Modern Branch-Cut-and-Price

Ruslan Sadykov

Université de Bordeaux

Institut de Mathématiques de Bordeaux

Thèse d'habilitation à diriger des recherches

présentée par

Ruslan Sadykov

SPÉCIALITÉ : MATHEMATIQUES APPLIQUEES ET CALCUL SCIENTIFIQUE

Modern Branch-Cut-and-Price

Date de soutenance : 4 décembre 2019

Devant la commission d'examen composée de :

François Clautiaux ..	Professeur, Université de Bordeaux	Garant
Bernard Gendron ...	Professeur, Université de Montréal	Rapporteur
Ivana Ljubic	Professeur, ESSEC Business School, Paris	Examinateur
Marco Lübbecke	Professeur, RWTH Aachen University ...	Rapporteur
Frédéric Semet	Professeur, Ecole Centrale de Lille	Rapporteur
Daniele Vigo	Professeur, University of Bologna	Examinateur
François Vanderbeck	Directeur général, Atoptima	Invité

– 2019 –

Acknowledgments

First of all I want to thank Leysan and my parents for their love and support. Without them it would be very difficult to be where I am now.

I am very grateful to my advisors and mentors Alexander Lazarev, Laurence Wolsey, Yuri Nesterov, Philippe Baptiste, and François Vanderbeck for guidance and scientific support. I am lucky I have met them in my life.

I would like to thank members of the HDR jury François Clautiaux, Bernard Gendron, Ivana Ljubic, Marco Lübbecke, Frédéric Semet, and Daniele Vigo for accepting this time-consuming charge.

I would like to mention Eduardo Uchoa and Artur Pessoa for making my sabbatical year in Brazil very productive and enjoyable. Despite of difficulties that Brazil experience at the moment, Brazilian people and Brazilian nature made our stay in this country unforgettable.

Finally, I would like to thank my peers and colleagues for their friendship and fruitful scientific discussions: Sasha Kvaratskhelia, Michael Baes, François Warichet, Peter Malkin, Hamish Waterer, Ives Pochet, Konstantine Kholodilin, Victor Turchin, Larissa Matveeva, Alexey Savvateev, Konstantine Artiuchine, Christophe Dürr, Yann Hendel, Pierre Pesneau, Andrew Miller, François Clautiaux, Boris Detienne, Matthieu Gérard, Quentin Viaud, Guillaume Marquez, Teobaldo Bulhões, Anand Subramanian, Christina Boeres, Fábio Protti, Christiana Bentes, Lucia Drummond, Thibaut Vidal, Shunji Tanaka, Yakov Zinder, Walid Klibi, and many others.

Contents

Chapter 1

Introduction

Mixed Integer Programming (MIP) is a leading methodology for solving combinatorial optimization problems. It has numerous applications in various industries. The main MIP solution approach is branch-and-cut which consists in iterative refining of a linear relaxation of the problem (by adding valid inequalities or cutting planes) and then applying the divide-and-conquer principle (branching). Modern MIP solvers implementing the branch-and-cut approach have been progressing at a fast pace for the last 30 years, and now they have found a widespread use around the globe. The branch-and-cut approach however has its limits. It is not efficient (i.e. does not scale well) for problems for which all known linear relaxations are not sufficiently tight, even after adding cutting planes. These are problems with complex combinatorial structure appearing for example in routing, scheduling, packing and other similar applications.

A radical way to improve the quality of linear relaxations is to substantially increase the number of variables. For such formulations however it is not possible to apply directly the standard branch-and-cut approach as variables should then be generated dynamically. An extension of branch-and-cut to this case is called branch-cut-and-price, in which the linear relaxation of the problem is solved by the column generation approach.

Branch-cut-and-price (BCP) is widely recognized to be a much more efficient approach than branch-and-cut for solving many practical problems. However, the former is not nearly as widespread as the latter. The usage of few available BCP solvers is manly limited to the academic world. We believe however that potential of BCP solvers is very high, and we try to contribute to their development as much as we can. This manuscript covers our work in this direction.

Some of our works do not concern column generation and branch-(cut-and-)price. They are briefly mentioned at the end of Section 3.6.

In the remaining of this chapter, we introduce column generation, branch-cut-and-price approaches, as well as some recently proposed techniques to improve their efficiency.

1.1 Column Generation

When presenting a column generation algorithm, in the literature authors usually start with an original compact MIP formulation (i.e. with a polynomial number of variables and constraints) and then apply to it a Dantzig-Wolfe reformulation. Often however there is no good compact MIP formulation for the problem at hand. In this case, many authors present an artificial cumbersome formulation which is needed only to be able to apply the Dantzig-Wolfe reformulation. Therefore, when presenting the column generation approach, we do not formulate the problem as a single original MIP, but directly use the subproblems and the master formulation which are linked by mapping beween their variables.

In short, *column generation* stays for the iterative method which solves linear programs with very large number of variables. Such a program is called *master problem* in the literature. "Very large" here means that it is practically impossible or time consuming to solve the linear program with all variables directly. Instead, at any moment we deal explicitly with only a subset of variables (others being implicitly fixed to zero) defining so-called *restricted master problem* (RMP). In each iteration of column generation, the RMP is solved to optimality by obtaining primal and dual optimal solutions. Knowing the dual solution, we can calculate the reduced costs of variables absent from the RMP and find the "best" variable according to it. The reduced cost of this column either tells us whether the primal solution of RMP is optimal for the master problem or not. In the latter case, the "best" variable is added to the RMP and the process repeats. The term "column generation" comes from the fact that variables correspond to columns in the linear programming matrix. We will use both terms "variable" and "column" interchangeably.

The process of determining the "best" column is rarely done by enumeration, i.e. by inspection of all variables one by one. In most cases, we use some structure of our problem, by which the set of column vectors corresponds to the set of feasible solutions of the *subproblem* or the union of all solutions of several subproblems. Those solutions are either defined explicitly by constraints or implicitly by on oracle, which returns a solution minimizing a linear objective function.

We will now formalize this description. Given set K of subproblems, let subproblem [SP^k], $k \in K$, be defined over set of variables $\{z_t^k : 1 \leqslant t \leqslant n^k\}$. Let $\{\bar{z}^p\}_{p \in P^k}$ be the set of n^k-dimensional points that correspond to feasible solutions of [SP^k]. We suppose that, given $\bar{c}^k \in \mathbb{R}^{n^k}$, we can find $p \in P^k$ minimizing $\bar{c}^k \bar{z}^p$ using the explicit subproblem definition or the oracle.

The problem is formulated as follows. There are variables x_j, $1 \leqslant j \leqslant n_1$, and variables y_s, $1 \leqslant s \leqslant n_2$. Let $\overline{Y}_s \in \mathbb{R}_+$ and $\underline{Y}_s \in \mathbb{R}_+$ denote upper and lower bounds on variable y_s, $1 \leqslant s \leqslant n_2$. The first $\bar{n}_1$ variables x and the first $\bar{n}_2$ variables y are defined to be integer. For each variable x_j, $1 \leqslant j \leqslant n_1$, $M(x_j) \subseteq \{z_t^k : 1 \leqslant t \leqslant n^k, k \in K\}$ defines its *mapping* into a non-empty subset of subproblem variables. Remark that mappings do not need to be disjoint, the same subproblem variable can be mapped to more than one variable x_j. In fact, $M^{-1}(z_t^k) = \{x_j \mid 1 \leqslant j \leqslant n_1, z_t^k \in M(x_j)\}$ denotes the inverse mapping, some M^{-1} sets may be empty. Variables w_k are pre-defined. They denote the subproblem solution cardinality. Let $\overline{W}_k \in \mathbb{Z}_+$ and $\underline{W}_k \in \mathbb{Z}_+$ denote upper and lower bounds on variable w_k, $k \in K$. We suppose that all

subproblems are distinct. In the case some subproblems are identical, we aggregate them by changing the bounds of variables w_k. For each solution $\bar{z}^p$, $p \in P^k$, $k \in K$, let λ_p^k be a non-negative integer variable. Coefficient $\bar{z}_t^p$ indicates the value of z_t^k in $\bar{z}^p$, $p \in P^k$.

We are now ready to give the problem formulation.

$$\text{Min} \quad \sum_{j=1}^{n_1} c_j x_j + \sum_{s=1}^{n_2} g_s y_s + \sum_{k \in K} f^k w^k \tag{1.1a}$$

$$\text{S.t.} \quad \sum_{j=1}^{n_1} \theta_{ij} x_j + \sum_{s=1}^{n_2} \phi_{is} y_s \geqslant d_i, \qquad i = 1, \ldots, m, \tag{1.1b}$$

$$x_j = \sum_{k \in K} \sum_{p \in P^k} \left(\sum_{z_t^k \in M(x_j)} \bar{z}_t^p \right) \lambda_p, \qquad j = 1 \ldots, n_1, \tag{1.1c}$$

$$w_k = \sum_{p \in P^k} \lambda_p, \qquad k \in K, \tag{1.1d}$$

$$\underline{W}_k \leqslant w_k \leqslant \overline{W}_k, \qquad k \in K, \tag{1.1e}$$

$$\underline{Y}_s \leqslant y_s \leqslant \overline{Y}_s, \qquad s = 1, \ldots, \bar{n}_2, \tag{1.1f}$$

$$\lambda_p \in \mathbb{Z}_+, \qquad p \in P^k,\ k \in K, \tag{1.1g}$$

$$x_j \in \mathbb{Z},\ y_s \in \mathbb{Z}, \qquad j = 1, \ldots, \bar{n}_1,\ s = 1, \ldots, \bar{n}_2. \tag{1.1h}$$

Equations (1.1a) and (1.1b) define a general objective function and m general constraints over those variables, respectively. The relation between variables x and λ is given by (1.1c), while the relation between w and λ is given by (1.1d). Depending on the problem's structure, integrality of variables λ may be relaxed without changing the set of optimal solutions. In the case variables x are restricted to be non-negative, the corresponding constraints are added in the form (1.1b).

Eliminating x, w variables and relaxing all integrality constraints, we obtain the master problem:

$$\text{Min} \quad \sum_{k \in K} \sum_{p \in P^k} \left(\sum_{j=1}^{n_1} c_j \sum_{z_t^k \in M(x_j)} \bar{z}_t^p + f^k \right) \lambda_p + \sum_{s=1}^{n_2} g_s y_s \tag{1.2a}$$

$$\text{S.t.} \quad \sum_{k \in K} \sum_{p \in P^k} \left(\sum_{j=1}^{n_1} \theta_{ij} \sum_{z_t^k \in M(x_j)} \bar{z}_t^p \right) \lambda_p + \sum_{s=1}^{n_2} \phi_{is} y_s \geqslant d_i, \qquad i = 1, \ldots, m, \tag{1.2b}$$

$$\underline{W}_k \leqslant \sum_{p \in P^k} \lambda_p \leqslant \overline{W}_k, \qquad k \in K, \tag{1.2c}$$

$$\underline{Y}_s \leqslant y_s \leqslant \overline{Y}_s, \qquad s = 1, \ldots, \bar{n}_2, \tag{1.2d}$$

$$\lambda_p \geqslant 0, \qquad p \in P^k,\ k \in K. \tag{1.2e}$$

Master problem (1.2) is solved by column generation as described above. Let π_i, $1 \leqslant i \leqslant m$, denote the dual variables, which correspond to constraints (1.2b). Let

ν^{k+} and ν^{k-}, $k \in K$, be the dual variables corresponding to constraints (1.2c). The reduced cost of subproblem variable z_t^k, $1 \leqslant t \leqslant n^k$, is then defined as:

$$\bar{c}_t^k = \sum_{x_j \in M^{-1}(z_t^k)} c_j - \sum_{i=1}^{m} \sum_{x_j \in M^{-1}(z_t^k)} \theta_{ij} \pi_i.$$

The reduced cost of a solution $\bar{z}^p$, $p \in P^k$, is

$$\bar{c}(\bar{z}^p) = f^k + \sum_{t=1}^{n^k} \bar{c}_t^k \bar{z}_t^p - \nu^{k+} - \nu^{k-}.$$

So, the pricing subproblems correspond to finding, for each $k \in K$, a solution $\bar{z}^p$, $p \in P^k$, with the minimum reduced cost.

For an original integer program

$$[\mathrm{F}] \equiv \min\{cx : \ \Theta x \geqslant d; \ Bx \geqslant b; \ x \in \mathbb{Z}_+^{n_1}\}, \tag{1.3}$$

following the discretization approach (Desrosiers and Lübbecke, 2011; Vanderbeck, 2000), the Dantzig-Wolfe reformulation (Dantzig and Wolfe, 1960) can be written in the form (1.1) by defining the subproblem explicitly $\{Bz \geqslant b; \ z \in \mathbb{Z}_+^{n_1}\}$ and using the unitary mapping $M(x_t) = \{z_t\}$.

1.2 Branch-Cut-and-Price (BCP)

First of all, this term may be written in different ways: "branch-and-cut-and-price" or "branch-price-and-cut". The former is supposed to be less grammatically correct. The latter would better reflect what the method is about: first we price columns and only then we add cuts. However, in this work we stick to traditional "branch-cut-and-price" which follows historical reasons as branch-and-cut was there before branch-and-price.

Branch-cut-and-price is the method to solve formulation (1.1). The basic BCP can be viewed as a simple combination of the branch-and-cut method to solve a MIP and the column generation approach which solves the linear relaxation (1.2) at every node of the search tree. Indeed, as long as branching constraints and cutting planes involve only variables y, x, and w, there is no special consideration to take into account: branching, cut separation and pricing are independent components of the BCP (Desaulniers et al., 2011).

However, branching on variables y, x, and w may not be sufficient and one may need to branch on variables λ. Usually this should be avoided, as branching on λ produces a highly unbalanced search tree. A way to do it is to introduce additional variables x and z. For example, for the bin packing problem, in addition to variables stating whether an item is packed to the bin one may introduce variables for each pair of items stating whether both of them are packed to the bin or not. Additional variables may however make the subproblem harder to solve. A similar alternative is the Ryan and Foster (1981) branching scheme. A more general branching rule is the one by Vanderbeck (2000, 2011) which implements a partial disaggregation

of identical subproblems. However, here again the subproblem difficulty may be affected, as one needs to impose bounds on variables z.

Absolute majority of works before 2005 considered the basic variant of branch-and-price and branch-cut-and-price, in which branching, cutting and pricing are independent (assuming that the Ryan and Foster branching is equivalent to adding a polynomial number of variables to the subproblems). Few exceptions include (non-exhaustively) the work by Nemhauser and Park (1991) on combination of column generation and separation of cutting planes over variables λ for the edge coloring problem, and the work of Vanderbeck (2000) on generic branching.

1.3 Modern Branch-Cut-and-Price

We do this distinction between basic branch-cut-and-price and the "modern" one to reflect the large changes in the method which were recently developed in the literature. For us, modern branch-cut-and-price is i) non-robust, ii) complex, and iii) generic.

Non-robustness

As mentioned above, "traditional" branch-cut-and-price is a simple combination of branch-and-price and branch-and-cut in which cutting and pricing are completely independent from each other. This means that after adding cutting planes to the master the structure of the pricing problem does not change. Such cuts are called robust (Pessoa et al., 2008) in the literature. In contrast to that, in modern BCP the algorithm for solving the pricing problem may be largely modified if non-robust cuts are present in the master. Moreover, cut separation routines may be affected by the information collected when running the pricing problem solver. For example, when the time of the latter becomes large, we may choose not to add cuts anymore and branch instead.

After the pioneering work (Nemhauser and Park, 1991) on non-robust cuts for edge coloring, in 2000s Chvátal-Gomory rank-1 cuts over variables λ were applied in the column generation context. Belov and Scheithauer (2006) used them for cutting stock problems, whereas Jepsen et al. (2008) employed them for vehicle routing. Unfortunately, rank-1 cuts have a large impact on the difficulty of the pricing problem; one cannot have too many active rank-1 cuts at the same time. However, recently a new variant of these cuts was proposed by Pecin et al. (2017b,c): limited-memory rank-1 cuts. These cuts are weaker that the original (i.e. full memory) cuts, but have a much smaller impact on the dynamic programming labeling algorithm to solve the pricing problem, which is the resource constrained shortest path. Actually, these cuts were designed in a special way to mitigate the impact on the solution difficulty of the pricing problem. Moreover, they are efficient only if the pricing problem is solved by dynamic programming. So here we have a "synergy" between cutting planes and pricing solver to achieve the state-of-the-art results for vehicle routing problems.

Complexness

Modern branch-cut-and-price algorithms are complex methods involving much more components than just column generation, cut separation and branching. These are stabilization, heuristic pricing, management of the pricing problem relaxation, variable fixing by reduced costs, column enumeration, primal heuristics, and strong branching. I will now introduce these components in more detail.

Column generation is an iterative procedure and it is prone to convergence issues such as dual oscillations, the tailing-off effect, and primal degeneracy. Stabilization techniques have been developed to reduce these drawbacks (Lübbecke and Desrosiers, 2005). They can be classified into four categories (Vanderbeck, 2005): imposing bounds on dual prices (Ben Amor et al., 2006), smoothing dual prices based on past information before passing them to the pricing problem (Wentges, 1997), penalizing the deviation of the dual solution from a stability center (du Merle et al., 1999; Briant et al., 2008), and working with interior dual solutions rather than extreme point dual solutions (Rousseau et al., 2007).

The pricing problem should not be necessarily solved to optimality each time. For column generation convergence it suffices to generate a negative reduced cost column in each iteration. Such columns can be found by a (meta)heuristic (Desaulniers et al., 2008) or by changing parameters of an exact solver (Gamrath and Lübbecke, 2010). Additional ways to solve heuristically the pricing problem are discussed by Desaulniers et al. (2002). Usually generating negative reduced cost columns without solving the pricing problem to optimality reduces a lot the column generation convergence time.

As defined in (Vanderbeck and Savelsbergh, 2006), a column is *redundant* when problem (1.1) admits an optimal solution in which the value of the corresponding variable is equal to zero. In many cases non-redundancy check is rather simple. One way to do it is to update upper and lower bounds on subproblem variables z based on problem's constraints. Standard preprocessing techniques for MIPs (Savelsbergh, 1994) or, equivalently, domain propagation (Gamrath and Lübbecke, 2010) can be used for this purpose. Columns satisfying updated bounds are called *proper* (Vanderbeck and Savelsbergh, 2006). Non-proper columns are redundant by definition. These columns can be excluded from the solution space of the pricing problem by imposing bounds on variables z. However, this may make the pricing problem harder to solve.

In the context of vehicle routing, proper columns correspond to elementary routes which pass by each client at most once. Imposing elementarity constraint in the pricing problem makes it much more difficult to solve. Therefore usually the elementarity constraint is relaxed. A relaxation which offers a good trade-off between the pricing problem difficulty and the lower bound obtained by solving the master problem is the *ng*-path relaxation (Baldacci et al., 2011b). In this relaxation, a neighbourhood is defined for every client. Non-elementary routes can be generated. However, a client i can be visited twice only if it is not in the neighbourhood of a client visited between two consecutive visits to i. So the pricing problem relaxation is controlled by clients' neighbourhoods. This relaxation can be dynamically strengthened (Roberti and Mingozzi, 2014) by augmenting *ng*-neighbourhoods based on the current fractional

solution. Thus, this process is similar to cut generation.

We have just discussed ways to detect redundant columns which do not participate in any feasible solution of the problem. Alternatively, assuming that an incumbent solution is available, we can try to detect columns which do not participate in a solution which improves on the incumbent (Vanderbeck, 2005). For that, a value is temporarily assigned to a variable and the master problem is resolved. If the dual bound obtained is worse that the incumbent solution value, this value can be removed from the domain of the variable. In general, this approach is expensive, as the master problem should be resolved many times. In some problem specific context however, it can be made efficient. For example, when the pricing problem is the (resource constrained) shortest path problem, fixing arc variables to zero one by one and checking the resulting dual bound is reasonably fast using arc reduced costs (Irnich et al., 2010).

Another approach for detecting redundant columns in the context of vehicle routing has been proposed by Baldacci et al. (2008). In this approach, one enumerates all elementary routes or columns which can participate in an improving solution. Given an optimal dual solution π, a column $\lambda_p \in P^k$ cannot participate in an improving solution if its reduced cost $\bar{c}(\bar{z}_p^k)$ is larger that the current primal-dual gap, i.e. difference between the primal incumbent solution value and the column generation dual bound value. If enumeration is succesfull, then all generated columns are added to the master, and the latter is solved as a MIP to optimality without further column generation. If the number of enumerated columns is large, then they can be put to the column pool, and the pricing from now on can be performed by inspection of the pool Contardo and Martinelli (2014). If the primal-dual gap is small, the column enumeration approach usually provides substantial time reduction for the branch-cut-and-price algorithm. A similar approach in a more general context is discussed by Rönnberg and Larsson (2019).

As it can be seen from previous paragraphs, having a good primal solution is important for the modern BCP. This makes the variable fixing and column enumeration much more efficient. Also in many applications, finding primal solutions is the main focus whereas proving optimality is less important. The three main classes of primal heuristics used in the column generation context are rounding, diving, and sub-MIPing heuristics. First ones iteratively round column values in the fractional solution. Diving heuristic combine rounding and the master problem resolving using repetitive application of column generation. Sub-MIPing heuristics fix certain variables and solve the remaining problem as a MIP. In the column generation context, the standard implementation is to fix to zero all columns absent from the current restricted master problem and solve the latter by a MIP solver. This kind of heuristics are often called *restricted master heuristics*. They are probably the most used heuristics in column generation and BCP approaches. A drawback of column generation based heuristics is that they are very often specific for the application in hand. Some work has been done however on generic primal heuristics for branch-and-price (Lübbecke and Puchert, 2012).

The last but not least BCP component we want to discuss is strong branching. As for general MIP, strong branching is very important for proving optimality of a solution. However, due to implementation difficulty, strong branching is not often used in

BCP algorithms. Every node in the branch-and-bound tree is expensive to evaluate as column generation is used for that. Therefore, strong branching in a BCP algorithm should be multi-phase. First, for a relatively large set of branching candidates we only resolve the restricted master problem without column generation. Secondly, a small subset of promising candidates is chosen and evaluated using incomplete column generation, in which the pricing problem is solved heuristically or the number of iterations is limited. Finally, one or few most promising candidates are evaluated completely, i.e. using complete column and cut generation. When selecting initial set of candidates, branching history is taken into account. For this, pseudocosts for all previously evaluated branching candidate are usually computed (Gamrath and Lübbecke, 2010). Selection of candidates between phases are usually based on the product rule (Achterberg, 2007). Recently, there has been shown a large positive impact of strong branching on BCP solution time in the context of vehicle routing applications (Røpke, 2012; Pecin et al., 2017b).

Generality

There exist at least two public generic branch-cut-and-price solvers:
GCG (`gcg.or.rwth-aachen.de`) (Gamrath and Lübbecke, 2010), and SAS Optimization (`www.sas.com/en_us/software/optimization.html`) (Galati, 2009). Another generic BCP solver BaPCod (`realopt.bordeaux.inria.fr/?page_id=2`) is not public (but in some cases is available by request). These solvers can be used just by providing a MIP formulation for the problem and possibly indicating the decomposition to be applied. However the performance of these solvers is superior to generic MIP solvers only for a small number of problems. This is because the pricing problem is solved by MIP solvers themselves. Most often, the BCP approach is efficient when a specialized algorithm is used to solve the pricing problem. In this case however the overall BCP algorithm becomes also problem specific.

There is a trend to make "non-MIP-based" BCP algorithms more generic, so they can be used for classes of problems and not for a single one. This implies that the pricing solver and its implementation should be as generic as possible. For the moment, this concerns mainly pricing solvers based on dynamic programming. These are solvers for the shortest path problem with resource constraints (Irnich and Desaulniers, 2005), and the minimum flow problem in *decision hypergraphs* (Martin et al., 1990). Such hyper-graph can be generated by *context-free grammars* which is a rather expressive tool to model many personnel scheduling problems (Côté et al., 2011). A generic branch-and-bound algorithm for problems modelled using decision diagrams (Bergman et al., 2016) may also be used as a pricing solver. This algorithm also uses dynamic programming to obtain bounds.

Up to now, rather generic BCP algorithms based on dynamic programming to solve the pricing problem has been proposed for vehicle routing, crew and personnel scheduling problems in (Desaulniers et al., 1998; Baldacci and Mingozzi, 2009; Côté et al., 2013).

Chapter 2

Generic Approaches

In this chapter, we review our works which concern generic components of branch-cut-and-price. Approaches proposed here can be applied to many different problems.

In Section 2.1, we present a way to reformulate a given MIP in an extended space, as well as the associated column-and-row generation approach to solve the linear relaxation of the reformulated problem. Our reformulation can be viewed as a generalization of the Dantzig-Wolfe reformulation. In the case when the standard column generation has convergence problems, the column-and-row generation approach sometimes allows us to obtain dual bounds of the same or nearly same quality in much less time.

In Section 2.2, we propose another approach to improve convergence of the standard column generation. This approach is based on dual price smoothing technique already known in the literature. Our main contribution here consists in making this approach "parameter-less". This allows one to use it in a generic BCP solver in a way that is transparent for the user. We also show that there is a synergy between this approach and the penalty function based stabilization.

In Section 2.3, we address an important question how to devise efficient primal heuristics which can be used in a generic BCP solver. We propose several algorithms which belong to the class of diving heuristics. We show that their efficiency is superior to the more conventional restricted master heuristic for some classical problems.

In Section 2.4, we propose a generic model for vehicle routing and related problems. A particularity of this model is that it incorporates the new concept of packing sets. By specifying packing sets, the user passes an additional information to the generic BCP solver. This information allows the solver to use recently proposed BCP components such as *ng*-path relaxation, limited memory rank-1 cuts, and route enumeration. Thus the solver can obtain the state-of-the-art performance while remaining a generic tool.

2.1 Column Generation for Extended Formulations

This section is based on the short conference paper (Sadykov and Vanderbeck, 2011) and the full journal paper (Sadykov and Vanderbeck, 2013b).

Working in an extended variable space allows one to develop tight reformula-

tions for mixed integer programs. However, the size of the extended formulation grows rapidly too large for a direct treatment by a MIP-solver. Then, one can work with inner approximations defined and improved by generating dynamically variables and constraints. When the extended formulation stems from subproblems' reformulations, one can implement column generation for the extended formulation using Dantzig-Wolfe decomposition paradigm. Pricing subproblem solutions are expressed in the variables of the extended formulation and added to the current restricted version of the extended formulation along with the subproblem constraints that are active for the subproblem solutions. This so-called "column-and-row generation" procedure is revisited here in an unifying presentation that generalizes the column generation algorithm and extends to the case of working with an approximate extended formulation. The interest of the approach is evaluated numerically on machine scheduling, and multi-echelon lot-sizing problems. We compare a direct handling of the extended formulation, a standard column generation approach, and the "column-and-row generation" procedure, highlighting a key benefit of the latter: lifting pricing problem solutions in the space of the extended formulation permits their recombination into new subproblem solutions and results in faster convergence.

2.1.1 Column-and-row generation procedure

Assume an original pure integer program [F] that can be stated in the form (1.3). Let the subsystem be defined by

$$X = \{Bx \geqslant b;\ x \in \mathbb{Z}^n_+\}. \tag{2.1}$$

Here $n = n_1$, $\Theta \in \mathbb{Q}^{m_1 \times n}$ and $B \in \mathbb{Q}^{m_2 \times n}$ are rational matrices, while $d \in \mathbb{Q}^{m_1}$ and $b \in \mathbb{Q}^{m_2}$ are rational vectors. Assume that X (respectively [F]) is a pure integer program that is feasible and bounded.

Assume now that there exists a polyhedron $Q = \{H\,z \geqslant h, z \in \mathbb{R}^e_+\}$, defined by a rational matrix $H \in \mathbb{Q}^{f \times e}$ and a vector $h \in \mathbb{Q}^f$, and a linear transformation T defining the projection: $z \in \mathbb{R}^e_+ \to x = (T\,z) \in \mathbb{R}^n_+$. Moreover, Q defines an extended formulation for $\mathrm{conv}(X)$, i.e., $\mathrm{conv}(X) = \mathrm{proj}_x\, Q = \{x = T\,z :\ H\,z \geqslant h, z \in \mathbb{R}^e_+\}$, and $Z = Q \cap \mathbb{Z}^e_+$ defines an extended IP-formulation for X, i.e., $X = \mathrm{proj}_x\, Z = \{x = T\,z : H\,z \geqslant h, z \in \mathbb{Z}^e_+\}$. T is a generalization of inverse mapping M^{-1} defined in Section 1.1.

The subproblem extended formulation immediately gives rise to a reformulation of [F], to which we refer by [R]:

$$[\mathrm{R}] \equiv \min\{c\,T\,z :\ \Theta\,T\,z \geqslant d;\ H\,z \geqslant h;\ z \in \mathbb{Z}^e_+\}.$$

The standard Dantzig-Wolfe reformulation approach is a special case of extended formulation where X is reformulated as: $X = \{x = \sum_{p \in P_x} \bar{x}^p \lambda_p : \sum_{p \in P_x} \lambda_p = 1, \lambda_p \in \{0,1\}^{|P_x|}\}$, and P_x defines the set of generators of X, i.e. the set of integer solutions of X. Then, the reformulation takes the form known as the master program, to which we refer by [M]:

$$[\mathrm{M}] \equiv \min\{\sum_{p \in P_x} c\,\bar{x}^p\,\lambda_p;\ \sum_{p \in P_x} \Theta\,\bar{x}^p\,\lambda_p \geqslant d;\ \sum_{p \in P_x} \lambda_p = 1;\ \lambda \in \{0,1\}^{|P_x|}\}. \tag{2.2}$$

Let $\{\bar{z}^p\}_{p\in P}$ be the enumerated set of solutions $\bar{z}^p$ of the extended subproblem formulation $Z \subseteq \mathbb{Z}^e_+$.

Definition 2.1. Given a solution $\bar{z}^p$ of Z, let $J(z^p) = \{j : z^p_j > 0\} \subseteq \{1,\ldots,e\}$ be the support of solution vector $\bar{z}^p$ and let $I(z^p) = \{i : H_{ij} \neq 0 \text{ for some } j \in J(z^p)\} \subseteq \{1,\ldots,f\}$ be the set of constraints of Q that involve some non zero components of z^p. The "restricted reformulation" [R$'$] defined by a subset $P' \subset P$ of feasible solutions to Z is:

$$[\mathrm{R}'] \equiv \min\{c\,T'\,z' :\ \Theta\,T'\,z' \geqslant d;\ H'\,z' \geqslant h';\ z' \in \mathbb{Z}^{|J'|}_+\},$$

where z' (resp. h') is the restriction of z (resp. h) to the components of $J' = \bigcup_{p\in P'} J(z^p)$, H' is the restriction of H to the rows of $I' = \bigcup_{p\in P'} I(z^p)$ and the columns of J', while T' is the restriction of T to the columns of J'.

The procedure to solve the linear relaxation of [R] by dynamic generation of its columns and rows is:

Step 0: Initialize the dual bound, $\beta := -\infty$, and the subproblem solution set P' so that the linear relaxation of [R$'$] is feasible.

Step 1: Solve the LP relaxation of [R$'$] and collect its value $v^{R'}_{LP}$ and the dual solution π associated to constraints $\Theta\,T'\,z' \geqslant d$.

Step 2: Solve the pricing problem: $\bar{z}(\pi) := \mathrm{argmin}\{(c - \pi\Theta)\,T\,z : z \in Z\}$.

Step 3: Compute the Lagrangian dual bound: $L(\pi) = \pi\,d + (c - \pi\Theta)\,T\,\bar{z}(\pi)$, and update the dual bound $\beta := \max\{\beta, L(\pi)\}$. If $v^{R'}_{LP} = \beta$, STOP.

Step 4: Update the current bundle, P', by adding solution $\bar{z}(\pi)$ and update the resulting restricted reformulation [R$'$] according to Definition 2.1. Then, goto Step 1.

The validity of the above column-and-row generation procedure for the extended reformulation [R] derives from the following Proposition 2.1. Note first that given a subset $P' \subset P$, one can define the associated set

$$G' = G(P') = \{g \in G :\ \bar{x}^g = T\,z^p \text{ for some } p \in P'\}$$

which in turn defines a restricted formulation [M$'$]. Let $v^{R'}_{LP}$, $v^{M'}_{LP}$ be the optimum value of LP relaxations of the restricted formulations [R$'$] and [M$'$].

Proposition 2.1. *(i)* $v^* = v^R_{LP} = v^M_{LP} \leqslant v^{R'}_{LP} \leqslant v^{M'}_{LP}$.

(ii) For any $\pi \geqslant 0$, the Lagrangian bound, $L(\pi) = \pi\,d + (c - \pi\theta)\,T\,\bar{z}(\pi)$, defines a valid dual bound for the LP relaxation of [R]. Hence, bound β, in the above procedure is a valid dual bound, i.e., $\beta \leqslant v^*$.

(iii) If $v^{R'}_{LP} = \beta$ (the stopping condition in Step 3 is satisfied), then $v^{R'}_{LP} = v^*$.

(iv) If (π,σ) is an optimal dual solution to the linear relaxation of [R$'$], and $v^{R'}_{LP} > \beta$, then $[(c - \pi\Theta)\,T - \sigma\,H]\bar{z}(\pi) < 0$ for the subproblem solution $\bar{z}(\pi)$ that is obtained in Step 2. Hence, some of the components of $\bar{z}(\pi)$ were not present in [R$'$] and have a negative reduced cost for the current dual solution (π,σ).

In the column-and-row generation procedure, pricing can be operated in the original variables, x, in Step 2. Indeed, $\min\{(c - \pi\Theta)\, T\, z : \ z \in Z\} \equiv \min\{(c - \pi\Theta)\, x : \ x \in X\}$. But, to implement Step 4, one needs to lift the solution $\bar{x}(\pi) := \operatorname{argmin}\{(c - \pi\Theta)\, x : x \in X\}$ in the z-space in order to add variables to [R], i.e., one must have a procedure to define $\bar{z}(\pi)$ such as $\bar{x}(\pi) = T\, \bar{z}(\pi)$.

The proposed procedure remains valid under weaker condition $\text{conv}(X) \subset \text{proj}_x\, Q$, i.e. when the projection of the extended formulation to x-space gives an outer approximation of the convex hull of X. In this case the obtained lower bound v^R_{LP} may not be as strong as the column generation one: $v^R_{LP} \leqslant v^* = v^M_{LP}$. The termination of the column-and-row generation procedure remains guaranteed. On termination, one may not have the value v^R_{LP} but one has at least as good valid dual bound β: $v^R_{LP} \leqslant \beta \leqslant v^* = v^M_{LP}$. Thus, once $v^{R'}_{LP} \leqslant \beta$, there is not real incentive to further consider columns $\bar{z}(\pi)$ with negative reduced cost components in the LP relaxation of [R′]. Although this may decrease $v^{R'}_{LP}$, there is no more guarantee that β increases in further iterations.

2.1.2 Interest of the approach

Both the column-and-row generation method for [R] and the standard column generation approach for [M] can be understood as ways to get around the issue of size arising in a direct solution of the extended formulation [R]. The primary interest for implementing column generation for [R] rather than for [M] is to exploit the inequality $v^{R'}_{LP} \leqslant v^{M'}_{LP}$ in (i) of Proposition 2.1. This inequality states that column-and-row generation applied to [R_{LP}] can converge faster than standard column generation applied to [M_{LP}] when there exist possible re-compositions of solutions in the LP relaxation of [R′] that would not be feasible in the LP relaxation of [M′]. In the literature (Valério de Carvalho, 1999), another motivation is put forward for using the column-and-row generation rather than standard column generation: [R] offers a richer model in which to define cuts or branching restrictions. Note however that although [R] provides new entities for branching or cutting decisions, one can implicitly branch or formulate cuts on the variables of [R] while working with [M] given that one does pricing in the z-space.

The drawbacks of a column-and-row approach, compared to applying standard column generation, are:

- having to handle a larger restricted linear program: the LP relaxation of [R′] has more variables and constraints than the LP relaxation of [M′] for a given P'.
- having to manage dynamic row generation along side column generation;
- having to face potential symmetries in the representation of solutions that might arise in the extended formulation;
- potentially having to use a subproblem oracle specific to the subproblem extended formulation, if lifting a subproblem solution in the extended space is not an option; this is an issue when pricing in the z-variable space requires higher complexity / computing times than in the x-variables.

We now illustrate the proposed approach on two problems: machine scheduling and multi-item lot-sizing. In our paper (Sadykov and Vanderbeck, 2013b), we also present results for bin packing and generalized assignment problems.

2.1.3 Illustration on machine scheduling

Consider first a single machine scheduling problem on a planning horizon T as studied by van den Akker et al. (2000). The problem is to schedule the jobs, $j \in J = \{1, \ldots, n\}$, a single one at the time, at minimum cost, which can be modeled as :

$$[F] \equiv \min \left\{ \sum_j c(S_j) : \; S_j + d_j \leqslant S_i \;\text{ or }\; S_i + d_i \leqslant S_j \; \forall (i,j) \in J \times J \right\} \tag{2.3}$$

where S_j denotes the start time of job j and d_j is its given duration (i.e. processing time). Disjunctive program (2.3) admits an extended formulation written in terms of decision variables $z_{jt} = 1$ if and only if job $j \in J \cup \{0\}$ starts at the outset of period $t \in \{1, \ldots, T\}$, where job 0 with processing time 1 models machine idle time. By convention, period t is associated with time interval $[t-1, t)$ and z_{jt} is only defined for $1 \leqslant t \leqslant T - d_j + 1$. The reformulation [R] takes the form:

$$\min \sum_{jt} c_{jt}\, z_{jt} \tag{2.4}$$

$$\sum_{t=1}^{T-d_j+1} z_{jt} = 1, \qquad \forall j \in J, \tag{2.5}$$

$$\sum_j z_{j1} = 1, \tag{2.6}$$

$$\sum_j z_{jt} = \sum_{j:\, t-d_j \geqslant 1} z_{j,t-d_j}, \qquad \forall t > 1, \tag{2.7}$$

$$z_{jt} \in \{0,1\}, \qquad \forall j, t, \tag{2.8}$$

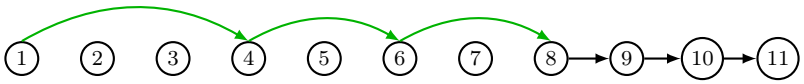
where (2.5) enforces the assignment of each job, (2.6) the initialization of the schedule, while (2.7) forbids the use of the machine for more than one job at the time: a job j can start in t only if one ends in t and therefore releases the machine. The formulation can be extended to the case in which m identical machines are available; then the right-hand-size of (2.6) is m and variable z_{0t} represents the number of idle machines at time t. One can also model in this way a machine with arbitrary capacity where jobs have unit capacity consumption. The objective can model any cost function that depends on job start times (or completion times).

Extended reformulation [R] has size $O(|J|\,T)$ which is pseudo-polynomial in the input size as $T \geqslant \sum_j d_j$. The subsystem defined by constraints (2.6-2.7) characterizes a flow that represents a "pseudo-schedule" satisfying non-overlapping constraints but not the single assignment constraints. A standard column generation approach based on subsystem (2.6-2.7) consists in defining reformulation:

$$[M] \equiv \min \left\{ \sum_{p \in P} c^p\, \lambda_p : \sum_{p \in P} \sum_{t=1}^{T-d_j+1} \bar{z}^p_{jt}\, \lambda_p = 1 \; \forall j, \; \sum_{p \in P} \lambda_p = m, \; \lambda_p \in \{0,1\} \; \forall p \in P \right\}$$

where P is the set of "pseudo-schedules": vector $\bar{z}^p$ and scalar c^p define the associated solution and cost for a solution $p \in P$. As done by van den Akker et al. (2000), reformulation [M] can be solved by column generation. The pricing subproblem [SP] is a shortest path problem: find a sequence of jobs and down-times to be scheduled on the single machine with possible repetition of jobs. The underlying graph is defined by nodes that represent periods and arcs $(t, t + d_j)$ associated to the processing of jobs $j \in J \cup \{0\}$ in time interval $[t - 1, t + d_j)$. Figure 2.1 illustrates such path for a numerical instance.

Figure 2.1: A path associated to a pseudo-schedule solution to the sub-problem for $T = 10$ and $J = \{1, \ldots, 4\}$ and $d_j = j$ for each $j \in J$. The sequence consists in scheduling job 3, then twice job 2 consecutively, and to complete the schedule with idle periods (represented by straight arcs).

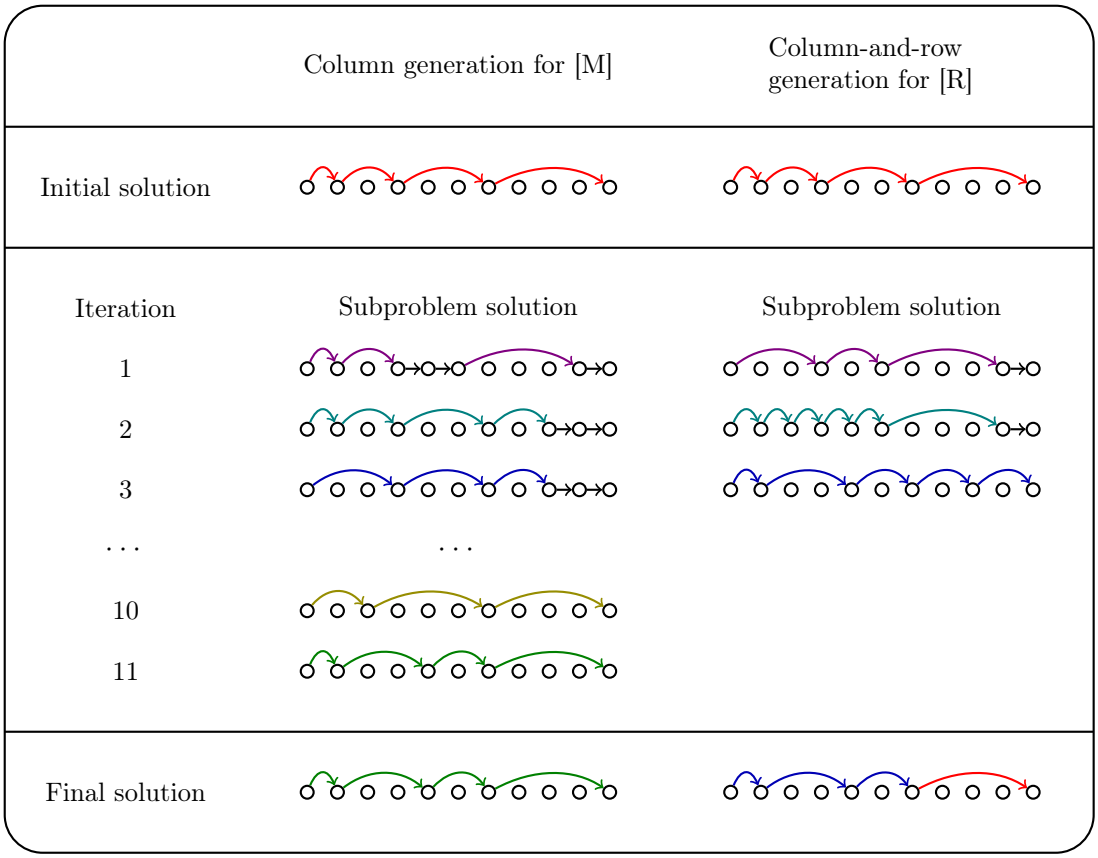


An alternative to the above standard column generation approach for [M] would be the proposed column-and-row generation for [R]. In it, we solve the shortest path pricing problem [SP] and add to [R] the components of the subproblem solution $\bar{z}^p$ in the time index formulation along with the flow conservation constraints that are binding for that solution. To illustrate the difference between the two approaches, Figure 2.2 shows several iterations of the column generation procedure for [M] and [R], for the numerical instance of Figure 2.1. Formulations [M] and [R] are initialized with the variable(s) associated to the same pseudo-schedule depicted in Figure 2.2 as the initial subproblem solution. Note that the final solution of [M] is the solution obtained as the subproblem solution generated at iteration 11; while, for formulation [R], the final solution is a recombination of the subproblem solution of iteration 3 and the initial solution.

For the computational experiments, we used instances with the total weighted tardiness objective function (this problem is denoted as $P \mid\mid \sum w_j T_j$). Instance size is determined by a triple $(n, m, d_{\max})$, where n is the number of jobs, m is the number of machines, and $d_{\max}$ is the maximum duration of jobs. Instances are generated using the procedure by Potts and Van Wassenhove (1985): integer durations d_j are uniformly distributed in interval $[1, 100]$ and integer weights w_j in $[1, 10]$ for jobs j, $j = 1, \ldots, n$, while integer due dates have been generated from the uniform distribution $[D(1 - TF - RDD/2)/m, D(1 - TF + RDD/2)/m]$, where $D = \sum_j d_j$, TF is the tardiness factor, RDD is the relative range of due dates, $TF, RDD \in \{0.2, 0.4, 0.6, 0.8, 1\}$. For each instance size, 25 instances were generated, one for each couple of parameters (TF, RDD).

In Table 2.1, we compare methods on these instances without using dual price smoothing as a stabilization technique. The table reports on column generation for [M_{LP}], column-and-row generation for [R_{LP}], and solving [R_{LP}] directly using Cplex. In both column generation and column-and-row generation, the master is

Figure 2.2: Solving the example by column versus column-and-row generation (assuming the same data as in of Figure 2.1). Each bended arc represents a job, and straight arcs represent idle periods.



initialized with a trivial heuristic solution. With column-and-row generation, a lot of recombinations occur during the "heading-in" phase at the outset of the algorithm when the dual information is still very poor. These recombinations slow down the master solution time, while not being very useful as variables generated during this initial phase are not likely to appear in the optimum solution. Hence, we adopt a hybrid approach (also reported in Table 2.1), starting the algorithm using pure column generation and switching to column-and-row generation only beyond the "heading-in" phase (precisely, when $L(\pi) > 0$). This hybrid technique induces time saving in solving the master at the expense of an increase in the number of iterations (by a factor 2 to 3). The results reveals that solving $[R_{LP}]$ directly using Cplex is not competitive. Thanks to the stabilization effect of column recombinations, column-and-row generation yields a significant reduction in the number of iterations compared to standard column generation. However the restricted master $[R'_{LP}]$ is typically much larger (and harder to solve) than $[M'_{LP}]$ (around twice the number of variables and 20 times the number of constraints). Despite this fact, column-and-

			Cplex 12.1 for [R_{LP}]	Col. gen. for [M_{LP}]		Col-and-row gen. for [R_{LP}]		Hyb. col-and-row gen. for [R_{LP}]	
m	n	$p_{\max}$	*cpu*	*it*	*cpu*	*it*	*cpu*	*it*	*cpu*
1	25	50	1.6	331	0.5	50	0.3	106	0.3
1	50	50	18.8	1386	19.7	76	2.9	207	2.8
1	100	50	304.2	8167	1449.5	104	24.8	354	19.4
1	25	100	7.1	337	0.9	72	0.9	124	0.8
1	50	100	132.6	1274	24.2	107	8.9	246	8.6
1	100	100	2332.0	8907	1764.4	144	90.3	455	61.3
2	25	100	4.1	207	0.3	63	0.2	97	0.2
2	50	100	109.2	645	5.7	94	1.7	173	1.9
2	100	100	3564.4	2678	115.5	117	14.3	319	14.9
4	50	100	18.7	433	1.5	90	0.6	167	0.7
4	100	100	485.7	1347	27.9	113	4.7	295	5.2
4	200	100	>2h	4315	409.7	148	36.1	561	39.7

Table 2.1: Computational results for Machine Scheduling

row generation is much faster. In our paper (Sadykov and Vanderbeck, 2013b), we have also tested the standard column generation and the column-and-row generation combined with dual pricing smoothing stabilization (Wentges, 1997). In this case, the difference between two methods is smaller. Still, however, the hybrid column-and-row generation is the best approach.

