997). Bison: A fast hybrid procedure for exactly

solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7):627 – 645. (Cited on pages 62 and 72.)

Schwerin, P. and Wäscher, G. (1997). The Bin-Packing Problem: A Problem Generator and Some Numerical Experiments with FFD Packing and MTP. *International Transactions in Operational Research*, 4(5-6):377–389. (Cited on page 72.)

Silva, E., Alvelos, F., and Valério de Carvalho, J. M. (2010). An integer programming model for two- and three-stage two-dimensional cutting stock problems. *European Journal of Operational Research*, 205(3):699 – 708. (Cited on pages 88, 89, 90, and 91.)

Simchi-Levi, D. (1994). New worst-case results for the bin-packing problem. *Naval Research Logistics*, 41(4):579–585. (Cited on page 13.)

Smith, K. A., Abramson, D., and Duke, D. (2003). Hopfield neural networks for timetabling: formulations, methods, and comparative results. *Comput. Ind. Eng.*, 44(2):283–305. (Cited on page 77.)

Valério de Carvalho, J. M. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Ann. Oper. Res.*, 86:629–659. (Cited on pages 23 and 74.)

Valério de Carvalho, J. M. (2002). LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273. (Cited on pages 17, 24, and 51.)

Vance, P. H. (1998). Branch-and-Price Algorithms for the One-Dimensional Cutting Stock Problem. *Comput. Optim. Appl.*, 9:211–228. (Cited on page 20.)

Wäscher, G. and Gau, T. (1996). Heuristics for the integer one-dimensional cutting stock problem: A computational study. *OR Spectrum*, 18:131–144. (Cited on pages 22 and 72.)

Wäscher, G., Haußner, H., and Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130. (Cited on page 14.)

Wolsey, L. A. (1977). Valid inequalities, covering problems and discrete dynamic programs. In P.L. Hammer, E.L. Johnson, B. K. and Nemhauser, G., editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 527 – 538. Elsevier. (Cited on pages 15, 23, 27, 30, 31, 32, 33, 37, and 38.)

List of Figures