2.1.4 Illustration on multi-item lot-sizing

The Multi-Item Lot-Sizing problem consists in planning production so as to satisfy demands d_t^k for item $k = 1, \ldots, K$ over a discrete time horizon with period $t = 1, \ldots, T$ either from stock or from production. The production of an item entails production stages (echelons) $e = 1, \ldots, E$, each of which takes place on a different machine that can only process one product in each period (under the so-called small bucket assumption). A compact formulation [F] is:

$$\min\{\sum_{ket}(c_{et}^k\, x_{et}^k + f_{et}^k\, y_{et}^k) : \tag{2.9}$$

$$\sum_k y_{et}^k \leqslant 1, \qquad \forall e, t, \tag{2.10}$$

$$\sum_{\tau=1}^{t} x_{e\tau}^k \geqslant \sum_{\tau=1}^{t} x_{e+1,\tau}^k, \qquad \forall k, e < E, t, \tag{2.11}$$

$$\sum_{\tau=1}^{t} x_{E\tau}^k \geqslant D_{1t}^k, \qquad \forall k, t, \tag{2.12}$$

$$x_{et}^k \leqslant D_{tT}^k\, y_{et}^k, \qquad \forall k, e, t, \tag{2.13}$$

$$x_{et}^k \geqslant 0 \qquad \forall k, e, t, \tag{2.14}$$

$$y_{et}^k \in \{0, 1\}, \qquad \forall k, e, t, \tag{2.15}$$

where variables x^k_{et} are the production of item k at echelon e in period t (at unit cost c^k_{et}) and y^k_{et} take value 1 if the production of item k at echelon e is setup in period t (at a fixed cost f^k_{et}); $D^k_{1t} = \sum_{\tau=1}^{t} d^k_\tau$. The stock values can be computed as $s^k_{et} = \sum_{\tau=1}^{t} x^k_{e\tau} - \sum_{\tau=1}^{t} x^k_{e+1,\tau}$; their costs have been eliminated (they are included in c^k_{et}).

There exists an optimal solution where at each echelon and period either there is an incoming stock or an incoming production but not both, i.e., such that $x^k_{et}\, s^k_{et} = 0\ \forall e, t$. Hence, production can be restricted to lots corresponding to an interval of demands. This dominance property can be exploited to solve single item subproblem by dynamic programming in polynomial time (Pochet and Wolsey, 2006). A backward dynamic program (DP) can be defined where the states are associated with quadruplets (e, t, a, b) denoting the fact of having at echelon e in period t accumulated a production that is covering exactly the demand D^k_{ab} for the final product of item k. It is defined for $t \leqslant a \leqslant b \leqslant T$ and $e = 1, \ldots, E$. The backward recursion is:

$$V(e,t,a,b) = \min\left\{V(e,t+1,a,b),\ \min_{l=a,\ldots,b}\{V(e+1,t,a,l) + c^k_{et}\, D^k_{al} + f^k_{et} + V(e,t+1,l+1,b)\}\right\}$$

for all $e = E, \ldots, 1$, $t = T, \ldots, 1$, $a = T, \ldots, 1$, and $b = T, \ldots, a$. By convention $V(e,t,a,b) = 0$ if $a > b$. The initialization is $V(E+1,t,a,b) = 0$. The optimum is given by $V^* = V(1,1,1,T)$.

From the dynamic program, one can reformulate the single item subproblem as selecting a decision tree in a hypergraph whose nodes are the states of the above DP. The DP transition can be associated to flow on hyperarcs: $z^k_{e,t,a,l,b} = 1$ if at echelon $e \in \{1, \ldots, E\}$ in period $t \in \{1, \ldots, T\}$ the production of item k is made to cover demands from period $a \in \{t, \ldots, T\}$ to period $l \in \{a-1, \ldots, T\}$, while the rest of demand interval, i.e. demands from period $l+1$ to period $b \in \{l, \ldots, T\}$, will be covered by production in future periods. If $l = a-1$, there is no production; this can only happen when $a > t$. While if $l = b$, the whole demand interval, D^k_{ab}, is produced in t. The associated cost, $c^k_{e,t,a,l,b}$, is $(c^k_{et}\, D^k_{al} + f^k_{et})$ if $l \geqslant a$ and zero if $l = a-1$. For the initial echelon $e = 1$, variables $z^k_{1,t,a,l,b}$ are only defined for $b = T$. For the first period $t = 1$, they are only defined for $a = t = 1$. This leads to reformulation [R]:

$$\min \sum_{e,t,a,l,b,k} c^k_{e,t,a,l,b}\, z^k_{e,t,a,l,b} \tag{2.16}$$

$$\sum_{e=1}^{E} \sum_{a,l,b,k:\, l \geqslant a, D^k_{al} > 0} z^k_{e,t,a,l,b}, \leqslant 1, \qquad \forall e, t \tag{2.17}$$

$$\sum_{l} z^k_{1,1,1,l,T} = 1, \qquad \forall k, \tag{2.18}$$

$$\sum_{l} z^k_{e,t,a,l,b} - \sum_{\tau \leqslant a} z^k_{e,t-1,\tau,a-1,b} - \sum_{\tau \geqslant b} z^k_{e-1,t,a,b,\tau} = 0, \qquad \forall k, e, t, a, b, \tag{2.19}$$

$$z^k_{e,t,a,l,b} \in \{0,1\}, \qquad \forall k, e, t, a, l, b., \tag{2.20}$$

which results from subproblem reformulation Z^k defined by constraints (2.18-2.20) for a fixed k. Note that constraints (2.19) are only defined for $t > 1$ and $b = T$ when $e = 1$; while when $e > 1$, they are only defined for $a = t$ when $t = 1$.

		Col. gen. for [M_{LP}]		Col-and-row gen. for [R_{LP}]	
K	T	*it*	*cpu*	*it*	*cpu*
2 echelons					
10	50	126	1.7	29	1.6
20	50	79	1.8	27	3.1
10	100	332	38.0	43	8.1
20	100	232	31.5	38	20.0
3 echelons					
10	50	187	11.8	38	5.5
20	50	112	12.0	33	9.8
10	100	509	454.5	49	36.4
20	100	362	520.4	48	103.1
5 echelons					
10	50	296	62.6	48	16.3
20	50	223	66.8	42	34.3
10	100	882	4855.9	61	134.0
20	100	750	4657.8	56	386.1

Table 2.2: Computational results for multi-echelon multi-item lot-sizing

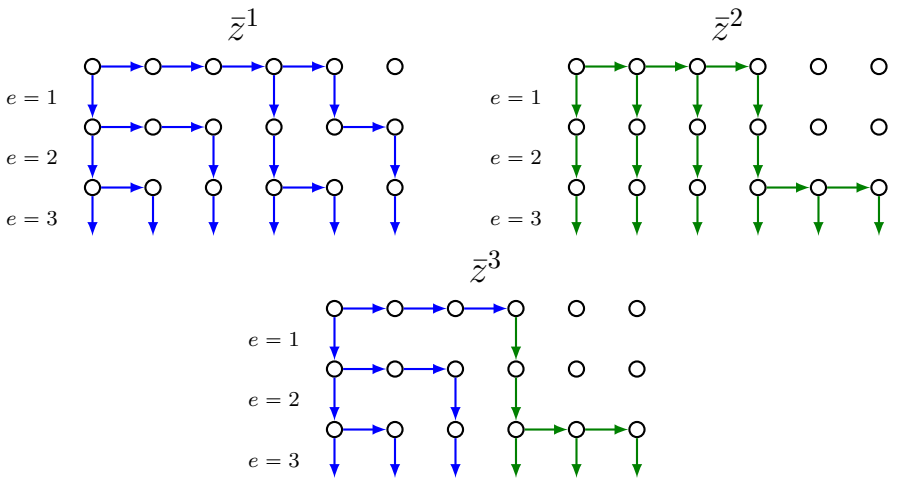
In Table 2.2, we compare standard column generation and pure column-and-row generation respectively for [M_{LP}] and [R_{LP}]. Dual price smoothing stabilization (Wentges, 1997) was used in the both approaches. Solving formulation [R_{LP}] directly with Cplex is impractical. The restricted master is initialized with a trivial heuristic solution. Results are averages over 5 instances generated randomly, with a number of items $K \in \{10, 20, 40\}$, a number of periods $T \in \{50, 100, 200, 400\}$, setup costs uniformly distributed in $[20, 100]$, production costs being zero, and storage cost h_e^k generated as $h_{e-1}^k + \gamma$, where γ is uniformly distributed in interval $[1, 5]$. For each period, there is a positive demand for 3 items on average. Demands are generated using a uniform distribution on interval $[10, 20]$.

For the column generation approach to [M_{LP}], instances get easier as the number of items increases. Indeed, instances with more items have fewer feasible solutions given the single mode constraints. The column-and-row generation clearly outperforms standard column generation on all instances except those with 2 echelons and 50 periods. The number of iterations for column-and-row generation is up to an order of magnitude smaller. This shows the benefit of recombinations of decision trees (as illustrated in Figure 2.3) that take place in this application. These experiments shows that very large extended formulations can be tractable when solved by column-and-row generation.

2.1.5 Related work

Approaches similar to the column-and-row generation procedure were previously described in the literature in application-specific context, such as bin packing (Valério de Carvalho, 1999), multi-commodity flow (Mamer and McBride, 2000; Löbel, 1998),

Figure 2.3: Two decision trees associated with subproblem solutions $\bar{z}^1$ and $\bar{z}^2$. Their recombination $\bar{z}^3$ is feasible for $[\mathrm{R}'_{LP}]$, but not feasible for $[\mathrm{M}'_{LP}]$, both defined by the same subset $P' \supseteq \{1, 2\}$ of solutions.



split delivery vehicle routing (Feillet et al., 2010), or network design (Feillet et al., 2010; Frangioni and Gendron, 2009). Convincing computational results of some of these papers indicate the interest of the method. Although the motivations of these studies are mostly application specific, methodological statements made therein are to some extend generic. Moreover, there are recent efforts to explore this approach further. In a study developed concomitantly to ours, Frangioni and Gendron (2013) present a "structured Dantzig-Wolfe decomposition" for which they adapt column generation stabilization techniques (from linear penalties to the bundle method). Compared to (Frangioni and Gendron, 2013), our generic presentation, relying on the definition of an extended formulation, assumes slightly less restrictive assumptions and extends to approximate extended formulation. Feillet et al. (2010) present a branch-and-price-and-cut algorithm where columns and cuts are generated simultaneously; their presentation considers approximate extended formulation but with a weaker termination condition. Muter et al. (2013) consider what they call a "simultaneous column-and-row generation" approach, but it takes the meaning of a nested decomposition, different from the method reviewed here: the subproblem has itself a decomposable structure.

Bienstock and Zuckerberg (2010) developed a similar approach for solving precedence-constrained scheduling problems arising in open pit mining. In their algorithm, the subproblem is the maximum closure problem. It is polynomially solvable and admits an "ideal" extended formulation. The main difference of the Bienstock and Zuckerberg (BZ) approach from ours is that the former does not fully disaggregate the subproblem solution $\bar{z}^p$ into individual variables z^p_t, $1 \leqslant t \leqslant n_1$, to be added to the restricted problem $[\mathrm{R}'_{LP}]$. Instead, the subproblem solutions are only partly disaggregated in order to keep the size of $[\mathrm{R}'_{LP}]$ reasonable but still allow the subproblem

solutions to recombine in the master. In addition, they can aggregate previously disaggregated subproblem solutions when it is advantageous. The BZ algorithm obtained excellent results for solving LP relaxation of the strong extended formulation [R] of the open pit mining problem. Muñoz et al. (2018) extended the BZ algorithm to a broader class of problems, in particular to the well-known resource constrained project scheduling problem.

2.2 Stabilization

This section is based on the long conference paper (Pessoa et al., 2013) and the full journal paper (Pessoa et al., 2018c).

The convergence of a column generation algorithm can be improved in practice by using stabilization techniques. Smoothing and proximal methods based on penalizing the deviation from the incumbent dual solution have become standards of the domain. Interpreting column generation as cutting plane strategies in the dual problem, we analyze the mechanisms on which stabilization relies. In particular, the link is established between smoothing and in-out separation strategies to derive generic convergence properties. For penalty function methods as well as for smoothing, we describe proposals for parameter self-adjusting schemes. Such schemes make initial parameter tuning less of an issue as corrections are made dynamically. Such adjustments also allow to adapt the parameters to the phase of the algorithm. We provide extensive test reports that validate our self-adjusting parameter scheme and highlight their performances. Our results also show that using smoothing in combination with penalty function yields a cumulative effect on convergence speed-ups.

We have been using the automatic (parameter-less) dual price smoothing stabilization presented in Section 2.2.2 for a majority of applications described in Chapter 3. Automatic smoothing is activated by default in our BaPCod solver as well as in the generic BCP solver for vehicle routing problems presented in Section 2.4. We know that our technique is used in the open-source GCG solver (Gamrath and Lübbecke, 2010), as well as in column generation based generic solver for employees scheduling and rostering solver developed internally by the EURODECISION company (personal communication).

2.2.1 Connection between the column generation procedure and the Lagrangian dual problem

Consider the column generation approach introduced in Section 1.1. To simplify the presentation, we assume the case of single subproblem [SP] and we drop index k. Consider a vector $\bar{\lambda} \in \mathbb{Z}_+^{|P|}$, and vector $\bar{x} \in \mathbb{Z}^{n_1}$ obtained from $\bar{\lambda}$ using (1.1c). We denote $x = Tz$ and $z = U\lambda$, where $T \in \mathbb{R}^{n_1 \times n}$, and $U \in \mathbb{R}^{n \times |P|}$. Then constraint (1.1c) can be rewritten as $x = TU\lambda$. For any Lagrangian penalty vector $\bar{\pi} \in \mathbb{R}_+^m$, the *Lagrangian function*,

$$\begin{aligned} L(\bar{\pi}, \bar{\lambda}, \bar{y}) &= c\bar{x} + g\bar{y} + f\mathbf{1}\bar{\lambda} + \bar{\pi}d - \bar{\pi}\Theta TU\bar{\lambda} - \bar{\pi}\Phi\bar{y} \\ &= \bar{\pi}d + (cTU - \bar{\pi}\Theta TU + f\mathbf{1})\bar{\lambda} + (g - \bar{\pi}\Phi)\bar{y} \end{aligned}$$

is optimized over λ and y to yield a valid dual bound for (1.1), by solving the *Lagrangian subproblem*:

$$[\text{LSP}(\bar{\pi})] \equiv L(\bar{\pi}) = \min_{\substack{\lambda \in \mathbb{Z}_+^{|P|}: \underline{W} \leqslant \mathbf{1}\lambda \leqslant \overline{W}, \\ \underline{Y} \leqslant y \leqslant \overline{Y}}} L(\bar{\pi}, \lambda, y).$$

The *Lagrangian dual function* is defined by $L : \pi \in \mathbb{R}_+^m \to L(\pi)$. Maximizing function L leads to the best dual bound that can be derived from the Lagrangian relaxation. The *Lagrangian dual problem* is defined as

$$[\text{LD}] \equiv \max_{\pi \in \mathbb{R}_+^m} L(\pi).$$

By dualizing constraints $\underline{W} \leqslant \mathbf{1}\lambda \leqslant \overline{W}$ and $\underline{Y} \leqslant y \leqslant \overline{Y}$, the Lagrangian dual problem can be written as a linear program:

$$[\text{LD}] \equiv \max_{\pi \in \mathbb{R}_+^m} \pi d + \min_{\substack{\lambda \in \mathbb{Z}_+^{|P|}, \\ \underline{W} \leqslant \mathbf{1}\lambda \leqslant \overline{W}, \\ \underline{Y} \leqslant y \leqslant \overline{Y}}} (cTU - \pi\Theta TU + f\mathbf{1})\lambda + (g - \pi\Phi)y \tag{2.21}$$

$$\equiv \max \big\{ \pi d + \nu^- \underline{W} - \nu^+ \overline{W} + \mu^- \underline{Y} - \mu^+ \overline{Y} : \tag{2.22}$$

$$\nu^- - \nu^+ + \pi\Theta TU \leqslant cTU + f\mathbf{1}, \tag{2.23}$$

$$\mu^- - \mu^+ + \pi\Phi = g, \tag{2.24}$$

$$\pi \in \mathbb{R}_+^m,\ \nu^-, \nu^+ \in \mathbb{R}_+,\ \mu^-, \mu^+ \in \mathbb{R}_+^{n_2} \big\} \tag{2.25}$$

$$\equiv \min \big\{ (cTU + f\mathbf{1})\lambda + gy : \tag{2.26}$$

$$\Theta TU\lambda + \Phi y \geqslant d, \tag{2.27}$$

$$\underline{W} \leqslant \mathbf{1}\lambda \leqslant \overline{W},\ \underline{Y} \leqslant y \leqslant \overline{Y}, \lambda \in \mathbb{R}_+^{|P|} \big\} \tag{2.28}$$

The formulation (1.2) is precisely the form (2.26)-(2.28) of [LD], which we denote as [M]. We now formally describe the column generation procedure to solve [M].

At some iteration τ, the restriction of [M] to columns defined from $P^\tau = \{p_1, \ldots, p_\tau\}$, is denoted by [M$^\tau$]. This *restricted master LP* is:

$$[\text{M}^\tau] \equiv \Big\{ \sum_{p \in P^\tau} (cT\bar{z}^p + f)\lambda_p + gy : \sum_{p \in P^\tau} \Theta T \bar{z}^p \lambda_p + \Phi y \geqslant d,$$
$$\underline{W} \leqslant \sum_{p \in P^\tau} \lambda_p \leqslant \overline{W},\ \underline{Y} \leqslant y \leqslant \overline{Y}, \lambda \in \mathbb{R}_+^{|P^\tau|} \Big\}$$

Linear program [M$^\tau$] is solved to optimality. Let $(\bar{\lambda}^\tau, \bar{y}^\tau)$ denote an optimal solution of [M$^\tau$]. The linear program dual of [M$^\tau$] is

$$[\text{DM}^\tau] \equiv \Big\{ \pi d + \nu^- \underline{W} - \nu^+ \overline{W} + \mu^- \underline{Y} - \mu^+ \overline{Y} : \nu^- - \nu^+ + \pi\Theta T\bar{z}^p \leqslant cT\bar{z}^p + f, \forall p \in P^\tau,$$
$$\mu^- - \mu^+ + \pi\Phi = g, \pi \in \mathbb{R}_+^m,\ \nu^-, \nu^+ \in \mathbb{R},\ \mu^-, \mu^+ \in \mathbb{R}_+^{n_2} \Big\}$$

Let $(\bar{\pi}^\tau, \bar{\nu}^\tau, \bar{\mu}^\tau)$ denote an optimal solution to [DM$^\tau$]. Using this dual solution, one searches for the most negative reduced cost column, by solving the subproblem:

$$\bar{z}^{p_{\tau+1}} \leftarrow \bar{z}(\bar{\pi}^\tau) := \operatorname*{argmin}_{\bar{z}^p :\, p \in P} \left\{ (c - \bar{\pi}^\tau\Theta)T\bar{z}^p \right\}.$$

If $(c-\bar{\pi}^\tau\Theta)T\bar{z}(\bar{\pi}^\tau) < \bar{\nu}^- - \bar{\nu}^+ - f$, then $\bar{z}(\bar{\pi}^\tau)$ defines a negative reduced cost column that is added to the restricted master. Otherwise, the current LP solution is optimal for [M]. Solutions $\tilde{\lambda}^\tau$ and $\tilde{y}^\tau$ which minimize the value of Lagrangian subproblem [LSP($\bar{\pi}^\tau$)] are then the following

$$\tilde{\lambda}_p^\tau = \begin{cases} \overline{W}, & p = p_{\tau+1} \text{ and } (c - \bar{\pi}^\tau\Theta)T\bar{z}^p + f \leqslant 0, \\ \underline{W}, & p = p_{\tau+1} \text{ and } (c - \bar{\pi}^\tau\Theta)T\bar{z}^p + f > 0, \\ 0, & p \neq p_{\tau+1}, \end{cases} \qquad \tilde{y}_s^\tau = \begin{cases} \overline{Y}_s, & g_s - \bar{\pi}^\tau\phi_{\cdot s} \leqslant 0, \\ \underline{Y}_s, & \text{otherwise.} \end{cases}$$

The above algorithm outputs a sequence of values for the Lagrangian price vector $\{\bar{\pi}^\tau\}_{\tau\geqslant 1}$ that converges towards an optimal dual price vector π^*.

Observation 2.1.

(i) The solution of the dual restricted master [DM$^\tau$] is such that $\pi^\tau = \text{argmax}_{\pi\in\mathbb{R}_+^m} L^\tau(\pi)$ *where $L^\tau(\pi)$ defines an approximation of the Lagrangian dual function L, considering only the subset of subproblem solutions $\{\bar{z}^{p_1}, \ldots, \bar{z}^{p_\tau}\}$, i.e.*

$$L^\tau : \pi \to \pi d + \min_{\substack{\lambda\in\mathbb{Z}_+^{|P|}: \\ \lambda_p=0\ \forall p\notin P^\tau, \\ \underline{W}\leqslant\mathbf{1}\lambda\leqslant\overline{W}}} \{(cTU - \pi\Theta TU + f)\lambda\} + \min_{\underline{Y}\leqslant y\leqslant\overline{Y}} \{(g - \pi\Phi)y\}$$

Function L^τ is an upper approximation of function L: $L^\tau(\pi) \geqslant L(\pi)\ \forall\pi \in \mathbb{R}_+^m$.

(ii) Solving [LSP(π^τ)] exactly serves four purposes simultaneously:

a. it yields the most negative reduced cost column: $\bar{z}^{p_{\tau+1}} = \bar{z}(\bar{\pi}^\tau)$ for [M];

b. it yields the most violated constraint defined by a subproblem solution $\bar{z}^p$ for [DM];

c. the constraint violation of the oracle solution $(\tilde{\lambda}^\tau, \tilde{y}^\tau)$ defines a sub-gradient of L at point $\bar{\pi}^\tau$:

$$\nabla L(\bar{\pi}^\tau) := d - \Theta TU\tilde{\lambda}^\tau - \Phi\tilde{y}^\tau. \tag{2.29}$$

d. the correct value of the Lagrangian function L is now known at point $\bar{\pi}^\tau$: $L(\bar{\pi}^\tau) = \bar{\pi}^\tau d + (cTU - \bar{\pi}^\tau\Theta TU + f\mathbf{1})\tilde{\lambda}^\tau + (g - \bar{\pi}^\tau)\tilde{y}^\tau$, and therefore this value remains unchanged in any further approximation of L, i.e., $L^{\tau'}(\bar{\pi}^\tau) = L(\bar{\pi}^\tau)\ \forall\tau' > \tau$.

There are following drawbacks of the column generation algorithm (Vanderbeck, 2005):

- *Dual oscillations:* Solutions $\bar{\pi}^\tau$ jump erratically. One extreme solution of the restricted dual master at iteration t, [DM$^\tau$], is followed by a different extreme point of [DM$^{\tau+1}$], leading to a behaviour often referred to as *"bang-bang"*. Because of these oscillations, it might be that $||\bar{\pi}^{\tau+1} - \pi^*|| > ||\bar{\pi}^\tau - \pi^*||$. Moreover, the dual bounds $L(\bar{\pi}^\tau)$ are converging non monotonically, with ups and downs in the value curve (the *yo-yo* phenomenon).

- *The tailing-off effect:* Towards the end of the algorithm, added inequalities in [DM$^\tau$] tend to cut only a marginal volume of the dual solution space, making progress very slow.
- *Primal degeneracy and alternative dual optimal solutions:* An extreme point $\bar{\lambda}$ of polyhedron [M$^\tau$] has typically fewer non zero values than the number of master constraints. The complementary dual solution solves a system with fewer constraints than variables that admits many alternative solutions. As a consequence, the method iterates between alternative dual solutions without making any progress on the objective value.

2.2.2 Smoothing stabilization techniques

Stabilization techniques are devised to accelerate the convergence of the column generation algorithm. Smoothing techniques belongs to one of standard families of such techniques. The dual solution $\bar{\pi}^\tau$ used for pricing is here "corrected" based on previous dual solutions. In particular, Neame (2000) proposes to define smoothed price as:

$$\tilde{\pi}^\tau = \alpha\tilde{\pi}^{\tau-1} + (1-\alpha)\bar{\pi}^\tau, \tag{2.30}$$

i.e., $\tilde{\pi}^\tau$ is a weighted sum of previous iterates: $\tilde{\pi}^\tau = \sum_{\tau'=1}^{\tau}(1-\alpha)\alpha^{\tau-\tau'}\pi^{\tau'}$ with "discount" factors that model the absolescence of "old" candidates. Wentges (1997) proposes another smoothing rule where:

$$\tilde{\pi}^\tau = \alpha\hat{\pi}^\tau + (1-\alpha)\bar{\pi}^\tau, \tag{2.31}$$

where $\hat{\pi}^\tau$ is the incumbent dual solution at iteration τ. So we have $\tilde{\pi}^\tau = \hat{\pi}^\tau + (1-\alpha)(\bar{\pi}^\tau - \hat{\pi}^\tau)$, which amounts to taking a step of size $(1-\alpha)$ from $\hat{\pi}^\tau$ in the direction of $\bar{\pi}^\tau$. In both rules, $\alpha \in [0,1)$ parameterizes the level of smoothing. The pricing problem is then solved using the smoothed prices, $\tilde{\pi}^\tau$, instead of π^τ:

$$\bar{z}(\tilde{\pi}^\tau) := \underset{\bar{z}^p:\ p\in P}{\operatorname{argmin}}\{(c - \tilde{\pi}^\tau\Theta)T\bar{z}^p\}. \tag{2.32}$$

Solving this modified pricing problem might not yield a negative reduced cost column, even when one exists for $\bar{\pi}^\tau$. This situation is the result of a *mis-pricing*. In this case, one can show that $L(\tilde{\pi}^\tau)$ improves on $L(\hat{\pi}^\tau)$. Thus, $\hat{\pi}^\tau$ is updated to $\tilde{\pi}^\tau$, and re-applying (2.30) or (2.31) with the same $\bar{\pi}^\tau$ solution leads to a new dual price vector that is closer to $\bar{\pi}^\tau$.

When using a smoothing scheme, instead of using the same α for all iterations, one can define iteration-dependent values α_τ. We name α-schedule, the procedure used to select values of α_τ dynamically. Intuitively, a large α can yield deeper cut if no mis-pricing occurs, while a small α can yield large dual bound improvement if a mis-pricing occurs. But a large α resulting in a mis-pricing or a small α with no mis-pricing may result in an iterate with little progress being made. If no smoothing is used, i.e., when $\alpha_\tau = 0\ \forall\tau \geqslant 1$, the procedure is a standard LP based column generation for which finite convergence is proven. For the Simplex algorithm, each basis is visited at most once, provided a cycle breaking rule is used. When smoothing is used on the other hand, the same basis can remain optimal for several iterations

in a sequence of mis-pricings. In the line of the asymptotic convergence proof for in-out separation of Ben-Ameur and Neto (2007), one can show finite convergence of the column generation procedure using smoothing.

Proposition 2.2. *Applying an LP-based column generation procedure to* (2.26)-(2.28) *while pricing on smoothed prices as set in* (2.32)*, using the smoothing scheme with Neame* (2.30) *or Wentges* (2.31) *rule, converges to a solution within an* ϵ*-optimality gap after a finite number of iterations, for any* $\epsilon > 0$ *and* α*-schedule with* $\alpha^\tau \in [0, 1)$ *for all iterations* τ*; i.e., for some* $\tau \in \mathbb{N}$*,* $(\bar{\pi}^\tau, \bar{\nu}^\tau, \bar{\mu}^\tau)$ *is an* ϵ*-optimal solution to* (2.22)-(2.25)*.*

However, asymptotically convergent algorithms might not be suitable for practical purposes. For instance, consider setting $\alpha = 0.8$ for all t. Then, the distance reduction in a mis-pricing sequence becomes small very quickly. In practice, it would be better to choose an α-schedule such that $\tilde{\pi}^\tau = \bar{\pi}^\tau$ after a small number of mis-pricing iterations. Given a static baseline α, we propose to adapt its value during a mis-pricing sequence in the following way. After k mis-prices, value of α_k is set to $\max\{0, 1-k\cdot(1-\alpha)\}$. Thus, we have $\alpha_k = 0$ after $k = \left\lceil \frac{1}{(1-\alpha)} \right\rceil$ mis-pricing iterations, at which point smoothing stops, as $\tilde{\pi}^\tau = \bar{\pi}^\tau$, which forces the end of a mis-pricing sequence.

So far we assumed a static baseline α provided as an input. Let us now consider how the user could be free from having to tune α for his application. In deriving an auto-adaptive α-schedule, we rely on local information. Our proposal is to decrease α when the sub-gradient at point $\tilde{\pi}^\tau$ indicates that a larger step from $\hat{\pi}^\tau$ would further increase the dual bound (i.e., when the angle of the ascent direction, $\nabla L(\tilde{\pi}^\tau)$, as defined in (2.29), and the direction $(\bar{\pi}^\tau - \hat{\pi}^\tau)$ is less than 90°), and vice versa. We now outline the column generation procedure with dynamic α-schedule based on sub-gradient information for a given initial α^0.

Step 0: Let $\tau \leftarrow 0$, initialize the master, solve it to obtain $\hat{\pi}^0 = \bar{\pi}^0$.

Step 1: Solve the master, obtain $\bar{\pi}^\tau$, $\hat{\pi}^{\tau+1} \leftarrow \hat{\pi}^\tau$.

Step 2: Let $k \leftarrow 0$, $\alpha \leftarrow \alpha^\tau$

Step 3: Solve the pricing problem and obtain $\bar{z}(\tilde{\pi}_k)$, where $\tilde{\pi}_k \leftarrow \alpha\hat{\pi}^\tau + (1-\alpha)\bar{\pi}^\tau$

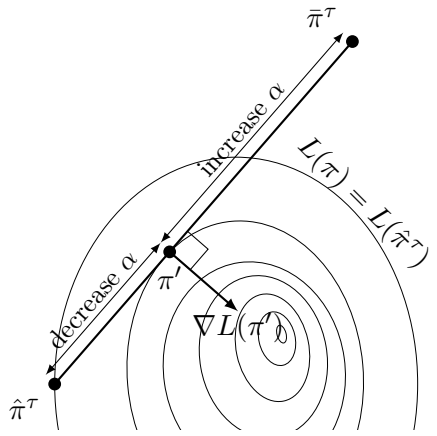
Step 4: If $L(\tilde{\pi}_k) > L(\hat{\pi}^{\tau+1})$, $\hat{\pi}^{\tau+1} \leftarrow \tilde{\pi}_k$

Step 5: If a mis-pricing occurs, then $k \leftarrow k+1$, $\alpha \leftarrow \max\{0, 1-k\cdot(1-\alpha^\tau)\}$, and go to Step 3.

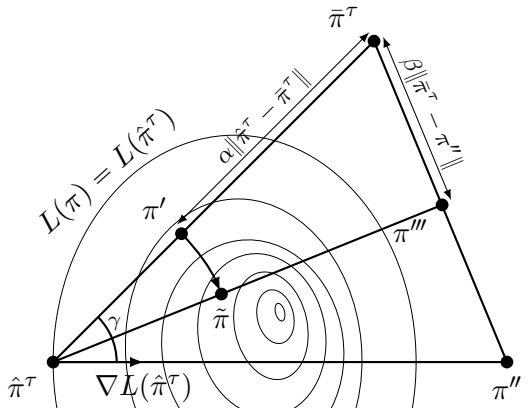
Step 6: If $\nabla L(\tilde{\pi}_0)(\bar{\pi}^\tau - \hat{\pi}^\tau) > 0$, $\alpha^{\tau+1} \leftarrow f_{\text{incr}}(\alpha^\tau)$; otherwise, $\alpha^{\tau+1} \leftarrow f_{\text{decr}}(\alpha^\tau)$.

Step 7: Let $\tau \leftarrow \tau + 1$ and goto Step 1.

Figure 2.4a offers an illustration of this procedure. The functions that we use for increasing and decreasing α are heuristic. We add to α 10% of the amount that is missing for it to reach the value one when increasing. Non-symmetrically, we



(a) Auto-adaptive α-schedule: condition for increasing or decreasing α^τ depending on whether $\tilde{\pi}$ is below or beyond the threshold value π' for which the angle is perpendicular



(b) Directional Smoothing: combining sub-gradient and Wentges smoothing

Figure 2.4: Illustrations for the dynamic α-schedule and directional smoothing

remove 0.1 from α when decreasing it. We define $f_{\text{incr}}(\alpha^\tau) = \alpha^\tau + (1 - \alpha^\tau) \cdot 0.1$, while $f_{\text{decr}}(\alpha^\tau) = \max\{0, \alpha^\tau - 0.1\}$. Such a choice for these functions comes from the following observation we had during preliminary computational experiments: α^τ should asymptotically approach value one when we need to increase stabilization, but we should be able to decrease α^τ sufficiently fast when strong stabilization is not needed anymore.

In an alternative procedure, we consider modifying the direction $(\bar{\pi}^\tau - \hat{\pi}^\tau)$ by twisting it towards the direction of ascent observed in $\tilde{\pi}$. The resulting method can be viewed as a hybridization of column generation with a sub-gradient method. When Wentges' rule (2.31) is used, the resulting hybrid method is related to the Volume algorithm by Barahona and Anbil (2000), where $\bar{\pi}^\tau$ is obtained by taking a step from the incumbent value $\hat{\pi}^\tau$ in a direction that combines previous iterate information with the current sub-gradient.

We call this hybrid procedure *directional smoothing*. In it, smoothed price vector $\tilde{\pi}$ is calculated as outlined below. It uses two parameters α and β.

Step 1: $\pi' = \hat{\pi}^\tau + (1 - \alpha)(\bar{\pi}^\tau - \hat{\pi}^\tau)$

Step 2: $\pi'' = \hat{\pi}^\tau + \frac{\nabla L(\hat{\pi}^\tau)}{\|\nabla L(\hat{\pi}^\tau)\|} \|\bar{\pi}^\tau - \hat{\pi}^\tau\|$

Step 3: $\pi''' = \beta\pi'' + (1 - \beta)\bar{\pi}^\tau$

Step 4: $\tilde{\pi}_i = \max\left\{0,\ \hat{\pi}_i^\tau + \frac{\|\pi' - \hat{\pi}^\tau\|}{\|\pi''' - \hat{\pi}^\tau\|} (\pi_i''' - \hat{\pi}_i^\tau)\right\}, \quad \forall 1 \leqslant i \leqslant m.$

Calculation of $\tilde{\pi}$ is illustrated in Figure 2.4b. In Step 1, π' is computed by applying smoothing. In Step 2, π'' is computed as the point located on the steepest ascent

direction at a distance from $\hat{\pi}^\tau$ equal to the distance to $\bar{\pi}^\tau$. In Step 3, a rotation is performed, defining target π''' as a convex combination between π'' and $\bar{\pi}^\tau$. Then, in Step 4, smoothed price vector is selected in direction $(\pi''' - \hat{\pi}^\tau)$ at the distance from $\hat{\pi}^\tau$ equal to $\|\pi' - \hat{\pi}^\tau\|$ and it is projected on the positive orthant. As in the case with non-directional smoothing, using modified dual prices can result in mis-pricing. When this occurs, we switch off directional smoothing by setting $\beta = 0$ in the next iterates of the mis-pricing sequence.

In the absence of mis-pricing, we fix the value of parameter β using a simple procedure that is based on our computational experiments that showed that the larger the angle γ between vectors $(\bar{\pi}^\tau - \hat{\pi}^\tau)$ and $(\pi'' - \hat{\pi}^\tau)$, the smaller the value for β should be. This is especially true at the end of column generation when angle γ is close to 90°, and large value β results in many mis-pricings. Based on this observation, our proposal is to use an adaptive β-schedule by setting $\beta = \cos\gamma$. As γ is always less than 90°, since vector $(\bar{\pi}^\tau - \hat{\pi}^\tau)$ is an ascent direction, $\beta \in (0, 1]$.

2.2.3 Combination with penalty function stabilization approaches

To avoid oscillation, one may attempt to remain in the proximity of the current incumbent dual solution $\hat{\pi}^\tau$ used as a *stability center*. This can be implemented by adding a penalty function to the dual objective to drive the optimization towards the stability center. Using the modified objective function, the restricted dual problem [DM$^\tau$] is replaced by

$$\pi^\tau = \underset{\pi \in \mathbb{R}^m_+}{\operatorname{argmax}} \{L^\tau(\pi) - \hat{S}(\pi)\}, \tag{2.33}$$

where the *penalty function* $\hat{S} : \pi \in \mathbb{R}^m_+ \rightarrow \mathbb{R}_+$ is typically convex, takes value zero at $\hat{\pi}$, and increases as $||\pi - \hat{\pi}^\tau||$ increases. The Bundle method (Briant et al., 2008) is a reference case of such proximal methods where value of $\hat{S}(\pi)$ depends quadratically on the distances from $\hat{\pi}^\tau$. One can also make use of a *piecewise linear penalty function* $\hat{S}$ (du Merle et al., 1999; Ben Amor et al., 2009) in order to ensure that the master problem is still a linear program (with additional artificial variables whose costs and bounds are chosen to model a piecewise linear stabilizing function).

Here we restrict our attention to 3-piece piecewise linear penalty functions that are further assumed to be symmetric. Our 3-piece function minimizes the $\mathcal{L}_1$ norm of the deviation with weight γ, once outside a confidence interval of size Δ. In other words, it penalizes each dual variable deviation $|\pi_i - \hat{\pi}^\tau_i|$ individually by a factor γ when it lies outside of the confidence interval $[\hat{\pi}^\tau_i - \Delta, \hat{\pi}^\tau_i + \Delta]$. Thus, the dual problem [DM] is modified by introducing, for each component $1 \leqslant i \leqslant m$, additional variables π^+_i and π^-_i which represent positive and negative deviation of π_i from the confidence interval:

$$\begin{aligned} \pi_i - \pi^+_i &\leqslant \hat{\pi}^\tau_i + \Delta, \quad 1 \leqslant i \leqslant m, \\ \pi_i + \pi^-_i &\geqslant \hat{\pi}^\tau_i - \Delta, \quad 1 \leqslant i \leqslant m. \end{aligned}$$

These constraints in [DM] correspond to new artificial variables ρ^+_i and ρ^-_i, $1 \leqslant i \leqslant m$, in the master problem [M]. In our paper (Pessoa et al., 2018c), we have also considered 5-piecewise linear penalty functions.

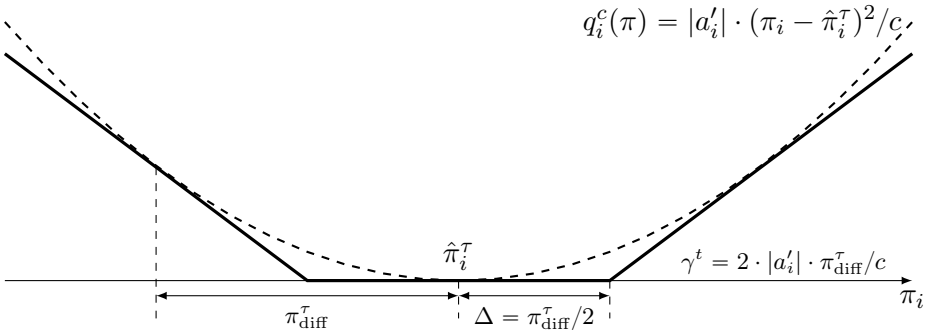


Figure 2.5: Curvature based penalty function in iteration τ

Now we present two schemes to update parameters γ and Δ. Both schemes rely on a single input parameter, which we denote κ. Thus, our proposal amounts to define self-adjusting γ and Δ-schedules. However, unlike for the parameterization of our smoothing scheme, we need here to select an input κ, that can be critical for the efficiency of the stabilization scheme. We say that a parameter is *critical* if its value affects the performance of the stabilization scheme by a "large margin" (say a factor ten in the number of iterations for instance) and the "best parameter value" differs for different problems. For all non-critical parameters, we used the same value for all problems tested in our computational experiments.

We developed a scheme for automated parameter adjustment that is "curvature-based": the idea is to approximate a quadratic function of the form: $q_i^c(\pi) = |a_i'| \cdot (\pi_i - \hat{\pi}_i^\tau)^2/c$, where c is the parameter which represents the function's curvature. Parameters γ and Δ are chosen in such a way that the resulting 3-piecewise linear penalty function is an outer approximation of $q_i^c(\pi)$ and tangent to it in 3 points, as pictured in Figure 2.5. Let π_{diff}^τ be the average component-wise difference between $\bar{\pi}^\tau$ and $\hat{\pi}^\tau$,

$$\pi_{\text{diff}}^\tau = \frac{\sum_{i=1}^m |\bar{\pi}_i^\tau - \hat{\pi}_i^\tau|}{m},$$

where π_{diff}^1 is the average component-wise difference between the dual solution on the first iteration and zero vector of the appropriate dimension. We set γ and Δ values at iteration τ as follows:

$$\Delta^\tau = \frac{\pi_{\text{diff}}^\tau}{2} \text{ and } \gamma^\tau = \frac{\pi_{\text{diff}}^\tau}{c}.$$

While the curvature c is set to $\pi_{\text{diff}}^1/\kappa$. So, κ is the only parameter of the stabilization scheme. If some artificial variables ρ_i^+ or ρ_i^- remain in the optimal solution to [M], the value of curvature c is increased, so as to reduce the weight of the penalty function in the master objective. It is multiplied by 10 in our procedure, but this factor is not critical.

The above "curvature-based" parameterization of the penalty function decreases the number of critical parameters from two to one. Its drawback however is to lead to modifying the restricted master objective coefficients and artificial variable bounds at each iteration, making re-optimization more computationally demanding. Moreover,

parameter κ remains very critical. A "small" value can make stabilization very weak, while a "large" value can make column generation converge to a solution which is "far away" from the true optimum of the unmodified master. Our attempts to come up with an auto-adjusting scheme for the curvature c (or equivalently for parameter κ) were not successful. Such an auto-adjusting scheme remains an interesting research goal for the future.

The alternative parameter adjustment scheme that we considered is an "explicit" setting of parameters γ and Δ. Our preliminary experiments on a range of different problems showed that a good setting of parameter γ is 0.9. Then, Δ is set to $\pi^1_{\text{diff}}/\kappa$. Again, κ is the only critical parameter to set. During the course of the column generation procedure, parameter Δ is divided by 2 each time all the artificial variables ρ_i^+ and ρ_i^- are absent from the solution of the restricted master. At the end of the column generation procedure, if there are artificial variables ρ_i^+ or ρ_i^- in the solution, the value of γ is decreased (reducing the impact of the penalty function in the objective allows the procedure to move to better solutions for the unmodified master): we divide it by 10. These two settings (division of Δ by 2 and division of γ by 10) are not critical.

Although, this "explicit" penalty function does not "stabilize as much" as the "curvature-based" penalty function, as shown by the comparison of the number of iterations in our experiment, it has the following advantage. Bounds on the artificial variables remain unchanged over the course of the algorithm (except if artificial variables remain in the optimal solution), thus the re-optimization of the restricted master is easier for the LP solver. This is beneficial for the overall running time.

2.2.4 Computational results

We have tested our stabilization approaches on 9 different problems: Generalized Assignment, Multi-Item Multi-Echelon Lot-Sizing, Multi-commodity Flow, Parallel Machine Scheduling, Capacitated Vehicle Routing, Multi-Activity Shift Scheduling, Bin Packing, Cutting Stock, and Vertex Coloring. Definition of problems and description of used instances are provided in our paper.

In first experiment, we assess numerically the stabilization effect of applying Wentges smoothing with static α-schedule versus our auto-adaptive schedule starting with $\alpha_0 = 0.5$. Additionally, we estimate the effect of using directional smoothing, with static and auto-adaptive value of parameter β, in combination with Wentges smoothing.

To tune the static α-value, we determine experimentally for each instance separately the best α (denoted *best*) among values in $\{0.1, 0.2, \ldots, 0.8, 0.9, 0.95\}$ as the value that yields the minimum total running time of the column generation procedure used to solve the master LP. Best value α depends highly on the application. Moreover, it differs a lot from one instance to the next even within the same application, as reflected by the range of best α reported in Table 2.3. Similarly, the best static β-value is determined by testing all β values in $\{0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$.

Table 2.3 gives an overview of column generation results with Wentges and directional smoothing. Five variants of column generation are compared: (i) that without any stabilization ($\alpha = 0$, $\beta = 0$), (ii) that with static Wentges stabilization

($\alpha = best$, $\beta = 0$), (iii) that with auto-adaptive Wentges stabilization ($\alpha = auto$, $\beta = 0$), (iv) that with combined automatic Wentges and static directional stabilization ($\alpha = auto$, $\beta = best$), and (v) that with combined automatic Wentges and directional stabilization ($\alpha = auto$, $\beta = auto$). The first column is the problem name; the second one reports the geometric mean of CPU time in seconds without stabilization; the third one gives the range of best α-values for each application; while the last four columns give the geometric means of the ratio of the solution time of variant (i) against variant (ii), (iii), (iv), and (v), respectively. The first observation is that our automatic Wentges smoothing scheme manages to reproduce the results of the smoothing with instance-wise tuned parameter for the first six problems. For the last three problems, it is not the case. For the Bin Packing and Cutting Stock problems, we conjecture that this is due to the very high number of identical pricing subproblems (at least 250). Because of this, the pricing problem can have a large number of mutually "perpendicular" optimal solutions that are generated on subsequent column generation iterations. This results in a sub-gradient that is perpendicular to the "in-out" direction; then the sub-gradient provides no indication on how to update parameter α. The explanation for the Vertex Coloring problem case is different: automatically adjusted dual prices make the pricing problem harder to solve. The second observation is that directional smoothing can improve the convergence significantly for some applications. Self-adjusting scheme for parameter β does not always reproduce results for the tuned static parameter. However, the advantage of the former is that it never deteriorates the performances, whereas static setting of β may cause that. An interesting observation is that automatic directional smoothing "corrects" the poorer performance of automatic Wentges smoothing when combined with it for the Bin Packing and Cutting Stock problems.

Application	Time of $\alpha, \beta = 0$	Range of best α	Ratio of solution time of variant $\alpha, \beta = 0$ to $\beta = 0$ and $\alpha = best$	Ratio of solution time of variant $\alpha, \beta = 0$ to $\beta = 0$ and $\alpha = auto$	Ratio of solution time of variant $\alpha, \beta = 0$ to $\alpha = auto$ and $\beta = best$	Ratio of solution time of variant $\alpha, \beta = 0$ to $\alpha = auto$ and $\beta = auto$
Generalized Assignment	75	$[0.3, 0.9]$	3.91	3.83	9.58	7.93
Lot Sizing	87	$[0.6, 0.95]$	3.41	4.09	4.46	5.99
Multi-Commodity Flow	914	$[0.1, 0.9]$	2.67	2.73	3.79	3.11
Machine Scheduling	34	$[0.7, 0.95]$	2.96	2.91	2.42	3.61
Vehicle Routing	5.5	$[0.2, 0.8]$	1.57	1.55	1.40	1.54
Shift Scheduling	4.7	$[0.7, 0.95]$	1.73	1.70	1.68	1.71
Bin Packing	4.1	$[0.2, 0.9]$	2.18	1.69	1.94	1.81
Cutting Stock	3.7	$[0.1, 0.95]$	1.16	0.98	1.43	1.32
Vertex Coloring	2.2	$[0.3, 0.8]$	1.22	1.01	(*)	(*)

Table 2.3: Overview of speedup factors for column generation with Wentges and directional smoothing; a (*) denotes cases where the time limit was reached

In the experiment reported here, we compare numerically automatic smoothing stabilization scheme, to a piecewise linear penalty function stabilization scheme, and to the combination of the two stabilization schemes. Five variants of column generation are compared: (i) that with automatic smoothing (`Smooth`); (ii) that with "curvature-based" 3-piecewise linear penalty function with tuned parameter κ (`Curv`); (iii) that with combination of the two previous stabilization methods (`SmoothCurv`);

	Smooth	Curv		SmoothCurv		Expl			SmoothExpl		
Application	T	κ	T	κ	T	pcs	κ	T	pcs	κ	T
Gen. Assignment	7.93	80	14.51	10	25.69	5	100	17.29	3	200	43.62
Lot Sizing	5.99	16	14.80	2	17.13	5	10	10.96	3	10	14.68
Multi-Com. Flow	3.11	5	4.17	1	5.29	3	20	5.29	3	4	7.28
Mach. Scheduling	3.61	4	1.77	2	4.19	5	10	2.20	3	10	5.74
Vehicle Routing	1.54	1.6	1.89	0.1	1.83	5	1	2.34	3	2	2.02
Shift Scheduling	1.71	25	0.73	25	1.33	3	0.2	1.06	3	0.1	1.73
Bin Packing	1.81	20	0.52	10	0.62	3	20	1.07	3	0.2	1.25
Cutting Stock	1.32	2	0.46	5	0.60	3	20	0.94	3	10	1.15

Table 2.4: Overview of results for column generation with smoothing and piecewise penalty function stabilization

(*iv*) that with "explicit" 3-piece and 5-piece linear penalty function with tuned parameter κ (`Expl`); and (*v*) that with the combination of automatic smoothing and "explicit" penalty function stabilization (`SmoothExpl`). We do not present results for the Vertex Coloring problem since the application of linear penalty function stabilization sometimes makes the pricing subproblem too hard to be solved in reasonable time. Where automatic smoothing is used, we mean the generic automated scheme that led to the best performance for the application concerned. As presented in the previous section, it is the combination of Wentges smoothing and directional smoothing for all applications, except for the Vehicle Routing and the Shift Scheduling problems where directional smoothing should not be applied. For each application and each of the last four variants of column generation, we determined experimentally the best value for parameter κ for which the geometric mean (among all instances of the problem) of the master LP solution time was the lowest.

Table 2.4 gives an overview of the comparative results for our 5 variants of column generation implementation: `Smooth`, `Curv`, `SmoothCurv`, `Expl`, `SmoothExpl`. For the last two variants, we report results for 3-piece or 5-piece penalty functions, depending on which variant gave the better results. Columns "pcs" indicate whether 3-piece of 5-piece function is used. Columns `T` in Table 2.4 give the geometric mean of the ratio of the solution time of the corresponding variant against the basic approach without any stabilization. The main observation here is that the combination of two stabilization techniques: smoothing and penalty function, outperforms any of the two techniques applied separately; this is the case for all problems except Vehicle Routing. Note also that the "explicit" variant of the penalty function stabilization performs better than "curvature-based" variant for all applications except Lot-Sizing. Results for the Shift Scheduling, Bin Packing, and Cutting Stock problems are different from others. For these three applications, the overall effect of stabilization by penalty functions does not decrease running times noticeably or may increase it, even though the number of iterations decreases. This is due to the increase of the number of (artificial) variables in the master that is critical in making the master harder to re-optimize.

From detailed results, which are not presented here, we observe that application of smoothing stabilization techniques deteriorates the solution time of standard column generation only for very few instances of the last three problems. Another important remark is that, for all applications except two, there are cases for which the speed-

up ratio of the best stabilization technique is one order of magnitude or better. For the Generalized Assignment problem, the maximum speed-up ratio is about 6500 reducing the solution time from more than 6 hours to less than 4 seconds for instance $D - 10 - 400$. This observation suggests that stabilization techniques should be considered systematically as a complement to basic Simplex-based column generation.

We would like to finish by the following remarks about our experience of using the stabilization techniques considered here inside Branch-Cut-and-Price algorithms. Although penalty function approaches have generally better performance than dual price smoothing, the former are harder to parameterize. The issue with parameterization gets worse when one starts to add cuts and to branch. This is because dual values corresponding to cuts and branching constraints have generally different magnitude than dual values corresponding to "core" constraints. As a result, we almost always employ only non-directional automatic dual price smoothing, which is activated by default in the BaPCod solver. Other stabilization techniques are used only in the case of severe convergence problems, and usually if column generation is used without cutting and without branching.

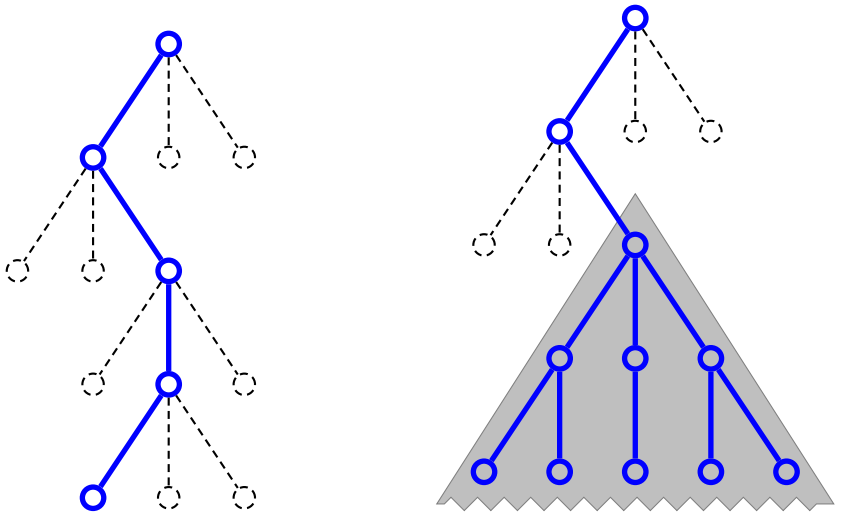
2.2.5 Related work

There are three families of column generation stabilization techniques according to Vanderbeck (2005). Techniques in the first family attempt to remin in the proximity of the current incumbent dual solution. Piecewise linear penalty functions were used by du Merle et al. (1999) and Ben Amor et al. (2009). The Bundle method uses (Briant et al., 2008) uses a quadratic penalty function. An alternative is to set proximality as a constraint giving rise a trust-region approach (Lemaréchal et al., 1995). Another alternative is to minimize the distance (according to a norm) subject to a parametric target constraint on the Lagrangian function value, leading to a level set approach (de Oliveira and Sagastizábal, 2014).

The second family groupes smoothing techniques addressed in Section 2.2.2. Two known examples are works by Wentges (1997) and Neame (2000). In our paper, we have shown that smoothing techniques are related to the in-out separation scheme of Ben-Ameur and Neto (2007) and Fischetti and Salvagnin (2010).

Techniques in the third family work with dual solutions in the interior of the dual feasible space. The analytic center cutting plane method by Goffin and Vial (2002) defines the next dual solution as the analytic center of the dual linear program augmented with an optimality cut. A more recent example of the centralization approach is the primal-dual column generation method proposed by Gondzio et al. (2013). Alternatives exist to implement centralization within a Simplex-based approach (Rousseau et al., 2007; Lee and Park, 2011).

One more way to stabilize column generation is to use dual-optimal inequalities (Ben Amor et al., 2006; Gschwind and Irnich, 2016). Column-and-row generation procedure presented in Section 2.1.1 and related approaches reviewed in Section 2.1.5 can also be viewed as ways to stabilize column generation. Another approach is to aggregate constraints in the master problem proposed by Elhallaoui et al. (2005). It reduces the dimension of the dual space and also accelerates convergence.



(a) Heuristic dive in an enumeration tree

(b) Exact solution of a Sub-MIP after a heuristic partial fixing of the solution

Figure 2.6: Illustration of two heuristic paradigms.

2.3 Primal Heuristics

This section is based on our short conference papers (Joncour et al., 2010; Pesneau et al., 2012; Sadykov et al., 2015b) and on the full journal paper (Sadykov et al., 2019).

Primal heuristics have become an essential component in mixed integer programming (MIP) solvers. Extending MIP based heuristics, our study outlines generic procedures to build primal solutions in the context of the branch-and-price approach and reports on their performance. Our heuristic decisions carry on variables of the Dantzig-Wolfe reformulation, the motivation being to take advantage of a tighter linear programming relaxation than that of the original compact formulation and to benefit from the combinatorial structure embedded in these variables. We focus on the so-called *diving* methods that use re-optimization after each LP rounding. We explore combinations with diversification-intensification paradigms such as *limited discrepancy search*, *sub-MIPing*, *local branching*, and *strong branching*. The dynamic generation of variables inherent to a column generation approach requires specific adaptation of heuristic paradigms. We manage to use simple strategies to get around these technical issues. Our numerical results on generalized assignment, cutting stock, and vertex coloring problems sets new benchmarks, highlighting the performance of diving heuristics as generic procedures in a column generation context and producing better solutions than state-of-the-art specialized heuristics in some cases.

2.3.1 Review of column generation based primal heuristics

Basic instruments of standard primal heuristic based on mathematical programming approaches are *rounding*, *diving* and *sub-MIPing*, along side selection rules such as *greedy* or *guided search*. A *rounding heuristic* consists in iteratively selecting a variable taking a fractional value in the solution of the LP relaxation and setting its lower bound to the rounded-up fractional value or its upper bound to the rounded-down value. The method is defined by a rule for selecting the variable and the rounding direction. Classical selection rules in this context are *least fractional*, *guided search*, *least pseudo-cost*, or *least infeasibility* among the rules reviewed in Berthold (2006).

Diving differs from simple rounding heuristic by the fact that the LP is re-optimized after bounding or fixing one or several variables. A diving heuristic can be understood as a heuristic search in an LP-based branch-and-bound tree: the search plunges deep into the enumeration tree by selecting a branch heuristically at each node, as illustrated in Figure 2.6a. The branching rule used in such context is typically quite different from the one used in an exact approach: in a diving heuristic, one is not concerned with balancing the tree, but one aims at finding good primal solutions quickly.

Sub-MIPing consists in restricting the MIP problem to a residual problem of smaller size by way of fixing (or bounding) some of its variable values; the restricted MIP is then solved exactly or approximately using a MIP solver. Thus, the method is essentially defined by a way of restricting the problem. For instance, one can restrict the problem by fixing some of its variables through a rounding or diving heuristic, as illustrated in Figure 2.6b; or by restricting the set of variables, implicitly fixing to zero variables of the complementary set.

Some of the heuristic paradigms that are mentioned in the literature can be seen as a specific implementation of *Sub-MIPing*: so are *Relaxation Induced Neighborhood Search* (RINS) (Danna et al., 2005), *local branching* (Fischetti and Lodi, 2003), and *crossover heuristic* (Berthold, 2006; Lübbecke and Puchert, 2012); others, like *feasibility pump* (Fischetti et al., 2005), reduce the set of solutions to be considered using soft-constraints (i.e., penalizing deviation).

Some primal heuristics have been routinely used in the column generation context. Beyond simple *greedy heuristics* (iterative greedy selection of a column into the partial solution) that are application specific, the most widely used heuristic is the so-called *restricted master heuristic*. This generic scheme can be seen as a sub-MIPing heuristic: the column generation formulation is restricted to a subset P' of variables λ_p, $p \in P'$, and it is solved as a static MIP. The restricted set P' is typically defined as the set of columns generated during the master LP solution. Hence, this method is also called *price-and-branch* (Desrosiers and Lübbecke, 2011). The main drawback of this approach is that the resulting restricted master integer problem may be infeasible in many cases, i.e., often the columns of P' may not be combined into an integer solution. In an application specific context, an ad-hoc recourse can be designed to "repair" infeasibility. Another typical implementation of such sub-MIPing heuristic is to define set P' using the columns of several constructive heuristic solutions, possibly augmented with the LP solution columns. This guarantees feasibility of a restricted master heuristic. The method can then be viewed as the search for

improving integer solutions in a neighborhood of the initial heuristic solutions (the neighborhood is defined by the columns set). However, the method often produces no better solutions than that of the initial heuristic solutions. Implementations of restricted master heuristics have been developed, for instance, for production planning, interval scheduling, network design, vehicle routing and delivery problems.

Close to being generic, but not quite, *rounding heuristics* procedures have been applied to the master LP solution. In the common case where the master formulation defines a set covering problem, a standard rounding strategy consists of 3 steps: (*i*) an initial partial solution is obtained by rounding downwards the master LP solution; (*ii*) then, columns LP values of which are closest to the next integer are then considered for round up while feasible; (*iii*) finally, an ad-hoc constructive procedure is used to complete the solution. *Local search* can be used to improve the solutions while implementing some form of diversification, but this is again typically application specific: one can remove some of the columns selected in the primal solution and reconstruct a solution with one of the above techniques. Rounding heuristics (sometimes coupled with local search) have been successfully applied on cutting stock, planning and vehicle routing problems. However, reaching feasibility remains a difficult issue that is handled in an application specific manner.

Diving heuristics are generic ways of "repairing" unfeasibilities. The residual master problem that remains after a rounding operation must be "cleaned up" before re-optimization, deleting all columns that could not be part of an integer solution to the residual problem (and hence would lead to an infeasibility if selected). Such preprocessing that is specific to a column generation context is presented in (Vanderbeck and Savelsbergh, 2006; Vanderbeck and Wolsey, 2010) : master constraints imply bounds on subproblem variables that lead to the definition of *proper columns*, i.e., columns that could take a non-zero integer value in an optimal solution to the residual master problem. We eliminate columns that become non-proper after rounding. This preprocessing is a key feature in diving heuristics. It helps to avoid the primal heuristic dead-ending with an infeasible solution. Furthermore, we focus on generating proper columns in future pricing: if the pricing problem solver can handle the bounded version of the column generation sub-problem, one can tighten lower and upper bounds on pricing problem variables to generate proper columns. Note that the re-optimization of the residual master might not necessarily be trivial and it can lead to generating new columns. This mechanism yields the "missing" complementary columns to build feasible solutions. If the residual master is however infeasible for a given partial solution, re-optimization can be a way to prove it early through a Simplex phase 1 and/or through preprocessing. Although it is an important feature for the success of the approach, re-optimization of the master LP after fixing can be time consuming. Tuning the level of approximation in this re-optimization allows one to control the computational effort. Diving heuristics were successfully used, for instance, on vehicle routing, inventory routing, crew rostering, bin packing, cutting stock, graph partitioning, machine scheduling, freight rail scheduling and lot-sizing problems.

Deriving general purpose primal heuristics based on the Dantzig-Wolfe reformulation raises some difficulties. Bounding a master variable in [M] or modifying its cost can be incompatible with the column generation approach in the sense that it

can induce modifications to the pricing problem that are not amenable to the pricing solver. Basically the issues are:

- Setting an upper bound on an existing column, as one might wish to do in diving heuristics, i.e., enforcing $\lambda_p \leqslant u_p$, yields an associated dual variable that must be accounted for in pricing (by modeling an extra cost for the specific solution $\bar{z}^p$). If constraint $\lambda_p \leqslant u_p$ are ignored when pricing, $\bar{z}^p$ might be wrongly regenerated as the best price solution. One could restrict the pricing problem to avoid regenerating $\bar{z}^p$, but this induces significant modifications to the pricing procedure that are as bad as accounting for the additional dual price.

- However, setting a lower bound on an existing column of the form $\lambda_p \geqslant l_p$ is trivial; this constraint can safely be ignored when pricing. Indeed, ignoring the dual price "reward" for generating this column, means that the pricing oracle overestimates its reduced cost and might not generate it; but the column needs not be generated since it is already in the master.

- Adding constraints involving individual variables λ_p as one wants to do for a local branching heuristic results in incompatibility issues of the same nature than adding upper bounds of the form $\lambda_p \leqslant u_p$.

- Increasing the cost of a variable λ_p in the objective function, as needed in the feasibility pump paradigm, means that λ_p will price out negatively according to the original pricing oracle, and hence it can be wrongly regenerated by the original oracle (which models the initial cost) as the best pricing problem solution.

- Decreasing the cost of a variable λ_g, however, is amenable to the unmodified column generation scheme, as using the original pricing oracle shall simply lead to overestimate the reduced cost of such already included column.

Thus, the only trivial operations in the master problem are lower bound setting and cost reductions for variables λ.

To simplify presentation, in next Sections 2.3.2-2.3.4 we suppose that in the master formulation (1.2) variables y are absent, and vector $\underline{W}$ of lower bounds is fixed to zero.

2.3.2 Pure diving

A pure *diving heuristic* is a simple depth-first heuristic search in the branch-and-price tree. At each tree node, a branch is heuristically selected based on a rounding strategy: it corresponds to rounding up or down a variable λ_p of the master LP solution. We denote this rounding as $\lceil \bar{\lambda}_p \rfloor$. To ensure compatibility with column generation, we translate such rounding operations into fixing a partial solution: rounding down variable λ_p means taking $\lfloor \lambda_p \rfloor$ copies of this column in the partial solution, while a round-up corresponds to taking $\lceil \lambda_p \rceil$ copies of this column. Thus, both round-up and round-down are implemented as setting a lower bound on the column use, and

the column remains in the formulation. After such rounding operation, the *residual master problem* and the pricing problem are modified by applying preprocessing techniques before re-optimization. The residual master problem is the master (1.2) in which the right-hand-size vector d and bound vector $\overline{W}$ are modified to take into account the current fixed partial solution. Then preprocessing changes bounds of variables x using updated vector d and then bounds of pricing problem variables z_t^k, $1 \leqslant t \leqslant n^k$, $k \in K$, are changed using mapping M between x and z.

A generic template of a pure diving procedure is given in Table 2.5 in a recursive form where τ is the iteration counter (it is different from the column generation iteration counter used in Section 2.2). The parameters of this procedure are elements: $P^{\tau-1}, d^{\tau-1}, \overline{W}^{\tau-1}$, which define the previous residual master [M$^{\tau-1}$]. Other parameters are the current partial solution, $\hat{\lambda}$, and the current rounding, $\tilde{\lambda}$, of the LP solution to the residual master [M$^{\tau-1}$]: $\tilde{\lambda}$ defines variables and their values which should be added to the current partial solution $\hat{\lambda}$. To start the diving procedure, we call PureDiving$(P^0, d, \overline{W}, 0, 0)$, where P^0 is the set of columns obtained while solving the linear relaxation of the initial master problem [M].

PureDiving$(P^{\tau-1}, d^{\tau-1}, \overline{W}^{\tau-1}, \hat{\lambda}, \tilde{\lambda})$

Step 1: Update the master: $d^\tau \leftarrow d^{\tau-1} - \sum_{p\in P} \Theta \bar{z}^p \tilde{\lambda}_p$, $\overline{W}^\tau \leftarrow \overline{W}^{\tau-1} - \sum_{p\in P^k} \tilde{\lambda}_p$. Update partial solution: $\hat{\lambda} \leftarrow \hat{\lambda} + \tilde{\lambda}$.

Step 2: Apply pre-processing to residual master and the associated pricing problem. Let P^τ denote the set of columns that remain proper. If residual master problem is shown infeasible, return.

Step 3: Solve the LP relaxation of the current residual master [M$^\tau$], let $\bar{\lambda}^\tau$ denote its LP solution. If the problem is shown infeasible through Phase I of the Simplex algorithm, return.

Step 4: Let $F = \{p \in P^\tau : \lfloor \bar{\lambda}_p^\tau \rfloor < \bar{\lambda}_p^\tau < \lceil \bar{\lambda}_p^\tau \rceil\}$. If $\hat{\lambda} + \{\bar{\lambda}_p^\tau\}_{p\in P^\tau \setminus F}$ defines a feasible primal solution to (1.1), record this solution.

Step 5: $\tilde{\lambda} \leftarrow 0$. If $F = \varnothing$, return.
Heuristically choose a column set $R \subseteq F$ and heuristically round their values: $\tilde{\lambda}_p \leftarrow \lceil \bar{\lambda}_p^\tau \rceil$ such that $\tilde{\lambda}_p > 0$ for $p \in R$.
Recursively call PureDiving$(P^\tau, d^\tau, \overline{W}^\tau, \hat{\lambda}, \tilde{\lambda})$.

Table 2.5: Pure diving heuristic

In *Step 2*, set P^τ denotes the set of columns that are suitable for the current residual master program:

$$P^\tau := \{p \in P^k : \; l_t^k \leqslant \bar{z}_t^p \leqslant u_t^k \quad \forall t = 1, \ldots, n^k, k \in K\} \tag{2.34}$$

where l_t^k and u_t^k are valid lower and upper bounds on pricing problem solution defining "proper" columns as in (Vanderbeck and Savelsbergh, 2006). Columns in

$P^{\tau-1}\backslash P^{\tau}$ are removed from residual master problem [M$^{\tau}$].

A central element of the procedure that drives the heuristic is *Step 5* in which we choose columns for rounding. Note that several columns can be taken into the solution simultaneously. After solving the current residual master LP, one selects in the partial solution one or several columns $p \in F = \{p \in P^{\tau} : \lfloor \bar{\lambda}_p^{\tau} \rfloor < \bar{\lambda}_p^{\tau} < \lceil \bar{\lambda}_p^{\tau} \rceil\}$, at heuristically set values $\lceil \bar{\lambda}_p^{\tau} \rceil$. One then checks whether the current partial solution defines a solution to the full-blown problem. The important feature of the diving procedure is preprocessing, master and pricing problems are updated to "proper" columns and the process reiterates while the residual master problem is not proven infeasible.

In our implementation of *Step 5*, we choose one column $p \in P$, whose value $\bar{\lambda}_p^{\tau}$ is closest to its nearest non-zero integer. Then, we round $\bar{\lambda}_p^{\tau}$ to its nearest non-zero integer. In our experiment, we noted that rounding one column at a time yields typically better results. This is probably because we make less macroscopic fixing and take the benefit of re-optimization and preprocessing of the modified master.

2.3.3 Diving with limited backtracking

Here, we consider a limited backtracking scheme. It relies on using the *Limited Discrepancy Search (LDS)* paradigm of Harvey and Ginsberg (1995). This original feature defines a scheme to diversify the search. It has a great impact in improving the performance of the pure diving method. We call this algorithm "Diving with LDS". Furthermore, we developed a specific implementation of a limited backtracking scheme for use when it is hard to find a feasible solution to the problem using pure diving. In this heuristic, that we call "Diving for Feasibility", backtracking is used to intensify the search towards the leaves of the search tree, exploring the neighborhood of the first dive solution, until a feasible solution is identified.

DivingWithLDS($P^{\tau-1}, d^{\tau-1}, \overline{W}^{\tau-1}, \hat{\lambda}, \tilde{\lambda}, T, h$)

Steps 1–4 are the same as in PureDiving

Step 5: **Repeat** the following

1. $\tilde{\lambda} \leftarrow 0$. If $F\backslash T = \varnothing$, return.
2. Heuristically choose a column set $R \subseteq F\backslash T$, and heuristically round their values: $\tilde{\lambda}_p \leftarrow \lceil \bar{\lambda}_p^{\tau} \rceil$ such that $\tilde{\lambda}_p > 0$ for $p \in R$.
3. Recursively call DivingWithLDS($P^{\tau}, d^{\tau}, \overline{W}^{\tau}, \hat{\lambda}, \tilde{\lambda}, T, h+1$).
4. $T \leftarrow T \cup R$.

While $|T| \leqslant$ *maxDiscrepancy* **and** $h \leqslant$ *maxDepth*.

Table 2.6: Diving heuristic with limited backtracking

The scheme is applied from the root node of the search tree. Backtracking is performed up to depth *maxDepth*. When a backtrack is performed back to a node,

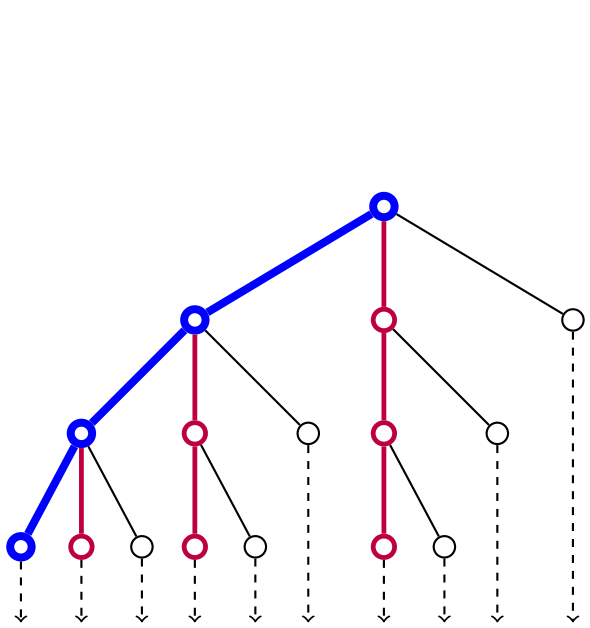
the backtracked branching decision is forbidden for other branches from that node and in the subtrees "rooted" at these branches; i.e., the column that was selected for rounding in the backtracked branch cannot be selected as a candidate for rounding in other branches. This is implemented by using a tabu list that includes forbidden branching decisions: the tabu list is a set of columns, denoted T, that are forbidden to be chosen for rounding. Backtracking branches are considered as long as the number of columns in the tabu list does not exceed *maxDiscrepancy* which is the second parameter of the scheme. The template for the diving heuristic with limited backtracking (named "diving with LDS") in a recursive form is presented in Table 2.6. This procedure is derived from the pure diving procedure with a modified *Step 5* and two additional parameters: the tabu list T and the current search tree depth h. To start the diving heuristic with limited backtracking, we call DivingWithLDS($P^0, d, \overline{W}, 0, 0, \varnothing, 1$).

Our implementation of *Step 5* of the diving heuristic with limited backtracking is similar to the one for the pure diving heuristic. We choose a column $p \in F \backslash T$, whose value $\bar{\lambda}_p^\tau$ is closest to its nearest non-zero integer, and we round $\bar{\lambda}_p^\tau$ to its nearest non-zero integer. Thus, our set R has cardinality one and our tabu set increases by one unit at the time. An example of the search tree for the diving heuristic with limited backtracking is depicted in Figure 2.7a. In this example, the instantiation of the parameters is *maxDepth* $= 3$, *maxDiscrepancy* $= 2$, and, as in our implementation, we round one column at a time. In this figure, the boldest branches have an empty tabu list, less bold branches have tabu list with one column, and thin branches have the tabu list with two columns.

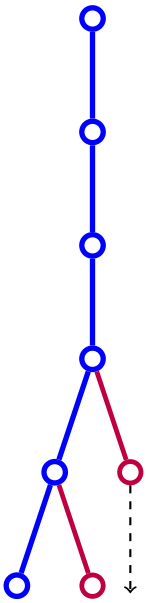
When the main goal is to find a feasible solution, we consider using backtracking toward the leaves of the depth-first-search dive in the branch-and-price tree. This algorithm that we call "Diving for Feasibility", entails a specific parametrization of the above DivingWithLDS procedure: *maxDepth* $= \infty$; additionally, the procedure stops as soon as a feasible solution is found at *Step 4* of PureDiving. Observe that given that we use a depth-first-search strategy, the backtracking will take place towards the leaves, by reconsidering in priority the last variable fixings. As the procedure stops as soon as a feasible solution is found, it is unlikely that the search backtracks up to the root. In our implementation we set *maxDiscrepancy* $= 1$, the resulting search scheme is illustrated in Figure 2.7b. This heuristic tends to yield a feasible solution in very small additional time compared to pure diving heuristic when the latter cannot find one.

2.3.4 Strong diving

Strong branching (Linderoth and Savelsbergh, 1999) is a strategy to make "intelligent" look-ahead branching decisions in an effort to reduce the size of the branch-and-bound tree when solving a mixed integer program. The idea is to select among several possible branching decisions by comparing the impact they have on improving the dual bound. For this, one performs a (possibly approximate) evaluation of the dual bound of the child nodes, i.e., one solves (approximately) the linear programming relaxation of the child nodes with the branching decision temporarily applied. Then, a branching decision is chosen which generates the best improvement in the



(a) "Diving with LDS" example for *maxDepth* = 3 and *maxDiscrepancy* = 2;



(b) "Diving for Feasibility" example for *maxDiscrepancy* = 1, assuming a feasible solution was found after 2 back-tracks

Figure 2.7: Illustrations for Diving with LDS and Diving for Feasibility

local dual bound computed from the child node's (approximate) dual bounds.

Our use of the strong branching paradigm in the diving procedure goes as follows. We choose several columns as candidates for rounding. The number of candidates is limited by the parameter *maxCandidates*. Then, for each candidate, we apply temporarily the rounding and compute the resulting dual bound for the associated residual master problem (making use of the column generation procedure). Contrary to the "classic" strong branching applied in branch-and-bound, the best candidate is that which generates the child node with the smallest change of the dual bound. Indeed, the heuristic explores a single branch in search for the best possible primal bounds. This is different from the aim of getting the largest dual bound improvement that is pursued in a "classic" branch-and-bound procedure.

The template for the strong diving heuristic in a recursive form is presented in Table 2.7. Our procedure is combined with the paradigm of the diving heuristic with limited backtracking. In *Step 2* and *Step 3*, we choose a candidate set of columns, and then each candidate is evaluated. *Step 5* of the DivingWithLDS is modified into *Step 4* in StrongDiving: for the next diving decision, we choose a column which rounding results in the smallest increase of the dual bound of the master problem linear relaxation. In our implementation, we define C to be the columns with values closest to their non-zero integers; their number is bounded by *maxCandidates*. In *Step 3*, we round λ_p to its nearest non-zero integer. To start the strong diving

StrongDiving($P^\tau, d^\tau, \overline{W}^\tau, \hat{\lambda}, T, h$)

Step 1: If $h = 1$, run Steps 2–4 of PureDiving with $\hat{\lambda} = 0$.

Step 2: Heuristically choose a column set $C \subseteq F \backslash T$, such that $|C| = \min\{maxCandidates, |F \backslash T|\}$.

Step 3: **For each** candidate $p \in C$

1. $\tilde{\lambda} \leftarrow 0$. Heuristically round λ_p: $\tilde{\lambda}_p \leftarrow \lceil \bar{\lambda}^\tau_p \rfloor$ such that $\tilde{\lambda}_p > 0$.
2. Call Steps 1–4 of PureDiving($P^\tau, d^\tau, \overline{W}^\tau, \hat{\lambda}, \tilde{\lambda}$). At the end of this call, save the following: $P^{\tau,p} \leftarrow P^\tau$, $d^{\tau,p} \leftarrow d^\tau$, $\overline{W}^{\tau,p} \leftarrow \overline{W}^\tau$, $\hat{\lambda}^p \leftarrow \hat{\lambda} + \tilde{\lambda}$. Also, let db^p be the value of the LP relaxation of $[\mathrm{M}^\tau]$ in Step 3 of this call.

Step 4: **Repeat** the following

1. Choose $p \in C$ with the smallest value db^p.
2. Recursively call StrongDiving($P^{\tau,p}, d^{\tau,p}, \overline{W}^{\tau,p}, \hat{\lambda}^p, T, h+1$).
3. $T \leftarrow T \cup \{p\}$, $C \leftarrow C \backslash \{p\}$.

While $|T| \leqslant maxDiscrepancy$ **and** $h \leqslant maxDepth$.

Table 2.7: Strong diving heuristic

heuristic, we call StrongDiving($P^0, d, \overline{W}, 0, \varnothing, 1$).

2.3.5 Other variants

Diving with restarts fixes a part of the solution to what was the incumbent solution and restarts the pure diving heuristic for the residual master problem. Diving combines with sub-MIPing consists in limiting the set of variables in the master to the subset of columns that were generated during the diving for feasibility heuristic. Our local branching heuristic is similar to the previous one, except that we add a constraint in the master to limit the deviation from the current incumbent solution.

The feasibility pump heuristic can be partially adapted to the column generation context. As indicated above, the cost of a master variable λ_p in the objective function can be decreased without an impact on the pricing problem. However, increasing the cost above the true cost of a variable is not possible. Therefore, changing the cost of master variables as required by the feasibility pump paradigm is possible, but with the limitation not to exceed the true cost.

These paradigms are described in our papers (Pesneau et al., 2012; Sadykov et al., 2019). We do not provide details here, as our experimental research revealed that these variants are not competitive with the variants described above.

2.3.6 Computational results

We have tested our heuristics on three applications: Generalized Assignment, Cutting Stock, and Vertex Coloring. Description of the problem, used test instances, and column generation parameters are given in our paper (Sadykov et al., 2019). The specific heuristic procedures that we compared are:

1. Pure Diving.
2. Diving for Feasibility with parameter $maxDiscrepancy = 1$.
3. Diving with LDS with parameters $maxDiscrepancy = 3$, $maxDepth = 2$.
4. Strong Diving with parameters $maxDiscrepancy = 3$, $maxDepth = 2$, $maxCandidates = 10$.
5. A pure Restricted Master.

Our numerical results are presented in Table 2.8: "Time" is the average running time of the heuristic in seconds; "Found" is the percentage of instances for which a feasible solution was found by the heuristic, "Opt" is the percentage of instances for which an optimal solution was found by the heuristic, and "Gap" is the average relative gap between the solution value found by the heuristic and the optimal solution value. The running time includes both the time to initially solve the master LP and the time to perform the heuristic diving. The gap is relative for generalized assignment and absolute for bin packing and vertex coloring.

	Generalized Assignment			Bin Packing			Vertex Coloring		
Heuristic	Time	Found	Gap	Time	Opt	Gap	Time	Opt	Gap
Restricted Master	26.50	55%	11.00%	224.37	5%	1.22	3.94	49%	0.54
Pure Diving	0.80	70%	0.37%	13.71	46%	0.54	0.94	71%	0.29
Diving for Feasib.	0.81	100%	0.39%	13.71	46%	0.54	0.94	71%	0.29
Diving with LDS	4.21	100%	0.10%	27.44	89%	0.11	1.38	88%	0.12
Strong Diving	33.45	100%	0.05%	67.42	90%	0.10	3.65	94%	0.06

Table 2.8: Computational comparison of heuristic variants on instances of three problems

In Table 2.8, the worst performance is clearly that of the restricted master heuristic: it found the least number of feasible solutions, and where a solution was obtained, the average gap is much larger than for other heuristics. The fastest heuristic is pure diving, but it does not always find a feasible solution. Diving for feasibility manages to "correct" this drawback of the pure diving heuristic with almost no additional time. Diving with restarts and local branching heuristics improve over diving for feasibility but require more running time. Diving with LDS improves significantly the quality of obtained solution but requires even more running time. The best solution quality is obtained by the strong diving heuristic, but it consumes an order of magnitude more time than the previous heuristic. Overall, we conclude that the diving with LDS heuristic offers the best tradeoff between quality and running time.

However, if one needs a fast heuristic, diving for feasibility can provide high quality solutions in small time.

We have also performed a computational comparison of our diving heuristic with LDS and the best heuristic available in the literature by Yagiura et al. (2006) for the generalized assignment problem. On the set of classic instances of types C, D, and E containing up to 1600 tasks and 80 agents, our heuristic was faster and obtained 0.026% average optimality gap and 4.1 average absolute gap. The heuristic by Yagiura et al. (2006) obtained 0.042% average optimality gap and 9.6 average absolute gap.

According to Posta et al. (2012), there were ten open classic instances. We have tried to improve the best known solutions for these instances. For this, we did seven runs of the diving heuristic with LDS. For each run, we set different parameters *maxDiscrepancy*, *maxDepth* (always ensuring that the total number of dives is equal to ten), different maximum number of columns in the master (master clean-up), and different parameter κ for the penalty function stabilization (see Section 2.2.3). It was already mentioned above that the master LP usually admits many alternative solutions. Therefore, a change of the master clean-up and stabilization parameters typically change the master LP solution obtained by column generation. Thus, the diving heuristics explore different parts in the search tree.

	Best known		Best run (instance specific)			Average (7 runs)	
Instance	Bound	Solution	Solution	Time	Red. gap	Time	Red. gap
D-20-200	12235	12244	12238	18	66%	22	3%
D-20-400	24563	24585	24567	82	82%	82	56%
D-40-400	24350	24417	24356	134	89%	145	72%
D-15-900	55404	55414	**54404**	80	100%	179	43%
D-30-900	54834	54868	54838	529	88%	505	61%
D-60-900	54551	54606	54554	1445	95%	1490	83%
D-20-1600	97824	97837	97825	744	92%	665	69%
D-40-1600	97105	97113	**97105**	3158	100%	7314	38%
D-80-1600	97034	97052	97035	10852	94%	10856	-48%
C-80-1600	16284	16289	16285	2186	80%	2572	80%

Table 2.9: Improved best known solutions for open instances of the generalized assignment problem.

In Table 2.9, for each instance, from left to right we report the instance name, the best known lower bound and solution value reported by Posta et al. (2012), the best solution value obtained by us, the time in seconds taken to obtain this value, the reduction in gap between the bound and the best known solution obtained by us, the average solution obtained over all seven runs, and the average running time. We have managed to improve the best known solutions for all ten open instances. Moreover, the average reduction of the gap between the best known solution and the best known lower bound is 89%. For two instances D-15-900 and D-40-1600, optimal solutions were found for the first time. The average solution value obtained over the seven runs is smaller than the previously best known solution for all instances except

one.

We would like to underline that such good results for open Generalized Assignment instances would not be possible if we did not use stabilization techniques discussed in Section 2.2. In fact, for these instances one need to combine and use all three techniques: automatic Wentges smoothing, automatic directional smoothing, and penalty function stabilization. For instances like D-20-1600, which has 80 items per machine and 1600 items in total, it is simply impossible to perform non-stabilized column generation until convergence (it would take weeks or even months of calculation). Using triple stabilization it is however possible in about 10 minutes not only to solve the LP relaxation of the set partitioning formulation to optimality but also to perform 10 dives in the diving heuristic with LDS.

2.4 Generic BCP for Vehicle Routing and Related Problems

This section is based on our long conference paper (Pessoa et al., 2019a) and on the full paper (Pessoa et al., 2019b) submitted to a journal.

Major advances were recently obtained in the exact solution of Vehicle Routing Problems (VRPs). Sophisticated Branch-Cut-and-Price (BCP) algorithms for some of the most classical VRP variants now solve many instances with up to a few hundreds of customers. However, adapting and reimplementing those successful algorithms for other variants can be a very demanding task. This work proposes a BCP solver for a generic model that encompasses a wide class of VRPs. It incorporates the key elements found in the best existing VRP algorithms: ng-path relaxation, rank-1 cuts with limited memory, path enumeration, and rounded capacity cuts; all generalized through the new concept of "packing set". This concept is also used to derive a branch rule based on accumulated resource consumption and to generalize the Ryan and Foster branch rule. Extensive experiments on several variants show that the generic solver has an excellent overall performance, in many problems being better than the best specific algorithms. Even some non-VRPs, like bin packing, vector packing and generalized assignment, can be modeled and effectively solved.

The solver is now available at `vrpsolver.math.u-bordeaux.fr` for download and for free academic use. The optimization algorithms and the Julia–JuMP user interface (Dunning et al., 2017) were released in a pre-compiled docker image. The demos are available for the Capacitated Vehicle Routing Problem, Vehicle Routing Problem with Time Windows, Heterogeneous Fleet Vehicle Routing Problem, Pickup and Delivery Problem with Time Windows, Team Orienteering Problem, Generalized Assignment Problem, Capacitated Arc Routing Problem, and Two-Echelon Capacitated Vehicle Routing Problem.

2.4.1 Exact algorithms for vehicle routing problems

Since its introduction by Dantzig and Ramser (1959), the Vehicle Routing Problem (VRP) has been one of the most widely studied in combinatorial optimization. Google Scholar indicates that 823 works containing the exact string "vehicle routing"

in the title were published only in 2018. VRP relevance stems from its direct use in the real systems that distribute goods and provide services, vital to the modern economies. Reflecting the large variety of conditions in those systems, the VRP literature is spread into dozens, perhaps hundreds, of variants. For example, there are variants that consider capacities, time windows, heterogeneous fleets, multiple depots, pickups and deliveries, optional customer visits, arc routing, etc.

In recent years, big advances in the exact solution of VRPs had been accomplished. A milestone was certainly the Branch-Cut-and-Price (BCP) algorithm of Pecin et al. (2017b), that could solve Capacitated VRP (CVRP) instances with up to 360 customers, a large improvement upon the previous record of 150 customers. That algorithm exploits many elements introduced by several authors, combining and enhancing them. In particular, the new concept of *limited memory cut* proved to be pivotal. Improvements of the same magnitude were later obtained for a number of classical variants like VRP with Time Windows (VRPTW) (Pecin et al., 2017a), Heterogeneous Fleet VRP (HFVRP) and Multi Depot VRP (MDVRP) (Pessoa et al., 2018b), and Capacitated Arc Routing (CARP) (Pecin and Uchoa, 2019). For all those variants, instances with about 200 customers are now likely to be solved, perhaps in hours or even days. However, there is something even more interesting: many instances with about 100 customers, that a few years ago would take hours, are solved in less than 1 minute. This means that many more real world instances can now be tackled by exact algorithms in reasonable times.

Unhappily, designing and coding each one of those complex and sophisticated BCPs has been a highly demanding task, measured on several work-months of a skilled team. In fact, this prevents the use of those algorithms in real world problems, that actually, seldom correspond exactly to one of the most classical variants. This work presents a framework that can handle most VRP variants found in the literature and can be used to model and solve many other new variants. In order to obtain state-of-the-art BCP performance, some key elements found in the best specific VRP algorithms had to be generalized. The new concept of *packing set* was instrumental for that.

The quest for general exact VRP algorithms can be traced back to Balinski and Quandt (1964), where a set partitioning formulation valid for many variants was proposed. That formulation had only turned practical in the 1980's and 1990's, when the Branch-and-Price (BP) method was developed. At that time, it was recognized that the pricing subproblems could often be modeled as Resource Constrained Shortest Path (RCSP) problems and solved by labeling algorithms, leading to quite generic methods (for example, Desaulniers et al. (1998)). However, those BP algorithms only worked well on problems with "tightly constrained" routes, like VRPTW with narrow time windows. Many variants, including CVRP, were much better handled by Branch-and-Cut (BC) algorithms using problem-specific cuts. In the late 2000's decade, after the work by Fukasawa et al. (2006), it became clear that the combination of cut and column generation performs better than pure BP or pure BC on almost all problems. Until today, BCP remains the dominant VRP approach. A first attempt of a generic BCP was presented in Baldacci and Mingozzi (2009), where 7 variants, all of them particular cases of the HFVRP, could be solved. Recently, in (Sadykov et al., 2017) we proposed a BCP for several particular cases of

the HFVRP with time windows. The framework we propose now is far more generic than that.

2.4.2 The model

Define directed graphs $G^k = (V^k, A^k)$, $k \in K$. Let $V = \bigcup_{k \in K} V^k$ and $A = \bigcup_{k \in K} A^k$. The graphs are not necessarily simple and may even have loops. Vertices and arcs in all graphs are distinct. Each graph has special source and sink vertices: v^k_{source} and v^k_{sink}. The source and sink may be distinct vertices, but may also be the same vertex. Define set R^k of resources. For each $r \in R^k$ and $a \in A^k$, $q_{a,r} \in \mathbb{R}$ is the consumption of resource r in arc a. Resources without any negative consumption are called *monotone*, otherwise they are *non-monotone*. Set R^k is divided into *main resources* R^k_M and *secondary resources* R^k_N. Main resources should be monotone. Moreover, there should not exist a cycle in G^k with zero consumption of all main resources. Therefore, unless G^k is acyclic, it is mandatory to have at least one main resource. The concept of main resource is directly related to key implementation issues. Secondary resources may be monotone or non-monotone. Finally, resources are also classified as *disposable* or *non-disposable*. By default, resources are assumed to be disposable. There are finite accumulated resource consumption intervals $[l_{a,r}, u_{a,r}]$, $a \in A^k$. Since in most applications these intervals are more naturally defined on vertices, we may define intervals $[l_{v,r}, u_{v,r}]$, $v \in V^k$, meaning that $[l_{a,r}, u_{a,r}] = [l_{v,r}, u_{v,r}]$ for every arc $a \in \delta^-(v)$ (i.e., entering v). A resource constrained path $p = (v^k_{\text{source}} = v_0, a_1, v_1, \ldots, a_{n-1}, v_{n-1}, a_n, v_n = v^k_{\text{sink}})$ over a graph G^k should have $n \geqslant 1$ arcs, $v_j \neq v^k_{\text{source}}$ and $v_j \neq v^k_{\text{sink}}$, $1 \leqslant j \leqslant n-1$, and is feasible if:

- for every $r \in R^k$ that is disposable, the accumulated resource consumption $S_{j,r}$ at visit j, $0 \leqslant j \leqslant n$, where $S_{0,r} = 0$ and $S_{j,r} = \max\{l_{a_j,r}, S_{j-1,r} + q_{a_j,r}\}$, does not exceed $u_{a_j,r}$;

- for every $r \in R^k$ that is non-disposable, the accumulated resource consumption $S_{j,r}$ at visit j, $0 \leqslant j \leqslant n$, where $S_{0,r} = 0$ and $S_{j,r} = S_{j-1,r} + q_{a_j,r}$, lies in the interval $[l_{a_j,r}, u_{a_j,r}]$.

Some feasible paths may not be elementary, some vertices or arcs being visited more than once. For each $k \in K$, let P^k denote the set of all feasible resource constrained paths in G^k. Each set P^k is finite, either because G^k is acyclic or because the main resources limit the number of times that each vertex or arc can be visited. Define $P = \bigcup_{k \in K} P^k$. As vertices and arcs in different graphs are distinct, paths in different graphs are also distinct.

For all $a \in A^k$ and $p \in P^k$, let $\bar{z}^p_a$ indicate how many times arc a appears in path p. The problem should be formulated exactly as stated in (1.1). Equations (1.1a) and (1.1b) define a general objective function and m general constraints over those variables, respectively. Constraints (1.1b) may even contain exponentially large families of cuts, provided that suitable procedures are given for their separation. However, by simplicity, we continue the presentation as if all the m constraints are explicitly defined. As stated in Section 1.1, for each variable x_j, $1 \leqslant j \leqslant n_1$,

$M(x_j) \subseteq A$ defines its *mapping* into a non-empty subset of the arcs. We slightly abuse notation here by defining mapping directly to arcs, and not to arc variables z_a^k. For each $k \in K$, $\underline{W}^k$ and $\overline{W}^k$ are given lower and upper bounds on the number of paths from G^k in a solution. The LP relaxation (1.2) of formulation (1.1) is solved by column generation as explained in Section 1.1.

2.4.3 Packing sets for generalizing state-of-the-art elements

Formulation (1.1) can be used to model most VRP variants and also some other non-VRP applications. It can be solved by a standard BP algorithm (or a standard robust BCP algorithm (Pessoa et al., 2008), if (1.1b) contains separated constraints), where the RCSP subproblems are handled by a labeling dynamic programming algorithm. However, its performance on the more classic VRP variants would be very poor when compared to the best existing specific algorithms. One of the main contributions of this work is a generalization of the key additional concepts found in those state-of-the-art algorithms, leading to the construction of a powerful and still quite generic BCP algorithm.

In order to do that, we introduce a new concept. Let $\mathcal{B} \subset 2^A$ be a collection of mutually disjoint subsets of A such that the constraints:

$$\sum_{p \in P} \left(\sum_{a \in B} \bar{z}_a^p \right) \lambda_p \leqslant 1, \quad B \in \mathcal{B}, \tag{2.35}$$

are satisfied by at least one optimal solution (x^*, y^*, λ^*) of formulation (1.1). In those conditions, we say that each element of $\mathcal{B}$ is a *packing set*. Note that a packing set can contain arcs from different graphs and not all arcs in A need to belong to some packing set. The definition of a proper collection $\mathcal{B}$ is application specific and part of the modeling task. It does not follow automatically from the analysis of formulation (1.1).

In many applications the packing sets are more naturally defined on vertices, so we also provide that modeling alternative. Let coefficient $\bar{z}_v^p$ indicate how many times vertex v appears in a path p. Let $\mathcal{B}^{\mathcal{V}} \subset 2^V$ be a collection of mutually disjoint subsets of V such that the constraints:

$$\sum_{p \in P} \left(\sum_{v \in B} \bar{z}_v^p \right) \lambda_p \leqslant 1, \quad B \in \mathcal{B}^{\mathcal{V}}, \tag{2.36}$$

are satisfied by at least one optimal solution (x^*, y^*, λ^*) of formulation (1.1). In those conditions, we say that the elements of $\mathcal{B}^{\mathcal{V}}$ are *packing sets on vertices*. Actually, in some symmetric problems there is a computational advantage in defining packing sets on vertices.

The following concepts — *ng*-paths, Limited Memory Rank-1 Cuts, path enumeration, accumulated consumption branching, and rounded capacity cuts — were originally proposed and used on the most classical VRP variants, often CVRP and VRPTW. In our proposed generalization, those problems will correspond to simple models where the packing sets in $\mathcal{B}^{\mathcal{V}}$ are the singletons formed by each customer vertex.

ng-paths

When modeling classical VRPs, one of the weaknesses of linear relaxation (1.2) is often the existence of non-elementary paths in P that can not be part of any integer solution. In those cases, one would like to eliminate all those paths from the definition of P. However, this would make the pricing subproblems much harder, to the point of becoming intractable in many cases. A good compromise between the linear relaxation strength and pricing difficulty can be obtained by the so-called ng-paths, introduced in Baldacci et al. (2011b).

In our more general context, we say that a path is $\mathcal{B}$-*elementary* if it does not use more than one arc in the same packing set of $\mathcal{B}$. Let P^k_{elem} be the subset of the paths in P^k that are $\mathcal{B}$-elementary, $P_{elem} = \bigcup_{k \in K} P^k_{\text{elem}}$.

Ideally, we would like to price only $\mathcal{B}$-elementary paths. Instead, we settle for generalized $\mathcal{B}$-ng-paths defined as follows. For each arc $a \in A$, let $NG(a) \subseteq \mathcal{B}$ denote the ng-set of a. A $\mathcal{B}$-ng-path may use two arcs belonging to the same packing set B, but only if the subpath between those two arcs passes by an arc a such that $B \notin NG(a)$. The ng-sets may be determined a priori; but also dynamically, like in Roberti and Mingozzi (2014) and Bulhoes et al. (2018d).

If the packing sets are being defined on vertices, there is the similar concept of $\mathcal{B}^{\mathcal{V}}$-elementary path: a path that does not use more than one vertex in the same packing set of $\mathcal{B}^{\mathcal{V}}$. We also denote by P^k_{elem} the subset of the paths in P^k that are $\mathcal{B}^{\mathcal{V}}$-elementary. In this context, for each vertex $v \in V$, let $NG(v) \subseteq \mathcal{B}^{\mathcal{V}}$ be the ng-set of v. A $\mathcal{B}^{\mathcal{V}}$-$ng$-path may use two vertices belonging to the same packing set B, but only if the subpath between those two vertices passes by a vertex v such that $B \notin NG(v)$.

When $\mathcal{B}$ or $\mathcal{B}^{\mathcal{V}}$ are clear from the context, we may still refer to $\mathcal{B}$-ng-paths or $\mathcal{B}^{\mathcal{V}}$-$ng$-paths simply as ng-paths.

Limited Memory Rank-1 Cuts

The Rank-1 Cuts (R1Cs) (Petersen et al., 2008; Bulhoes et al., 2018b) are a generalization of the Subset Row Cuts proposed by Jepsen et al. (2008). Here, they are further generalized as follows. Consider a collection of packing sets $\mathcal{B}$. A Chvátal-Gomory rounding of constraints (2.35), using a non-negative multiplier ρ_B for each $B \in \mathcal{B}$, yields:

$$\sum_{p \in P} \left\lfloor \sum_{B \in \mathcal{B}} \rho_B \sum_{a \in B} \bar{z}^p_a \right\rfloor \lambda_p \leqslant \left\lfloor \sum_{B \in \mathcal{B}} \rho_B \right\rfloor . \tag{2.37}$$

Those R1Cs are potentially very strong, but each added cut makes the pricing subproblems significantly harder.

The limited memory technique Pecin et al. (2017c) is essential for mitigating that negative impact. In this technique, a R1C, characterized by its vector of multipliers ρ, is associated to a *memory arc-set* $A(\rho) \subseteq A$. The limited-memory R1C (lm-R1C) is defined as:

$$\sum_{p \in P} \alpha(\rho, A(\rho), p) \lambda_p \leqslant \left\lfloor \sum_{B \in \mathcal{B}} \rho_B \right\rfloor , \tag{2.38}$$

where the coefficient $\alpha(\rho, A(\rho), p)$ is computed as in the pseudo-code that describes Function α.

Function $\alpha(\rho, A, p = (v_0, a_1, v_1, \ldots, a_{n-1}, v_{n-1}, a_n, v_n))$

```
1 α ← 0, S ← 0;
2 for j = 1 to n do
3     if a_j ∉ A(ρ) then
4         S ← 0;
5     if a_j ∈ B ∈ B then
6         S ← S + ρ_B;
7         if S ⩾ 1 then
8             S ← S − 1, α ← α + 1;
9 return α;
```

If $A(\rho) = A$, constraints (2.37) and (2.38) are identical. Otherwise, variables λ_p corresponding to paths p passing by arcs $a \notin A(\rho)$ may have their coefficients decreased. However, if the memory sets are adjusted in such a way that variables λ_p with positive values in the current linear relaxation have the same coefficients that they would have in (2.37), the resulting lm-R1C is as effective as the original R1C. Yet, if the final $A(\rho)$ is a small subset of A, as usually happens, the impact in the pricing is much reduced.

If the model defines its packing sets in vertices, the R1Cs are defined in a similar way. There is a non-negative multiplier ρ_B for each $B \in \mathcal{B}^{\mathcal{V}}$ and the cut is:

$$\sum_{p \in P} \left\lfloor \sum_{B \in \mathcal{B}^{\mathcal{V}}} \rho_B \sum_{v \in B^V} \bar{z}_v^p \right\rfloor \lambda_p \leqslant \left\lfloor \sum_{B \in \mathcal{B}^{\mathcal{V}}} \rho_B \right\rfloor . \tag{2.39}$$

Given a memory arc-set $A(\rho) \subseteq A$ corresponding to the vector ρ, the lm-R1C is defined as:

$$\sum_{p \in P} \alpha(\rho, A(\rho), p) \lambda_p \leqslant \left\lfloor \sum_{B \in \mathcal{B}^{\mathcal{V}}} \rho_B \right\rfloor , \tag{2.40}$$

where Function α is the same, except that the condition in line 5 is replaced by $(v_j \in B \in \mathcal{B}^{\mathcal{V}})$.

Regardless of if the packing sets are being defined on arcs or on vertices, it is possible to use lm-R1Cs where the memories are defined by vertex-sets. In this case, a *memory vertex-set* $V(\rho) \subseteq V$ should be assigned to the lm-R1C corresponding to vector ρ. Function α should receive $V(\rho)$ instead of $A(\rho)$ as parameter and the condition in line 3 should be changed to $(v_j \notin V(\rho))$.

Memory vertex-sets perform better for most instances of some classical VRPs. This happens because R1C memory adjustment converges in less iterations in that case. In the other hand, memory arc-sets may be better for some harder instances; because they allow for a finer memory adjustment, leading to less impact in the pricing.

Path Enumeration

The path enumeration technique was proposed by Baldacci et al. (2008), and later improved by Contardo and Martinelli (2014). It consists in trying to enumerate into a pool all paths in a certain set P^k that can possibly be part of an improving solution. After a successful enumeration, the corresponding pricing subproblem k can be solved by inspection, saving time. Moreover, standard fixing by reduced costs can be used to remove paths from the pools. If the enumeration has already succeeded for all $k \in K$ and once the total number of paths in the tables is reduced to a reasonably small number (say, less than 10,000), the formulation restricted to those paths can be given and directly solved by a general MIP solver.

In our context, we try to enumerate all paths $p \in P^k_{\text{elem}}$ such that $\bar{c}(\bar{z}^p) < UB - LB$, where UB is the best known integer solution cost, and LB the value of the current linear relaxation. Moreover, if two paths p and p' in P^k map to variables λ_p and $\lambda_{p'}$ with identical coefficients in the essential constraints in (1.1b), the one with a larger cost is *dominated* and can be dropped. The *essential constraints* are those that are required to make the formulation valid, constraints in (1.1b) added only to strengthen the linear relaxation are not essential.

However, the enumeration procedure would be highly inefficient if the dominance could only be checked for pairs of complete paths. Instead, it is necessary to perform dominance over the partial paths ($\mathcal{B}$-elementary paths starting at the source vertex) that are being constructed along the procedure. Our procedure uses the following dominance rule: if p and p' are partial paths ending at the same vertex and having already visited exactly the same packing sets in $\mathcal{B}$ (regardless of the visitation order), the one with larger cost (breaking ties arbitrarily) is considered dominated and dropped. No complete path in $P^k_{\textbf{elem}}$ that is the completion of a dominated partial path will be produced.

A sufficient condition for enumeration is the following: *every two feasible partial $\mathcal{B}$-elementary paths starting in v^k_{source} that end in the same vertex and map to different coefficients in some essential constraint in* (1.1b) *should have visited different subsets of $\mathcal{B}$*. We remark that this condition can not be checked automatically. In fact, in general it is not even possible to automatically determine what are the essential constraints in (1.1b). It is up to the modeller to prove that the provided model satisfies the sufficient condition, so enumeration can be used. Happily, in many models it is easy to prove that the condition is satisfied. However, if it is not satisfied, it is up to the modeler to prove that the enumeration is valid for his model directly from the dominance rule. Otherwise, the enumeration should be turned off.

Branching

Branching over individual x and y variables (or over constraints defined over those variables) is simple and do not change the structure of the pricing subproblems. In many models this kind of branching is sufficient for correctness. However, there are models where constraints (1.1g) need to be explicitly enforced. However, branching over individual λ variables should be avoided due to a big negative impact in the pricing and also due to highly unbalanced branch trees (Vanderbeck and Wolsey, 2010). The model offers two ways of branching over sets of λ variables:

- Choose distinct sets B and B' in $\mathcal{B}$. Let $P(B, B') \subseteq P$ be the subset of the paths that contain arcs in both B and B'. The branch is over the value of $\sum_{p \in P(B,B')} \lambda_p$, either 0 or 1. This is a generalization of the Ryan and Foster (1981) branch rule. It is still to be avoided if possible, because it makes the pricing harder. However, using that scheme leads to more balanced search trees.

- Choose $B \in \mathcal{B}$, $r \in R^k_M$, and a certain threshold value q^*: in the left child make $u_{a,r} = q^*$, for all $a \in B \cap G^{k(a)}$; in the right child make $l_{a,r} = q^*$. This *branching over the accumulated consumption of a resource* generalizes the strategy proposed by Gélinas et al. (1995). The branching is not likely to be complete, in the sense that some fractional λ solutions can not be eliminated by it. However, it does not increase the pricing difficulty and it may work well in practice, postponing (and even avoiding) the use of a branching that makes pricing harder.

Rounded Capacity Cut Separators

The Rounded Capacity Cuts (RCCs), first proposed for CVRP by Laporte and Nobert (1983), are still useful on modern BCP algorithms for that problem and also for a number of other VRP variants. Moreover, a very good heuristic separation routine is available for it in CVRPSEP library (Lysgaard, 2003). So, we decided to introduce the concept of *RCC separator* as a feature of our model.

The RCC separator can only be used if the packing sets are defined on vertices. For a vertex $v \in V$, define $B(v)$ as the packing set of $\mathcal{B}^{\mathcal{V}}$ that contains v, $B(v) = \varnothing$ if v is not in any packing set. An RCC separator is defined by setting a capacity Q and a demand function $h : \mathcal{B}^{\mathcal{V}} \cup \varnothing \rightarrow \mathbb{R}_+$ such that $h(\varnothing) = 0$ and is valid if there exists an optimal solution (x^*, y^*, λ^*) of formulation (1.1) such that:

1. $\sum_{j=0}^{n} h(B(v_j)) \leqslant Q$, for all $p = (v_0, a_1, v_1, \ldots, a_{n-1}, v_{n-1}, a_n, v_n) \in P$ with $\lambda^*_p \geqslant 1$;

2. for all $B \in \mathcal{B}^{\mathcal{V}}$ such that $h(B) > 0$, the corresponding constraints in (2.36) should be satisfied with equality by (x^*, y^*, λ^*).

Again, it is up to the modeler to prove that the separator included in the model is valid.

Given a valid RCC separator, if $S \subseteq \mathcal{B}^{\mathcal{V}}$, $h(S)$ denotes $\sum_{B \in S} h(B)$ and $\bar{z}^p_S$ is the number of times that an arc in path $p \in P$ enters in S. We say that an arc (v_{j-1}, v_j) enters in S if $B(v_{j-1}) \notin S$ and $B(v_j) \in S$. A Rounded Capacity Cut is the following valid inequality:

$$\sum_{p \in P} \bar{z}^p_S \lambda_p \geqslant \left\lceil \frac{h(S)}{Q} \right\rceil . \tag{2.41}$$

Cuts in format (2.41) are robust. The dual variable of the cut corresponding to an $S \subseteq \mathcal{B}^{\mathcal{V}}$ is simply subtracted from the reduced cost of all arcs entering S.

It is possible to define multiple RCC separators in the same model, each one having its demand function and capacity. This can be useful for modeling VRPs

where routes are constrained by multiple dimensions. Packing sets would have zero demand in the dimensions, in which they do not "participate".

2.4.4 Model Examples

We provide here two examples to illustrate the modeling capabilities of our solver. Other examples are given in our paper (Pessoa et al., 2019b).

Generalized Assignment Problem (GAP)

Data: Set T of tasks; set K of machines; capacity Q^k, $k \in K$; assignment cost c_t^k and machine load h_t^k, $t \in T$, $k \in K$.

Goal: Find an assignment of tasks to machines such that the total load in each machine does not exceed its capacity, with minimum total cost.

Model: RCSP generator graphs $G^k = (V^k, A^k)$ for each $k \in K$: $V^k = \{v_t^k : t = 0, \ldots, |T|\}$, $A^k = \{a_{t+}^k = (v_{t-1}^k, v_t^k), a_{t-}^k = (v_{t-1}^k, v_t^k) : t = 1, \ldots, |T|\}$, $v_{\text{source}}^k = v_0^k$, $v_{\text{sink}}^k = v_{|T|}^k$ (see Figure 2.8); $R^k = R_M^k = \{r^k\}$; $q_{a_{t+}^k, r^k} = h_t^k, q_{a_{t-}^k, r^k} = 0, t \in T$; $[l_{v_t^k, r^k}, u_{v_t^k, r^k}] = [0, Q^k]$, $t \in T \cup \{0\}$. Integer variables x_t^k, $t \in T$, $k \in K$. The formulation is:

$$\text{Min} \quad \sum_{t \in T} \sum_{k \in K} c_t^k x_t^k \tag{2.42a}$$

$$\text{S.t.} \quad \sum_{k \in K} x_t^k = 1, \qquad t \in T. \tag{2.42b}$$

We have $\underline{W}_k = 0$, $\overline{W}_k = 1$, $k \in K$; $M(x_t^k) = \{a_{t+}^k\}, t \in T, k \in K$. $\mathcal{B} = \bigcup_{t \in T} \{\{a_{t+}^k : k \in K\}\}$. Branching is over the x variables. Enumeration is on.

Graphs G^k, illustrated in Figure 2.8, are designed to model binary knapsack constraints: each path in P^k corresponds to a possible assignment of a set of tasks to machine k. The basic formulation in this model is defined as follows. The objective function (2.42b) corresponds to the general objective (1.1a) and constraints (2.42b) to the general constraints (1.1b). The definition of variables x as integer yields integrality constraints corresponding to (1.1h). Constraints (1.1c) are indirectly defined by the RCSP graphs and by the mapping. Finally, constraints (1.1d) are indirectly defined the graphs and by the values of $\underline{W}_k$ and $\overline{W}_k$. A collection of packing sets is provided, so the features described in Section 2.4.3, that extend the basic formulation, can be used. In this model, the validity of the chosen $\mathcal{B}$ is a clear consequence of constraints (2.42b) and of the mapping. However, in other problems, the validity of the packing sets provided by the modeler may not be obvious. All constraints in (2.42b) are essential. It can be checked that the enumeration sufficient condition is satisfied, so the enumeration procedure can be used.

Capacitated Vehicle Routing Problem (CVRP)

Data: Undirected graph $G' = (V, E)$, $V = \{0, \ldots, n\}$, 0 is the depot, $V_+ = \{1, \ldots, n\}$ are the customers; positive cost c_e, $e \in E$; positive demand h_i, $i \in V_+$; vehicle capacity Q.

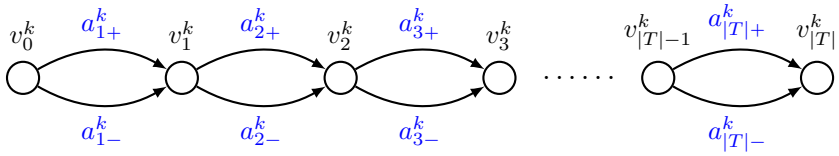


Figure 2.8: GAP model graph, RCSPs correspond to binary knapsack solutions.

Goal: Find a minimum cost set of routes, starting and ending at the depot, visiting all customers and such that the sum of the demands of the customers in a route does not exceed vehicle capacity.

Model: A single graph $G = (V, A)$, $A = \{(i,j),(j,i) : \{i,j\} \in E\}$, $v_{\text{source}} = v_{\text{sink}} = 0$; $R = R_M = \{1\}$; $q_{a,1} = (h_i + h_j)/2$, $a = (i,j) \in A$ (define $h_0 = 0$); $l_{i,1} = 0, u_{i,1} = Q, i \in V$. Integer variables x_e, $e \in E$. The formulation is:

$$\text{Min} \quad \sum_{e \in E} c_e x_e \tag{2.43a}$$

$$\text{S.t.} \quad \sum_{e \in \delta(i)} x_e = 2, \qquad i \in V_+, \tag{2.43b}$$

$$x_e \leqslant 1, \qquad e \in E \backslash \delta(0). \tag{2.43c}$$

We have $\underline{W} = \lceil \sum_{i=1}^{n} h_i/Q \rceil$, $\overline{W} = n$; $M(x_e) = \{(i,j),(j,i)\}$, $e = \{i,j\} \in E$. $\mathcal{B}^{\mathcal{V}} = \cup_{i \in V_+} \{\{i\}\}$. RCC separator on $(\cup_{i \in V_+} \{(\{i\}, h_i)\}, Q)$. Branching on x variables. Enumeration is on.

Constraints (2.43c) are separated (by inspection) as user cuts. The packing sets are defined on vertices. In this problem, defining the resource consumption in a symmetric way ($q_{(i,j),1} = q_{(j,i),1}$) improves the efficiency of the pricing, as discussed in (Sadykov et al., 2017). As constraints (2.43c) are not essential, the enumeration condition over (2.43b) is satisfied. The function $h : \mathcal{B}^{\mathcal{V}} \cup \varnothing \rightarrow \mathbb{R}_+$ for the RCC separator is defined as the set of all pairs $(B, d(B))$ for which $h(B) \neq 0$.

2.4.5 Computational results and perspectives

Algorithms used in the implementation of the solver are generalizations of already published algorithms. The most critical component is the algorithm for solving the RCSP pricing problems. We use our bucket graph based variant (Sadykov et al., 2017) of the bi-directional labeling dynamic programming algorithm (Righini and Salani, 2006). Automatic smoothing stabilization approach proposed in Section 2.2 and diving primal heuristics suggested in Section 2.3 are also employed. The technique by Held et al. (2012) to calculate the safe dual bounds has been utilized for the bin packing problem. Strong branching technique similar to (Røpke, 2012; Pecin et al., 2017b) has also been used. Further implementation details are given in our papers (Sadykov et al., 2017; Pessoa et al., 2019b).

In Table 2.10, we show computational results for 13 applications. The first column is the problem acronym: Capacitated Vehicle Routing Problem (CVRP), Vehicle Routing Problem with Time Windows (VRPTW), Heterogeneous Fleet Vehicle Routing Problem (HFVRP), Multi-Depot Vehicle Routing Problem (MDVRP), Pickup and Delivery Problem with Time Windows (PDPTW), Team Orienteering Problem

Problem	Data set	#	Size	TL	Gen. BCP	Best Publication
CVRP	E-M	12	51-200	10h	12 (61s)	**12 (49s)** Pecin et al. (2017b)
	X	58	101-393	60h	**36 (147m)**	34 (209m) Uchoa et al. (2017)
VRPTW	Solomon Hard	14	100	1h	**14 (5m)**	13 (17m) Pecin et al. (2017a)
	Gehring Homb	60	200	30h	**56 (21m)**	50 (70m) Pecin et al. (2017a)
HFVRP	Golden	40	50-100	1h	**40 (144s)**	39 (287s) Pessoa et al. (2018b)
MDVRP	Cordeau	11	50-360	1h	**11 (6m)**	11 (7m) Pessoa et al. (2018b)
PDPTW	Ropke Cordeau	40	60-150	1h	**40 (5m)**	33 (17m) Gschwind et al. (2018)
	LiLim	30	200	1h	3 (56m)	**23 (20m)** Baldacci et al. (2011a)
TOP	Chao class 4	60	100	1h	**55 (8m)**	39 (15m) Bianchessi et al. (2018)
CTOP	Archetti	14	51-200	1h	**13 (7m)**	6 (35m) Archetti et al. (2013)
CPTP	Archetti open	28	51-200	1h	**24 (9m)**	0 (1h) Bulhoes et al. (2018a)
VRPSL	Bulhoes et al.	180	31-200	2h	**159 (16m)**	49 (90m) Bulhoes et al. (2018a)
GAP	OR-Lib class D	6	100-200	2h	5 (40m)	**5 (30m)** Posta et al. (2012)
	Nauss	30	90-100	1h	**25 (23m)**	1 (58m) Gurobi Optimization (2017)
BPP	Falkenauer T	80	60-501	10m	80 (16s)	**80 (1s)** Brandão and Pedroso (2016)
	Hard28	28	200	10m	28 (17s)	**28 (4s)** Delorme and Iori (2018)
	AI	250	200-1000	1h	**160 (25m)**	140 (28m) Wei et al. (2019)
	ANI	250	200-1000	1h	**103 (35m)**	97 (40m) Wei et al. (2019)
VPP	Classes 1,4,5,9	40	200	1h	**38 (8m)**	13 (50m) Heßler et al. (2018)
CARP	Eglese	24	77-255	30h	**22 (36m)**	22 (43m) Pecin and Uchoa (2019)

Table 2.10: Generic solver vs. best specific algorithms on 13 problems.

(TOP), Capacitated Team Orienteering Problem (CTOP), Capacitated Profitable Tour Problem (CPTP), Vehicle Routing Problem with Service Levels (VRPSL), Generalized Assignment Problem (GAP), Bin Packing Problem (BPP), Vector Packing Problem (VPP), Capacitated Arc Routing Problem (CARP). Second column refers to data sets, the third indicates the number of instances. Next is the time limit per instance. The last two columns show the results obtained by our generic solver and by the best (to our knowledge) published results for the data set. For each algorithm, we give the number of instances solved within the time limit, the average time in brackets (geometric mean time if the time limit is 10 hours or more), and its reference. For instances not solved, the time limit is considered as the solution time. Best results are marked in bold. The performance of our solver depends significantly on initial primal bounds given by the user. In the experiments, we always used the same primal bounds as in the works we compare with.

The results presented in Table 2.10 show that the generic BCP significantly outperforms the state-of-the-art for VRPTW, TOP, CTOP, CPTP, VRPSL, and VPP. A noticeably better performance is achieved for CVRP and HFVRP. For MDVRP, GAP, BPP and CARP, the generic BCP is comparable to the best performing algorithms in the literature. Results are mixed for PDPTW. Worse performance for LiLim instances can be explained by the fact that the generic BCP does not incorporate some labeling algorithm acceleration techniques specific to PDPTW. For the RopkeCordeau instances however, generic state-of-the-art BCP elements mitigate the effect of lacking ad-hoc enhancements.

For each problem, the details concerning the models and the parameterization

employed, instances considered, initial primal bounds used, as well as detailed analysis of computational results are available in our paper (Pessoa et al., 2019b). Using long runs of our solver, we have been able to close six open CVRP instances of the X set, containing from 284 to 548 clients. Details are also given in the paper.

Modeling a typical VRP variant, like those in our tests, requires around 100 lines of Julia code (not counting input/output code). This means that a user can already have a good working algorithm in a day. After that, several days of computational experiments for parameter tuning may yield an improved performance. However, there are variants where additional work on separation routines for problem specific cuts may be needed for top performance.

Furthermore, we believe that there is plenty of room for "creative modeling", where users may find original ways of fitting new problems into the proposed model. In fact, as already demonstrated on generalized assignment problem and on bin/vector packing problems, not only VRP variants can be efficiently treated. It may be also possible to model scheduling, network design problems, as well as problems from other discrete optimization subareas. As the VRP technology available in the solver is quite advanced, there is a chance of obtaining the state-of-the-art performance.

As future work, we plan to further extend the modeling capabilities of the VRP solver. We believe that the most promising course for that is to add the possibility of using other types of resources in the models. This may include resources with arc consumption dependent on its own accumulated consumption or even dependent on accumulated consumption of other resources, resources with soft or multiple interval limits, non-linear and stochastic resources, and others, as discussed in Irnich (2008) and in Parmentier (2019). However, devising and implementing algorithms that support any of those more complex resources, in an efficient way and preserving the compatibility with all the existing features of our solver, will be a major challenge.

Chapter 3

Applications

In this chapter, we review our works on particular problems, both academic and real-life ones.

In Section 3.1 we consider the freight railcar flow problem arising in Russia. This problem can be formulated as a multi-commodity flow problem. We show that the column-and-row generation approach proposed in Section 2.1 allows us to solve the LP relaxation of this problem faster than an LP solver and the standard column generation.

In Section 3.2 we propose a Branch-and-Price algorithm for the Bin Packing Problem with Conflicts. We suggest a novel algorithm to solve the pricing problem which is the knapsack problem with conflits. We also apply our diving heuristic with limited backtracking proposed in Section 2.3.3. Our algorithm is able to solve all test instances for the problem proposed earlier in the literature.

In Section 3.3 we consider the two-dimensional guillotine cutting-stock problem with leftovers arising in the glass cutting industry. We develop the column generation algorithm to solve a relaxation of the problem. A novel partial enumeration technique is proposed to strengthen the relaxation. Our diving heuristic with limited backtracking proposed in Section 2.3.3 is applied to obtain feasible solutions of the problem. Our experimental results reveal that it outperforms constructive and evolutionary heuristics for the problem on the set of industrial instances.

In Section 3.4 we work with a rostering problem. Here the objective is to compute a team schedule for a fixed roster of employees in order to minimize the over- and under-coverage of different parallel activities along a planning horizon. We propose a Branch-and-Price algorithm and adapt to the problem the diving heuristic described in Section 2.3.2. We report results of computational experiments on our set of industrial instances.

In Section 3.5 we examine the robust counterpart of the classic Capacitated Vehicle Routing Problem. We consider the knapsack budget polytope as the uncertainty set for the customer demands. We show that it is possible to reformulate this robust problem as a deterministic heterogeneous fleet vehicle routing problem. The later is then solved by our generic BCP solver described in Section 2.4. We propose problem-specific cuts and a an iterated local search heuristic to improve the performance. Our computational results show a large superiority of our approach over the state-of-the-art on the set of literature instances.

Finally, in Section 3.6 we briefly mention our results for other problems we worked on.

3.1 Freight Railcar Flow Problem

This section is based on our long conference paper (Sadykov et al., 2013) and on our short conference paper (Sadykov et al., 2015a).

We consider a variant of the freight railcar flow problem. In this problem, we need 1) to choose a set of transportation demands between stations in a railroad network, and 2) to fulfill these demands by appropriately routing the set of available railcars, while maximizing the total profit. We formulate this problem as a multi-commodity flow problem in a large space-time graph. Three approaches are proposed to solve the Linear Programming relaxation of this formulation: direct solution by an LP solver, a column generation approach based on the path reformulation, and the "column generation for extended formulations" approach presented in Section 2.1. In the latter, the multi-commodity flow formulation is solved iteratively by dynamic generation of arc flow variables. Three approaches have been tested on a set of real-life instances provided by one of the largest freight rail transportation companies in Russia. Instances with up to 10 millions of arc flow variables were solved within minutes of computational time.

3.1.1 Problem description

In Russia, the regulation separates the activity of forming and scheduling freight trains from the activity of managing the fleet of freight railcars. A state company is in charge of the first activity. Freight railcars are owned by several independent companies. Every such company is quite limited in transportation decisions due to the separation of activities. A company which owns a fleet of railcars can only accept or refuse a transportation demand. Then it must assign railcars to accepted demands. In some cases, the company has a possibility to slightly modify the execution date of a demand, which gives more flexibility to the decision process but makes it more complicated.

Thus, an operational plan of such a company is determined by 1) a set of accepted transportation demands, 2) for each demand, its execution date and the set of cars assigned to it, and 3) empty cars movements to supply each demand. As the company is commercial, a reasonable criterion for the quality of an operational plan is the profit generated by it. The profit is determined by the difference between the price collected for fulfilling transportation demands and the costs paid to the state company for exploiting the railroad network.

In this paper, we study the problem of finding the most profitable operational plan for a company which owns and manages a fleet of railcars. This problem was formulated by the mathematical modeling department of one of the largest such companies in Russia.

The railroad network consists of a set of stations. Travel times and costs are known for each "origin-destination" pair of stations. Times are measured in days and

rounded up. The cost for an empty car transfer depend on the type of the latest product this car has transported, as explained in the introduction.

Number of cars, their initial locations and availability dates are known. Cars are divided into types. The type of a car determines types of products which can be loaded on this car. The route of a car consists of a sequence of alternating loaded and empty movements between stations. Cars can wait at stations before and after fulfilling transportation demands. In this case, a charge is applied. Daily rate of this charge depends on the demand before (or after) the waiting period.

Each transportation demand is defined by a (maximum) number of cars compatible with the product that should be taken from an origin station to a destination station. Some demands can be fulfilled partially. In this case, the client communicates the minimum number of cars which should be delivered. Thus, the total number of transported cars for every accepted demand should be between the minimum and maximum number.

The client specifies the availability date of the product and the delivery due date which cannot be exceeded. The demand transportation time is known. This allows us to determine the latest date at which the transportation must start. The profit we gain for meeting the demand depends on the date the transportation of a loaded car starts. In practice, the contract is concluded for transportation of each car separately. Thus the profit we gain for delivering cars with the product of a same demand at a certain date depends linearly on the number of cars. Note that the profit function already takes into account the charges paid for using the railroad network.

We now specify notations for the data of the problem. Following sets are given.

- I — set of stations.
- K — set of car types.
- U — set of product types
- Q — set of demands
- S — set of "sources" which specify initial state of cars.
- T — set of periods (planning horizon).

For each station, $i \in I$ we know sets R_i^1 and R_i^2 of standing daily rates for cars waiting to be loaded and waiting after unloading.

For each demand $q \in Q$ we know:

- $i_q \in I$ — origin station
- $j_q \in I$ — destination station
- $u_q \in U$ — type of product to be transported
- $K_q \subseteq K$ — set of car types, which can be used for this demand
- $n_q^{\max}$ — number of cars needed to completely fullfil the demand
- $n_q^{\min}$ — minimum number of car needed to partially fullfil the demand

- $\tau_q \in T$ — demand availability, i.e. the period starting from which the transportation of the product can start
- g_q — maximum delay for starting the transportation
- ρ_{qt} — profit from delivery of one car with the product, transportation of which started at period t, $t \in [r_q, r_q + g_q]$
- $d_q \in \mathbb{Z}_+$ — transportation time of the demand
- $r_q^1 \in R_{i_q^1}^1$ — daily standing rate charged for one car waiting before loading the product at origin station
- $r_q^2 \in R_{i_q^2}^2$ — daily standing rate charged for one car waiting after unloading the product at destination station

For each car type $k \in K$, we can obtain set Q_k of demands, which a car of type k can fulfil.

For each source $s \in S$, we are given:

- $\vec{i}_s \in I$ — station where cars are located
- $\vec{k}_s \in K$ — type of cars
- $\vec{r}_s \in T$ — period, starting from which cars can be used
- $\vec{r}_s \in R_{i_s}^2$ — daily standing rate charged for cars
- $\vec{u}_s \in U$ — type of the latest delivered product
- $\vec{n}_s \in \mathbb{N}$ — number of cars in the source

For each car type $k \in K$, we can obtain set of sources $S_k = \{s \in S : \vec{k}_s = k\}$.

Additionally, functions $F(k, i, j, u)$ and $D(k, i, j)$ are given which specify cost and duration of transportation of one empty car of type $k \in K$ from station $i \in I$ to station $j \in I$ under the condition that the type of the latest delivered product is $u \in U$ (for the cost).

3.1.2 Mathematical model

We represent movements of cars of each type $k \in K$ by commodity k. For each commodity $k \in K$, we introduce a directed graph $G^k = (V^k, A^k)$. Set V^k of vertices is divided into two subsets V^{k1} and V^{k2} which represent respectively states in which cars stand at a station before being loaded and after being unloaded. A vertex $v_{itr}^{k1} \in V^{k1}$ represents stay of cars of type k waiting to be loaded at station $i \in I$ at daily rate $r \in R_i^1$ at period $t \in T$. Flow balance b_v of this vertex $v = v_{itr}^{k1}$ is zero. A vertex $v_{itru}^{k2} \in V_c^{k2}$ represents stay of cars of type k after being unloaded at station $i \in I$ at daily rate $r \in R_i^2$ at period $t \in T$. Here $u \in K$ is the type of unloaded product. Flow balance b_v of this vertex $v = v_{itru}^{k2}$ is determined as follows:

$$b_v = \begin{cases} \vec{n}_s, & \exists s \in S_k : \vec{i}_s = i, \vec{\tau}_s = t, \vec{r}_s = r, \vec{u}_s = u, \\ 0, & \text{otherwise.} \end{cases}$$

Additionally, there is a single terminal vertex with flow balance equal to $-\sum_{s \in S_c} \vec{n}_s$.

There are three types of arcs in A^k: waiting, empty transfer, and loaded transfer arcs.

- A waiting arc $a^{k\alpha}_{itru}$ represents waiting of cars of type k from period $t \in T$ to $t+1$ at station $i \in I$ at daily rate $r \in R^\alpha_i$ before being loaded ($\alpha = 1$) or after being unloaded ($\alpha = 2$). $u \in U$ is the type of unloaded product in case $\alpha = 2$. This arc goes from vertex $v^{k\alpha}_{itru}$ to vertex $v^{k\alpha}_{i,t+1,r,u}$, or to the terminal vertex if $t+1 \notin T$. Cost of this arc is r.

- An empty transfer arc $a^k_{ijtr'r''u}$ represents a transfer of empty cars of type k waiting at station $i \in I$ at daily rate $r' \in R^2_i$ to station $j \in I$ where they will wait at daily rate $r'' \in R^1_j$, such that the type of latest unloaded product is $u \in U$, and transfer starts at period $t \in T$. This arc goes from vertex $v^{k2}_{itr'u}$ to vertex $v^{k1}_{jt'r''}$, or to the terminal vertex if $t' \notin T$, where $t' = t + D(k,i,j)$. Cost of this arc is $F(k,i,j,u)$.

- A loaded transfer arc a^k_{qt} represents transportation of the product of demand $q \in Q$ by cars of type k starting at period $t \in T \cap [\tau_q, \tau_q + g_q]$. This arc goes from vertex $v^{k1}_{i_q t r^1_q}$ to vertex $v^{k2}_{j_q, t+d_q, r^2_q, u_q}$, or to the terminal vertex if $\{t + d_q\} \notin T$. The cost of this arc is $-\rho_{qt}$.

A small example of graph G^k is depicted in Figure 3.1. In this example, there is only one "before" vertex and one "after" vertex for each time period and each station. In real-life examples, there are several rows of "before" and "after" vertices for each station.

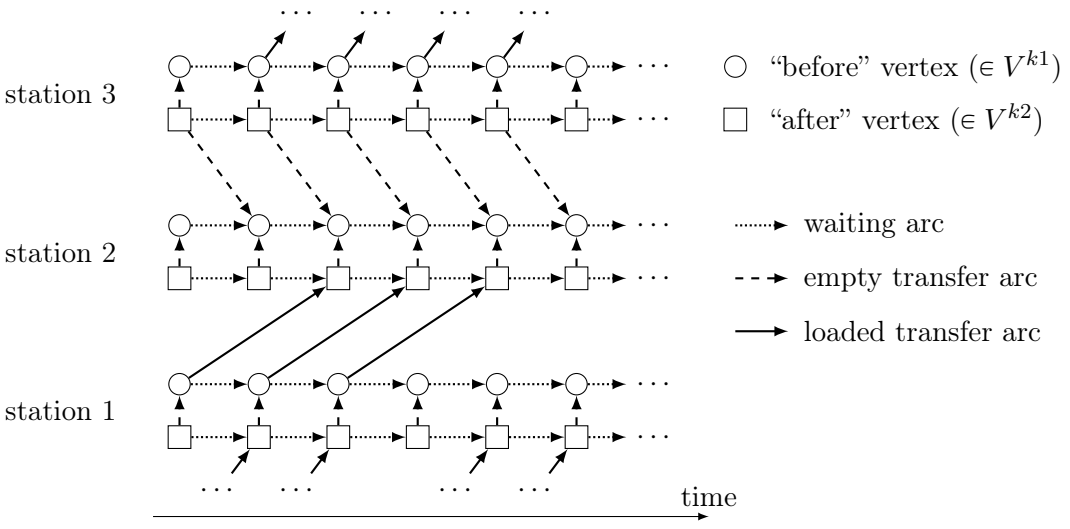


Figure 3.1: An example of graph G^k

We denote as A^k_q the set of all loaded transfer acs related to demand $q \in Q_k$: $A^k_q = \{a^k_{q't} \in A^k : \ q' = q\}$. Also we denote as $\delta^+(v)$ and $\delta^-(v)$ the sets of incoming and outgoing arcs for vertex v.

From now on, graph G^k is assumed to be trivially preprocessed: we remove vertices with degree two (replacing appropriately incident arcs), and remove every vertex (together with incident arcs) such that there is no path from any source to it or there is no path from it to the terminal vertex.

For each commodity $k \in K$ and for each arc $a \in A^k$, we define an integer variable z_a^k which represents the flow size of commodity k along arc a. Cost of arc $a \in A^k$ is denoted as c_a^k. Additionally, for each demand $q \in Q$, we define a binary variable y_q which indicates whether demand q is accepted or not.

Now we are able to present the multi-commodity flow formulation [MCF] for the problem.

$$\min \sum_{k \in k} \sum_{a \in A^k} c_a^k z_a \tag{3.1a}$$

$$n_q^{\min} y_q \leqslant \sum_{k \in K_q} \sum_{a \in A_q^k} z_a^k \leqslant n_q^{\max} y_q, \quad \forall q \in Q, \tag{3.1b}$$

$$\sum_{a \in \delta^-(v)} z_a^k - \sum_{a \in \delta^+(v)} z_a^k = b_v, \qquad \forall k \in k, v \in V^k, \tag{3.1c}$$

$$z_a^k \in \mathbb{Z}_+, \qquad \forall k \in K, a \in A^k, \tag{3.1d}$$

$$y_q \in \{0,1\}, \qquad \forall q \in Q. \tag{3.1e}$$

Constraints (3.1b) specify that the number of cars assigned to accepted demand q should be between $n_q^{\min}$ and $n_q^{\max}$. Constraints (3.1c) are flow conservation constraints for each commodity. As formulation [MCF] generalizes the standard multi-commodity flow problem (where variables y are fixed to one), our problem is NP-hard in the strong sense.

3.1.3 Solution approaches

Formulation (3.1) can be viewed as formulation [R] of Section 2.1.1, in which (3.1b) and (3.1e) are linking constraints, and (3.1c)–(3.1d) define the subsystem $\{Hz \geqslant h;\ z \in \mathbb{Z}_+^{|A|}\}$, where $A = \bigcup_{k \in K} A^k$. The latter can be decomposed into subsystems $\{H^k z^k \geqslant h^k;\ z^k \in \mathbb{Z}_+^{|A^k|}\}$, one for each commodity $k \in K$. Every such subsystem defines a feasible flow of commodity $k \in K$ in graph G^k.

The assumption of Section 2.1.1 is fulfilled as $\{H^k z^k \geqslant h^k;\ z^k \in \mathbb{R}_+^{|A^k|}\}$ defines the convex hull of the subsystem corresponding to commodity $k \in K$. Thus, we can apply the column-and-row generation approach to solve the linear relaxation of [R]. Given a dual solution $\pi^+, \pi^- \in \mathbb{R}_+^{|Q|}$ corresponding to constraints (3.1b), the pricing problem for commodity $k \in K$ is

$$\bar{z}^k(\pi^+, \pi^-) = \operatorname*{argmin}_{z^k} \left\{ (c^k - \pi^- \Omega^k + \pi^+ \Omega^k) z^k :\ \Delta^{k-} z^k - \Delta^{k+} z^k = b,\ z^k \in \mathbb{Z}_+^{|A^k|} \right\},$$

where Ω^k is the binary matrix to determine the link between arcs in A^k and demands in Q: $\Omega_{qa}^k = 1$ if and only if $a \in A_q^k$; Δ^{k-} and Δ^{k+} are the binary adjacency matrices for graph G^k: $\Delta_{va}^{k-} = 1$ ($\Delta_{va}^{k+} = 1$) if and only if $a \in \delta^-(v)$ ($a \in \delta^+(v)$), $a \in A^k$, $v \in V^k$.

Alternatively, we can use the standard column generation approach in which we define as P^k_{flow} the set of all solutions $\bar{z}^p$ to $\left\{\Delta^{k-} z^k - \Delta^{k+} z^k = b,\ z^k \in \mathbb{Z}_+^{|A^k|}\right\}$, i.e the set of all feasible flows $p \in P^k_{flow}$. Such an approach would not be efficient computationally due to severe convergence issues, as the number of demands covered by a flow is very large.

Instead, we adopt the column generation approach based on paths. Let P^k_s, be the set of routes for a particular car, i.e. the set of paths in graph P^k, $k \in K$, starting at source $s \in S^k$, $S^k = \{s \in S :\ \vec{k}_s = k\}$. Let $\bar{z}^p \in \{0,1\}$ be the characteristic vector of path $p \in P^k_s$ in graph G^k starting at node $s \in S_k$, i.e. $\bar{z}^p_a = 1$ if and only if path p uses arc $a \in A^k$. Let c^p be the cost of path $p \in P^k_s$: $c^p = c^k \bar{z}^p$. Let A^k_q be the set of loaded transfer arcs in A^k which represent transportation of demand $q \in Q$: $A^k_q = \left\{a^k_{qt} :\ t \in T \cap [\tau_q, \tau_q + g_q]\right\}$. Then the problem can be reformulated using variables λ_p, $p \in P^k_s$, $k \in K$, $s \in S_k$.

$$\min \sum_{k \in k} \sum_{s \in S_k} \sum_{p \in P^{ks}} c^p \lambda_p \tag{3.2a}$$

$$n^{\min}_q y_q \leqslant \sum_{k \in k} \sum_{s \in S^k} \sum_{p \in P^k_s} \sum_{a \in A^k_q} \bar{z}^p_a \lambda_p \leqslant n^{\max}_q y_q, \quad \forall q \in Q, \tag{3.2b}$$

$$\sum_{p \in P^k_s} \lambda_p = \vec{n}_s, \qquad \forall k \in k, s \in S^k, \tag{3.2c}$$

$$\lambda_p \in \mathbb{Z}_+, \qquad p \in P^k_s, k \in K, s \in S_k, \tag{3.2d}$$

$$y_q \in \{0,1\}, \qquad \forall q \in Q. \tag{3.2e}$$

The LP relaxation of (3.2) is solved by column generation. The pricing problem is decomposed into subproblems, one for each commodity $k \in K$, and for each source $s \in S^k$. So, the problem of finding a variable λ with the minimum reduced cost can be solved by a sequence of shortest path problems between each source $s \in S^k$ and the terminal vertex for every commodity $k \in K$. To accelerate the solution of the pricing problem, instead of searching the shortest path separately for each source, in each graph G^k, we can find a minimum cost in-tree to the terminal vertex from every source in S^k. As directed graphs G^k are acyclic (each arc except those from V^{k2} to V^{k1} induces a time increase), the complexity of this procedure is linear in the number of arcs in A^k.

This procedure is quite fast, but its disadvantage consists in significant "over-covering" of demands. This means that many generated paths contain arcs corresponding to the same demands, i.e. much more cars are assigned to these demands than needed. This has a bad impact on the convergence of column generation. Therefore, we developed an iterative procedure which heuristically constructs a solution to the original problem with demand profits modified by the current dual solution values. Then all paths which constitute this solution are added to the master. On each iteration, we search for a shortest path tree and then remove covered demands and cars assigned to them for the next iteration. The heuristic stops when either all demands are covered, or all cars are assigned, or maximum number of iterations is reached. Details of this procedure are given in our paper.

3.1.4 Numerical results

The test instances were provided to us by the mathematical modeling departement of *JSC Freight One*, which is one of the largest freight rail transportation companies in Russia. We have numerically tested the following three approaches:

1. Direct solution of the LP relaxation of formulation (3.1) by the *Clp* LP solver. Before applying the LP solver the formulation is preprocessed by a non-trivial problem specific procedure. This procedure is not public and it was not available to us. Moreover, the open-source solver *Clp* was specifically modified to better tackle formulation $(MCF)_{LP}$. Thus, this approach was applied inside the company. We tried to solve the formulation with only trivial preprocessing by the default version of both LP solvers *Clp* and *Cplex*, but our solution times on a comparable computer were significantly larger. Therefore, for the comparison, we use the solution times communicated to us by the company. We denote this approach as DIRECT.

2. Solution of the LP relaxation of formulation (3.2) by standard column generation. Automatic dual price smoothing stabilization proposed in Section 2.2.2 is used. We denote this approach as COLGEN.

3. Solution of the LP relaxation of formulation (3.1) by the column-and-row generation procedure proposed in Section 2.1.1. The pricing problem here is solved using the minimum cost flow algorithm implemented in C++ library *Lemon*. To improve convergence of the algorithm, the master is initialized with the full set of waiting arcs. Contrary to DIRECT, only a trivial procedure was applied to preprocess the formulation. We denote this approach as COLROWGEN. The same stabilization approach is also used.

Further implementation details are given in our paper (Sadykov et al., 2013).

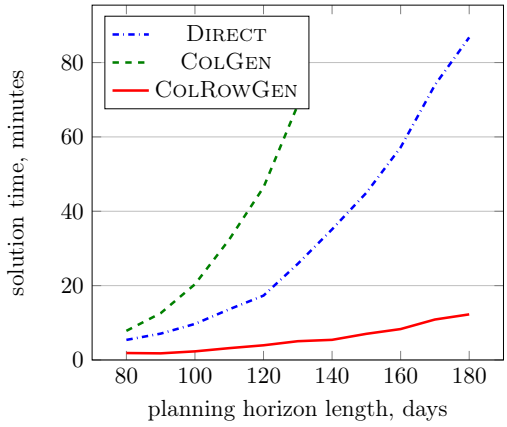
The first test set consists of 3 instances. Characteristics of these instances and results for 3 tested approaches for these instances are presented in Table 3.1.

Instance name	`x3`	`x3double`	`5k0711q`
Number of stations	371	371	1'900
Number of demands	1'684	3'368	7'424
Number of car types	17	17	1
Number of cars	1'013	1'013	15'008
Number of sources	791	791	11'215
Time horizon, days	37	74	35
Total number of vertices, thousands	62	152	22
Total number of arcs, thousands	794	2'846	1'843
Solution time for DIRECT	20s	1h34m	55s
Solution time for COLGEN	22s	7m53s	8m59s
Solution time for COLROWGEN	3m55s	>2h	43s

Table 3.1: The first set of instances: characteristics and numerical results

The difference in performance of the approaches DIRECT and COLROWGEN on instances `x3` and `x3double` can be explained by the problem specific preprocessing. Although we are not aware of preprocessing details, we know that it is based on similarities between car types. For instance `5k0711q` in which there is only one car type, difference between two approaches is much smaller. Note that this instance has been artificially created from the real-life one by merging car types into one.

The second test set consists of instances with larger planning horizon length. These instances contain 1'025 stations, up to 6'800 demands, 11 car types, 12'651 cars, and 8'232 sources. The planning horizon length is from 80 to 180 days. The full graph $G = (\bigcup_{k \in K} V^k, \bigcup_{k \in K} A^k)$ for the largest instance contains about 300 thousand nodes and 10 million arcs. For these instances, the two best approaches are DIRECT and COLROWGEN. The comparison of the solution times is presented in Figure 3.2.



Horizon	DIRECT	COLROWGEN
80	5m24s	1m52s
90	7m05s	1m47s
100	9m42s	2m19s
110	13m38s	3m11s
120	17m19s	3m57s
130	25m52s	5m03s
140	35m08s	5m25s
150	44m58s	7m02s
160	57m11s	8m19s
170	1h13m58s	10m53s
180	1h26m46s	12m16s

Figure 3.2: Solution times for test instances with larger planning horizon length

An important observation is that the algorithm COLROWGEN generally converges in less than 10 iterations (and always in less than 15 iterations). The restricted master on the final iteration contains only about 3% of the arc flow variables.

In our consequent short conference paper (Sadykov et al., 2015a) we have showed that by applying combined stabilization approach described in Section 2.2.3, we can improve the running time of the approach COLGEN. It becomes competitive with COLROWGEN for instances with the planning horizon up to 140 days. However, instances with larger time horizon are still better solver by the column-and-row generation approach.

To obtain integer solutions for the problem, a diving heuristic from Section 2.3 can be applied when the COLGEN approach is used. In Sadykov et al. (2015a), we have showed that the pure diving heuristic always obtains optimal solutions for the test instances with the planning horizon up to 140 days (solution value is equal to the column generation lower bound). Experiments conducted inside the company also showed that a simple rounding of the solution of the LP relaxation of formulation (3.1) produces very good results.

3.1.5 Related work

To our knowledge, the closest model considered in the literature is the freight car flow problem faced by a Brazilian logistics operator and described by Fukasawa et al. (2002). In this paper, authors proposed a similar integer multi-commodity flow model and solved it using a simple preprocessing and an Mixed Integer Programming (MIP) solver. The main difference with our model is the availability of a fixed train schedule. In their model cars must be assigned to trains to be transported, and one needs to consider loading and unloading times.

Another similar car flow model has been considered by Holmberg et al. (1998). In this model, one searches only for a flow of empty cars, the flow of loaded cars being fixed. Thus, a heuristic iterative procedure is applied to optimize the total flow of cars.

A paper which is related to our research in terms of the solution approach applied is due to Löbel (1998), who considered a vehicle scheduling problem arising in public mass transit. This problem is modeled by a multi-commodity flow model formulation, the LP relaxation of which is solved by dynamically generating arc variables.

3.2 Bin Packing with Conflicts

This section is based on our journal paper (Sadykov and Vanderbeck, 2013a).

The bin packing problem with conflicts consists in packing items in a minimum number of bins of limited capacity while avoiding joint assignments of items that are in conflict. Our study demonstrates that a generic implementation of a Branch-and-Price algorithm using specific pricing oracle yields comparatively good performance for this problem. We use our black-box Branch-and-Price solver BaPCod, relying on its generic branching scheme and primal heuristics. We developed a dynamic programming algorithm for pricing when the conflict graph is an interval graph, and a depth-first-search branch-and-bound approach for pricing when the conflict graph has no special structure. The exact method is tested on instances from the literature where the conflict graph is an interval graph, as well as harder instances that we generated with an arbitrarily conflict graph and larger number of items per bin. Our computational experiment sets new benchmark results for the problem, closing all open instances of the literature in one hour of CPU time.

3.2.1 Formulations of the problem

Formally, the Bin Packing Problem with Conflicts (BPPC) can be described as follows. We are given a set K of identical bins of capacity Q, a set $V = \{1, 2, \ldots, n\}$ of items characterized by a non-negative capacity consumption $q_i \leqslant Q$, and a conflict graph $G = (V, E)$, where E is a set of edges such that $(i, j) \in E$ when i and j are in conflict. The problem is to assign items to bins, using a minimum number of bins, while ensuring that the total weight of the items assigned to a bin does not exceed the bin capacity Q, and that no two items that are in conflict are assigned to the same bin. The number of bins $|K| \leqslant n$ is assumed to be large enough to guarantee

feasibility; more precisely it is a valid upper bound on the number of bins in an optimal solution.

For this problem, one can use the set covering reformulation used by Fernandes Muritiba et al. (2010); Elhedhli et al. (2011). Let P be the family of all the subsets of items which are not in conflict and fit into one bin. Each subset $p \in P$ is defined by an indicator vector $\bar{z}^p$ ($\bar{z}_i^p = 1$ if item i is in set p) and associated with a binary variable λ_p taking value 1 if the corresponding subset of items is selected to fill one bin. The formulation is:

$$\min \sum_{p \in P} \lambda_p \tag{3.3a}$$

$$s.t. \quad \sum_{p \in P} \bar{z}_i^p \lambda_p = 1, \qquad i = 1, \ldots, n, \tag{3.3b}$$

$$\lambda_p \in \{0, 1\}, \quad p \in P. \tag{3.3c}$$

Here, constraints 3.3b) require that each item is assigned to a bin.

Formulation (3.3) is tackled using a branch-and-price approach: at each node of a branch-and-bound tree, the linear relaxation of (3.3) is solved by column generation, as discussed in Section 1.1.

Let $\bar{\pi}$ be a current dual solution of the restricted master problem. Then, the pricing problem can be formulated as

$$\max \sum_{i=1}^{n} \bar{\pi}_i z_i \tag{3.4a}$$

$$s.t. \quad \sum_{i=1}^{n} q_i z_i \leqslant Q, \qquad i = 1, \ldots, n, \tag{3.4b}$$

$$z_i + z_j \leqslant 1, \qquad (i, j) \in E, \tag{3.4c}$$

$$z_i \in \{0, 1\}, \quad i = 1, \ldots, n. \tag{3.4d}$$

Model (3.4) is the Knapsack Problem with Conflicts (KPC).

3.2.2 Algorithms for the knapsack problem with conflicts

In selecting an algorithm to solve the KPC with interval and arbitrary conflict graphs, the first obvious choice is to apply an IP solver to formulation (3.4). However, our preliminary tests showed that very efficient IP solvers such as CPLEX are not fast enough to be called many times during the column generation procedure. An alternative specialized branch-and-cut algorithm for the KPC was proposed by Hifi and Michrafy (2007). It is faster than CPLEX only on a small fraction of instances with conflict graph density of around 1%. Therefore, we developed our own specialized algorithms for the KPC.

First we consider the Knapsack Problem with Interval Conflict Graphs (KPICG). Formally, a graph $G = (V, E)$ is an interval graph if, to each vertex $v \in V$, one can associate an open interval $I_v = (a_v, b_v)$ for $a_v, b_v \in \mathbb{R}$ with $a_v < b_v$, such as two distinct vertices $u, v \in V$ are adjacent in G if and only if $I_u \cap I_v \neq \varnothing$. The family $\{I_v\}_{v \in V}$ is an interval representation of G. See Figure 3.3 for an illustration.

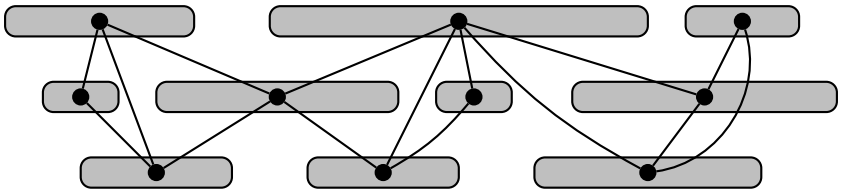


Figure 3.3: An interval graph and its interval representation

Recently, Pferschy and Schauer (2009) proposed a pseudo-polynomial algorithm for the KPC with chordal conflict graphs which is a super-class of interval graphs. The complexity of their algorithm is $O(nQ^2)$. Its pseudo-polynomial complexity of order two makes this procedure too time consuming to be used for pricing within a branch-and-price algorithm. We designed a dynamic programming algorithm with a lower complexity for the special case of interval graph. Our algorithm exploits the interval representation of the conflict graph: $\{I_i\}_{i\in V}$.

Definition 3.1. Considering an interval conflict graph, $G = (V, E)$, assume the items, $i \in V$, are indexed in non-decreasing order of the right endpoints of the corresponding intervals $\big\{I_i = (a_i, b_i)\big\}_{i\in V}$ (the ties are resolved arbitrarily), i.e., $b_i \leqslant b_j$ if $i < j$. Let $C_i^<$ denote the set of items with indexes smaller than i that are not in conflict with i:

$$C_i^< = \{j : \ j < i, (i, j) \notin E\}, \quad \forall i \in V.$$

Let $prev_i$ be the item in $C_i^<$ with the largest index (or 0 if $C_i^< = \varnothing$):

$$prev_i = \begin{cases} \max\{j : \ j \in C_i^<\}, & C_i^< \neq \varnothing, \\ 0, & C_i^< = \varnothing, \end{cases} \qquad \forall i \in V.$$

Observation 3.1. *Consider an interval conflict graph, $G = (V, E)$. Given the item indexing of Definition 3.1, for every pair $i, j \in V$, such that $1 \leqslant j \leqslant prev_i$, i and j are not in conflict, i.e., $(i, j) \notin E$, while when $prev_i < j < i$, i and j are in conflict, i.e., $(i, j) \in E$.*

Let $S(i, q)$ be the value of an optimal solution of the KPICG for the first i items and knapsack size q. With this notation, the solution of model (3.4) gives $S(n, Q)$. By Observation 3.1, the solution set associated with $S(i, q)$ either includes item i and cannot include any items j such as $prev_i < j < i$; or it does not include item i and it reproduces the solution set for $S(i - 1, q)$. Therefore,

$$S(i, q) = \max\big\{S\big(prev_i, q - q_i\big) + \bar{\pi}_i, \ S(i - 1, q)\big\}. \tag{3.5}$$

The value $S(n, Q)$ is the solution to KPICG. The associated solution set B can be retrieved by backtracking from value $S(n, Q)$ to value $S(0, 0)$. The dynamic programming algorithm stemming from the recurrence relation (3.5) is formally presented as Function `DP`. It is easy to see that the time and the space complexity of the dynamic programming algorithm are both $O(nQ)$ once the values *prev* are known. This complexity is more tractable in practice than that of the algorithm by Pferschy and Schauer (2009).

Function DP$(n, \bar{\pi}, q, Q, prev)$

```
for q ← 0 to Q do S(0, q) ← 0 ;
for i ← 1 to n do
    for q ← 0 to q_i − 1 do
        S(i, q) ← S(i − 1, q), l(i, q) ← 0;
    for q ← q_i to Q do
        S(i, q) ← S(i − 1, q), l(l, q) ← 0;
        if S(i, q) < S(prev_i, q − q_i) + π̄_i then
            S(i, q) ← P(prev_i, q − q_i) + π̄_i, l(i, q) ← 1;
q ← Q, Ī ← ∅, i ← n;
while i > 0 do
    if l(i, q) = 1 then Ī ← Ī ∪ {i}, q ← q − q_i, i ← prev_i ;
    else i ← i − 1 ;
return Ī;
```

Next, we consider the knapsack problem with an arbitrary conflict graph. We developed a recursive enumeration procedure for the KPC that combines the classic depth-first-search based branch-and-bound algorithm for the 0-1 Knapsack Problem with the enumeration algorithm for solving the maximum clique (or maximum independent set) problem by Carraghan and Pardalos (1990). The latter also makes use of a depth-first-search strategy, while dual bounds are obtained by simply ignoring all conflicts between free vertices, i.e. vertices which have not yet been fixed via branching decisions.

Definition 3.2. For each item $i \in V$, we define the list C_i of items in conflict with i. At any node of the enumeration tree, we denote by I^1 the set of items that have been selected in the current partial knapsack solution and by I^0 the items that have been fixed to 0. The set $F = (V \backslash (\cup_{i \in I^1} C_i \cup I^1 \cup I^0))$ denotes free items that are not fixed to either 0 or 1 in previous branching decisions and are not in conflict with items in I^1. Assume that items are indexed in the non-decreasing order of their "efficiency", i.e., of their ratio $\bar{\pi}_i / q_i$. Then, $succ_i(F)$ denotes the item following i in the sorted sub-list of items of F. By extension $succ_0(F)$ denote the first element in F, while $last(F)$ the last element in F and $succ_{last(F)}(F) = n + 1$.

During the depth-first-search, upper (dual) bounds UB are computed at each node of the tree by solving the continuous relaxation of the residual knapsack problem on set F, ignoring conflict constraints:

$$UB = \max \sum_{i \in F} \bar{\pi}_i z_i + \sum_{i \in I^1} \bar{\pi}_i \tag{3.6a}$$

$$s.t. \quad \sum_{i \in F} q_i z_i \leqslant Q - \sum_{i \in I^1} q_i, \quad i \in F, \tag{3.6b}$$

$$0 \leqslant x_i \leqslant 1, \qquad i \in F. \tag{3.6c}$$

As the items in F are sorted according to their efficiencies, problem (3.6) can be solved in $O(n)$ time using a greedy algorithm.

If the current upper bound UB is smaller or equal to the value LB of the incumbent solution, we prune the node, putting an end to further recursive calls to the enumeration procedure. Otherwise, we branch: for each item $i \in F$, we consider a child node where i is added to I^1 and all items of F that precede i in the ordering are added to I^0. As the items in F and in the conflict list C_i of the i-th item in F are sorted in the same order, each child node can be created in time $O(n)$. Thus, the time spent per node is linear. The recursive enumeration procedure for KPC is formally presented as Function $\texttt{Node}(\rho, q, I^1, F, LB, \bar{I})$, where ρ is the current profit, q the current weight, I^1 the set of items fixed to 1, F the set of free items, LB the current lower bound, and $\bar{I}$ the associated current incumbent solution. Our depth-first-search branch-and-bound algorithm for KPC is invoked by calling $\texttt{Node}(0, 0, \varnothing, V, 0, \mathbf{0})$.

Function Node$(\rho, q, I^1, F, LB, \bar{I})$

```
1  if ρ > LB then LB ← ρ, Ī ← I^1;
2  UB ← q, q' ← q, i ← succ_0(F);
3  while q' < Q and i ⩽ last(F) do
4      if q' + q_i ⩽ Q then
5          UB ← UB + π̄_i, q' ← q' + q_i, i ← succ_i(F);
6      else
7          UB ← UB + (Q − q') · (π̄_i/q_i), q' ← W;
8  if UB ⩽ LB then return Ī;
9  i ← succ_0(F);
10 while ρ + (Q − q) · (π̄_i/q_i) > LB and i ⩽ last(F) do
11     j ← succ_i(F), F ← F\{i};
12     if q + q_i ⩽ Q then Ī ← Node(ρ + π̄_i, q + q_i, I^1 ∪ {i}, F\C_i, LB, Ī);
13     i ← j;
14 return Ī;
```

3.2.3 Computational results

In the branch-and-price algorithm we use the generic branching scheme proposed by Vanderbeck (2011) that was specially designed to preserve the structure of the pricing problem. The scheme proceeds by progressively partitioning into column classes the set P of feasible pricing problem solutions and by implementing separate pricing on each class. A class is defined by restricting the solution set via fixing some variables to zero or one. Hence, pricing over a class can be done using the initial oracle since the latter can handle some variable fixing. The implementation developed in Vanderbeck (2011) guarantees that the number of created classes remains polynomial in the input size. Fractional master solutions are eliminated by adding branching constraint in the master that force an integer lower bound on the number of columns selected in each defined class. The dual bounds after branching are proved

to be as strong as if branching constraints had been defined in the subproblem.

To obtain feasible solutions of the problem, we use the diving heuristic with limited backtracking proposed in Section 2.3.3.

In our paper (Sadykov and Vanderbeck, 2013a) we have showed that, in the standard literature instances for the problem generated by Gendreau et al. (2004), the conflicts form an interval graph. The detailed description of the test instances as well as experimental setup are also given in our paper.

In our numerical experiments, we first compared our algorithm and the algorithm of Fernandes Muritiba et al. (2010), which we denote FMIMT. In comparison to FMIMT, we tested three versions of our algorithm: (1) using our branch-and-price approach with specialized DP pricing but without the primal heuristic, (2) running the algorithm with the column generation based primal heuristic, but relying on the branch-and-bound oracle for pricing in a general conflict graph instead of the specialized DP pricing oracle for interval graphs (3) the full-blown branch-and-price approach with DP pricing and primal heuristic.

	FMIMT		Our w/o heur.		Our w/o DP sp		Our w DP sp & heur.	
class	¬opt	av. time	¬opt	av. time	av. time	max. time	av. time	max. time
t60	0	38.7	0	1.0	0.9	27.9	0.8	6.5
t120	5	1860.3	1	156.0	26.9	1971.3	37.8	2956.4
t249	4	1582.1	2	334.2	30.0	235.5	29.3	130.6
t501	4	3163.6	0	245.9	407.7	2818.7	189.1	960.8
u120	0	29.4	0	3.4	2.6	55.7	2.8	26.2
u250	0	107.1	0	23.9	13.5	32.4	12.5	35.9
u500	5	2195.4	1	318.0	132.2	501.2	70.3	154.9
u1000	2	1911.9	0	1401.2	940.2	3335.6	437.6	1133.1

Table 3.2: Comparison of our algorithm with the algorithm of Fernandes Muritiba et al. (2010)

In Table 3.2, we report the number of unsolved instances within the time limit, denoted ¬opt, out of 90 instances (except for our algorithm with the primal heuristic that solved all instances to optimality); the average solution time; and the maximum solution time for our algorithm with primal heuristic. The time limit was 10 hours for FMIMT and 1 hour for our algorithm. Our algorithm with the primal heuristic solved all instances to optimality and it is faster by an order of magnitude than FMIMT. Using the heuristic allowed us to solve 4 more instances and it speeds up our algorithm considerably. Additionally, we observed that our root node lower bound was equal to the optimal solution for all instances but 3. All but 4 instances were solved at the root node (thanks to the primal heuristic). Note also that using specialized DP pricing oracle reduces the running time of the algorithm only for large instances (500 items and more).

Secondly, we compare the above three versions of our algorithms (without and with DP pricing or primal heuristic) to the algorithm of Elhedhli et al. (2011), which we denote ELGN. In Table 3.3, we report the number of unsolved instances within the time limit, denoted ¬opt (except for our algorithm with primal heuristic that solved all instances to optimality); the average solution time; and the maximum solution time. The algorithm ELGN includes a primal heuristic: a rounding procedure with

	ELGN		Our w/o heur.		Our w/o DP sp		Our w DP sp & heur.	
class	¬opt	av.time	¬opt	av.time	av.time	max.time	av.time	max.time
t60	0	3.2	0	0.9	0.9	8.9	1.3	7.7
t120	3	118.6	0	5.0	5.5	34.0	6.9	30.0
t249	10	398.0	4	157.1	65.0	1024.9	53.8	383.2
u120	0	47.0	0	2.4	3.2	15.4	3.7	9.4
u250	1	183.1	1	37.0	21.5	99.5	21.2	73.3
u500	13	1253.8	5	277.5	214.6	1479.9	115.2	310.4

Table 3.3: Comparison of our algorithm with the algorithm of Elhedhli et al. (2011)

no backtracking. Results of Table 3.3 indicates that our algorithm with DP pricing and primal heuristic is an order of magnitude faster. All instances were solved at the root node thanks to the primal heuristic. The root node lower bound is equal to the optimal solution for all tested instances.

	PH		DH		DH LDS	
class	gap	time	gap	time	gap	time
t60	0.45%	37.5	0.18%	0.7	0%	0.8
t120	0.62%	40.0	0.48%	3.5	0.03%	5.1
t249	0.39%	51.9	0.29%	21.3	0%	29.3
t501	0.21%	58.9	0.16%	130.3	0%	189.1
u120	0.10%	22.4	0.16%	2.3	0%	2.8
u250	0.21%	52.1	0.10%	11.8	0%	12.5
u500	0.20%	69.7	0.05%	66.2	0%	70.3
u1000	0.22%	107.8	0.02%	412.5	0%	437.6

Table 3.4: Comparison of our primal heuristics with the population heuristic of Fernandes Muritiba et al. (2010).

Fernandes Muritiba et al. (2010) have proposed a population based heuristic (PH) for the problem. It consists in a tabu search algorithm and a diversification procedure. In Table 3.4, we compare the two variants of our primal heuristic with PH: a pure diving approach (DH) without the Limited Discrepancy Search (meaning that the `maxDiscrepancy` parameter is equal to 0) and the variant used in the above test (DH with LDS) with the parameters `maxDiscrepancy` = 2 and `maxDepth` = 3. The results of Table 3.4 show that DH is faster than PH for instances with less than 500 items and produce on average significantly better solutions than the population heuristic (except for class "u120"). DH LDS is only slightly slower than the PH and produces optimal solutions for all instances except one.

In the paper, we have also proposed two sets of new instances for the problem: instances with arbitrary conflict graphs and instances with a larger number of items per bin. These instances are experimentally shown to be harder the literature ones.

3.2.4 Related work

The BPPC was first considered by Jansen (1999) who developed approximation algorithms for special cases of conflict graphs. Several computational studies on the problem have recently been published. Gendreau et al. (2004) have evaluated six heuristics and lower bounding strategies for the problem. Different heuristics, lower bounds and an exact algorithm based on a branch-and-price approach were proposed by Fernandes Muritiba et al. (2010). A special purpose branch-and-price algorithm was developed by Elhedhli et al. (2011).

Khanafer et al. (2010) developed procedures for fast calculation of lower bounds for the problem. The concepts of dual-feasible and data-dependent dual-feasible functions have been used. Brandão and Pedroso (2016) applied their general arc-flow formulation for the BPPC. Similar to us, they solved all instances with interval conflict graph. However, their results for harder instances with arbitrary conflict graph were worse than ours. Gschwind and Irnich (2016) proposed stabilized column generation algorithm based on dual inequalities to find lower bounds for the BPPC.

Recently, Bettinelli et al. (2017) improved our branch-and-bound algorithm for solving the KPC with an arbitrary conflict graph. Their algorithm uses a stronger lower bound derived from dynamic programming. Finally, Wei et al. (2019) claimed to have better results for the BPPC than our branch-and-price algorithm. They developed a branch-cut-and-price approach which uses subset-row non-robust cuts, as well as a specialized labelling algorithm to solve the pricing problem.

3.3 2D Guillotine Cutting Stock Problem with Leftovers

This section is based on our journal paper (Clautiaux et al., 2019b).

In the two-dimensional guillotine cutting-stock problem, the objective is to minimize the number of large plates used to cut a list of small rectangles. We consider a variant of this problem, which arises in glass industry when different bills of order (or batches) are considered consecutively. For practical organisation reasons, leftovers are not reused, except the large one obtained in the last cutting pattern of a batch, which can be reused for the next batch. The problem can be decomposed into an independent problem for each batch.

We focus on the one-batch problem, the objective of which is to minimize the total width of the cutting patterns used. We propose a diving heuristic based on column generation, in which the pricing problem is solved using dynamic programming (DP). This DP generates so-called non-proper columns, i.e. cutting patterns that cannot participate in a feasible integer solution of the problem. We show how to adapt the standard diving heuristic to this "non-proper" case while keeping its effectiveness. We also introduce the partial enumeration technique, which is designed to reduce the number of non-proper patterns in the solution space of the dynamic program. This technique strengthens the lower bounds obtained by column generation and improves the quality of the solutions found by the diving heuristic. Computational results are reported and compared on classical benchmarks from the literature as well as on new instances inspired from glass industry data. According to these results, variants of the proposed diving heuristic outperform constructive and evolutionary

heuristics.

3.3.1 Problem description

We study an industrial glass cutting problem related to the manufacturing of double-paned windows. The industrial process is organized as follows. First a set of large rectangular plates is received from a glass manufacturer. All plates have the same width and height, and are stored in a specialized area of the factory. Then each plate is iteratively retrieved from the storage area, put on a cutting table and cut into smaller rectangular pieces. The obtained glass pieces are then sent to the second production unit where windows are assembled and sent to customers. We focus on the first part of the process: cutting the small rectangular pieces and minimizing the wasted material.

Due to the physics of glass, each cutting pattern is made of so-called *guillotine cuts*. These cuts are made in a straight line from one border of the (sub)plate to the other. This is mandatory in the glass industry, otherwise the glass breakage process may lead to cracks that propagate through the whole glass plate. According to cutting device limitations, consider cutting patterns that emanate from a 4-stage guillotine-cut process. First-stage cuts are applied to the originale plate. Second-stage cuts are applied to subplates that are obtained by first-stage cuts, and so on. Due to the large width of plates, the first-stage cuts are always vertical. Then second, third, and fourth-stage cuts are respectively horizontal, vertical and horizontal. Only so-called *restricted cuts* are considered, i.e. cuts of length equal to an item width or height. Special requirement related to restricted cuts is that one of the two subplates is immediately initialized with an item of width or height equal to the length of the performed cut. Our initial experimental observations as well as those reported by Furini et al. (2016) for a related problem confirm that using only restricted cuts does not deteriorate the solutions in most cases. Moreover restricted cuts facilitate handling of plates by operators. An example of a valid cutting pattern is depicted in Figure 3.4.

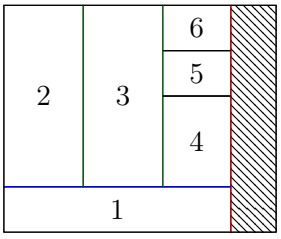
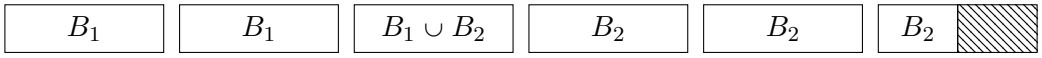


Figure 3.4: Representation of a four-stage guillotine cutting pattern with restricted cuts

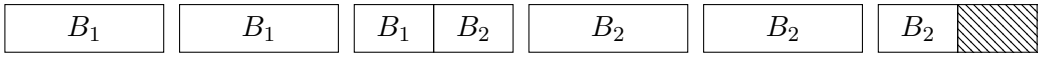
Our cutting problem has two uncommon specificities that derive from the practical industrial process described above. The first is that production is decomposed into batches. This is due to the fact that there is a limited intermediate storage area between the two production units. Thus the set of glass pieces to produce during a day is pre-decomposed into batches such that each batch fits to the storage. Batches are cut in a predetermined specific order that takes into account the due

dates of customer orders. In a given batch, the exact specified quantity of ordered glass pieces has to be cut. It is forbidden to have overproduction or underproduction. The number of plates is always sufficient to cut all ordered pieces.

The second specificity is the way leftovers are handled. There is no specific area to store leftovers from previous cutting patterns, mostly because there is no standard size for the orders, and organization costs that would be entailed are expected to be larger than the cost of the raw material saved by reusing all leftovers. Therefore, almost all leftovers are recycled, and cannot be used for subsequent batches. Only one leftover piece is kept from each batch: the one related to the last plate used. This subplate remains on the cutting device. Its height must be equal to the height of the large plates. Figure 3.5 depicts feasible/infeasible solutions. Therefore, the cost of a solution in a batch is not the number of bins used, but the total width of bins used (considering that all bins but the last are used entirely).



(a) Infeasible solution since it is not allowed to mix items from different batches in a cutting pattern



(b) Feasible solution. The leftover from batch B_1 is used to cut items from batch B_2

Figure 3.5: Representation of a solution for two batches B_1 and B_2.

We call our industrial problem *consecutive two-dimensional guillotine cutting-stock problems with leftover* (C-2DG-CSP-L). It is NP-hard since it generalizes the classical cutting-stock problem. It is also combinatorially complex and the practical instances that we need to solve are large: 10 to 15 batches each having about 150 different items, for which the total demand can be as large as 400 copies, and the plates have a large size 6000×3000 compared to the item sizes. The C-2DG-CSP-L can be decomposed into independent problems for each batch. Our paper focuses on the one-batch problem, which we call *two-dimensional guillotine cutting-stock problems with leftover* (2DG-CSP-L). A solution to the 2DG-CSP-L is depicted in Figure 3.6.

Figure 3.6: Representation of a solution for the 2DG-CSP-L corresponding to a batch B. The first plate has a smaller width, since it is the leftover from the previous batch. The right-hand part of the last plate will be re-used in the next batch.

3.3.2 Column generation approach

Let us first present our notations. The set of items is denoted I. Each item $i \in I$ is a rectangular glass piece to cut with dimensions $q_i \times h_i$, called width and height, and a demand (or number of copies to cut) equal to d_i. Each item i can be rotated, in which case, its dimensions become $h_i \times q_i$. Item set I is partitioned into an ordered list of distinct batches $(I_1, \ldots, I_B)$, which have to be processed independently in the

order given. To cut items, an unlimited number of identical glass plates (bins) of dimension $Q \times H$ are available. In the remainder of the paper, we assume that all input data are integer.

For 2DG-CSP-L, one needs to pack set I_b of items of batch $b \in \{1, \ldots, B\}$ to three type of plates: a leftover plate from batch $b-1$ of given dimension $Q' \times H$ (type 1), an unlimited number of standard plates of dimension $Q \times H$ (type 2), and a single potential leftover plate of dimension $Q \times H$ (type 3); and only a part of the width of the plate of type 3 is used (it is equals to the width of the cutting pattern assigned to it). Using these three types, the objective is to minimize the total used plate width and assuring that exactly one plate of type 1 and at most one plate of type 3 are used respectively. For the first batch ($b = 1$), we assume that $Q' = 0$, i.e. the plate of type 1 is not available.

To formulate the 2DG-CSP-L, let P^k be the set of all valid cutting patterns for a plate of type $k \in \{1, 2, 3\}$. Let $\bar{x}_i^p$ and $\bar{q}^p$ be the number of items $i \in I$ in cutting pattern $p \in P^k$ and its width. Let also λ_p be an integer variable which is equal to the number of times pattern $p \in P^k$ is used in the solution. Then the 2DG-CSP-L can be formulated as

$$\min \sum_{p \in P^1} Q' \lambda_p + \sum_{p \in P^2} Q \lambda_p + \sum_{p \in P^3} \bar{q}^p \lambda_p \tag{3.7a}$$

$$\sum_{p \in P^1 \cup P^2 \cup P^3} \bar{x}_i^p \lambda_p = d_i, \qquad \forall i \in I, \tag{3.7b}$$

$$\sum_{p \in P^k} \lambda_p = 1, \qquad \forall k \in \{1, 3\}, \tag{3.7c}$$

$$\lambda_p \in \mathbb{Z}_+, \qquad \forall p \in P^2, \tag{3.7d}$$

$$\lambda_p \in \{0, 1\}, \qquad \forall p \in P^k, k \in \{1, 3\}. \tag{3.7e}$$

Formulation (3.7) is tackled using a branch-and-price approach: at each node of a branch-and-bound tree, the linear relaxation of (3.7) is solved by column generation, as discussed in Section 1.1.

Let $(\bar{\pi}, \bar{\mu})$ be the current dual solution of the restricted master problem, where vector $\bar{\pi}$ is associated to constraints (3.7b) and vector $\bar{\mu} = (\bar{\mu}_1, \bar{\mu}_3)$ is associated to constraints (3.7c). Then, the pricing problem is to find a cutting pattern $p \in P^1 \cup P^2 \cup P^3$ that minimizes the following reduced cost

$$\bar{c}(\bar{x}^p) = -\sum_{i \in I} \bar{\pi}_i \bar{x}_i^p + \begin{cases} Q' - \bar{\mu}_1, & p \in P^1, \\ Q, & p \in P^2, \\ \bar{q}^p - \bar{\mu}_3, & p \in P^3. \end{cases} \tag{3.8}$$

Obviously, the pricing problem decomposes into three subproblems, one for each plate type. Each pricing subproblem is a bounded four-stage restricted guillotine-cut two-dimensional knapsack problem. A dynamic programming algorithm (DP) for the unbounded case of this problem was presented by us in Clautiaux et al. (2018). To be self-contained, we recall the dynamic programming recursion here.

When item rotations are allowed, set I of items is duplicated in the subproblem to include rotated item copies. The two copies of an item i share the same dual value $\bar{\pi}_i$ from the RMP, and the same production bound d_i.

Since we use restricted cuts, we consider two types of dynamic programming states: those related to a restricted cut (used to initiate a strip), and those that are used to complete a strip.

Let (q, h, s), $q \in \{1, \ldots, Q\}$, $h \in \{1, \ldots, H\}$, $s \in \{1, \ldots, 3\}$, be the state corresponding to the situation in which a rectangular part of dimension $q \times h$ of the plate is to be separated from the current plate using a guillotine cut of stage s. Let $(\overline{q, h, s})$, $q \in \{1, \ldots, Q\}$, $h \in \{1, \ldots, H\}$, $s \in \{2, \ldots, 4\}$, be the state corresponding to the same situation with the additional restriction that the next cut should obtain a single item copy. For a given state (q, h, s), let $U(q, h, s)$ be the maximum value of a configuration obtained from this state. Note that this value only depends on the dual values $\bar{\pi}$ associated to item copies, and an additional term for type 3 plate only, which is equal to the total width used. It is sufficient to account for this restricted width in stage 1 cuts only. Then $U(Q, H, 1)$ equals to the pricing subproblem optimum value (ignoring constant values in (3.8)).

Let $\mathcal{Q}(q, h)$ and $\mathcal{H}(q, h)$ be the set of all possible widths and heights of items which fit into rectangle $q \times h$.

$$\mathcal{Q}(q, h) = \bigcup_{i \in I:\ q_i \leqslant q, h_i \leqslant h} \{q_i\}, \quad \mathcal{H}(q, h) = \bigcup_{i \in I:\ q_i \leqslant q, h_i \leqslant h} \{h_i\}.$$

We now give the recursion for computing values $U(q, h, s)$ and $U(\overline{q, h, s})$ for type 2 plates.

$$U(q, h, 1) = \max \left\{ 0, \max_{q' \in \mathcal{Q}(q,h)} \left\{ U(\overline{q', h, 2}) + U(q - q', h, 1) \right\} \right\}$$

$$U(q, h, 2) = \max \left\{ 0, \max_{h' \in \mathcal{H}(q,h)} \left\{ U(\overline{q, h', 3}) + U(q, h - h', 2) \right\} \right\}$$

$$U(q, h, 3) = \max \left\{ 0, \max_{q' \in \mathcal{Q}(q,h)} \left\{ U(\overline{q', h, 4}) + U(q - q', h, 3) \right\} \right\}$$

$$U(\overline{q, h, 2}) = \max_{i \in I:\ q_i = q, h_i \leqslant h} \{ \bar{\pi}_i + U(q, h - h_i, 2) \}$$

$$U(\overline{q, h, 3}) = \max_{i \in I:\ h_i = h, q_i \leqslant q} \{ \pi_i + U(q - q_i, h, 3) \}$$

$$U(\overline{q, h, 4}) = \max \left\{ 0, \max_{i \in I:\ q_i = q, h_i \leqslant h} \left\{ \bar{\pi}_i + U(\overline{q, h - h_i, 4}) \right\} \right\}$$

In the recursive formulae for $U(q, h, s)$ and $U(\overline{q, h, 4})$, the alternative with zero value corresponds to turning the remaining rectangular part of the plate into waste.

For the type 1 plate generation, the recursions are identical, only initialization needs to be adapted to the available width. For type 3 plates, the recursive formula for $U(q, h, 1)$, becomes

$$U(q, h, 1) = \max \left\{ 0, \max_{q' \in \mathcal{Q}(q,h)} \left\{ U(\overline{q', h, 2}) - q' + U(q - q', h, 1) \right\} \right\}$$

where term $-q'$ represents the penalty for the width consumed.

3.3.3 Diving heuristic with non-proper columns

A column generation based diving heuristic can be used to find feasible solutions for the 2DG-CSP-L. The diving heuristics presented in Section 2.3 require one to use *proper* columns, *i.e.* variables that could take a non-zero value in an integer solution of the residual master problem. In our context, a variable λ_p, $p \in P^1 \cup P^2 \cup P^3$, is proper if $\bar{x}_i^p \leqslant d_i, \forall i \in I$. Therefore, in each pricing subproblem we should impose upper bounds on the number of copies of items in the cutting pattern. However when solving the pricing problem by the dynamic programming algorithm given in Section 3.3.2 we consider only the unbounded version. In the presence of upper bounds, the pricing problem becomes significantly harder to solve to optimality. A possible solution to this issue is to solve the pricing problem heuristically, using algorithms presented in our paper (Clautiaux et al., 2019b). However, our preliminary experiments showed that this approach deteriorates significantly the quality of solutions obtained by the diving heuristic.

In this work, we propose a variant of the diving heuristic which uses non-proper columns. A previous study of Cintra et al. (2008) and our preliminary experiments showed that the lower bound obtained by solving the master problem (MP) with non-proper columns is close to the one obtained when using exclusively proper columns. As the quality of diving heuristics depend mainly on the strength of the MP bound, we may then expect that a "non-proper" diving heuristic will be efficient for our problem.

Our "non-proper" diving heuristic proceeds as follows. Remember that at each iteration, the residual master problem is solved by column generation. Both proper and non-proper columns may be generated. However, the partial solution can only be augmented with proper columns. Therefore, given a fractional solution $\bar{\lambda}$, we choose a *proper* variable λ_p with a value closest to its nearest non-zero integer. If such variable exists, we proceed the same way as in the basic diving heuristic. If there is no such "*proper*" variable, we choose a column λ_p with the smallest reduced cost (with respect to the optimal dual solution of the master problem) among

- all proper columns contained in the current RMP;
- and proper columns generated by solving the bounded pricing problem by heuristic algorithms presented in our paper (Clautiaux et al., 2019b)

As usual, this column is then added to the partial solution with the value equal to the nearest non-zero integer of $\bar{\lambda}_p$. In particular, if $\bar{\lambda}_p = 0$ then $\lambda_p = 1$ is included in the partial solution.

Our preliminary experiments showed that fixing type 3 columns λ_p, $p \in P^3$, early in the search has a negative impact on the quality of solutions obtained. Therefore, we adopt the following modification. Cutting patterns $p \in P^3$ are never added to the partial solution before patterns $p \in P^1 \cup P^2$. As there is exactly one pattern $p \in P^3$ in any feasible solution, once it is added to a partial solution, the latter should become complete. Therefore, each time the partial solution is augmented with a cutting pattern of type 1 or 2, we verify heuristically whether the remaining item copies can be cut into one plate of dimension $Q \times H$. If it is possible, we produce a cutting pattern $p \in P^3$ including all remaining item copies and minimizing heuristically its

width $\bar{q}^p$. Then this pattern is used to complete the solution, and the diving heuristic terminates.

One can develop further this idea and, each time the partial solution is augmented, to formulate the residual 2DG-CSP-L and solve it with heuristics described in our paper. Calls to these heuristics are done iteratively for each plate until the residual problem instance is closed. This modification can be seen as a combination of the diving and pricing heuristics. This combined heuristic is presented in Algorithm 1. A boolean parameter *lastPlateOnly* is used to set how to evaluate the residual 2DG-CSP-L instance: building only a solution for the last plate or a complete solution to the residual 2DG-CSP-L instance.

Algorithm 1: The combined pricing heuristics and diving heuristic with non-proper columns

$P^* \leftarrow$ solution of the 2DG-CSP-L with demands d by the evolutionary algorithm
$d' \leftarrow d$, $P^{\text{part}} \leftarrow \varnothing$
repeat
  Solve the MP with upper bounds d' by column generation and obtain solution $\bar{\lambda}$
  $P^{\text{prop}} \leftarrow \{p \in P^1 \cup P^2 : \bar{\lambda}_p > 0, \bar{x}^p \leqslant d'\}$ (set of proper patterns in the solution)
  if $P^{\text{prop}} \neq \varnothing$ **then**
    $p' \leftarrow \text{argmin}_{p \in P^{\text{prop}}} \{| \bar{\lambda}_p - \lceil \bar{\lambda}_p \rfloor |\}$
  else
    $P^{\text{RMP}} \leftarrow$ set of patterns of type 1 and 2 in the RMP
    $P^{\text{heur}} \leftarrow$ set of heuristic solutions to the pricing problem of type 1 and 2
    $p' \leftarrow \text{argmin}_{p \in P^{\text{RMP}} \cup P^{\text{heur}}} \{\bar{c}_p\}$
  $P^{\text{part}} \leftarrow P^{\text{part}} \cup \{p'\}$, $d' \leftarrow d' - \bar{x}^{p'} \cdot \lceil \bar{\lambda}_p \rfloor$
  if *lastPlateOnly* $= true$ **then**
    $p^3 \leftarrow$ heuristic solution to the pricing problem of type 3
    if $d' - \bar{x}^{p^3} = \mathbf{0}$ **then**
      $d' \leftarrow \mathbf{0}$
      if $cost\big(P^{\text{part}} \cup \{p^3\}\big) < cost(P^*)$ **then** $P^* \leftarrow P^{\text{part}} \cup \{p^3\}$
  else
    $P^{\text{evol}} \leftarrow$ heuristic solution to the residual 2DG-CSP-L with demand d'
    if $cost\big(P^{\text{part}} \cup P^{\text{evol}}\big) < cost(P^*)$ **then** $P^* \leftarrow P^{\text{part}} \cup P^{\text{evol}}$
until $d' = 0$
return P^*

To introduce the Limited Discrepancy Search (LDS) in our "non-proper" diving heuristic, we need to ensure that at least one non-tabu column is produced by our pricing problem heuristic. This can be achieved by generating a sufficient number of different cutting patterns, i.e. the size of set P^{heur} in Algorithm 1 is strictly larger that the current size of the tabu list.

3.3.4 Partial enumeration

Although the diving heuristic can be adapted to handle non-proper columns, still the quality of the solutions may be decreased in comparison with the "proper" case.

In this section, we propose a modification to the dynamic program that partially takes into account upper bounds on the number of item copies in order to favour the generation of proper cutting patterns. As it will be seen from our computational results, this approach allows one to improve the quality of the lower bound obtained by solving the master problem as well as the quality of solutions produced by the diving heuristic. This comes at the cost of a slightly larger dynamic program. Nevertheless we show below that this can be controlled by a suitable configuration.

On one hand, complete enumeration of all feasible patterns would take into account the bound constraints, but the computational cost would be huge. On the other hand, the dynamic program has a reasonable computational cost, but it does not take into account the bound constraints. Our idea is to mix both approaches, by replacing some parts of DP by a partial enumeration. When one uses this technique, the set of feasible solutions of the pricing problem contains non-proper patterns, but the number of such patterns is greatly decreased. Thus the quality of lower bounds obtained by column generation is improved.

This idea is implemented using so-called *meta-items*, each one representing a partial vertical or horizontal stack of item copies satisfying production upper bounds. When restricted states $U(\overline{q,h,s})$, $s = 2,3,4$ are considered, instead of choosing one item to initiate the stripe, we choose a meta-item (or equivalently the set of items that it represents). There is potentially an exponential number of possible meta-items to initiate the stripe. Therefore, we introduce an additional parameter δ, which restricts the possible meta-items produced by only considering items whose width/height is close enough to the size of the stripe.

Formally, let $\bar{x}_i^m$ be the number of copies of item $i \in I$ included into meta-item m. Given three values $0 < q \leqslant Q$, $0 < h \leqslant H$, and $0 < \delta \leqslant \min_{i\in I} q_i$, we define the following set $\mathcal{M}^{\text{vert}}(q,h,\delta)$ of vertical meta-items. Each meta-item $m \in \mathcal{M}^{\text{vert}}(q,h,\delta)$ forms in the cutting pattern a partial vertical stack of width q containing copies of items $i \in I$ such that $q - \delta < q_i \leqslant q$ and $h_i \leqslant h$. Items that do not belong to m may only be cut in other vertical stacks or in the same stack above the item copies in m. In addition, copies of items $i \in I$ such that $\bar{x}_i^m > 0$ may only be cut in other vertical stacks. Formally:

$$m \in \mathcal{M}^{\text{vert}}(q,h,\delta) \Leftrightarrow \begin{cases} \exists i \in I, q_i = q : \ \bar{x}_i^m > 0, \\ \bar{x}_i^m > 0 \Rightarrow q - \delta < q_i \leqslant q \text{ and } h_i \leqslant h, \quad \forall i \in I, \\ \bar{x}_i^m \leqslant d_i, \quad \forall i \in I, \\ \sum_{i\in I} \bar{x}_i^m h_i \leqslant H. \end{cases}$$

The first condition ensures that one item has width q (and thus a restricted pattern is built). The second condition ensures that the size of the items in the meta-item satisfies the requested limitations. The third condition ensures that the meta-item satisfies the production bound constraints. The fourth condition ensures that the meta-item height does not exceed the plate height.

Analogously, given values $0 < q \leqslant Q$, $0 < h \leqslant H$, and $0 < \delta \leqslant \min_{i\in I} h_i$, we

define the following set $\mathcal{M}^{\text{hor}}(h,q,\delta)$ of horizontal meta-items:

$$m \in \mathcal{M}^{\text{hor}}(h,q,\delta) \Leftrightarrow \begin{cases} \exists i \in I, h_i = h: \ \bar{x}_i^m > 0, \\ \bar{x}_i^m > 0 \Rightarrow h - \delta < h_i \leqslant h \text{ and } q_i \leqslant q, \quad \forall i \in I, \\ \bar{x}_i^m \leqslant d_i, \quad \forall i \in I, \\ \sum_{i \in I} \bar{x}_i^m q_i \leqslant Q. \end{cases}$$

For each meta-item $m \in \mathcal{M}$, we define its total value $\hat{\pi}_m = \sum_{i \in I} \bar{\pi}_i \bar{x}_i^m$. For each vertical meta-item $m \in \mathcal{M}^{\text{vert}}$, we define its total height $\hat{h}_m = \sum_{i \in I} \bar{x}_i^m h_i$, and for each horizontal meta-item $m \in \mathcal{M}^{\text{hor}}$, we define its total width $\hat{q}_m = \sum_{i \in I} \bar{x}_i^m q_i$.

Note that, by definition, $\mathcal{M}^{\text{vert}}(q,h,\delta) = \varnothing$ if $q \notin \mathcal{Q}(Q,H)$, and $\mathcal{M}^{\text{hor}}(h,q,\delta) = \varnothing$ if $h \notin \mathcal{H}(Q,H)$. Suppose now that for each $q \in \mathcal{Q}(Q,H)$ a value δ_q, $0 \leqslant \delta_q \leqslant \min_{i \in I} q_i$, is fixed, and for each $h \in \mathcal{H}(Q,H)$ a value δ_h, $0 \leqslant \delta_h \leqslant \min_{i \in I} h_i$, is fixed. Then the recursive formulae for states $\overline{(q,h,s)}$ can be rewritten in the following way without loss of any proper patterns from the set of feasible solutions.

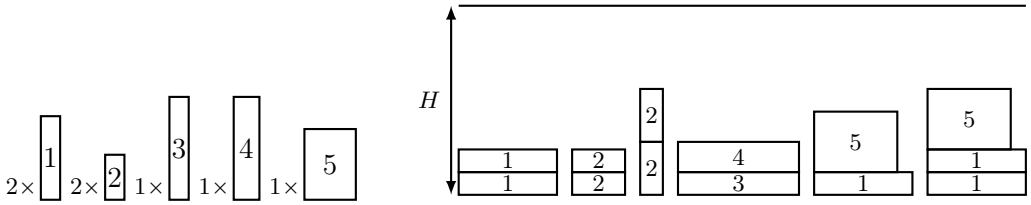
$$U(\overline{q,h,2}) = \begin{cases} \max\limits_{m \in \mathcal{M}^{\text{vert}}(q,h,\delta_q):\, \hat{h}_m \leqslant h} \left\{ \hat{\pi}_m + U(q, h - \hat{h}_m, 2) \right\}, & \text{if } \delta_q > 0, \\ \max\limits_{i \in I:\, q_i = q, h_i \leqslant h} \left\{ \bar{\pi}_i + U(q, h - h_i, 2) \right\}, & \text{if } \delta_q = 0, \end{cases}$$

$$U(\overline{q,h,3}) = \begin{cases} \max\limits_{m \in \mathcal{M}^{\text{hor}}(h,q-\delta_q,\delta_h):\, \hat{q}_m \leqslant q} \left\{ \hat{\pi}_m + U(q - \hat{q}_m, h, 3) \right\}, & \text{if } \delta_h > 0, \\ \max\limits_{i \in I:\, h_i = h, q_i \leqslant q - \delta_q} \left\{ \bar{\pi}_i + U(q - q_i, h, 3) \right\}, & \text{if } \delta_h = 0, \end{cases}$$

$$U(\overline{q,h,4}) = \begin{cases} \max\left\{ 0, \max\limits_{m \in \mathcal{M}^{\text{vert}}(q,h-\delta_h,1)} \{\hat{\pi}_m\} \right\}, & \text{if } \delta_q > 0, \\ \max\left\{ 0, \max\limits_{i \in I:\, q_i = q, h_i \leqslant h - \delta_h} \left\{ \bar{\pi}_i + U(\overline{q, h - h_i, 4}) \right\} \right\} & \text{if } \delta_q = 0, \end{cases}$$

If all values δ are fixed to zero, the modified dynamic program reduces to the original one presented in Section 3.3.2. The larger the values δ are, the fewer non-proper cutting patterns are generated and, at the same time, the larger is the number of meta-items. So, there is a trade-off between the complexity (or the size) of the dynamic program and the strength of the approximation of the space of proper cutting patterns by the space of feasible solutions of the dynamic program. We parametrize this trade-off by defining thresholds $\Delta^{\text{size}} \geqslant 0$ on the size of the sets of meta-items and $\Delta^{\text{diff}} \geqslant 0$ on values δ, respectively for dimension q and h. Given these thresholds, values δ are determined the following way. For each $q \in \mathcal{Q}(Q,H)$, we set δ_q to the largest value $\delta \leqslant \min\left\{\Delta_q^{\text{diff}}, \min_{i \in I} q_i\right\}$ such that $\mid \mathcal{M}^{\text{vert}}(q,H,\delta) \mid \leqslant \Delta_q^{\text{size}}$. As $\mathcal{M}^{\text{vert}}(q,H,0) = \varnothing$ for any q, such value δ always exists. Analogously, for each $h \in \mathcal{H}(Q,H)$, we set δ_h to the largest value $\delta \leqslant \min\left\{\Delta_h^{\text{diff}}, \min_{i \in I} h_i\right\}$ such that $\mid \mathcal{M}^{\text{hor}}(h,Q,\delta) \mid \leqslant \Delta_h^{\text{size}}$. Note again that $\mathcal{M}^{\text{hor}}(h,Q,0) = \varnothing$ for any h. The sets of meta-items are computed by enumeration.

To illustrate the impact of partial enumeration on the column generation lower bound, let us consider the instance depicted in Figure 3.7. We suppose that the width of the plate is large enough to cut all demanded pieces. Thus the objective is to minimize the width of the cutting pattern. In Figure 3.8a, we show the fractional solution obtained by solving the master problem by column generation in which



(a) Pieces and their demands (b) All vertical meta-items different from a single item piece

Figure 3.7: An illustrative instance

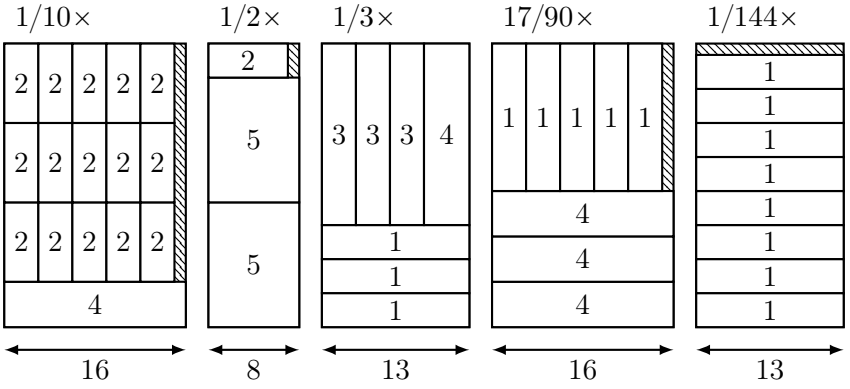
the pricing problem is solved by the standard dynamic programming. All patterns participating in this solution are non-proper, and most of these patterns are "highly" non-proper, i.e. the number of cut copies exceeds largely the demand. This solution's value is 13.046, which gives a lower bound on the optimal value. In Figure 3.8b we present the solution of the master problem involving only variables corresponding to patterns generated by the dynamic programming with partial enumeration. Only one pattern in the solution is non-proper, and the number of cut copies exceeds the item demand only by one. The value of the lower bound obtained by column generation is increased to 14.333, thus cutting 66% of the gap with the optimal solution of value 15 depicted in Figure 3.8c.

3.3.5 Computational results

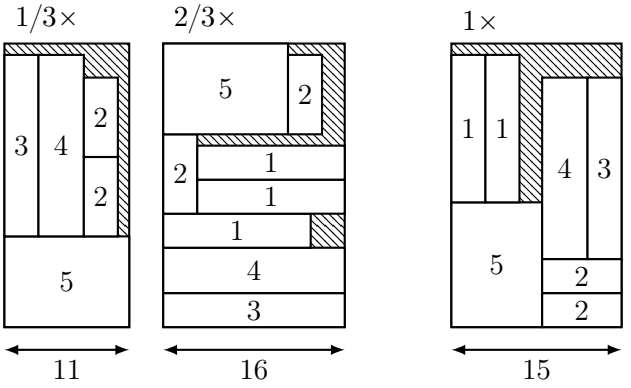
The computational tests were performed on two groups of instances: literature and industrial ones. Their description are given in our paper.

We first estimate the impact of partial enumeration on the column generation lower bound and on the quality of the diving heuristic. We characterize the configuration of partial enumeration in the pricing problem with four values: Δ_q^{size}, Δ_h^{size}, Δ_w^{diff}, and Δ_h^{diff}. We consider here three settings. The first corresponds to the standard dynamic program with no enumeration: $\Delta_1 = (\Delta_q^{\text{size}} = 0, \Delta_h^{\text{size}} = 0, \Delta_q^{\text{diff}} = 0, \Delta_h^{\text{diff}} = 0)$. The second corresponds to enumerating items with the same height h_i for odd cutting stages and the same width q_i for even cutting stages: $\Delta_2 = (\Delta_q^{\text{size}} = 1000, \Delta_h^{\text{size}} = 1000, \Delta_q^{\text{diff}} = 1, \Delta_h^{\text{diff}} = 1)$. Our third setting corresponds to enumerating items with the same height h_i for odd cutting stages and enumerating items with different width q_i for the second stage: $\Delta_3 = (\Delta_q^{\text{size}} = 1000, \Delta_h^{\text{size}} = 1000, \Delta_q^{\text{diff}} = \infty, \Delta_h^{\text{diff}} = 1)$. The size of the dynamic program corresponding to configuration Δ_3 increases by at most 33% in comparison with the size of the standard dynamic program.

When partial enumeration is applied, many non-proper patterns are excluded from the solution space of the pricing problem. Therefore, the lower bound obtained by column generation may be improved. In the next set of experiments we computationally estimate this improvement. We have implemented four variants of column generation. The first one, denoted as cg_{MIP}, uses a MIP flow formulation to solve the pricing problem and generates only proper columns. Thus, the best possible "proper" lower bound is obtained at the expense of a large solution time. The three



(a) Fractional solution with standard dynamic programming



(b) Fractional solution with partial enumeration

(c) Optimal solution

Figure 3.8: Impact of partial enumeration on the master problem solution

other variants use dynamic programming with partial enumeration to solve the pricing problem. We denote them as cg_{Δ_1}, cg_{Δ_2}, and cg_{Δ_3} depending on the partial enumeration configuration.

Results are reported in Table 3.5. The first column reports the instance class and its total number of instances between brackets. Class name C corresponds to literature instances, R corresponds to industrial ones. The number after I gives the number of items $|I|$. The next three columns present results for the variant cg_{MIP}: number of instances for which the column generation converged within one hour, average time (t) in seconds, and the average primal-dual gap (gap) in percentage from the best known solution. Next three columns give the average gap (gap_{comp}) for other three variants of column generation. In order to have a correct comparison, averages in columns gap_{comp} are calculated only for instances for which the variant cg_{MIP} converged. Note that the column generation variants with dynamic programming converged within the time limit for all instances. Thus in columns gap_{all} and t_{all}, we give the average gap and the average solution time among all instances for all configurations.

From computational results reported in Table 3.5, it can be seen that dynamic

	cg_{MIP}			gap_{comp},%			gap_{all}, %			t_{all}		
Instances	*#opt*	t	*gap*, %	cg_{Δ_1}	cg_{Δ_2}	cg_{Δ_3}	cg_{Δ_1}	cg_{Δ_2}	cg_{Δ_3}	cg_{Δ_1}	cg_{Δ_2}	cg_{Δ_3}
C-I20 (100)	62	249.0	1.99	3.82	3.14	3.04	3.79	3.07	3.00	0.3	0.3	0.3
C-I40 (100)	55	1358.1	1.15	1.86	1.53	1.52	2.73	2.39	2.38	1.4	1.4	1.4
C-I60 (100)	23	1047.0	0.68	1.00	0.79	0.79	1.82	1.57	1.57	3.0	3.1	3.1
C-I80 (100)	16	506.4	0.60	0.88	0.65	0.65	1.48	1.29	1.29	5.3	5.5	5.5
C-I100 (100)	14	503.3	0.59	0.69	0.64	0.64	1.27	1.14	1.14	9.0	9.4	9.3
R-I25 (45)	20	1486.0	0.66	0.96	0.74	0.72	1.98	1.69	1.65	1.0	1.0	1.0
R-I50 (45)	4	1879.5	0.60	0.60	0.60	0.60	1.30	1.21	1.20	4.8	4.7	4.9
R-I100 (45)	0	-	-	-	-	-	0.63	0.59	0.58	26.6	25.2	27.8

Table 3.5: Comparison of different column generation variants

programming is orders of magnitude faster than MIP for solving the pricing problem. Partial enumeration allows one to obtain a lower bound which is closer to the "proper" lower bound, at least for the easiest instances that can be tackled by cg_{MIP}. We were not able to determine how close is the lower bound obtained by cg_{Δ_3} to the "proper" bound for larger and harder instances, as the latter bound is very time consuming to obtain. Application of partial enumeration significantly increases the quality of lower bounds obtained by column generation at almost no cost. Therefore, it offers a good trade-off between quality of lower bounds and total running time. It can also be seen that column generation is slower for industrial instances. This is expected as the running time of the dynamic program depends on the plate size, which is larger for the instances in the second group.

In the next experiment, we compare five heuristics for the 2DG-CSP-L:

(i) evolutionary algorithm (*ea*);

(ii) algorithm (*iub*), which combines evolutionary algorithm with list heuristics;

(iii) diving heuristic denoted ($div_{\varnothing}$) without partial enumeration in the pricing problem and with simple evaluation of the residual problem in the diving (parameter $lastPlateOnly = true$);

(iv) diving heuristic denoted (*div*) with partial enumeration Δ_3 in the pricing problem and with simple evaluation of the residual problem in the diving (parameter $lastPlateOnly = true$);

(v) combination of the diving heuristic and the evolutionary algorithm with complete evaluation of the residual problem in the diving (parameter $lastPlateOnly = false$), which we denote as (*ediv*).

The first two heuristics are presented in our paper (Clautiaux et al., 2019b). The evolutionary heuristic is similar to Hadjiconstantinou and Iori (2007). We also consider variants with Limited Discrepancy Search with parameters `maxDiscrepancy` = 3 and `maxDepth` = 2 for the last two heuristics and denote them as (div_{32}) and ($ediv_{32}$). Note that diving heuristics are always initialized with the solution produced by heuristic (*ea*).

In Table 3.6, we report the average gap in percentage from the best known solution and the average time in seconds.

	ea		iub		$div_{\varnothing}$		div		div_{32}		$ediv$		$ediv_{32}$	
Instances	gap	t	gap	t	gap	t	gap	t	gap	t	gap	t	gap	t
C-I20 (100)	3.47	1	2.11	1	2.09	1	2.26	1	1.73	1	1.87	1	1.64	1
C-I40 (100)	3.64	1	1.79	4	1.59	2	1.42	2	0.55	4	0.86	3	0.37	8
C-I60 (100)	3.68	1	1.80	13	1.13	3	1.09	4	0.39	9	0.55	8	0.21	27
C-I80 (100)	4.14	2	1.94	32	0.99	6	0.78	6	0.19	17	0.36	19	0.07	79
C-I100 (100)	4.47	4	1.84	65	0.75	10	0.56	10	0.15	28	0.32	36	0.02	170
average	3.88	2	1.90	23	1.31	5	1.22	5	0.60	12	0.79	14	0.46	57
R-I25 (45)	2.66	1	2.56	2	1.73	2	1.40	2	0.91	5	1.17	2	0.75	7
R-I50 (45)	1.93	2	1.83	16	1.02	8	0.54	10	0.17	53	0.33	19	0.01	112
R-I100 (45)	1.51	8	1.37	175	0.61	48	0.30	55	0.08	369	0.18	155	0.00	1224
average	2.03	4	1.92	65	1.12	20	0.75	23	0.38	143	0.56	59	0.26	448

Table 3.6: Comparison of heuristics for the 2DG-CSP-L

From Table 3.6 one can see that diving algorithms clearly outperform the first two heuristics. Heuristic (ea) is the fastest but also produces solutions of the worst quality. Heuristic (iub) improves the solution quality at the expense of much larger running time. However, it struggles with real-life instances, as the solution improvement over (ea) for them is very small. Diving algorithms ($div_{\varnothing}$) and (div) significantly outperform heuristic (iub) both in terms of running time and solution quality. Partial enumeration improves the effectiveness of the diving heuristic for most instances at a very small cost. This technique is especially useful for large instances. The combination ($ediv$) of the diving heuristic and the evolutionary algorithm further improves the quality of the obtained solutions at a cost of a reasonable increase of running time. The best solutions on average are obtained by diving heuristics with Limited Discrepancy Search. However, the running time of these heuristics is quite long especially for large real-life instances. Thus, to our opinion, the heuristics (div) and ($ediv$) offer the best trade-off between solution quality and running time.

In our paper, we have also compared the above heuristics on a set of industrial instances for the multi-batch problem C-2DG-CSP-L. The detailed results are presented in our paper. From these results, it stems that heuristic (ea) is faster than the other heuristics. More expensive heuristic (iub) improves on (ea) only marginally. Much better results are obtained by diving heuristics. The standard diving heuristic (div) saves up to 1.4% of plates on average. The extended diving heuristic ($ediv$) saves up to 1.5% of plates on average. Moreover the gap with the lower bound is at most 0.7% of plates on average using (div). For ($ediv$) this drops to 0.6% on average. In our opinion, heuristic (div) offers the best "solution quality – running time" trade-off. Even if its running time reaches 2 hours for the largest instances, its application in practice is still realistic. These instances correspond to a one day planning horizon. Therefore, spending two hours to obtain a solution seems to be reasonable.

Our diving heuristic with non-proper columns is generic and could be applied to other two-dimensional guillotine cutting-stock problems, for example with different number of stages or with non-restricted cuts. For this, one needs to adapt the dynamic program to solve pricing problem. It would be interesting to see how the heuristic running time evolves for such problem variants. Generalisation to the case

with plate defects is especially useful for practical purposes. However, this generalisation is not straightforward. In the future, we plan to study the whole industrial process in the glass factory, including inventory and batching decisions. This problem would include batch scheduling, inventory management and cutting problems, and would be a real challenge for our methodologies.

3.3.6 Related work

The first study on 2D packing problems appeared in Gilmore and Gomory (1965). Therein, the 2D bin packing problem was solved by column generation and a dynamic programming algorithm for the pricing problem. For the three-stage version of the problem, Vanderbeck (2001) used a nested decomposition, solving the pricing problem using Dantzig-Wolfe reformulation. When the pricing problem is too hard to solve, a level approach can be used instead as outlined by Puchinger and Raidl (2007). It combines different methods such as heuristics, meta-heuristics or ILP models. Another approach based on a large ILP model has been developed by Macedo et al. (2010).

Column generation or pattern based heuristics have also been proposed. Alvarez-Valdes et al. (2002) used a simple and a more elaborated ad-hoc rounding heuristics. The same authors also used a truncated restricted master heuristic. Furini et al. (2012) employed pure diving, restricted master, and diving with sub-MIPing heuristics, if one uses the terminology defined in Section 2.3.1. Since the pricing problem is time consuming, the previous authors used heuristics to solve it most of the time. Finally, Cintra et al. (2008) worked with column generation approach with non-proper columns. They solved the residual problem obtained after initial rounding using an ad-hoc heuristic.

Recurrence relations for dynamic programming to solve the 2D guillotine knapsack problem (2DG-KP) were initially introduced by Beasley (1985). Dolatabadi et al. (2012) combined dynamic programming and implicit enumeration of patterns to solve the 2DG-KP problem with unlimited number of stages. In Clautiaux et al. (2018), we proposed a hypergraph based label setting algorithm for the the 2DG-KP.

A limited number of works consider leftovers. An application of the cutting-stock problem with leftovers to glass cutting was treated by Puchinger et al. (2004). The authors have designed heuristics, meta-heuristics and heuristic branch-and-bound methods to solve the problem. Recently, similar approaches have been proposed by Dusberger and Raidl (2015). They developed a Variable Neighbourhood Search based on a Ruin-and-Recreate principle. Andrade et al. (2016) proposed direct MILP models. We are not aware of existing works studying the variant of the problem with leftovers and four stage cutting.

3.4 Multi-Activity Tour Scheduling

This section is based on our journal paper (Gérard et al., 2016).

In this paper, we address a multi-activity tour scheduling problem with time varying demand. The objective is to compute a team schedule for a fixed roster

of employees in order to minimize the over-coverage and the under-coverage of different parallel activity demands along a planning horizon of one week. Numerous complicating constraints are present in our problem: all employees are different and can perform several different activities during the same day-shift, lunch breaks and pauses are flexible, demand is given for 15 minutes periods. Employees have feasibility and legality rules to be satisfied, but the objective function does not account for any quality measure associated with each individual's schedule. More precisely, the problem mixes simultaneously days-off scheduling, shift scheduling, shift assignment, activity assignment, pause and lunch break assignment.

To solve this problem, we developed four methods: a compact Mixed Integer Linear Programming model, a branch-and-price like approach with a nested dynamic program to solve heuristically the subproblems, a diving heuristic and a greedy heuristic based on our subproblem solver. The computational results, based on both real cases and instances derived from real cases, demonstrate that our methods are able to provide good quality solutions in a short computing time. Our algorithms are now embedded in a commercial software, which is already in use in a mini-mart company.

3.4.1 Problem description

Employee scheduling is an important issue in retail as personnel wages account for a large part of their operational costs. This problem raises considerable computational difficulties, especially when certain factors are considered, such as employee availability, fairness, strict labor rules, highly variable work demand, mixed full and part-time contracts, etc. In this work, we study a real-life multi-activity tour scheduling problem with highly heterogeneous employees and flexible working hours. Given a fixed set of employees, the objective is to construct their work schedule or planning that minimizes the distance to the ideal coverage of the demand. Numerous complicating factors described in the literature are taken into account and, to the best of our knowledge, this paper is one of the first attempts, in parallel with Restrepo et al. (2016), to combine days-off scheduling, shift scheduling, shift assignment, activity assignment, pause and lunch break assignment.

The problem consists in scheduling a fixed workforce to maximize the fit to a given time-varying demand. The planning horizon consists of D consecutive days. Each day is divided into the same number of successive time periods of equal length (15 minutes in this paper). Set T represents the different time periods in the discrete planning horizon. The set of heterogeneous employees is denoted by K.

The whole set of activities that employees can carry out is divided into two distinct groups: set A of *production activities* related to work demands, and set E of *pause activities* related to non-productive activities. In our retail context, a production activity can represent, for example, the welcome desk, a cash desks line or a meat counter. Each employee $k \in K$ has a set of production activities $A^k(t)$ that he/she can perform at time period t. Set $E^k(t)$ contains a pause if employee can take it at time period t; this set is empty otherwise. The beginning and the length of a pause are strictly constrained by the personalized pause policy of the company agreement. An employee k is unavailable at time period t if $A^k(t) \cup E^k(t) = \varnothing$.

In this case, the planning computed for employee k cannot contain any activity at time t. Note that if an employee is unavailable the entire day, then a day-off has to be scheduled. Some employees may be pre-assigned to activities for certain time periods. In this case, finding a schedule that respects this pre-assigned tasks is a part of the problem.

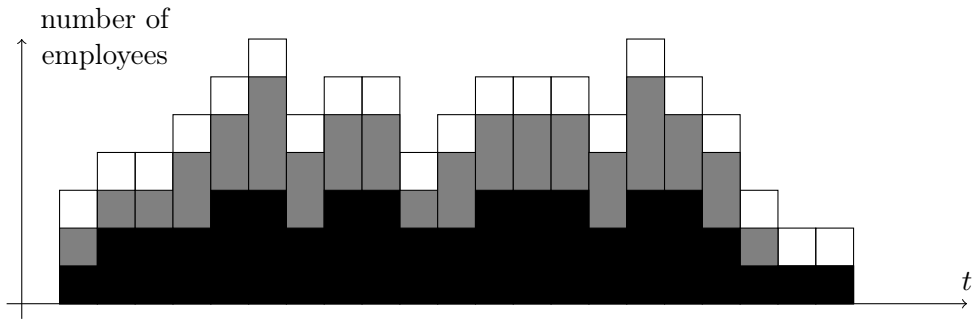


Figure 3.9: Representation of the workload for a production activity : the ideal number of employees required to cover the demand is in gray, the thresholds of critical undercoverage and overcoverage are given respectively in black and white.

The work demand DE_{at} represents the ideal number of employees needed to realize production activity a in the best possible conditions during time period t: see the representation given in Figure 3.9. Satisfying exactly the demand is not mandatory : in most cases it is not possible. In this case, either an under-coverage, or an over-coverage is produced. Furthermore, if over-coverage (respectively under-coverage) exceeds the given threshold OV_{at} (respectively UN_{at}), then it becomes critical and indicates that too many (respectively too few) employees have been assigned to activity a during time period t.

Our objective is to construct a feasible team schedule that minimizes the sum of the over-coverage and under-coverage costs for the whole planning horizon and all production activities.

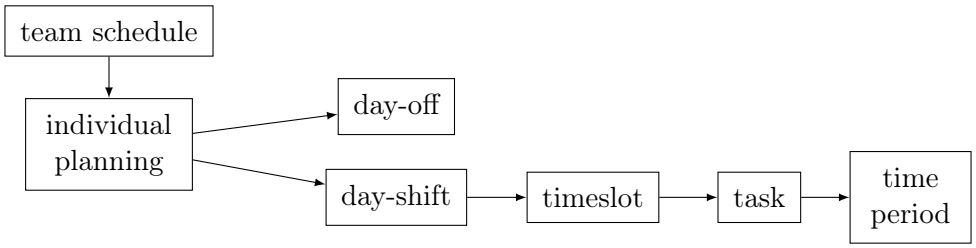


Figure 3.10: Hierarchical structure of a team schedule.

A feasible solution follows a hierarchical structure shown in Figure 3.10. For each level of the hierarchy, there is an associated set of constraints. This flexible structure does not rely on the use of a pre-computed day-shift or individual planning library, since the number of possibilities is far too large.

- *team schedule* consists of a set of $|K|$ valid employee plannings.

- An *individual planning* for employee k is a set of successive *day-shifts* and *days-off* over a week. Two consecutive day-shifts are separated by a *rest break*.
- A *day-off* represents a special day when employee k does not participate in any activity. Deciding whether or not an employee takes a day-off is part of the optimization process (but some days-off are mandatory if the employee is unavailable).
- A *day-shift* consists of one *timeslot* or two timeslots separated by a *lunch break*.
- A *timeslot* is a non-empty sequence of *tasks* where different activities are carried out successively and continuously. Two consecutive tasks cannot be related to the same activity. The set of possible beginning times of all timeslots of employee k is denoted as B^k. This set contains disjoint intervals, some of them are for the first timeslot of a day, others are for the second.
- A *task* is a time interval where a single activity a is performed over contiguous time periods. Activity a can be either a production activity or a pause.

In this work, we take into account constraints that we have encountered in real-life customer contexts. Each employee has his own set of planning constraints and each constraint has its own parameters. At each level of the team schedule hierarchy, duration and numerical constraints have to be satisfied. In Table 3.7, we list these constraints grouped by levels of the hierarchy. Note that *duration* of entities possibly include breaks (pauses and lunches), whereas *working time* equals to the "net duration" that excludes the breaks. Furthermore, an important feature in this problem is that each employee has a target of weekly working time LE_e that must be met exactly.

We stress the fact that each employee is different: he/she has his own skills, potential pre-assignment and availability for each time period, etc. A day-shift designed for an employee e is not likely to be valid for another employee e'.

Pauses are not included in the working time. There is at most one pause assigned per timeslot. The pause is assigned if and only if the duration of the timeslot is at least four hours (including the pause duration). A pause must be located in the second third of its timeslot, and its duration is exactly one time period. Some pauses can be initially set at some time periods as pre-assignment constraints. In our settings, each pause is positioned inside an existing task. The two parts of task before and after the pause are considered as a unique task, i.e. the two constitute a single task with one begining and one end. Note that pauses are different from lunch breaks in our models: a lunch break separates the day-shift into two timeslots.

3.4.2 Solution approaches

Let P^k denote the set of individual plannings (or columns) for employee k. Each planning $p \in P^k$ is represented by a binary matrix $\bar{x}^p$, where $\bar{x}^p_{at} = 1$ if and only if employee k is assigned to activity a at time period t in planning p.

A binary variable λ_p, $p \in P^k$, determines whether individual planning p is chosen for employee k. Continuous variables y^+_{at}, y^-_{at}, $y^{\mathrm{crit}+}_{at}$, and $y^{\mathrm{crit}+}_{at}$, $t \in T$, $a \in A$, repre-

A task of employee $k \in K$ performing activity $a \in A$	
duration	$\in [\mathrm{DK}_a^{k-}, \mathrm{DK}_a^{k+}]$
A timeslot of employee $k \in K$ beginning at time b	
beginning time b	$\in B^k$
finishing time	$\in [\mathrm{FO}_b^{k-}, \mathrm{FO}_b^{k+}]$
number of tasks	$\in [\mathrm{NO}_b^{k-}, \mathrm{NO}_b^{k+}]$
A day-shift of employee $k \in K$ on day d	
beginning time	$\in [\mathrm{BD}_d^{k-}, \mathrm{BD}_d^{k+}]$
finishing time	$\in [\mathrm{FD}_d^{k-}, \mathrm{FD}_d^{k+}]$
working time	$\in [\mathrm{LD}_d^{k-}, \mathrm{LD}_d^{k+}]$
duration	$\leqslant \mathrm{DD}_d^{k+}$
number of timeslots	$\in [1, 2]$
rest time between timeslots	$\in [\mathrm{RD}_d^{k-}, \mathrm{RD}_d^{k+}]$
minimum working time of at least one timeslot	$\geqslant \mathrm{TD}_d^{k-}$
A weekly individual planning of employee $k \in K$	
target working time	$= \mathrm{LE}^k$
number of day-shifts	$\in [\mathrm{NE}^{k-}, \mathrm{NE}^{k+}]$
number of consecutive day-shifts	$\leqslant \mathrm{ME}^{k+}$
rest time between consecutive day-shifts	$\geqslant \mathrm{RE}^{k-}$

Table 3.7: Planning constraints over an horizon of one week

sent, respectively, over-coverage, under-coverage, critical over-coverage, and critical under-coverage of the demand of activity a at time period t.

The cost function is piecewise linear. It depends on slack variables related to demand constraints. For a given solution $\bar{\lambda}$ and a given production activity a and a time period t, the coverage of the demand can be computed as $\mathrm{DE}_{at} - \sum_{k \in K} \sum_{p \in P^k} \bar{x}_{at}^p \bar{\lambda}_p$. We distinguish over-coverage y_{at}^+ (resp. under-coverage y_{at}^-) from *critical* over-coverage $y_{at}^{\mathrm{crit}+}$ (resp. *critical* under-coverage $y_{at}^{\mathrm{crit}-}$) that occurs when the over-coverage (resp. under-coverage) is greater than OV_{at} (resp. UN_{at}). When critical over/under-coverage is reached, a larger unit cost has to be paid.

The problem can be formulated as follows:

$$\min \sum_{a \in A} \sum_{t \in T} \mathrm{CO}_a y_{at}^+ + \mathrm{CO}_a^{\mathrm{crit}} y_{at}^{\mathrm{crit}+} + \mathrm{CU}_a y_{at}^- + \mathrm{CU}_a^{\mathrm{crit}} y_{at}^{\mathrm{crit}-} \tag{3.9a}$$

$$\text{s.t.} \sum_{k \in K} \sum_{p \in P^k} \bar{x}_{at}^p \lambda_p - (y_{at}^+ + y_{at}^{\mathrm{crit}+}) + (y_{at}^- + y_{at}^{\mathrm{crit}-}) = \mathrm{DE}_{at}, \quad \forall t \in T, \forall a \in A, \tag{3.9b}$$

$$\sum_{p \in P^k} \lambda_p = 1, \qquad \forall k \in K, \tag{3.9c}$$

$$\mathrm{UN}_{at} \geqslant y^{-}_{at} \geqslant 0, \qquad \forall t \in T, \forall a \in A, \tag{3.9d}$$

$$\mathrm{OV}_{at} \geqslant y^{+}_{at} \geqslant 0, \qquad \forall t \in T, \forall a \in A, \tag{3.9e}$$

$$y^{\mathrm{crit}+}_{at}, y^{\mathrm{crit}-}_{at} \geqslant 0, \qquad \forall t \in T, \forall a \in A, \tag{3.9f}$$

$$\lambda_p \in \{0,1\}, \quad \forall k \in K, p \in P^k. \tag{3.9g}$$

The piecewise objective function (3.9a) minimizes the total cost of over-coverage and under-coverage over the planning horizon and production activities. Constant values $\mathrm{CO}_a \in \mathbb{R}_+$ and $\mathrm{CU}_a \in \mathbb{R}_+$ represent, respectively, the unitary costs of over-coverage and under-coverage for production activity a. Constant values $\mathrm{CO}^{\mathrm{crit}}_a \in \mathbb{R}_+$ and $\mathrm{CU}^{\mathrm{crit}}_a \in \mathbb{R}_+$ represent respectively the costs of critical over-coverage and under-coverage for production activity a. Critical over-coverage and critical under-coverage have larger costs: $\mathrm{CU}_a < \mathrm{CU}^{\mathrm{crit}}_a$ and $\mathrm{CO}_a < \mathrm{CO}^{\mathrm{crit}}_a$. Constraints (3.9b) link the decision variables and calculate the gap between the produced work and the work demand DE_{at} for each time period and each production activity. Constraints (3.9c) assign exactly one individual planning to each employee $k \in K$.

Formulation (3.9) is tackled using a branch-and-price approach: at each node of a branch-and-bound tree, the linear relaxation of (3.7) is solved by column generation, as discussed in Section 1.1.

The pricing problem decomposes into $|K|$ independent subproblems (one for each employee). Let $\bar{\pi}$ be the vector of dual values related to master problem constraints (3.9b) and $\bar{\mu}$ be the vector of dual values related to master problem constraints (3.9c). The subproblem for employee k consists in finding a feasible individual planning $\bar{x}^p$, $p \in P^k$, with the minimum reduced cost

$$\bar{c}(\bar{x}^p) = -\bar{\mu}_k - \sum_{t \in T} \sum_{a \in A} \bar{\pi}_{at} \bar{x}^p_{at}. \tag{3.10}$$

The heuristic dynamic programming algorithm to solve the pricing subproblems is given below in Section 3.4.3.

We now describe briefly three heuristic algorithms we propose to solve the problem. Their details are given in our paper.

The first algorithm is the heuristic branch-and-price (as the pricing problem is solved by a heuristic). In it we branch on a triple (k, a, t), i.e. on whether employee k performs activity a in period t or not.

The second algorithm is an application of the pure diving heuristic presented in Section 2.3.2. To decrease the running time, we introduce the time limit for column generation at each node of the diving heuristic. When this time limit is reached, we use the current master solution values for fixing the next column, even if this solution is not optimal.

The third algorithm is the greedy heuristic. The employee plannings are computed by our nested dynamic program presented below in Section 3.4.3. The plannings are individually generated one by one and added iteratively to the solution. The objective function of the subproblems is based on the residual work demand RE_{at}, which corresponds to the remaining work demand, taking into account the individual plannings already in the current partial solution. Each time an individual planning is computed, the residual work demand RE_{at} is updated, and the method

is run again with the remaining set of employees. At initialization, RE_{at} have the same value as the work demand DE_{at}. In the objective function of the subproblem for employee e, the cost $\bar{\pi}_{at}$ determines whether activity a is performed during time period t. This cost is calculated by the relation:

$$\bar{\pi}_{at} = \begin{cases} -\mathrm{RE}_{at}/\mathrm{DE}_{at} - 1, & \text{if } \mathrm{RE}_{at} \geqslant 0, \\ -\mathrm{RE}_{at}/\mathrm{DE}_{at}, & \text{otherwise.} \end{cases}$$

3.4.3 Nested dynamic program for the pricing problems

Pricing subproblems are solved by the same algorithm for all employees. In this section, we drop index k to simplify the presentation, i.e. we consider that employee k is fixed.

The pricing problem can be modelled as a resource constrained shortest path problem (Irnich and Desaulniers, 2005) or using dynamic program based on context-free grammars as in (Côté et al., 2013). In our paper (Gérard et al., 2016) we discuss limits of these approaches applied to our problem. Finally, we adopt an alternative approach based on a nested dynamic program. It uses the following specific structure of our problem.

- There are a large number of resource constraints, but only a subset of them are active at a given node.
- Many paths share identical subpaths. Due to the hierarchical structure of the planning, the best day-shift for a given day is likely to be used in many non-dominated partial solutions.

A similar nested dynamic programming approach is described by Dohn and Mason (2013) to find the best individual plannings in a nurse rostering problem by using 3 levels and 2 segmentations.

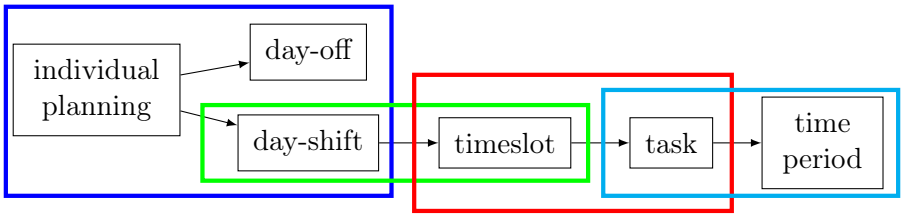


Figure 3.11: Nested dynamic programming segmentation.

We have adapted the nested method to the specific features of our problem by using 5 levels and 4 segmentations illustrated in Figure 3.11. We build an individual planning by combining day-shifts constituted by one or several timeslots, themselves composed of tasks. To manage easily path dominance rules and symmetries, the dynamic programming algorithm is segmented into several sub-problems according to the hierarchical structure of the planning. At each level, the design of a given entity consists in combining the valid entities of the level immediately below.

Let $\alpha(b, f, a)$ be the reduced cost of the task in which the employee starts activity $a \in A$ at period b, and finishes it at period f. Note that this task is valid for the employee, if it respects the duration bounds and employee skills and pre-assignments. We set the reduced cost of an invalid task to $+\infty$. Then, the formula for the reduced cost calculation is:

$$\alpha(b, f, a) = \begin{cases} -\sum_{t=b}^{f} \bar{\pi}_{at}, & \text{if } f - b + 1 \in [\mathrm{DK}_a^-, \mathrm{DK}_a^+],\ a \in A(t), \forall t \in [b, f], \\ +\infty, & \text{otherwise.} \end{cases} \tag{3.11}$$

Let $\beta(b, f, n, a)$ be the best reduced cost of a partial timeslot, which starts at period b, finishes at period f, contains a sequence of n consecutive tasks, the first of which does not perform activity a. The following recursion formula is used for the reduced cost calculation:

$$\beta(b, f, n, a) = \begin{cases} \min\limits_{a' \neq a} \{\alpha(b, f, a')\}, & \text{if } n = 1, \\ \min\limits_{\substack{f' \in [b, f-1], \\ a' \neq a}} \{\alpha(b, f', a') + \bar{\beta}(f' + 1, f, n - 1, a')\}, & \text{otherwise.} \end{cases}$$

We denote $\beta(b, d, n, -)$ the best reduced cost without imposing the constraint on the first task activity:

$$\bar{\beta}(b, f, n, -) = \min_{a \in A} \beta(b, f, n, a).$$

Let now $\gamma(b, f)$ be the best reduced cost of a complete timeslot, which starts at period b and finishes at period f. Note that this timeslot is valid for the employee, if its starting time is in B, it respects the completion and duration bounds, and the bounds on the number of tasks it contains. We set the reduced cost of an invalid timeslot to $+\infty$. Then, the formula for the reduced cost calculation is:

$$\gamma(b, f) = \begin{cases} \min\limits_{n \in [\mathrm{NO}_b^-, \mathrm{NO}_b^+]} \beta(b, d, n, -), & \text{if } b \in B,\ f \in [\mathrm{FO}_b^-, \mathrm{FO}_b^+], \\ +\infty, & \text{otherwise.} \end{cases} \tag{3.12}$$

Note that at this moment the pause policy may not be respected, as until now pauses are not included in timeslots. After calculating values γ, every timeslot without a pause and lasting more than four hours is replaced by one timeslot with a pause. For practical purposes, this is done in a greedy manner: we put the pause to a period in the second third of the timeslot such that its reduced cost is minimized. The pause replaces the corresponding work period such that the duration of timeslot is not increased. Note that this greedy approach for inserting pauses makes the whole dynamic programming procedure heuristic (sub-optimal solutions may be generated).

If a pre-assigned pause is contained inside a timeslot, and it is not positioned in the second third of it, such a timeslot is declared invalid, and its cost is set to $+\infty$. The same happens if the employee cannot take any pause, i.e. $E(t)$ is empty for all time moments in the second third of the timeslot.

Let $\hat{\gamma}(b, f)$ be the best reduced cost of a timeslot, which starts at period b, finishes at period f, and respects the pause policy. Let $\ell(b, f)$ be this timeslot's working time, which can be uniquely determined from its duration $(f - d + 1)$ according to the pause policy.

Let now $\delta^\kappa(d,b,f,\ell)$ be the best reduced cost of a day-shift of day d that starts at period b, completes at period f, contains $\kappa \in \{1,2\}$ timeslots and ℓ working periods. A valid day-shift should satisfy starting, completion, working time bounds and the daily pre-assignments. Let set Ω_d contain the set of valid triples (b,f,ℓ):

$$\Omega_d = \left\{ (b,f,\ell) : \begin{array}{l} b \in [\mathrm{BD}_d^-, \mathrm{BD}_d^+],\ f \in [\mathrm{FD}_d^-, \mathrm{FD}_d^+], \\ \ell \in [\mathrm{LD}_d^-, \mathrm{LD}_d^+],\ f - b + 1 \leqslant \mathrm{DD}_d^+ \end{array} \right\}.$$

The formula for the day-shift containing one timeslot is:

$$\delta^1(d,b,f,\ell) = \begin{cases} \hat{\gamma}(b,f), & \text{if } (b,f,\ell) \in \Omega_d, \ell = \ell(b,f); \\ +\infty, & \text{otherwise.} \end{cases} \tag{3.13}$$

The formula for the day-shift containing two slots separated by a lunch break is:

$$\delta^2(d,b,f,\ell) = \min_{f',b'} \begin{cases} \hat{\gamma}(b,f') + \hat{\gamma}(b',f), & \text{if } (b,f,\ell) \in \Omega_d,\ \ell = \ell(b,f') + \ell(b',f), \\ & b' - f' - 1 \in [\mathrm{RD}_d^-, \mathrm{RD}_d^+], \\ & \ell(b,f') \geqslant \mathrm{TD}_d \text{ or } \ell(b',f) \geqslant \mathrm{TD}_d, \\ +\infty, & \text{otherwise.} \end{cases} \tag{3.14}$$

The best reduced cost $\delta(d,b,f,\ell)$ of a day-shift with one or two timeslots can now be computed :

$$\delta(d,b,f,\ell) = \min\left\{\delta^1(d,b,f,\ell), \delta^2(d,b,f,\ell)\right\}.$$

Now we seek the best combination of day-shifts and days-off that designs a valid individual planning given its total working time LE and its number of day-shifts in $[\mathrm{NE}^-, \mathrm{NE}^+]$. This is also called a *tour scheduling problem* for a single employee. Let $\eta^0(d,n,\ell)$ be the best reduced cost of a partial employee planning for the first d days, which contains n day-shifts and ℓ working periods, and ends with a day-off. Let also $\eta^1(d,f,n,\ell)$ be the best reduced cost of a partial employee planning for the first d days, which contains n day-shifts and ℓ working periods, and ends at period f with a day-shift. These reduced costs are calculated using the following recursions. We set $\eta^0(0,0,0) = 0$, all other values $\eta^0(0,f,\ell)$ are set to $+\infty$. Also we set $\eta^1(d,f,n,\ell) = +\infty$ if $d \leqslant 0$ or $f \notin [\mathrm{FD}_d^-, \mathrm{FD}_d^+]$.

The formula for $\eta^0(d,n,\ell)$ is the following:

$$\eta^0(d,n,\ell) = \min\left\{\eta^0(d-1,n,\ell), \min_{f \in [\mathrm{FD}_{d-1}^-, \mathrm{FD}_{d-1}^+]} \left\{\eta^1(d-1,f,n,\ell)\right\}\right\}$$

The formula for $\eta^1(d,f,n,\ell)$, $f \in [\mathrm{FD}_d^-, \mathrm{FD}_d^+]$, is:

$$\eta^1(d,f,n,\ell) = \min\left\{\hat{\eta}^0(d,f,n,\ell), \min_{f' \in [\mathrm{FD}_{d-1}^-, \mathrm{FD}_{d-1}^+]} \left\{\hat{\eta}^1(d,f',f,n,\ell)\right\}\right\}, \tag{3.15}$$

where $\hat{\eta}^0(d,f,n,\ell)$ is the best reduced cost with the condition that the employee had a day-off on day $d-1$, and $\hat{\eta}^1(d,f,f',n,\ell)$ is the best reduced cost with the condition

that the employee had a day-shift of day $d-1$ finishing at time f'. $\hat{\eta}^0$ and $\hat{\eta}^1$ are computed as follows:

$$\hat{\eta}^0(d,f,n,\ell) = \min_{\substack{b\in[\mathrm{BD}_d^-,\mathrm{BD}_d^+],\\ \ell'\in[\mathrm{LD}_d^-,\mathrm{LD}_d^+]}} \left\{\eta^0(d-1,n-1,\ell-\ell') + \delta(d,b,f,\ell')\right\},$$

$$\hat{\eta}^1(d,f',f,n,\ell) = \min_{\substack{b\in\left[\max\{f'+\mathrm{RE}^-,\,\mathrm{BD}_d^-\},\,\mathrm{BD}_d^+\right],\\ \ell'\in[\mathrm{LD}_d^-,\mathrm{LD}_d^+]}} \left\{ \begin{array}{l} \eta^1(d-1,f',n-1,\ell-\ell') \\ +\delta(d,b,f,\ell') \end{array} \right\}. \quad (3.16)$$

During this step, the algorithm deals also with the maximum number of successive day-shifts without day-off. In recursion (3.15)), $\eta^1(d,f,n,\ell)$ should be written $\eta^1(d,f,n,m,\ell)$, where m is the number of consecutive day-shifts ending at day d such that $m \leqslant \mathrm{ME}^+$. However, we decided to omit the full recursion for the sake of simplification.

The best reduced cost of an individual planning can be computed as

$$\min_{n\in[\mathrm{NE}^-,\mathrm{NE}^+]} \left\{ \eta^0(D,n,\mathrm{LE}), \min_{f\in[\mathrm{FD}_D^-,\mathrm{FD}_D^+]} \eta^1(D,f,n,\mathrm{LE}) \right\}.$$

The nested dynamic programming algorithm takes too much time because of a large number of states. In order to accelerate the algorithm, we heuristically delete some states. Details are given in our paper.

3.4.4 Computational results

Our customer data comes from a company of mini-marts. All instances are defined over one week divided into 15 minutes periods. In the customer data, almost 10% of employees work only on Saturday, while the others may work at most five days. Around 70% of the employees can only perform one type of production activity. Most of the employees have a small flexibility in their schedule: usual beginning and finishing time of timeslots can be shifted by one hour.

The cost coefficients in the objective function are the following: $\mathrm{CO}_a = 1$, $\mathrm{CO}_a^{\mathrm{crit}} = 2$, $\mathrm{CU}_a = 2$, $\mathrm{CU}_a^{\mathrm{crit}} = 5$. For work demand DE_{at}, critical over-coverage occurs when strictly more than $\mathrm{OV}_{at} = \mathrm{DE}_{at} + 1$ employees are assigned to production activity a at time period t, while critical under-coverage occurs when strictly less than $\mathrm{UN}_{at} = \lceil \mathrm{DE}_{at}/2 \rceil$ employees are assigned to the activity. We use a representative set of twelve customer data. They have a different number of employees $|K|$, and a different number of production activities $|A|$.

We ran different methods on the customer data: the greedy heuristic, dive2, dive10 and dive30 (the diving heuristic with the cumulated column generation time limit of 2, 10 and 30 minutes, respectively), the heuristic branch-and-price algorithm, and the compact MIP model. The latter is described in the electronic supplement of our paper. In the method dive$T$, we limit the column generation time at each node to $60 \cdot T/|K|$ seconds. The best solution found by heuristics dive2, dive10 and dive30 is used for the initialization of the branch-and-price. The time reported for the branch-and-price does not include the time spent by the heuristic.

Data	$\|K\|$	$\|A\|$	LB_{triv}	greedy	dive2	dive10	dive30	B&P	LB_{Lagr}	compact
A1-7	5	1	204	390	335	335	335	325	322	325
A1-9	5	1	288	423	299	299	299	299	299	299
A1-0	10	1	334	528	393	393	393	393	393	393
A1-3	10	1	152	326	228	228	228	228	227	228
A3-5	25	3	680	1077	832	834	832	824 T	820	890 T
A3-9	25	3	866	1181	984	960	987	954 T	951	999 T
A3-1	30	3	880	1285	970	962	954	954 T	936	1095 T
A3-2	30	3	358	931	592	551	543	529 T	488	739 T
A5-5	42	5	140	1111	918	909	852	852 T	804	3179 T
A5-6	42	5	303	1254	1029	942	925	925 T	884	1298 T
A5-0	45	5	404	1713	1522	1510	1504	1504	1504	–
A5-1	45	5	412	1793	1529	1525	1533	1513 T	1508	–

Table 3.8: Solution values obtained by our methods for the customer data

Data	$\|K\|$	$\|A\|$	greedy	dive2	dive10	dive30	B&P	col. gen.	compact
A1-7	5	1	1	5	4	4	45	2	9
A1-9	5	1	1	7	6	5	11	11	8
A1-0	10	1	1	8	8	8	4	4	31135
A1-3	10	1	1	14	13	13	2045	13	8962
A3-5	25	3	2	118	482	833	T	105	T
A3-9	25	3	2	118	574	1164	T	117	T
A3-1	30	3	2	119	520	1163	T	546	T
A3-2	30	3	2	127	617	1615	T	2642	T
A5-5	42	5	3	131	592	1650	T	3663	T
A5-6	42	5	3	135	592	1679	T	1244	T
A5-0	45	5	2	123	527	1369	2166	2166	–
A5-1	45	5	2	125	529	1102	T	177	–

Table 3.9: Running time (in seconds) of our methods on the customer data

Tables 3.8 and 3.9 summarize the results obtained (respectively the objective function value of the solution found, and the execution time) with our customer data. In column LB_{Lagr}, we report the "heuristic" Lagrangian lower bound computed at the root node of the B& P. Column "LB_{triv}" is a trivial lower bound computed by taking into account only the overall demand and the total working time of the team. In the tables, the mark "T" in the table means that the corresponding approach did not terminate within the time limit of 24 hours. The mark "–" means that the MIP solver failed due to memory issues.

The running time of the greedy heuristic is very small, even for instances with 45 employees. The difference between the value of the greedy solution and the best known one can be large. The diving heuristics are much more effective to find solutions close to the heuristic lower bound. The relative gap between the solutions obtained by the dive2 heuristic and the branch-and-price is greater than 10% only for two instances. It may happen that the diving gives better results when a smaller

computing time is set. However, different experiments, not reported here, showed that giving more time to the diving heuristic at each node generally improves the result.

The branch-and-price method terminates for five instances out of twelve within 24 hours of computation time. For three instances the bound is tight at the root node, for six instances the initial upper bound was improved. When the MIP solver is able to find an optimal solution, its value is equal to the solution value found by the heuristic branch-and-price.

In our paper, we also present results for randomly generated instances. They are similar to the results we present here.

3.4.5 Related work

Since the seminal work of Dantzig (1954), a large quantity of research papers have developed models and methods to assist managers and planners in their employee scheduling tasks. Ernst et al. (2004) provided a comprehensive literature review of classical studies on this problem.

As the literature volume of the subject is huge, we mention only some works which are based on column generation. Demassey et al. (2006) developed a hybrid IP-CP (Integer Programming and Constraint Programming) approach for an employee timetabling problem with one day time horizon. In this column generation approach, the pricing problem is solved by constraint programming. Côté et al. (2013); Boyer et al. (2014); Restrepo et al. (2016) use branch-and-price to solve very general multi-activity shift scheduling problems. Their approaches rely on the description of shifts using a context-free grammar. Brunner and Stolletz (2014) use an ad-hoc branch-and-price method to solve a tour scheduling problem. The main ingredients of their approach are the use of variables related to day-shifts, which are recombined in the master problem, and stabilization strategies to reduce the number of column generation iterations. Another work by Dohn and Mason (2013) uses branch-and-price in the context of employee scheduling. They use a nested dynamic programming approach, which is well-suited to the structure of their problem. Recently, to solve a multi-activity tour scheduling problem, Restrepo et al. (2018) proposed a combination of Benders decomposition and column generation. They claimed that the latter outperformed a previously suggested branch-and-price algorithm.

Some works suggested to solve the Lagrangian relaxation of the problem by an alternative approach to the column generation. In a continuation of our work, Gérard (2015) used a sub-gradient algorithm when solving a tour scheduling problem with the time horizon up to four weeks. A similar approach was applied by Hernández-Leandro et al. (2019) for a shift scheduling problem. Pan et al. (2019) proposed a dual ascent heuristic to solve the Lagrangian relaxation.

3.5 Robust CVRP with Knapsack Uncertainty

This section is based on our paper submitted to a journal (Pessoa et al., 2018a).

We examine the robust counterpart of the classical Capacitated Vehicle Routing Problem (CVRP). We consider two types of uncertainty sets for the customer demands: the classical budget polytope introduced by Bertsimas and Sim (2003), and the partitioned budget polytope proposed by Gounaris et al. (2013). We show that using the set-partitioning formulation it is possible to reformulate our problem as a deterministic heterogeneous vehicle routing problem. Thus, many state-of-the-art techniques for exactly solving deterministic VRPs can be applied to the robust counterpart, and a modern branch-cut-and-price algorithm can be adapted to our setting by keeping the number of pricing subproblems strictly polynomial. More importantly, we introduce new techniques to significantly improve the efficiency of the algorithm. We present analytical conditions under which a pricing subproblem is infeasible. This result is general and can be applied to other combinatorial optimization problems with knapsack uncertainty. We also introduce robust capacity cuts which are provably stronger than the ones known in the literature. Finally, a fast iterated local search algorithm is proposed to obtain heuristic solutions for the problem. Using our branch-cut-and-price algorithm incorporating existing and new techniques, we are able to solve to optimality all but one open instances from the literature.

3.5.1 Robust model

In this work we address the robust CVRP by using a formulation that affects customers to individual routes, where a route is a path starting and ending at the depot and going at most once to each other node of the graph. In this formulation, the uncertainty on the clients demands constrains the set of feasible routes: we need to rely on *robust routes*, that is, routes having a total demand that does not exceed the capacity of the vehicle for any demand vector in the given uncertainty set.

The purpose of this section is to show how the set of *robust routes* can be expressed as the union of sets of classical routes, albeit for different demand and capacity values. Some of the techniques introduced next are classical in the robust combinatorial optimization literature, while others are novel and could benefit to other robust problems with capacity constraints, such as the bin-packing problem (Song et al., 2018), among others. For this reason, we present our approach in a general context and consider a general combinatorial optimization problem with n variables. The feasibility set of the problem is $\{z \in Z^0, \sum_{j=1}^{n} d_j z_j \leqslant Q\}$, where $d \in \mathbb{R}^n_+$ denotes the vector of weights, $Q \in \mathbb{R}_+$ is the available capacity, and $Z^0 \subseteq \{0,1\}^n$ is a discrete set describing the combinatorial structure of the problem at hand. For instance, in the case of the CVRP any $\bar{z} \in Z^0$ represents a route, while d_i is the demand of client i and Q is the capacity of each vehicle.

Uncertainty polytopes

The simplest polyhedral uncertainty set is the box $[\bar{d}, \bar{d} + \hat{d}] \subset \mathbb{R}^n_+$ defined by the vectors $\bar{d}, \hat{d} \in \mathbb{R}^n_+$, where $\bar{d}$ represents the nominal values and $\hat{d}$ the deviations. Notice that it is irrelevant to consider downward deviations of d in our context because we focus on capacity constraints so that downward deviations do not lead to infeasibility.

The box is usually not considered as good choice of uncertainty set as it is overly conservative. Indeed, it contains the vector $\bar{d} + \hat{d}$ having each component at its peak value, which seldom occurs in practice. For that reason, classical uncertainty polytopes cut the box by one or more linear constraints. We focus in this paper on the set

$$\mathcal{D} \equiv \left\{ d \in [\bar{d}, \bar{d} + \hat{d}] : \sum_{j \in V_i} \omega'_j d_j \leqslant b'_j,\ j = 1, \ldots, m \right\},$$

where $V_1, \ldots, V_m$ form a partition of $\{1, \ldots, n\}$, $\omega' \in \mathbb{R}^n_+$ and $b' \in \mathbb{R}^m_+$.

Set $\mathcal{D}$ is general enough to encompass two classical uncertainty polytopes from the robust optimization literature. The first one is the budgeted polytope introduced in Bertsimas and Sim (2003), widely used in the robust optimization literature,

$$\mathcal{D}^{\text{card}} \equiv \left\{ d \in [\bar{d}, \bar{d} + \hat{d}] : \sum_{j=1}^{n} \frac{d_j - \bar{d}_j}{\hat{d}_j} \leqslant \Gamma \right\},$$

obtained from $\mathcal{D}$ by setting $m = 1$, $\omega'_j = 1/\hat{d}_i$ for $j = 1, \ldots, n$, and $b'_1 = \Gamma + \sum_{j=1}^{n} \bar{d}_j/\hat{d}_j$. The second one was previously introduced for the CVRP by Gounaris et al. (2013) and is defined by

$$\mathcal{D}^{\text{part}} \equiv \left\{ d \in [\bar{d}, \bar{d} + \hat{d}] : \sum_{j \in V_i} (d_j - \hat{d}_j) \leqslant a_i,\ i = 1, \ldots, m \right\},$$

obtained from $\mathcal{D}$ by setting $\omega'_j = 1$ for each $j = 1, \ldots, n$ and $b'_i = a_i + \sum_{j \in V_i} \bar{d}_j$.

Our purpose is to reformulate the robust feasibility set

$$\mathcal{Z} \equiv \left\{ z \in Z^0 : \sum_{j=1}^{n} d_j z_j \leqslant Q,\ \forall d \in \mathcal{D} \right\}$$

as the union of a limited number of sets of the form $\{z \in Z^0, \sum_{j=1}^{n} d'_j z_j \leqslant Q'\}$ for some $d' \in \mathbb{R}^n_+$ and $Q' \in \mathbb{R}_+$. For the robust CVRP, this reformulation will have two advantages:

1. Reformulating the robust CVRP as a nominal heterogeneous CVRP so that the generic solver presented in Section 2.4 can be used for the robust CVRP.

2. A key part of the branch-cut-and-price algorithm lies in the generation of new routes, by solving pricing problems of the form

$$\min\{\bar{c}z : z \in \mathcal{Z}\}, \tag{3.17}$$

where $\bar{c}$ is the reduced cost vector. The above reformulation implies that the robust pricing problems can be solved through a sequence of nominal pricing problems.

Reducing robust problems to deterministic ones

In the following, we use classical techniques from robust combinatorial optimization, first introduced by Bertsimas and Sim (2003), to reformulate $\mathcal{Z}$ as the union of deterministic sets. To ease the derivations that follow, we express any $d \in \mathcal{D}$ as $d_j = \hat{d}_j + \xi_j \hat{d}_j$, where the uncertain parameter ξ_j measures the fraction of deviation $\hat{d}_j$ affected to d_j. Thus, each $d \in \mathcal{D}$ is in one-to-one correspondance with a vector ξ in the polytope

$$\Xi \equiv \left\{ \xi \in [0,1]^n : \sum_{j \in V_i} \omega_j \xi_j \leqslant b_i,\ i = 1, \dots, m \right\}.$$

Hence, the robust capacity constraint of $\mathcal{Z}$ can be reformulated as

$$\begin{aligned} &\sum_{j=1}^{n} d_j z_j \leqslant Q,\ \forall d \in \mathcal{D} \ \Leftrightarrow\ \max_{d \in \mathcal{D}} d_j z_j \leqslant Q \\ &\Leftrightarrow\ \bar{d}^\top z + \max_{0 \leqslant \xi \leqslant 1} \left\{ \sum_{j=1}^{n} \xi_j \hat{d}_j z_i : \sum_{j \in V_i} \omega_j \xi_j \leqslant b_i,\ i = 1, \dots, m \right\} \leqslant Q \end{aligned} \tag{3.18}$$

Next, we introduce the vectors of dual variables $\theta \in \mathbb{R}^m$ and $\zeta \in \mathbb{R}^n$. Recalling that $\{V_i, i = 1, \dots, m\}$ forms a partition of $\{1, \dots, n\}$, we let $i(j)$ be the only value of i such that $j \in V_i$ and replace the linear programming problem from the left-hand side of (3.18) by its dual:

$$\bar{d}^\top z + \min_{\theta, \zeta \geqslant 0} \left\{ b^\top \theta + \sum_{j=1}^{n} \zeta_j,\ \text{s.t. } \zeta_j + \omega_i \theta_{i(j)} \geqslant \hat{d}_i z_i,\quad j = 1, \dots, n \right\} \leqslant Q, \tag{3.19}$$

following the classical reformulation technique in robust linear optimization. In the dual problem from (3.19), each variable ζ_j belongs to a single constraint, in addition to the non-negative constraint. Therefore, as the cost of each ζ_j is positive, it can be replaced by $\max\{0, \hat{d}_j z_j - \omega_i \theta_{i(j)}\}$ for each $j = 1, \dots, n$. This leads to the equivalent constraint

$$\begin{aligned} &\bar{d}^\top z + \min_{\theta \in \mathbb{R}^m_+} \left\{ b^\top \theta + \sum_{j=1}^{n} \max\{0, \hat{d}_j z_j - \omega_j \theta_{i(j)}\} \right\} \\ &= \bar{d}^\top z + \min_{\theta \in \mathbb{R}^m_+} \left\{ b^\top \theta + \sum_{j=1}^{n} \max\{0, \hat{d}_j - \omega_j \theta_{i(j)}\}\, z_j \right\} \\ &= \min_{\theta \in \mathbb{R}^m_+} \left\{ b^\top \theta + \sum_{j=1}^{n} (\bar{d}_j + \max\{0, \hat{d}_j - \omega_j \theta_{i(j)}\})\, z_j \right\} \leqslant Q. \end{aligned} \tag{3.20}$$

When θ is fixed, the left-hand side of (3.20) becomes a deterministic capacity constraint with capacity $Q - b^\top \theta$ and weight $d^\theta_j = \bar{d}_j + \max\{0, \hat{d}_j - \omega_j \theta_{i(j)}\}$ for each $j = 1, \dots, n$. Because of the sense of the inequality, (3.20) is equivalent to

$$\bigcup_{\theta \in \mathbb{R}^m_+} \{ b^\top \theta + (d^\theta)^\top z \leqslant Q \}. \tag{3.21}$$

Expression (3.21) has rewritten the robust constraint as the union of feasible solution sets (hereinafter referred to as *feasibility sets*), each of which is characterized by a single deterministic capacity constraint. Yet, the union is indexed by the infinite set $\mathbb{R}^m_+$, limiting its usefulness. Fortunately, not all $\theta \in \mathbb{R}^m_+$ need to be considered in (3.21). Let us define $\theta^0_i = 0$ for $i = 1, \ldots, m$, $\theta^j_{i(j)} = \hat{d}_j/\omega_j$ for $j = 1, \ldots, n$, and introduce the set

$$\Theta = (\{0\} \cup \{\theta^j_{i(j)} : j \in V_1\}) \times \cdots \times (\{0\} \cup \{\theta^j_{i(j)} : \ j \in V_m\}) \subset \mathbb{R}^m_+. \tag{3.22}$$

Recall that $\mathcal{Z}$ has been defined as $\left\{z \in Z^0 : \ \sum_{j=1}^n d_j z_i \leqslant Q, \ \forall d \in \mathcal{D}\right\}$. The following theorem shows that if $z \in Z^0$, only $\theta \in \Theta$ needs to be considered in (3.21). Its proof is given in the appendix of our paper (Pessoa et al., 2018a).

Theorem 3.1. $\mathcal{Z} = \bigcup_{\theta \in \Theta} \{z \in Z^0 : \ (d^\theta)^\top z \leqslant Q - b^\top \theta\}$

The theorem implies immediately that the robust pricing problem (3.17) can be rewritten as follows.

Corollary 3.1. $\min\{\bar{c}z : \ z \in \mathcal{Z}\} = \min_{\theta \in \Theta} \left\{\min\{\bar{c}z : \ z \in Z^0, (d^\theta)^\top z \leqslant Q - b^\top \theta\}\right\}$.

The above results are particularly useful when m is small or $\{\theta^j_{i(j)} : \ j \in V_i\}$ does not contain too many elements, which is the case for $\mathcal{D}^{\text{card}}$ and $\mathcal{D}^{\text{part}}$, respectively. In the case of $\mathcal{D}^{\text{part}}$, we see that formula (3.22) leads to $\Theta^{\text{part}} = \{0, 1\}^m$, the cardinality of which does not depend on n. This means that the number of determinisic problems involved in the reformulation does not depend on the dimension of the robust problem (assuming that m is constant). For the CVRP for instance, we obtain that the number of deterministic problems involved in Theorem 3.1 does not depend on the size of the considered graphs. Thus, if $\omega_j = \hat{d}_j$ for each $j = 1, \ldots, n$, then $\Theta = \Theta^{\text{part}}$.

In the case of $\mathcal{D}^{\text{card}}$, we obtain $\Theta = \{0, \hat{d}_1, \hat{d}_2, \ldots, \hat{d}_n\}$. In fact, for that set a stronger result is known.

Theorem 3.2 (Lee and Kwon (2014))**.** *Suppose $\mathcal{D} = \mathcal{D}^{\text{card}}$ and, w.l.o.g., that $\hat{d}_1 \geqslant \hat{d}_2 \geqslant \cdots \geqslant \hat{d}_n \geqslant \hat{d}_{n+1} = 0$. Define $\Theta^{\text{card}} = \{\hat{d}_{\Gamma+1}, \hat{d}_{\Gamma+3}, \hat{d}_{\Gamma+5}, \ldots, \hat{d}_{\Gamma+\gamma}, 0\}$ where γ is the largest odd integer such that $\Gamma + \gamma < n + 1$. For any $z \in \{0, 1\}^n$, we have* $\operatorname{argmin}_{\theta \in \mathbb{R}_+} \left\{b\theta + (d^\theta)^\top z\right\} \cap \Theta^{\text{card}} \neq \varnothing$.

Theorem 3.2 implies that for the uncertainty set $\mathcal{D}^{\text{card}}$, the robust feasibility set $\mathcal{Z}$ can be reformulated as the union of roughly $\frac{n-\Gamma}{2}$ deterministic feasibility sets.

Corollary 3.2. *If $\mathcal{D} = \mathcal{D}^{\text{card}}$, $\mathcal{Z} = \bigcup_{\theta \in \Theta^{\text{card}}} \{z \in Z^0 \,|\, (d^\theta)^\top z \leqslant Q - b\theta\}$.*

Reducing the cardinality of Θ

The following contains a new idea to reduce the number of elements of Θ that need to be considered in Theorem 3.1 and Corollary 3.2. We outline next its bottom line, based on two main steps. Let us denote the feasibility sets involved in Theorem 3.1

and Corollary 3.2 as $\mathcal{Z}_\theta \equiv \{z \in Z^0 : (d^\theta)^\top z \leqslant Q - b^\top\theta\}$ for each $\theta \in \Theta$. The first step introduces a smaller set $\tilde{\mathcal{Z}}_\theta \subseteq \mathcal{Z}_\theta$ for each $\theta \in \Theta$. Sets $\tilde{\mathcal{Z}}_\theta$ do not have the structure of the original deterministic problem (they can be much more complex) so we do not wish to use them in the decomposition from Theorem 3.1. However, we can prove that we can remove from Θ any θ such that $\tilde{\mathcal{Z}}_\theta = \varnothing$, essentially replacing Theorem 3.1 by $\mathcal{Z} = \bigcup_{\theta\in\Theta:\ \tilde{\mathcal{Z}}_\theta \neq \varnothing} \mathcal{Z}_\theta$. As proving $\tilde{\mathcal{Z}}_\theta = \varnothing$ is hard in general, the second step introduces sufficient conditions for testing whether $\tilde{\mathcal{Z}}_\theta$ is empty. These sufficient conditions amount to run quick heuristic algorithms in a pre-processing phase, before the branch-cut-and-price algorithm is started.

Let us now detail the two steps of the approach. First, motivated by the complementary slackness conditions between the maximization problem from (3.18) and its dual, we define for any $\theta \in \Theta$ the set

$$\tilde{\mathcal{Z}}_\theta \equiv \left\{ z \in \mathcal{Z}_\theta :\ b_i \leqslant \sum_{j\in V_i:\hat{d}_j \geqslant \theta_i} \omega_j z_j,\ \forall i \in \{\ell \in \{1,\dots,m\} :\ \theta_\ell > 0\} \right\}.$$

We see that for each $\theta \in \Theta$, $\tilde{\mathcal{Z}}_\theta$ contains up to m constraints in addition to those already present in $\mathcal{Z}_\theta$. Hence, considering the counterpart of Corollary 3.1 for $\tilde{\mathcal{Z}}_\theta$ would involve solving $\min\{\bar{c}z :\ z \in \tilde{\mathcal{Z}}_\theta\}$ for each $\theta \in \Theta$. Since optimizing over set $\tilde{\mathcal{Z}}_\theta$ can be cumbersome, we will use the set only to remove from Θ any vector θ such that $\tilde{\mathcal{Z}}_\theta = \varnothing$, by proving

$$\mathcal{Z} = \bigcup_{\theta\in\Theta:\ \tilde{\mathcal{Z}}_\theta \neq \varnothing} \mathcal{Z}_\theta. \tag{3.23}$$

Proving (3.23) is enough to reduce the number of feasibility sets considered in Theorem 3.1 to $\{\theta \in \Theta :\ \tilde{\mathcal{Z}}_\theta \neq \varnothing\} \subseteq \Theta$. However, we need to prove a slightly stronger result to encompass also the case of Corollary 3.2 because the latter relies on Θ^{card} instead of Θ. For that reason, the following theorem introduces a technical assumption that considers any set $\Theta^* \subseteq \Theta$ large enough to contain all minimizers of the left-hand side of (3.20). Its proof is provided the appendix of our paper.

Theorem 3.3. *Let $\Theta^* \subseteq \Theta$ be such that $\operatorname{argmin}_{\theta\in\mathbb{R}^m_+}\left\{b^\top\theta + (d^\theta)^\top z\right\} \cap \Theta^* \neq \varnothing$ for each $z \in Z^0$. Then, it holds that $\mathcal{Z} = \bigcup_{\theta\in\Theta^*:\ \tilde{\mathcal{Z}}_\theta \neq \varnothing} \mathcal{Z}_\theta$ and $\min\{\bar{c}z :\ z \in \mathcal{Z}\} = \min_{\theta\in\Theta^*:\ \tilde{\mathcal{Z}}_\theta \neq \varnothing} \{\min\{\bar{c}z :\ z \in \mathcal{Z}_\theta\}\}$.*

For the second step of our approach, we propose fast alternatives to detect the infeasibility of $\tilde{\mathcal{Z}}_\theta$, avoiding its combinatorial structure which can be cumbersome to handle. Specifically, we check the feasibility of the relaxation of $\tilde{\mathcal{Z}}_\theta$, denoted $\hat{\mathcal{Z}}_\theta$, defined by relaxing the combinatorial structure Z^0 to $\{0,1\}^n$. Formally, we define $I(\theta) = \{i \in \{1,\dots,m\} :\ \theta_i > 0\}$, $\hat{V}_i(\theta) = \{j \in V_i :\ \hat{d}_j \geqslant \omega_j\theta_i\}$, and $\hat{\mathcal{Z}}_\theta \equiv \left\{z \in \{0,1\}^n :\ b^\top\theta + (d^\theta)^\top z \leqslant Q, \sum_{j\in\hat{V}_i(\theta)} \omega_j z_i \geqslant b_i, \forall i \in I(\theta)\right\}$. Since $\tilde{\mathcal{Z}}_\theta \subseteq \hat{\mathcal{Z}}_\theta$, proving $\hat{\mathcal{Z}}_\theta = \varnothing$ implies $\tilde{\mathcal{Z}}_\theta = \varnothing$. Moreover, we show that $\hat{\mathcal{Z}}_\theta$ can be verified in pseudo-polynomial time.

Lemma 3.4. *The feasibility of $\hat{\mathcal{Z}}_\theta$ can be tested in pseudo-polynomial time by solving m knapsack problems.*

The proof can be found in our paper. For the special case $\mathcal{D}^{\text{card}}$, checking the feasibility of $\hat{\mathcal{Z}}_\theta$ is much simpler.

Lemma 3.5. *When $\mathcal{D} = \mathcal{D}^{\text{card}}$, $\hat{\mathcal{Z}}_\theta \neq \varnothing$ if and only if*

$$\min_S \left\{ \sum_{j \in S} d_j^\theta : \; S \subseteq \{j \in \{1, \ldots, n\} : \hat{d}_j \geqslant \theta\}, |S| = \Gamma \right\} \leqslant Q - \Gamma\theta,$$

which can be answered in polynomial time.

For the special case $\mathcal{D}^{\text{part}}$ and assuming that $\hat{d} = \kappa\bar{d}$ for some scalar $\kappa > 0$ (which is true for all literature instances), we can provide an easy sufficient condition for $\hat{\mathcal{Z}}_\theta$ to be empty, by considering the linear programming relaxation of $\hat{\mathcal{Z}}_\theta$.

Lemma 3.6. *When $\mathcal{D} = \mathcal{D}^{\text{part}}$ and $\hat{d} = \kappa\bar{d}$ for some scalar $\kappa > 0$, $(b^\top\theta)/\kappa > Q - b^\top\theta$ implies $\hat{\mathcal{Z}}_\theta = \varnothing$.*

Lemmas 3.5 and 3.6 are applied in the pre-processing phase.

3.5.2 Solution approach

Now we use the results from Section 3.5.1 to reformulate the robust CVRP as a heterogeneous vehicle routing problem. Let $G = (V, A)$ be a complete digraph with nodes $V = \{0, 1, \ldots, n\}$ and arcs $\{(j, j') \in V \times V : j \neq j'\}$. Node $0 \in V$ represents the unique depot, and each node $j \in V^0 = V \backslash \{0\}$ corresponds to a customer with demand $d_j \in \mathbb{R}_+$. The depot hosts H homogeneous vehicles of capacity Q. Each vehicle incurs a transportation cost $c_{jj'} \in \mathbb{R}_+$ if it traverses the arc $(j, j') \in A$. The objective is to find a set of H routes starting and ending at the depot, each one serving a total demand of at most Q, such that each customer is visited exactly once and the total transportation cost is minimized.

We define P^0 as the set of all routes in G starting and ending at the depot. For each $p \in P^0$, we denote the cost of the route by c^p, and indicate whether node j pertains to the route by the binary value $\bar{z}_j^p$. Then, we describe the set of feasible routes for the CVRP with demand vector $\bar{d} \in \mathbb{R}_+^n$ as

$$P = \left\{ p \in P^0 : \; \sum_{j \in V_0} \bar{z}_j^p \bar{d}_i \leqslant Q \right\}.$$

Following Section 3.5.1, the set of robust routes is defined as

$$P^{\text{robust}} \equiv \left\{ p \in P^0 : \; \sum_{j \in V_0} \bar{z}_j^p d_j \leqslant Q, \quad \forall d \in \mathcal{D} \right\}.$$

Let $\Theta^* \subseteq \mathbb{R}_+^m$ be a set satisfying the assumption from Theorem 3.3 and $\tilde{\Theta} \equiv \{\theta \in \Theta^* : \tilde{P}_\theta^{\text{robust}} \neq \varnothing\}$ where $\tilde{P}_\theta^{\text{robust}}$ is the counterpart of $\tilde{\mathcal{Z}}_\theta$ for P^{robust}. Theorem 3.3 implies that

$$P^{\text{robust}} = \bigcup_{\theta \in \tilde{\Theta}} P_\theta^{\text{robust}}, \tag{3.24}$$

where $P_\theta^{\text{robust}} = \left\{p \in P^0 : \sum_{j \in V_0} \bar{z}_j^p d_j^\theta \leqslant Q - b^\top \theta\right\}$. Equation (3.24) underlines that the set of routes that are feasible for the robust capacity constraint is nothing else than the union of sets of routes feasible for different deterministic capacity constraints.

We will now formulate our robust CVRP problem using the generic model presented in Sections 2.4.2 and 2.4.3 in the same way as it is done for examples given in Section 2.4.4. As indicated above, our problem can be formulated as a deterministic heterogeneous vehicle routing problem. In the latter, the set K of vehicle types corresponds to set $\tilde{\Theta}$. Let $k(\theta) \in K$ be vehicle type corresponding to value $\theta \in \tilde{\Theta}$.
Model: Graphs $G^k = (V^k, A^k)$, sets of vertices $V^k = \{v_0^k, \dots, v_n^k\}$, sets of arcs $A^k = \{(v_i^k, v_j^k), (v_j^k, v_i^k) : \{i, j\} \in E\}$, source and sink vertices $v_{\text{source}}^k = v_{\text{sink}}^k = v_0^k$, $k \in K$. Each graph has one main resource $R^k = R_M^k = \{r^k\}$ corresponding to the vehicle capacity. Resource consumption of arcs in $A^{k(\theta)}$ depends on θ: $q_{a,1}^{k(\theta)} = (d_j^\theta + d_{j'}^\theta)/2$, $a = (v_j^{k(\theta)}, v_{j'}^{k(\theta)}) \in A^{k(\theta)}$, $\theta \in \tilde{\Theta}$. We define $d_0^k = 0$ for all $k \in K$. Resource consumption bounds of vertices in $V^{k(\theta)}$ also depend on θ: $l_{v_j^k, r^k} = 0, u_{v_j^k, r^k} = Q - b^\top \theta, v_j^k \in V^k$, $k = k(\theta) \in K$. We have integer variables x_e per edge in the original graph, $e = (j, j')$, $j, j' \in V$, $j < j'$. The formulation is:

$$\text{Min} \sum_{e \in E} c_e x_e \tag{3.25a}$$

$$\text{S.t.} \sum_{e \in \delta(j)} x_e^k = 2, \qquad j \in V^0, \tag{3.25b}$$

$$\sum_{e \in \delta(0)} x_e = 2H. \tag{3.25c}$$

Bounds on the number of paths from G^k in the solution are $\underline{W}^k = 0$, $\overline{W}^k = H$. Mapping between variables and arcs is $M(x_e) = \{(v_j^k, v_{j'}^k), (v_{j'}^k, v_j^k)\}_{k \in K}$, $e = (j, j') \in E$. Packing sets are defined on vertices as $\mathcal{B}^{\mathcal{V}} = \cup_{j \in V^0} \{\{v_j^k : k \in K\}\}$. We branch on variables x. We tried to branch on assignment of clients to graphs, but it was determined to be not efficient compurtationally. Elementary route enumeration is used.

Additionally we separate the problem-specific capacity inequalities presented in our paper (Pessoa et al., 2018a). These inequalities are defined in the space of variables x. Thus, they are robust and can be used with our generic BCP solver. Our capacity inequalities can be used in conjunction with another capacity inequalites for the robust CVRP proposed by Gounaris et al. (2013). For the specific case when $\mathcal{D} = \mathcal{D}^{\text{part}}$ and demand deviations are proportional to their nominal values, we propose strengthened capacity inequalities in our paper, and we show that they strictly dominate inequalities by Gounaris et al. (2013).

In our paper we also develop an iterated search heuristic with variable neighbourhood search which handles both uncertainty polytopes $\mathcal{D}^{\text{card}}$ and $\mathcal{D}^{\text{part}}$. We show experimentally that our heuristic outperforms experimentally both the AMP heuristic proposed by Gounaris et al. (2016) and its combination with the branch-and-price algorithm by Gounaris et al. (2013).

To solve model (3.25), we use our generic BCP solver presented in Section 2.4. The value of the best solution obtained by our iterated search heuristic is used as initial upper bound for the solver.

3.5.3 Computational results

For our experiments, we use instances proposed by Gounaris et al. (2013) for the uncertainty polytope $\mathcal{D}^{\text{part}}$ as well as our instances for the uncertainty polytope $\mathcal{D}^{\text{card}}$. Both sets of instances are based on 90 classical CVRP instances. Description of the computational setup and the procedure to generate the instances is given in our paper.

We report in Table 3.10 consolidated results of the proposed branch-cut-and price method (BCP) and its comparison against AMP+BC. The latter is the branch-and-cut algorithm by Gounaris et al. (2013) initialised by the heuristic by Gounaris et al. (2016). The table contains five columns reporting statistical data of the BCP root node, followed by three columns regarding the complete BCP runs and other three columns about AMP+BC. The BCP root columns report the average gaps between the pure column generation lower bounds and the best know solution costs (gap 0), the mean runtime to obtain such bounds (t. 0), the average gaps between the final root node lower bounds and the best know solution costs (gap 1), and the mean runtime to obtain such bounds (t. 1). For both BCP and AMP+BC, the corresponding last two columns report the mean total runtime (t.) and the number of instances for which the solution optimality has been proved (opt). For both methods, a runtime of 86,700 seconds is used when the instance cannot be solved within this time. Moreover, for BCP, the first column gives the average number of branch-cut-and-price nodes (nod.), and for AMP-BC, the first column gives the final gap between the obtained lower bound and the best known solution cost (gap).

Inst.		BCP root				BCP			AMP+BC		
class	#	gap 0	t. 0	gap 1	t. 1	nod.	t.	opt.	gap	t.	opt.
A	26	2.16%	0.70	0.00%	2.91	1.00	2.91	26	1.97%	3440.31	12
B	23	3.68%	1.31	0.01%	5.95	1.05	5.98	23	1.39%	250.96	13
E	11	2.31%	2.79	0.00%	11.40	1.00	11.40	11	2.19%	573.01	5
F	3	3.01%	139.10	0.30%	309.40	5.37	833.42	2	1.10%	55.76	2
M	3	1.66%	12.49	0.20%	52.44	3.33	153.51	3	2.70%	40681.81	1
P	24	1.27%	0.51	0.00%	1.47	1.00	1.48	24	2.09%	976.36	10
all	90	2.34%	1.17	0.02%	4.43	1.11	4.75	89	1.87%	981.90	43

Table 3.10: Branch-cut-and-price results for $\mathcal{D}^{\text{part}}$

From Table 3.10, it can be seen that the proposed BCP outperforms AMP+BC for all instance classes except F, where both methods solve two out of three instances having 44, 71 and 134 customers. In this case, the two smaller instances are harder for BCP because they have relatively long routes. Overall, BCP solves all instances but one while less than half of them are solved by AMP+BC. Although the only open instance has been tried for more than 24 hours without success, all other instances have been solved in less than 2 hours (7,200 seconds). The mean runtime of BCP

is two orders of magnitude smaller than that of AMP+BC. It is remarkable that almost all instances are solved at the root node. Note however that the root node for BCP includes the resolution of IP problems generated through the enumeration of all useful elementary routes by CPLEX, when such problems are small enough.

Applying the pre-processing based on reducing the cardinality of Θ, we could remove nearly 80% of the subproblems, and 22 of 90 instances were reduced to deterministic homogeneous CVRP (with one subproblem). The literature capacity cuts are already very effective, closing more than 60% of the gap. Yet, our new capacity cuts close 33% of the remaining gap, which is a significant improvement.

Table 3.11 summarizes the results of running BCP for the new benchmark instances. Each BCP run was limited to 2 hours. In the table, the columns follow the same structure as in Table 3.10.

Inst.		BCP root				BCP		
class	#	gap 0	t. 0	gap 1	t. 1	nod.	t.	opt.
A	27	2.19%	2.50	0.02%	17.88	1.11	18.81	27
B	23	5.14%	3.44	0.23%	41.48	1.50	69.54	20
E	13	1.91%	5.69	0.03%	24.84	1.16	27.79	13
F	2	2.00%	154.53	0.00%	222.81	1.00	222.83	2
M	2	4.03%	39.31	0.00%	108.21	1.00	108.23	2
P	23	1.59%	2.28	0.00%	10.74	1.00	10.75	23
all	90	2.79%	3.48	0.07%	22.47	1.17	26.47	87
Low $\hat{d}$	90	2.79%	3.80	0.03%	26.14	1.13	28.60	89
High $\hat{d}$	90	2.68%	3.36	0.16%	19.69	1.35	25.84	86
Low Γ	90	2.74%	4.28	0.10%	27.71	1.20	30.97	89
High Γ	90	2.67%	2.66	0.07%	14.77	1.17	17.20	87
Low Q	90	3.10%	3.79	0.43%	26.89	1.37	34.40	85
High Q	90	2.92%	3.83	0.18%	23.91	1.41	33.80	84

Table 3.11: Branch-cut-and-price results for $\mathcal{D}^{\text{card}}$

From Table 3.11, it can be seen that the overall performance of the proposed methods for the new benchmark set is roughly similar to that observed for the literature instances: most of the gap left by the column generation lower bound is closed but the combination of the proposed cuts with the literature cuts previously proposed for the deterministic version, and only a few instances could not be solved exactly withing two hours of runtime. A more detailed analysis however reveals that there are some cases where instances can still be challenging for the proposed algorithms. For the main class, the three instances that could not be solved belong to the class B, having 50, 56 and 63 customers. It is worth mentioning that four larger instances of the same class could be solved relatively easily. The main reason for not solving such instances is that all of them have root gaps larger than 1.5%, which is more than 20 times larger than the average gap for the whole benchmark set in the main configuration. Regarding other configurations, it can be seen that the instances with lower demand deviations are easier for BCP. We also observe that the

root gaps of BCP are significantly larger for high demand deviations, and for both higher and lower capacities, in the latter case being more than six times larger than in the main configuration. For the lower capacity, we observed that the higher root gap is compensated by the stronger effect of fixing by reduced cost and enumeration in these instances, which leads to a number of solved instances that is not very much different from the main configuration. Overall, only 23 out of 630 instances could not be solved exactly within 2 hours, ranging from 50 to 100 customers, where 18 of them are from the class B, 4 are from the class E and 1 is from the class A.

The number of pricing problems left after the preprocessing based on reducing the cardinality of Θ is roughly 15% smaller than before. The literature cuts close 40% of the gap for these instances, and our new ones close nearly 40% of the remaining gap.

3.5.4 Related work

The first study on the robust CVRP dates back to Sungur et al. (2008) who consider a variant of the robust CVRP where travel time is uncertain and the total travel time of each vehicle is bounded. As it was already mentioned above, Gounaris et al. (2013) and Gounaris et al. (2016) studied our problem with the uncertainty polytope $\mathcal{D}^{\text{part}}$. Gounaris et al. (2013) also considered the relationship between the robust CVRP and its chance-constrained distributionally robust counterpart. The latter problem is addressed more recently by Ghosal and Wiesemann (2018) where the authors characterize ambiguity sets that make the problem amenable to efficient numerical solutions. Dinh et al. (2018) developed branch-cut-and-price algorithms for the chance-constrained vehicle routing problem.

In parallel to the research devoted to the robust CVRP, the robust VRP with time windows and uncertain travel times has also been the subject of recent research. This starts with Agra et al. (2013) proposed different formulations for the problem, assessed on instances inspired by maritime transportation. More recently, Munari et al. (2019) addressed the CVRP with time windows using different compact and extended formulations. They also proposed a branch-cut-and-price algorithm. Their approach is however slower then ours as the complexity of solving the pricing problem depends on the definition of the uncertainty set. Indeed, it appears that time windows are harder to handle in the robust context than capacity restrictions, which is also detailed from a theoretical viewpoint by Pessoa et al. (2015).

Recently, Subramanyam et al. (2018) studied robust variant of a broad class of heterogeneous vehicle routing problems.

3.6 Other Applications

In this section, I briefly review other applications we considered recently.

In our journal paper Bulhoes et al. (2018d), we consider the Minimum Latency Problem (MLP), a variant of the well-known Traveling Salesman Problem in which the objective is to minimize the sum of waiting times of customers. This problem arises in many applications where customer satisfaction is more important than the total time spent by the server. The paper presents a novel branch-and-price algorithm

for MLP that strongly relies on new features for the *ng*-path relaxation, namely: (1) a new labeling algorithm with an enhanced dominance rule named multiple partial label dominance; (2) a generalized definition of *ng*-sets in terms of arcs, instead of nodes; and (3) a strategy for decreasing *ng*-set sizes when those sets are being dynamically chosen. Also, other elements of efficient exact algorithms for vehicle routing problems are incorporated into our method, such as reduced cost fixing, automatic dual stabilization presented in Section 2.2.2, route enumeration and strong branching. Computational experiments over TSPLIB instances are reported, showing that several instances not solved by the current state-of-the-art method can now be solved.

In our paper submitted to a journal (Bulhoes et al., 2018c), we deal with a very generic class of scheduling problems with identical/uniform/unrelated parallel machine environment. It considers well-known attributes such as release dates or sequence-dependent setup times and accepts any objective function defined over job completion times. Non-regular objectives are also supported. We introduce a branch-cut-and-price algorithm for such problems that makes use of non-robust cuts, i.e., cuts which change the structure of the pricing problem. This is the first time that such cuts are employed for machine scheduling problems. The algorithm also embeds other important techniques such as strong branching, reduced cost fixing and dual stabilization presented in Section 2.2.2. Computational experiments over literature benchmarks showed that the proposed algorithm is indeed effective and could solve many instances to optimality for the first time.

In our journal paper Pessoa et al. (2018b) we present a Branch-Cut-and-Price algorithm for the Heterogeneous Fleet VRP, including the related Multi-Depot VRP and Site Dependent VRP. The algorithm includes elements found in previous HFVRP algorithms (like route enumeration and Extended Capacity Cuts) and also elements (like limited memory R1Cs) only found in the most recent algorithms for the two most classical homogeneous VRP variants, Capacitated VRP and VRP with Time Windows. However, many of those elements were adapted in order to take advantage of the existence of several distinct subproblems, corresponding to each vehicle type. The computational results obtained were good. It seems that typical instances with up to 200 customers can now be expected to be solved to optimality (often in long runs). In contrast, the best previous algorithms had difficulties on solving instances with 100 customers. The new algorithm can also find several optimal solutions that can not be found by existing heuristic methods. An important additional contribution of this paper is introducing a carefully designed instance generator and using it to obtain new insights on the characteristics that make instances harder to be solved. We believe that the proposed set XH will be very useful to the VRP community, allowing a good benchmark of future heuristic and exact approaches for the HFVRP.

In our paper submitted to a journal (Marques et al., 2019), we propose a branch-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem in which delivery of products from a depot to customers is performed using intermediate depots called satellites. Our algorithm uses the generic BCP solver presented in Section 2.4. In addition, we make some problem-specific contributions. First, we introduce a new route based formulation for the problem which does not use variables

to determine product flows in satellites. Second, we introduce a new branching strategy which significantly decreases the size of the branch-and-bound tree. Third, we introduce a new family of satellite supply inequalities, and we empirically show that it improves the quality of the dual bound at the root node of the branch-and-bound tree. These inequalities are robust and thus can be used together with the solver by implementing the separation procedure in a callback function. Finally, extensive numerical experiments reveal that our algorithm can solve to optimality all literature instances with up to 200 customers and 10 satellites for the first time and thus double the size of instances which can be solved to optimality.

In our paper submitted to a journal (Ben Mohamed et al., 2019), we consider the two-echelon stochastic multi-period capacitated location-routing problem (2E-SM-CLRP). In it, one has to decide at each period on the number and the location of warehouse platforms as well as intermediate distribution platforms; while fixing the capacity of the links between them. The system must be dimensioned to enable an efficient distribution of goods to customers under a stochastic and time-varying demand. In the second echelon, the goal is to construct vehicle routes that visit customers from operating distribution platforms. The objective is to minimize the total expected cost. For this hierarchical decision problem, the model is a two-stage stochastic program with integer recourse. The decisions in the first-stage include location and capacity decisions to be fixed at each period over the planning horizon, while routing decisions in the second echelon are determined in the second-stage. We develop a Benders decomposition approach to solve the 2E-SM-CLRP. In our approach, the location and capacity decisions are taken by solving the Benders master problem. When these first-stage decisions are fixed, the resulting subproblem is a multi-depot vehicle-routing problem with limited capacities (CVRP-CMD) that is solved by our generic BCP solver presented in Section 2.4. Computational experiments show that several instances of realistic size can be solved optimally, and that relevant managerial insights are derived on the behavior of the design decisions under the stochastic multi-period characterization of the planning horizon.

Finally, we would like to mention the research which is not related to column generation and branch-(cut-)and-price. This research is devoted to scheduling and packing problems. Two joint papers with Philippe Baptiste propose innovative Mixed Integer Programming (MIP) formulations for single machine scheduling problems (Baptiste and Sadykov, 2009, 2010). Later we worked on the problem of scheduling of malleable jobs (they can be processed on several machine simultaneously) and devised an important dominance property (Sadykov, 2012a). It can be used for reducing the search space when solving the problem. Then we were interested in a truck scheduling problem arising in cross-docking terminals. We have showed that a special case of this problem is NP-hard and another special case can be solved in a polynomial time by a dynamic programming algorithm (Sadykov, 2012b). More recently our collaboration with Prof. Shunji Tanaka from Kyoto university led to an efficient branch-and-bound algorithm for the two machine flow-shop problem with the total flow time criterion (Detienne et al., 2016). An efficient dynamic programming labelling algorithm was proposed for the four-stage two-dimensional guillotine-cut bounded knapsack problem (Clautiaux et al., 2018).

Chapter 4

Perspectives

In this chapter, research directions are described which we think are important to follow in short and medium term.

In Section 4.1 we talk about possible ways to improve our generic BCP solver presented in Section 2.4. We discuss how on can possibly improve its efficiency and applicability. Alternative approaches to formulate and solve the subproblems in a generic way are also outlined.

In Section 4.2 we discuss problems which are important in practice, but challenging to solve with the current state-of-the-art approaches. These are problems involving synchronization of activities and resources, integrated problems, and problems with uncertain data.

4.1 Improving our BCP solver

Our generic solver discussed in Section 2.4 is arguably the most complex BCP algorithm ever implemented. However, we believe that it can still be much improved. In Section 4.1.1 we discuss possible ways to improve its efficiency for "core" problems such as classical vehicle routing. Then, we argue in Section 4.1.2 that modelling possibilities of our solver can further be extended for a significantly broader class of problems. Finally, in Section 4.1.3 we discuss alternative approaches to formulate and solve the subproblems.

4.1.1 Efficiency

Our generic BCP algorithm has the state-of-the-art performance when applied to several classic combinatorial optimization problems. Many researches worked on exact approaches for these problems. Thus, one may think that no any further significant progress for exact solution of these problems can be obtained. However, very recently a breakthrough has been achieved for exact solution of classic vehicle routing problems (Pecin et al., 2017b). The size of instances which can be solved to optimality has been doubled. This suggests that further significant progress is still possible. We see an opportunity in developing or extending the following techniques.

- The route enumeration technique by Baldacci et al. (2008) allows one in many

cases avoid branching and solve many instances at the root. Nevertheless branching is an important part of a BCP algorithm if one wants to solve hard or large instances. However, the deeper we go in the branch-and-bound tree, the harder is to find good branching candidates even when using strong branching. Standard way to branch (on edges of the graph) does not always provide good candidates. Thus we need to develop alternative branching strategies. One possible strategy is to branch on sets of vertices, i.e. on the number of paths visiting this set or on the number of arcs on the border of this set participating in the solution. This strategy can provide much better candidates, as in the case of Travelling Salesman Problem (Applegate et al., 2006). However, such candidates are more difficult to find, as there are much more sets of vertices than arcs in a graph. Novel and original approaches to search for good candidates are needed.

- Another important technique which is mentioned in Section 1.3 is graph filtering (or arc elimination) using reduced costs (Irnich et al., 2010). Until now, this technique is used in one way: if the current reduced cost of an arc is larger than the current primal-dual gap (assuming an optimal dual solution) then this arc is eliminated from the graph. Other implications can also be used. For example, if the total reduced cost of a subset of arcs exceeds the current primal-dual gap, then at least one arc in this subset does not participate in an optimal solution. Ways to exploit this and similar implications can be derived. We can also devise branching strategies based on arc reduced costs, i.e. branching strategies which are based not only on a primal solution of the relaxation but also on a dual solution.

- Although dual price smoothing stabilization proposed in Section 2.2.2 is used in our BCP algorithm, in some cases column generation convergence is still slow. Often this happens when solving instances with long paths, i.e. with columns having non-zero coefficients in many master constraints. We have shown experimentally in Section 2.2.4 that other stabilization techniques such as based on penalty functions can be more efficient than dual price smoothing. However, parameterisation of these techniques in a generic algorithm is still a challenge, especially after adding many cutting planes and branching constraints. Thus, further work on automatic parameterisation of different stabilization approaches should be performed.

- Limited-memory Chvatal-Gomory rank-1 cuts (Pecin et al., 2017b) are instrumental for obtaining state-of-the-art performance for many problems. They significantly improve the dual bound obtained by column generation. However, when solving instances with short paths, i.e. short routes in vehicle routing, rank-1 cuts obtained by Ghvatal-Gomory rounding of many set-packing constraints (2.35) (six and more) are necessary for obtaining strong bounds. Separating such cuts is a challenge, and efficient algorithm for this task should be devised.

- Pricing problem solution time is usually the bottleneck of our BCP algorithm. Reducing the running time of the labeling algorithm we use is an important

task. One of possibilities is its parallelization. Modern computers have usually several central processing units (CPU) and many graphics processing units (GPU). CPU and GPU parallelisation of dynamic programming brings usually significant speed-ups, as in Boschetti et al. (2017).

4.1.2 Applicability

Although our solver implements by far the most generic state-of-the-art BCP algorithm for vehicle routing problems, its applicability is limited. Our goal is to extend the applicability of the solver to many other problems and some other classes of problems.

Our approach can readily be used to model and solve some problems which are non-VRPs. Some of the examples we identified are in scheduling (Sadykov and Wolsey, 2006), network design (Balakrishnan et al., 2017), resource allocation (Kramer et al., 2019), and human organ exchange (Mak-Hau, 2017). It is however unclear what will be the performance of the solver for each of these problems.

At the moment our solver supports only the most simple type of resources, for which the arc cost and the resource consumption on arcs are constants and do not depend on the accumulated resource consumption. There exists several generalizations. One can use resource extension functions as in Irnich (2008). The case where the arc cost depends on accumulated resource consumption is discussed in Ioachim et al. (1998). Finally, Parmentier (2019) presents efficient algorithm for the case with non-linear and stochastic resources. Our aim is to support in a generic way a class of resources which is as broad as possible.

The only families of cutting planes supported by the solver are Chvatal-Gomory rank-1 cuts and Rounded Capacity Cuts. Other generic families of valid inequalities may also be considered. For example, several families exist for knapsack-type constraints. Separation of such valid inequalities may be important to improve the quality of column generation bounds. Sometimes, knapsack constraints can be defined not only for variables mapped to arcs, but also for variables mapped to the resources consumption of paths. Valid inequalities for such constraints are non-robust. We have already initiated the work to separate such non-robust cuts and to devise techniques to support them in the pricing problem. Our preliminary results in Liguori et al. (2019) show that there is some potential benefits of this approach.

The primal heuristics implemented in our solver are restricted master and various diving heuristics presented in Section 2.3. Performance of these heuristics can be improved by combination with path enumeration as we did in Marques et al. (2019). However, all these heuristics are quite slow in comparison with recent meta-heuristics like (Vidal et al., 2014) which are also quite generic. Moreover, our heuristics does not scale well. In fact, the performance of our solver in many cases depends heavily on initial upper bounds obtained by running problem-specific heuristics. We need to develop a fast generic heuristic generating good quality solutions which can be used as initial upper bounds. This task is very challenging. Generic MIP heuristics like the one by Rothberg (2007) may give us inspiration.

Another important task is to develop heuristics based on our solver for large-scale instances which are out of reach to be solved to optimality. In fact, recent

studies show that so-called *matheuristics* (i.e. heuristics based on mathematical programming techniques) can be competitive with more traditional meta-heuristics for complex and large-scale problems. One of matheuristic paradigms is to optimize subparts of solutions until a local optimum is reached, which is called POPMUSIC (Taillard and Voß, 2018). This method can also be viewed as exploring a very large neighbourhood using an exact algorithm. In fact, using exact exploration of a large neighbourhood one can quickly find solutions which are much more time consuming to obtain with more conventional exploration heuristics.

Another development direction concerns the solver implementation. For the moment, the solver relies on BaPCod (Vanderbeck et al., 2017), a prototype code which is difficult to maintain and extend. Moreover, the interface is written using old versions of Julia and JuMP (Dunning et al., 2017). Our goal to is develop an alternative generic BCP code called *Coluna*: `github.com/atoptima/Coluna.jl` and connect the solver interface with this new code.

4.1.3 Alternative approaches to formulate and solve the subproblems

The state-of-the-art performance of our BCP solver can be explained by the fact that we rely on an efficient labelling algorithm to solve the subproblems, i.e. to efficiently obtain resource constrained paths with small reduced cost in a suitable graph. In other words, we exploit the "resource constrained path structure" of problems. However, there exist some alternative modelling approaches. For example, the subproblems can be formulated in other generic paradigms, for example using decision diagrams (Bergman et al., 2016) or context-free grammars (Côté et al., 2013). Labelling algorithm may be replaced by an alternative solution method like the Pulse framework Lozano and Medaglia (2013) or the Successive Sublimation Dynamic Programming (Clautiaux et al., 2019a). Generalization from resource constrained paths in graphs to resource-constrained flows in hypergraphs (Clautiaux et al., 2018) allows one to model a broader class of problems.

In our works we suppose that the structure of the problem to solve has already been detected, i.e. the subproblem have been identified and formulated in a convenient way. Often in practice users are not always able to identify the structure of their problem. A common way to formulate a problem in practice is to use Mixed Integer Programming (MIP). If a problem is formulated using MIP, it makes sense to try to automatically identify its block-diagonal structure. There are methods to do it proposed for example by Bergner et al. (2015) and Khaniyev et al. (2018). These works can be viewed as complementary to ours. We find that the main drawback of these approaches is that automatically detected subproblems are then soved by generic MIP solvers. This usually results in a slow BCP algorithm. A challenge here is to detect subproblems not only as general MIPs but also structures of other types, as network flows, resource constrained paths in a graph, etc. It is however not clear whether it is possible to do at all. For the moment, we think that the best way is to educate the user how to identify the structure of their problems and to communicate this structure to the solver.

4.2 Towards practical problems

The applications we consider in this manuscript are mostly academic problems. In practice it is very rare to encounter applications which are pure academic problems or can be reduced to them. Therefore, it is important to think about solution methods for more practical problems. In this section we describe problems which integrate aspects found in many real-life applications and present possible approaches to tackle them. In Section 4.2.1, we consider problems involving synchronization of activities or resources. Then, in Section 4.2.2 we talk about integrated problems, i.e. problems involving decision of different kinds or taken on different levels. Finally, in Section 4.2.3 we see how data uncertainty can be taken into account in the solution approaches.

4.2.1 Synchronization

In practice, vehicle routing and scheduling problems often involve synchronization of activities and ressources. It can take form of precedence constraints between jobs or visits to clients, positive or negative time lags, or synchronization of different types of vehicles or other ressources. Examples are numerous. These are shop problem in scheduling, consistency problems in periodic vehicle routing, pickup and delivery problems, project scheduling, and many others. Real-life instances of such problems are considered to be out of reach of exact methodes and they are usually tackled by heuristic approaches. However, to estimate the quality of available heuristics we definitely need at least techniques to obtain good lower bounds. This is very challenging.

One possible approach is hybridization of mathematical programming techniques with constraint programming. The latter has recently achieved an important progress by incorporating lazy clause generation technique originated in SAT solvers, see for example Schutt et al. (2013). This technique allows one to solve much faster very constrained problems. Many problems with synchronization fall into this class. However, constraint programming still has enormous difficulties to deal with problems with sum-type objective functions, such as total earliness-tardiness in scheduling or total travelling distance in vehicle routing. Thus, complementarity between mathematical programming and constraint programming becomes obvious. However, integrating constraint programming (CP) techniques to BCP approaches has not been much successful until now. Despite the fact that both CP and BCP approaches are based on enumeration trees, the reasons of their success is different. BCP approaches are successful when the root bound is strong so that the enumeration tree is small. On the contrary, modern CP approaches usually have large enumeration trees. They "prefer" branching a lot in order to quickly discover conflicts, and then they find "explanations" of these conflicts and generate new constraints based on these explanations. Probably CP techniques can be useful for BCP methods when run in parallel or in the pre-processing stage.

Another approach is to take into account synchronization constraints progressively: first we relax them, and then we impose some of these constrains by generating cuts or adding additional resources. Elementarity constraints are treated in the

same way in modern BCPs. The success of dynamic *ng*-paths (Roberti and Mingozzi, 2014) and dynamic state-space relaxations (DSSR) (Martinelli et al., 2014) for vehicle routing problems relies on this approach. Our experiments on the most difficult Solomon instances of the VRPTW show that using dynamic *ng*-paths we can generate almost the same gap (0.91%) as with only elementary paths (0.89%) by taking only 45% of the time. Synchronization can sometimes be modelled using flow variables as, for example, synchronization between echelons in the two-echelon vehicle routing problem. We showed in Marques et al. (2019) that these flow variables can be replaced by an exponential number of constraints planes using "max-flow-min-cut" relation. These constraints can then be generated dynamically as "core" cutting planes. We now plan to extend this approach to the two-echelon vehicle routing problem with time windows, in which in addition we need to insure time synchronization between echelons, and to the vehicle routing problem with cross-docking.

Problems involving consolidation of shipments also involve synchronization constraints, as consolidated shipments shown be synchronized in time. A standard approach to solve such kind of problem is discretization and formulation as a flow problem with side constraints in a very large space-time network. These formulation are often unsolvable due to their size. Recently, new approaches have been proposed by Clautiaux et al. (2017) and Boland et al. (2017) to dynamically manage the size of these large-scale networks. We see an opportunity in integrating these approaches in a modern BCP with resource constrained shortest path subproblems in order to solver problems with very large graphs.

4.2.2 Integration

Recently, there has been an increase of the interest to solve integrated optimization problems. These problems address jointly two and more decision stages. Such problems may involve for example integration of production and outbound distribution (Fu et al., 2018), facility location and vehicle routing (Schneider and Drexl, 2017), inventory management and vehicle routing (Coelho et al., 2014). Integrated problems are difficult. They often involve synchronization between stages, different time scale in different stages and thus uncertainty in data (Ben Mohamed et al., 2019).

Exact approaches are often limited to small instances of integrated problems. Real-life instances are usually tackled by heuristic approaches. Nevertheless, for estimating quality of these heuristics we need approaches which obtain good quality lower bounds for large scale instances. We think that BCP algorithms are good candidates to obtain such bounds. Column generation approach has however several drawbacks when applied to large-scale integrated problems. When solving pure academic problems, the bottleneck of BCP algorithms is usually the algorithm to solve the pricing problem. On the contrary, when dealing with integrated real-life problems, the master problem may grow very large and the algorithm to solve the restricted master LP may become a bottleneck. Some approaches have been proposed in the literature to tackle this problem, for example a dynamic aggregation of constraints in the master (Elhallaoui et al., 2005). In recent conference presentations it was proposed to use machine learning techniques to help to converge the

column generation faster in the case of "heavy" master problem. Machine learning is either used to choose "good" columns from many columns generated by the pricing problem or to obtain constraint aggregation to initialize the master problem. We have recently experienced difficulties when the restricted master problem LP could not be solved in a reasonable time by a modern LP solver when tackling large-scale instances of the two-echelon vehicle routing problem. We need to understand reasons of such behaviour and propose solutions to overcome this drawback.

Column generation-based algorithms may not only be useful for finding lower bounds, but also for obtaining good feasible solutions for integrated problems. A perspective direction here is to develop matheuristics. Recently column generation-based matheuristics showed excellent results for some integrated problems (Amarouche et al., 2018; Wang et al., 2018). A common approach is to use heuristics and/or column generation to obtain interesting columns and then solve the restricted master problem as a MIP with a time limit. Then the set of columns can be possible updated and the restricted master is resolved. It would be very useful to come with some generic guidelines how to construct the set of columns which have a high chance to be combined into a good feasible solution. A generic heuristic of this type would also be highly welcome, at least for a wide class of problems.

4.2.3 Uncertainty

Interest in solving many combinatorial optimization problems exactly, in particular vehicle routing problems, is limited in practice. In many cases, the data cannot be determined exactly, i.e. vehicle transition times and costs, client demands. In such cases, generating a solution which is 1-2% better does not make sense. We need to take data uncertainty into account when deciding which solution is better. Several approaches to do this exist in the literature: stochastic optimization, (distributionally) robust optimization, data-driven optimization, chance constrained optimization. Unfortunately, they are computationally much more difficult than conservative approaches (for example, adding slacks). We need to find a good trade-off between computational difficulty and conservativeness.

In what concerns vehicle routing, up to now uncertainty has been taken into account mainly when dealing with basic problems such as CVRP (Gounaris et al., 2013; Dinh et al., 2018; Ghosal and Wiesemann, 2018; Oyola et al., 2018) or VRP with time windows (Agra et al., 2013; Munari et al., 2019). The variant with time windows is more challenging to solve. Therefore, alternative criteria for good solutions under travel time uncertainty has been proposed recently (Zhang et al., 2019). Another approach is to approximate travel time uncertainty using collected data about congestion. This results in time dependent vehicle routing problems (Gendreau et al., 2015). Unfortunately, the above approaches result in problems which are time consuming to solve. Further progress is needed to obtain faster algorithms, in the same vein as our recent work on robust CVRP (Pessoa et al., 2018a). The variant with time windows is especially interesting for us.

As it is mentioned in Section 4.2.2, integrated problems often involve different time scale for different decisions. For example, decisions to open/close/rent depots can be taked monthly or yearly, whereas routing should be done daily. Together

with data uncertainty, this results in multi-stage stochastic problems. A multi-stage problem can be approximated by two-stage problems, as we did in Ben Mohamed et al. (2019). However, such approximation may not be satisfactory in terms of the solution quality. Other "computationally feasible" approaches should be developed which better approximate multi-stage structure of such problems.

Bibliography

Achterberg, T., 2007. Constraint integer programming. Ph.D. thesis, Technische Universitat Berlin.

Agra, A., Christiansen, M., Figueiredo, R., Hvattum, L. M., Poss, M., Requejo, C., 2013. The robust vehicle routing problem with time windows. Computers and Operations Research 40 (3), 856 – 866.

Alvarez-Valdes, R., Parajon, A., Tamarit, J. M., May 2002. A computational study of lp-based heuristic algorithms for two-dimensional guillotine cutting stock problems. OR Spectrum 24 (2), 179–192.

Amarouche, Y., Guibadj, R. N., Moukrim, A., 2018. A Neighborhood Search and Set Cover Hybrid Heuristic for the Two-Echelon Vehicle Routing Problem. In: Borndörfer, R., Storandt, S. (Eds.), 18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018). Vol. 65 of OpenAccess Series in Informatics (OASIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 11:1–11:15.

Andrade, R., Birgin, E., Morabito, R., 2016. Two-stage two-dimensional guillotine cutting stock problems with usable leftover. International Transactions in Operational Research 23 (1-2), 121–145.

Applegate, D. L., Bixby, R. E., Chvatál, V., Cook, W. J., 2006. The Traveling Salesman Problem: A Computational Study. Princeton University Press.

Archetti, C., Bianchessi, N., Speranza, M., 2013. Optimal solutions for routing problems with profits. Discrete Applied Mathematics 161 (4–5), 547–557.

Balakrishnan, A., Li, G., Mirchandani, P., 2017. Optimal network design with end-to-end service requirements. Operations Research 65 (3), 729–750.

Baldacci, R., Bartolini, E., Mingozzi, A., 2011a. An exact algorithm for the pickup and delivery problem with time windows. Operations Research 59 (2), 414–426.

Baldacci, R., Christofides, N., Mingozzi, A., 2008. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. Mathematical Programming 115, 351–385.

Baldacci, R., Mingozzi, A., 2009. A unified exact method for solving different classes of vehicle routing problems. Mathematical Programming 120 (2), 347–380.

Baldacci, R., Mingozzi, A., Roberti, R., 2011b. New route relaxation and pricing strategies for the vehicle routing problem. Operations Research 59 (5), 1269–1283.

Balinski, M. L., Quandt, R. E., 1964. On an integer program for a delivery problem. Operations Research 12 (2), 300–304.

Baptiste, P., Sadykov, R., 2009. On scheduling a single machine to minimize a piecewise linear objective function: A compact MIP formulation. Naval Research Logistics 56 (6), 487–502.

Baptiste, P., Sadykov, R., 2010. Time-indexed formulations for scheduling chains on a single machine: An application to airborne radars. European Journal of Operational Research 203 (2), 476 – 483.

Barahona, F., Anbil, R., 2000. The volume algorithm: producing primal solutions with a subgradient method. Mathematical Programming 87 (3), 385–399.

Beasley, J. E., 1985. Algorithms for unconstrained two-dimensional guillotine cutting. The Journal of the Operational Research Society 36 (4), 297–306.

Belov, G., 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 (1), 85 – 106.

Ben-Ameur, W., Neto, J., 2007. Acceleration of cutting-plane and column generation algorithms: Applications to network design. Networks 49 (1), 3–17.

Ben Amor, H., Desrosiers, J., Valério de Carvalho, J. M., 2006. Dual-Optimal Inequalities for Stabilized Column Generation. Operations Research 54 (3), 454–463.

Ben Amor, H. M., Desrosiers, J., Frangioni, A., 2009. On the choice of explicit stabilizing terms in column generation. Discrete Applied Mathematics 157 (6), 1167 – 1184.

Ben Mohamed, I., Klibi, W., Sadykov, R., Şen, H., Vanderbeck, F., 2019. A Benders decomposition approach for the two-echelon stochastic multi-period capacitated location-routing problem. HAL 02178459, Inria.

Bergman, D., Cire, A. A., van Hoeve, W.-J., Hooker, J. N., 2016. Discrete optimization with decision diagrams. INFORMS Journal on Computing 28 (1), 47–66.

Bergner, M., Caprara, A., Ceselli, A., Furini, F., Lübbecke, M. E., Malaguti, E., Traversi, E., Feb 2015. Automatic dantzig–wolfe reformulation of mixed integer programs. Mathematical Programming 149 (1), 391–424.

Berthold, T., 2006. Primal heuristics for mixed integer programs. Master's thesis, Technischen Universität Berlin.

Bertsimas, D., Sim, M., Sep 2003. Robust discrete optimization and network flows. Mathematical Programming 98 (1), 49–71.

Bettinelli, A., Cacchiani, V., Malaguti, E., 2017. A branch-and-bound algorithm for the knapsack problem with conflict graph. INFORMS Journal on Computing 29 (3), 457–473.

Bianchessi, N., Mansini, R., Speranza, M. G., 2018. A branch-and-cut algorithm for the team orienteering problem. International Transactions in Operational Research 25 (2), 627–635.

Bienstock, D., Zuckerberg, M., 2010. Solving lp relaxations of large-scale precedence constrained problems. In: Eisenbrand, F., Shepherd, F. (Eds.), Integer Programming and Combinatorial Optimization. Vol. 6080 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 1–14.

Boland, N., Hewitt, M., Marshall, L., Savelsbergh, M., 2017. The continuous-time service network design problem. Operations Research forthcoming.

Boschetti, M. A., Maniezzo, V., Strappaveccia, F., 2017. Route relaxations on gpu for vehicle routing problems. European Journal of Operational Research 258 (2), 456 – 466.

Boyer, V., Gendron, B., Rousseau, L.-M., 2014. A branch-and-price algorithm for the multi-activity multi-task shift scheduling problem. Journal of Scheduling 17 (2), 185–197.

Brandão, F., Pedroso, J. a. P., 2016. Bin packing and related problems: General arc-flow formulation with graph compression. Computers & Operations Research 69, 56 – 67.

Briant, O., Lemaréchal, C., Meurdesoif, P., Michel, S., Perrot, N., Vanderbeck, F., 2008. Comparison of bundle and classical column generation. Mathematical Programming 113 (2), 299–344.

Brunner, J. O., Stolletz, R., 2014. Stabilized branch and price with dynamic parameter updating for discontinuous tour scheduling. Computers and Operations Research 44 (0), 137 – 145.

Bulhoes, T., Hà, M. H., Martinelli, R., Vidal, T., 2018a. The vehicle routing problem with service level constraints. European Journal of Operational Research 265 (2), 544 – 558.

Bulhoes, T., Pessoa, A., Protti, F., Uchoa, E., 2018b. On the complete set packing and set partitioning polytopes: Properties and rank 1 facets. Operations Research Letters 46 (4), 389 – 392.

Bulhoes, T., Sadykov, R., Subramanian, A., Uchoa, E., 2018c. On the exact solution of a large class of parallel machine scheduling problems. Cadernos do LOGIS 2018/3, Universidade Federal Fluminense.

Bulhoes, T., Sadykov, R., Uchoa, E., May 2018d. A branch-and-price algorithm for the minimum latency problem. Computers & Operations Research 93, 66–78.

Carraghan, R., Pardalos, P. M., 1990. An exact algorithm for the maximum clique problem. Operations Research Letters 9 (6), 375 – 382.

Cintra, G., Miyazawa, F., Wakabayashi, Y., Xavier, E., 2008. Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. European Journal of Operational Research 191 (1), 61 – 85.

Clautiaux, F., Detienne, B., Guillot, G., Feb. 2019a. Dynamic programming approaches for the temporal knapsack problem. Tech. Rep. 02044832, HAL-Inria.

Clautiaux, F., Hanafi, S., Macedo, R., Émilie Voge, M., Alves, C., 2017. Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints. European Journal of Operational Research 258 (2), 467 – 477.

Clautiaux, F., Sadykov, R., Vanderbeck, F., Viaud, Q., 2018. Combining dynamic programming with filtering to solve a four-stage two-dimensional guillotine-cut bounded knapsack problem. Discrete Optimization 29, 18–44.

Clautiaux, F., Sadykov, R., Vanderbeck, F., Viaud, Q., May 2019b. Pattern-based diving heuristics for a two-dimensional guillotine cutting-stock problem with leftovers. EURO Journal on Computational Optimization In Press.

Coelho, L. C., Cordeau, J.-F., Laporte, G., 2014. Thirty years of inventory routing. Transportation Science 48 (1), 1–19.

Contardo, C., Martinelli, R., 2014. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. Discrete Optimization 12, 129 – 146.

Côté, M.-C., Gendron, B., Rousseau, L.-M., 2011. Grammar-based integer programming models for multiactivity shift scheduling. Management Science 57 (1), 151–163.

Côté, M.-C., Gendron, B., Rousseau, L.-M., 2013. Grammar-based column generation for personalized multi-activity shift scheduling. INFORMS Journal on Computing 25 (3), 461–474.

Danna, E., Rothberg, E., Pape, C. L., Jan 2005. Exploring relaxation induced neighborhoods to improve mip solutions. Mathematical Programming 102 (1), 71–90.

Dantzig, G. B., 1954. A comment on Edie's "Ttraffic delays at toll booths". Journal of the Operations Research Society of America 2 (3), 339–343.

Dantzig, G. B., Ramser, J. H., 1959. The truck dispatching problem. Management Science 6 (1), 80–91.

Dantzig, G. B., Wolfe, P., 1960. Decomposition principle for linear programs. Operations Research 8 (1), 101–111.

de Oliveira, W., Sagastizábal, C., 2014. Level bundle methods for oracles with on-demand accuracy. Optimization Methods and Software 29 (6), 1180–1209.

Delorme, M., Iori, M., 2018. Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. INFORMS Journal on Computing accepted.

Demassey, S., Pesant, G., Rousseau, L.-M., 2006. A cost-regular based hybrid column generation approach. Constraints 11 (4), 315–333.

Desaulniers, G., Desrosiers, J., loachim, I., Solomon, M. M., Soumis, F., Villeneuve, D., 1998. A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems. Springer US, Boston, MA, pp. 57–93.

Desaulniers, G., Desrosiers, J., Solomon, M. M., 2002. Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems. Springer US, Boston, MA, pp. 309–324.

Desaulniers, G., Desrosiers, J., Spoorendonk, S., 2011. Cutting planes for branch-and-price algorithms. Networks 58 (4), 301–310.

Desaulniers, G., Lessard, F., Hadjar, A., 2008. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. Transportation Science 42 (3), 387–404.

Desrosiers, J., Lübbecke, M. E., 2011. Branch-price-and-cut algorithms. In: Wiley Encyclopedia of Operations Research and Management Science. American Cancer Society.

Detienne, B., Sadykov, R., Tanaka, S., 2016. The two-machine flowshop total completion time problem: Branch-and-bound algorithms based on network-flow formulation. European Journal of Operational Research 252 (3), 750 – 760.

Dinh, T., Fukasawa, R., Luedtke, J., Nov 2018. Exact algorithms for the chance-constrained vehicle routing problem. Mathematical Programming 172 (1), 105–138.

Dohn, A., Mason, A., 2013. Branch-and-price for staff rostering: An efficient implementation using generic programming and nested column generation. European Journal of Operational Research 230 (1), 157 – 169.

Dolatabadi, M., Lodi, A., Monaci, M., 2012. Exact algorithms for the two-dimensional guillotine knapsack. Computers & Operations Research 39 (1), 48 – 53.

du Merle, O., Villeneuve, D., Desrosiers, J., Hansen, P., 1999. Stabilized column generation. Discrete Mathematics 194 (1-3), 229–237.

Dunning, I., Huchette, J., Lubin, M., 2017. JuMP: A modeling language for mathematical optimization. SIAM Review 59 (2), 295–320.

Dusberger, F., Raidl, G. R., 2015. Solving the 3-staged 2-dimensional cutting stock problem by dynamic programming and variable neighborhood search. Electronic Notes in Discrete Mathematics 47, 133 – 140.

Elhallaoui, I., Villeneuve, D., Soumis, F., Desaulniers, G., 2005. Dynamic Aggregation of Set-Partitioning Constraints in Column Generation. Operations Research 53 (4), 632–645.

Elhedhli, S., Li, L., Gzara, M., Naoum-Sawaya, J., 2011. A Branch-and-Price Algorithm for the Bin Packing Problem with Conflicts. INFORMS Journal on Computing 23 (3), 404–415.

Ernst, A., Jiang, H., Krishnamoorthy, M., Sier, D., 2004. Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153 (1), 3 – 27.

Feillet, D., Gendreau, M., Medaglia, A. L., Walteros, J. L., 2010. A note on branch-and-cut-and-price. Operations Research Letters 38 (5), 346 – 353.

Fernandes Muritiba, A. E., Iori, M., Malaguti, E., Toth, P., 2010. Algorithms for the Bin Packing Problem with Conflicts. INFORMS Journal on Computing 22, 401–415.

Fischetti, M., Glover, F., Lodi, A., Sep 2005. The feasibility pump. Mathematical Programming 104 (1), 91–104.

Fischetti, M., Lodi, A., Sep 2003. Local branching. Mathematical Programming 98 (1), 23–47.

Fischetti, M., Salvagnin, D., 2010. An in-out approach to disjunctive optimization. In: Lodi, A., Milano, M., Toth, P. (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Vol. 6140 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 136–140.

Frangioni, A., Gendron, B., 2009. 0-1 reformulations of the multicommodity capacitated network design problem. Discrete Applied Mathematics 157 (6), 1229 – 1241.

Frangioni, A., Gendron, B., 2013. A stabilized structured Dantzig-Wolfe decomposition method. Mathematical Programming 140 (1), 45–76.

Fu, L.-L., Aloulou, M. A., Artigues, C., Aug 2018. Integrated production and outbound distribution scheduling problems with job release dates and deadlines. Journal of Scheduling 21 (4), 443–460.

Fukasawa, R., de Aragão, M. P., Porto, O., Uchoa, E., 2002. Solving the freight car flow problem to optimality. Electronic Notes in Theoretical Computer Science 66 (6), 42 – 52.

Fukasawa, R., Longo, H., Lysgaard, J., Aragão, M. P. d., Reis, M., Uchoa, E., Werneck, R. F., 2006. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Mathematical Programming 106 (3), 491–511.

Furini, F., Malaguti, E., Durán, R. M., Persiani, A., Toth, P., 2012. A column generation heuristic for the two-dimensional two-staged guillotine cutting stock problem with multiple stock size. European Journal of Operational Research 218 (1), 251 – 260.

Furini, F., Malaguti, E., Thomopulos, D., 2016. Modeling two-dimensional guillotine cutting problems via integer programming. INFORMS Journal on Computing 28 (4), 736–751.

Galati, M., 2009. Decomposition methods for integer linear programming. Ph.D. thesis, Lehigh University.

Gamrath, G., Lübbecke, M., 2010. Experiments with a generic Dantzig-Wolfe decomposition for integer programs. In: Festa, P. (Ed.), Experimental Algorithms. Vol. 6049 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 239–252.

Gélinas, S., Desrochers, M., Desrosiers, J., Solomon, M. M., Dec 1995. A new branching strategy for time constrained routing problems with application to backhauling. Annals of Operations Research 61 (1), 91–109.

Gendreau, M., Ghiani, G., Guerriero, E., 2015. Time-dependent routing problems: A review. Computers and Operations Research 64, 189 – 197.

Gendreau, M., Laporte, G., Semet, F., 2004. Heuristics and lower bounds for the bin packing problem with conflicts. Computers and Operations Research 31 (3), 347 – 358.

Gérard, M., 2015. Heuristiques basées sur la génération de colonnes pour un problème de planification du personnel. Ph.d. thesis (in french), University Lille I.

Gérard, M., Clautiaux, F., Sadykov, R., 2016. Column generation based approaches for a tour scheduling problem with a multi-skill heterogeneous workforce. European Journal of Operational Research 252 (3), 1019 – 1030.

Ghosal, S., Wiesemann, W., 2018. The distributionally robust chance constrained vehicle routing problem. Tech. Rep. 6759, Optimization Online.

Gilmore, P. C., Gomory, R. E., 1965. Multistage cutting stock problems of two and more dimensions. Operations Research 13 (1), 94–120.

Goffin, J.-L., Vial, J.-P., 2002. Convex nondifferentiable optimization: A survey focused on the analytic center cutting plane method. Optimization Methods and Software 17 (5), 805–867.

Gondzio, J., González-Brevis, P., Munari, P., 2013. New developments in the primal–dual column generation technique. European Journal of Operational Research 224 (1), 41 – 51.

Gounaris, C. E., Repoussis, P. P., Tarantilis, C. D., Wiesemann, W., Floudas, C. A., 2016. An adaptive memory programming framework for the robust capacitated vehicle routing problem. Transportation Science 50 (4), 1239–1260.

Gounaris, C. E., Wiesemann, W., Floudas, C. A., 2013. The robust capacitated vehicle routing problem under demand uncertainty. Operations Research 61 (3), 677–693.

Gschwind, T., Irnich, S., 2016. Dual inequalities for stabilized column generation revisited. INFORMS Journal on Computing 28 (1), 175–194.

Gschwind, T., Irnich, S., Rothenbächer, A.-K., Tilk, C., 2018. Bidirectional labeling in column-generation algorithms for pickup-and-delivery problems. European Journal of Operational Research 266 (2), 521 – 530.

Gurobi Optimization, L., 2017. Gurobi optimizer reference manual, version 7.5. `http://www.gurobi.com`, [Online; accessed 18-July-2019].

Hadjiconstantinou, E., Iori, M., 2007. A hybrid genetic algorithm for the two-dimensional single large object placement problem. European Journal of Operational Research 183 (3), 1150 – 1166.

Harvey, W. D., Ginsberg, M. L., 1995. Limited discrepancy search. In: Proceedings of the 14th international joint conference on Artificial intelligence (IJCAI'95). Vol. 1. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 607–615.

Held, S., Cook, W., Sewell, E. C., 2012. Maximum-weight stable sets and safe lower bounds for graph coloring. Mathematical Programming Computation 4 (4), 363–381.

Hernández-Leandro, N. A., Boyer, V., Salazar-Aguilar, M. A., Rousseau, L.-M., 2019. A matheuristic based on lagrangian relaxation for the multi-activity shift scheduling problem. European Journal of Operational Research 272 (3), 859 – 867.

Heßler, K., Gschwind, T., Irnich, S., 2018. Stabilized branch-and-price algorithms for vector packing problems. European Journal of Operational Research 271 (2), 401 – 419.

Hifi, M., Michrafy, M., 2007. Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. Computers and Operations Research 34 (9), 2657 – 2673.

Holmberg, K., Joborn, M., Lundgren, J. T., 1998. Improved empty freight car distribution. Transportation Science 32 (2), 163–173.

Ioachim, I., Gélinas, S., Soumis, F., Desrosiers, J., 1998. A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. Networks 31 (3), 193–204.

Irnich, S., Jan 2008. Resource extension functions: properties, inversion, and generalization to segments. OR Spectrum 30 (1), 113–148.

Irnich, S., Desaulniers, G., 2005. Shortest Path Problems with Resource Constraints. Springer US, Boston, MA, pp. 33–65.

Irnich, S., Desaulniers, G., Desrosiers, J., Hadjar, A., 2010. Path-reduced costs for eliminating arcs in routing and scheduling. INFORMS Journal on Computing 22 (2), 297–313.

Jansen, K., 1999. An approximation scheme for bin packing with conflicts. Journal of Combinatorial Optimization 3 (4), 363–377.

Jepsen, M., Petersen, B., Spoorendonk, S., Pisinger, D., 2008. Subset-row inequalities applied to the vehicle-routing problem with time windows. Operations Research 56 (2), 497–511.

Joncour, C., Michel, S., Sadykov, R., Sverdlov, D., Vanderbeck, F., 2010. Column generation based primal heuristics. Electronic Notes in Discrete Mathematics 36, 695 – 702.

Khanafer, A., Clautiaux, F., Talbi, E.-G., 2010. New lower bounds for bin packing problems with conflicts. European Journal of Operational Research 206 (2), 281 – 288.

Khaniyev, T., Elhedhli, S., Erenay, F. S., 2018. Structure detection in mixed-integer programs. INFORMS Journal on Computing 30 (3), 570–587.

Kramer, A., Lalla-Ruiz, E., Iori, M., Voß, S., 2019. Novel formulations and modeling enhancements for the dynamic berth allocation problem. European Journal of Operational Research 278 (1), 170–185.

Laporte, G., Nobert, Y., Jun 1983. A branch and bound algorithm for the capacitated vehicle routing problem. Operations-Research-Spektrum 5 (2), 77–85.

Lee, C., Park, S., 2011. Chebyshev center based column generation. Discrete Applied Mathematics 159 (18), 2251 – 2265.

Lee, T., Kwon, C., Dec 2014. A short note on the robust combinatorial optimization problems with cardinality constrained uncertainty. 4OR 12 (4), 373–378.

Lemaréchal, C., Nemirovskii, A., Nesterov, Y., Jul 1995. New variants of bundle methods. Mathematical Programming 69 (1), 111–147.

Liguori, P. H. P. V., Mahjoub, A. R., Sadykov, R., Uchoa, E., 2019. A branch-and-cut-and-price algorithm for the capacitated location-routing problem. In: Book of abstracts of the Tenth Triennial Symposium on Ttransportation Analysis

TRISTAN X. pp. http://www.tristan2019.org/wp--content/uploads/2019/07/TRISTAN--Abstracts.pdf.

Linderoth, J. T., Savelsbergh, M. W. P., 1999. A computational study of search strategies for mixed integer programming. INFORMS Journal on Computing 11 (2), 173–187.

Löbel, A., 1998. Vehicle scheduling in public transit and lagrangean pricing. Management Science 44 (12), 1637–1649.

Lozano, L., Medaglia, A. L., 2013. On an exact method for the constrained shortest path problem. Computers & Operations Research 40 (1), 378 – 384.

Lübbecke, M., Puchert, C., 2012. Primal heuristics for branch-and-price algorithms. In: Klatte, D., Lüthi, H.-J., Schmedders, K. (Eds.), Operations Research Proceedings 2011. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 65–70.

Lübbecke, M. E., Desrosiers, J., 2005. Selected topics in column generation. Operations Research 53 (6), 1007–1023.

Lysgaard, J., 2003. CVRPSEP: a package of separation routines for the capacitated vehicle routing problem. http://econ.au.dk/research/researcher-websites/jens-lysgaard/cvrpsep/, [Online; accessed 18-July-2019].

Macedo, R., Alves, C., de Carvalho, J. V., 2010. Arc-flow model for the two-dimensional guillotine cutting stock problem. Computers and Operations Research 37 (6), 991 – 1001.

Mak-Hau, V., Jan 2017. On the kidney exchange problem: cardinality constrained cycle and chain problems on directed graphs: a survey of integer programming approaches. Journal of Combinatorial Optimization 33 (1), 35–59.

Mamer, J. W., McBride, R. D., 2000. A decomposition-based pricing procedure for large-scale linear programs: An application to the linear multicommodity flow problem. Management Science 46 (5), 693–709.

Marques, G., Sadykov, R., Deschamps, J.-C., Dupas, R., April 2019. An improved branch-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem. HAL 02112287, Inria.

Martin, R. K., Rardin, R. L., Campbell, B. A., 1990. Polyhedral Characterization of Discrete Dynamic Programming. Operations Research 38 (1), 127–138.

Martinelli, R., Pecin, D., Poggi, M., 2014. Efficient elementary and restricted non-elementary route pricing. European Journal of Operational Research 239 (1), 102 – 111.

Munari, P., Moreno, A., De La Vega, J., Alem, D., Gondzio, J., Morabito, R., 2019. The robust vehicle routing problem with time windows: Compact formulation and branch-price-and-cut method. Transportation Science 53 (4), 1043–1066.

Muñoz, G., Espinoza, D., Goycoolea, M., Moreno, E., Queyranne, M., Letelier, O. R., Mar 2018. A study of the bienstock–zuckerberg algorithm: applications in mining and resource constrained project scheduling. Computational Optimization and Applications 69 (2), 501–534.

Muter, I., Birbil, c. I., Bülbül, K., 2013. Simultaneous column-and-row generation for large-scale linear programs with column-dependent-rows. Mathematical Programming 142 (1-2), 47–82.

Neame, P. J., 2000. Nonsmooth dual methods in integer programming. Ph.D. thesis, University of Melbourne, Department of Mathematics and Statistics.

Nemhauser, G. L., Park, S., 1991. A polyhedral approach to edge coloring. Operations Research Letters 10 (6), 315 – 322.

Oyola, J., Arntzen, H., Woodruff, D. L., Sep 2018. The stochastic vehicle routing problem, a literature review, part i: models. EURO Journal on Transportation and Logistics 7 (3), 193–221.

Pan, S., Calvo, R. W., Akplogan, M., Létocart, L., Touati, N., 2019. A dual ascent heuristic for obtaining a lower bound of the generalized set partitioning problem with convexity constraints. Discrete Optimization 33, 146 – 168.

Parmentier, A., Apr 2019. Algorithms for non-linear and stochastic resource constrained shortest path. Mathematical Methods of Operations Research 89 (2), 281–317.

Pecin, D., Contardo, C., Desaulniers, G., Uchoa, E., 2017a. New enhancements for the exact solution of the vehicle routing problem with time windows. INFORMS Journal on Computing 29 (3), 489–502.

Pecin, D., Pessoa, A., Poggi, M., Uchoa, E., 2017b. Improved branch-cut-and-price for capacitated vehicle routing. Mathematical Programming Computation 9 (1), 61–100.

Pecin, D., Pessoa, A., Poggi, M., Uchoa, E., Santos, H., 2017c. Limited memory rank-1 cuts for vehicle routing problems. Operations Research Letters 45 (3), 206 – 209.

Pecin, D., Uchoa, E., 2019. Comparative analysis of capacitated arc routing formulations for designing a new branch-cut-and-price algorithm. Transportation Science accepted.

Pesneau, P., Sadykov, R., Vanderbeck, F., 2012. Feasibility pump heuristics for column generation approaches. In: Klasing, R. (Ed.), Experimental Algorithms. Vol. 7276 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 332–343.

Pessoa, A., de Aragão, Marcus, M. P., Uchoa, E., 2008. Robust branch-cut-and-price algorithms for vehicle routing problems. In: Golden, B., Raghavan, S., Wasil,

E. (Eds.), The Vehicle Routing Problem: Latest Advances and New Challenges. Vol. 43 of Operations Research/Computer Science Interfaces. Springer US, pp. 297–325.

Pessoa, A., Poss, M., Sadykov, R., Vanderbeck, F., 2018a. Branch-and-cut-and-price for the robust capacitated vehicle routing problem with knapsack uncertainty. Cadernos do LOGIS 2018/1, Universidade Federal Fluminense.

Pessoa, A., Sadykov, R., Uchoa, E., 2018b. Enhanced branch-cut-and-price algorithm for heterogeneous fleet vehicle routing problems. European Journal of Operational Research 270 (2), 530–543.

Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F., 2013. In-out separation and column generation stabilization by dual price smoothing. In: Bonifaci, V., Demetrescu, C., Marchetti-Spaccamela, A. (Eds.), Experimental Algorithms. Vol. 7933 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 354–365.

Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F., 2018c. Automation and combination of linear-programming based stabilization techniques in column generation. INFORMS Journal on Computing 30 (2), 339–360.

Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F., 2019a. A generic exact solver for vehicle routing and related problems. In: Lodi, A., Nagarajan, V. (Eds.), Integer Programming and Combinatorial Optimization. Vol. 11480 of Lecture Notes in Computer Science. Springer International Publishing, Cham, pp. 354–369.

Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F., 2019b. A generic exact solver for vehicle routing and related problems. Cadernos do LOGIS 2019/2, Universidade Federal Fluminense.

Pessoa, A. A., Di Puglia Pugliese, L., Guerriero, F., Poss, M., 2015. Robust constrained shortest path problems under budgeted uncertainty. Networks 66 (2), 98–111.

Petersen, B., Pisinger, D., Spoorendonk, S., 2008. Chvátal-Gomory Rank-1 Cuts Used in a Dantzig-Wolfe Decomposition of the Vehicle Routing Problem with Time Windows. Springer US, Boston, MA, pp. 397–419.

Pferschy, U., Schauer, J., 2009. The knapsack problem with conflict graphs. Journal of Graph Algorithms and Applications 13 (2), 233–249.

Pochet, Y., Wolsey, L. A., 2006. Production Planning Using Mixed Integer Programming. Springer.

Posta, M., Ferland, J. A., Michelon, P., 2012. An exact method with variable fixing for solving the generalized assignment problem. Computational Optimization and Applications 52, 629–644.

Potts, C. N., Van Wassenhove, L. N., 1985. A Branch and Bound Algorithm for the Total Weighted Tardiness Problem. Operations Research 33 (2), 363–377.

Puchinger, J., Raidl, G. R., 2007. Models and algorithms for three-stage two-dimensional bin packing. European Journal of Operational Research 183 (3), 1304 – 1327.

Puchinger, J., Raidl, G. R., Koller, G., 2004. Solving a real-world glass cutting problem. In: Gottlieb, J., Raidl, G. R. (Eds.), Evolutionary Computation in Combinatorial Optimization. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 165–176.

Restrepo, M. I., Gendron, B., Rousseau, L.-M., 2016. Branch-and-price for personalized multiactivity tour scheduling. INFORMS Journal on Computing 28 (2), 334–350.

Restrepo, M. I., Gendron, B., Rousseau, L.-M., 2018. Combining benders decomposition and column generation for multi-activity tour scheduling. Computers and Operations Research 93, 151 – 165.

Righini, G., Salani, M., 2006. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. Discrete Optimization 3 (3), 255 – 273.

Roberti, R., Mingozzi, A., 2014. Dynamic ng-path relaxation for the delivery man problem. Transportation Science 48 (3), 413–424.

Rönnberg, E., Larsson, T., 2019. An integer optimality condition for column generation on zero–one linear programs. Discrete Optimization 31, 79 – 92.

Røpke, S., 2012. Branching decisions in branch-and-cut-and-price algorithms for vehicle routing problems. Presentation in Column Generation 2012.

Rothberg, E., 2007. An evolutionary algorithm for polishing mixed integer programming solutions. INFORMS Journal on Computing 19 (4), 534–541.

Rousseau, L.-M., Gendreau, M., Feillet, D., 2007. Interior point stabilization for column generation. Operations Research Letters 35 (5), 660 – 668.

Ryan, D. M., Foster, B. A., 1981. An integer programming approach to scheduling. In: Wren, A. (Ed.), Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling. North-Holland, Amsterdam, pp. 269 – 280.

Sadykov, R., 2012a. A dominant class of schedules for malleable jobs in the problem to minimize the total weighted completion time. Computers and Operations Research 39 (6), 1265 – 1270.

Sadykov, R., Dec 2012b. Scheduling incoming and outgoing trucks at cross docking terminals to minimize the storage cost. Annals of Operations Research 201 (1), 423–440.

Sadykov, R., Lazarev, A. A., Pessoa, A., Uchoa, E., Vanderbeck, F., 2015a. The prominence of stabilization techniques in column generation: the case of freight transportation. In: Sixth International Workshop on Freight Transportation and Logistics. Ajaccio, France, pp. 87–90.

Sadykov, R., Lazarev, A. A., Shiryaev, V., Stratonnikov, A., 2013. Solving a freight railcar flow problem arising in russia. In: Frigioni, D., Stiller, S. (Eds.), 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems. Vol. 33 of OpenAccess Series in Informatics (OASIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 55–67.

Sadykov, R., Uchoa, E., Pessoa, A., October 2017. A bucket graph based labeling algorithm with application to vehicle routing. Cadernos do LOGIS 2017/7, Universidade Federal Fluminense.

Sadykov, R., Vanderbeck, F., 2011. Column generation for extended formulations. Electronic Notes in Discrete Mathematics 37, 357 – 362.

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

Sadykov, R., Vanderbeck, F., 2013b. Column generation for extended formulations. EURO Journal on Computational Optimization 1 (1-2), 81–115.

Sadykov, R., Vanderbeck, F., Pessoa, A., Tahiri, I., Uchoa, E., 2019. Primal heuristics for branch-and-price: the assets of diving methods. INFORMS Journal on Computing 31 (2), 251–267.

Sadykov, R., Vanderbeck, F., Pessoa, A., Uchoa, E., 2015b. Column generation based heuristic for the generalized assignment problem. In: Simpósio Brasileiro de Pesquisa Operacional. Porto de Galinhas, Brazil, pp. 3624–3631.

Sadykov, R., Wolsey, L. A., 2006. Integer programming and constraint programming in solving a multimachine assignment scheduling problem with deadlines and release dates. INFORMS Journal on Computing 18 (2), 209–217.

Savelsbergh, M. W. P., 1994. Preprocessing and probing techniques for mixed integer programming problems. ORSA Journal on Computing 6 (4), 445–454.

Schneider, M., Drexl, M., Dec 2017. A survey of the standard location-routing problem. Annals of Operations Research 259 (1), 389–414.

Schutt, A., Feydy, T., Stuckey, P. J., Wallace, M. G., Jun 2013. Solving rcpsp/max by lazy clause generation. Journal of Scheduling 16 (3), 273–289.

Song, G., Kowalczyk, D., Leus, R., 2018. The robust machine availability problem – bin packing under uncertainty. IISE Transactions In Press.

Subramanyam, A., Repoussis, P. P., Gounaris, C. E., 2018. Robust optimization of a broad class of heterogeneous vehicle routing problems under demand uncertainty. Tech. Rep. 1810.04348, arXiv.

Sungur, I., nez, F. O., Dessouky, M., 2008. A robust optimization approach for the capacitated vehicle routing problem with demand uncertainty. IIE Transactions 40 (5), 509–523.

Taillard, É. D., Voß, S., 2018. POPMUSIC. Springer International Publishing, Cham, pp. 687–701.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., Subramanian, A., 2017. New benchmark instances for the capacitated vehicle routing problem. European Journal of Operational Research 257 (3), 845 – 858.

Valério de Carvalho, J. M., 1999. Exact solution of bin-packing problems using column generation and branch-and-bound. Annals of Operations Research 86, 629–659.

van den Akker, J., Hurkens, C., Savelsbergh, M., 2000. Time-indexed formulations for machine scheduling problems: Column generation. INFORMS Journal on Computing 12 (2), 111–124.

Vanderbeck, F., 2000. On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. Operations Research 48 (1), 111–128.

Vanderbeck, F., 2001. A nested decomposition approach to a three-stage, two-dimensional cutting-stock problem. Management Science 47 (6), 864–879.

Vanderbeck, F., 2005. Implementing mixed integer column generation. In: Desaulniers, G., Desrosiers, J., Solomon, M. M. (Eds.), Column Generation. Springer US, pp. 331–358.

Vanderbeck, F., 2011. Branching in branch-and-price: a generic scheme. Mathematical Programming 130 (2), 249–294.

Vanderbeck, F., Sadykov, R., Tahiri, I., 2017. BaPCod — a generic Branch-And-Price Code. `https://realopt.bordeaux.inria.fr/?page_id=2`, [Online; accessed 18-July-2019].

Vanderbeck, F., Savelsbergh, M. W. P., 2006. A generic view of Dantzig-Wolfe decomposition in mixed integer programming. Operations Research Letters 34 (3), 296–306.

Vanderbeck, F., Wolsey, L. A., 2010. Reformulation and decomposition of integer programs. In: Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., Wolsey, L. A. (Eds.), 50 Years of Integer Programming 1958-2008. Springer Berlin Heidelberg, pp. 431–502.

Vidal, T., Crainic, T. G., Gendreau, M., Prins, C., 2014. A unified solution framework for multi-attribute vehicle routing problems. European Journal of Operational Research 234 (3), 658 – 673.

Wang, K., Zhen, L., Wang, S., Laporte, G., 2018. Column generation for the integrated berth allocation, quay crane assignment, and yard assignment problem. Transportation Science 52 (4), 812–834.

Wei, L., Luo, Z., Baldacci, R., Lim, A., 2019. A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. INFORMS Journal on Computing accepted.

Wentges, P., 1997. Weighted dantzig–wolfe decomposition for linear mixed-integer programming. International Transactions in Operational Research 4 (2), 151–162.

Yagiura, M., Ibaraki, T., Glover, F., 2006. A path relinking approach with ejection chains for the generalized assignment problem. European Journal of Operational Research 169 (2), 548 – 569.

Zhang, Y., Baldacci, R., Sim, M., Tang, J., May 2019. Routing optimization with time windows under uncertainty. Mathematical Programming 175 (1), 263–305.