

# Branching in Branch-and-Price: a Generic Scheme

François Vanderbeck (fv@math.u-bordeaux1.fr)

Applied Mathematics, University Bordeaux 1, F-33405 Talence Cedex, France

Working paper U-05-14, November 2005

## Abstract

In Branch-and-Price algorithms, implementing a branching scheme that is both efficient and well suited for the column generation procedure can be challenging. A standard branching scheme can be implemented in the space of the compact formulation to which the Dantzig-Wolfe reformulation was applied; but this may be inefficient. Specialized branching rules have been proposed for use in Branch-and-Price; but they may imply modifications to the pricing problem. This is a main concern because the efficient pricing problem solver on which the method relies might become obsolete beyond the root node. This paper presents a branching scheme for use as a default in a generic Branch-and-Price code that only relies on the pricing oracle of the root node for column generation. It proceeds by recursively partitioning the set of columns. Branching constraints are enforced in the pricing problem instead of being dualized in a Lagrangian way. The pricing problem is solved by a limited number of calls to the pricing oracle. Bin packing and cutting stock problems are solved to integrality with this approach (the first that does not require expanding the variable space). The paper brings a unifying framework on the issue of branching in Branch-and-Price.

**Keywords:** Integer Programming, Dantzig-Wolfe reformulation, Branch-and-Price.

## Introduction

Many mixed integer programming problems have decomposable structure that makes them well suited for Dantzig-Wolfe reformulation. Then, a Branch-and-Price algorithm can be an efficient solution approach. However, branching can be challenging: the branching scheme must combine good properties in terms of leading to integrality of the solution,

yielding dual bound improvements, and being “compatible” with the column generation procedure. In particular, branching constraints may result in modifications to the structure of the pricing problem and impair its tractability. This is a main concern since the decomposition approach typically relies on the availability of an efficient pricing problem solver.

The issue of branching in a branch-and-price context has not been fully sorted out to date, because, for many practical applications, branching is straightforward: it can be implemented in a standard way in the space of the compact formulation to which the Dantzig-Wolfe reformulation was applied. Villeneuve et al. [19] suggest that one can always proceed by using standard branching in an “original” formulation and re-apply Dantzig-Wolfe reformulation to the problem augmented with branching constraints. However, when the decomposition involves multiple identical subproblems, this can induce symmetry. Specialized branching rules for use in Branch-and-Price have been developed. Ryan and Foster [10] proposed a scheme for application that can be reformulated as a set partitioning problem, a generalization of which was developed by Vanderbeck [15] (both schemes result in structure modifications to the pricing problem).

The scheme proposed in this paper unifies the previously proposed approaches. It can be seen as a specific implementation of the scheme of [15] where fractional solutions are cut off by bounding the number of columns selected from specific subsets. However, it can be implemented using the original pricing problem oracle while avoiding the symmetry drawback in the case of multiple identical subproblems. Moreover, branching constraints are not dualized in a Lagrangian way and therefore they induce better improvements in dual bounds. We show the one-to-one correspondence between enforcing our branching constraints in the Dantzig-Wolfe reformulation and implicitly fixing some bounds on the variables in the original formulation.

A motivation for this work is the development of a generic code for branch-and-price. Previous attempts have bounced on the issue of branching: tool boxes such as Minto [11], Abacus [12] or BCP [8] leave it to the user to implement an application specific branching scheme; while other codes are developed for a specific class of applications (such as the vehicle routing problem and its variants), examples are Gencol [6] and Maestro [4]. The branching scheme of this paper permits to get passed this dead lock. It applies across applications and it requires no input from the user (the same pricing problem solver can be

used after branching). It leads to a possible black box implementation of branch-and-price (a prototype named *BaPCod* is introduced in [17]). The necessary features are (i) the automation of the Dantzig-Wolfe reformulation process (the user defines of a mixed integer programming problem in terms of variables and constraints, identifies subproblems, and provides the associated solvers, but he does not need to explicitly define master, columns, their reduced cost, or the Lagrangian bounds); (ii) a column generation procedure with default initialization, stabilization and a selection of solver for the master [3]; and (iii) a default branching scheme.

The paper is organized as follows. Section 1 reviews the Dantzig-Wolfe reformulation principle. In Section 2, we characterize problems for which branching can be implemented in a straightforward way in the Dantzig-Wolfe reformulation. We also review existing branching schemes and introduce the scheme of this paper. Then, we restrict our attention to a binary model to simplify the detailed presentation. (The binary model also corresponds to a practical strategy of branching on binary components of the 0-1 form of an integer program.) However, hints on extensions to the integer case are provided. In Section 3, we show how to obtain a solution to the original problem from that of the Dantzig-Wolfe reformulation. This transformation shall be useful to translate the impact of our branching constraints in the original formulation.

Section 4 shows how to branch at the root node and Section 5 how to implement column generation after branching. In Section 6, we derive the expression of the dual bound after branching. In Sections 7 and 8, we explain how branching can be carried out beyond the root node and the simplification that arise from preprocessing. In Section 9, we compare our scheme to that of Ryan-and-Foster in the special case of a set partitioning problem. Section 10 presents preliminary computational experiments on bin packing and cutting stock problems (problems are solved to integer optimality through branching alone and using only a standard knapsack solver for pricing). The conclusion summarizes the scheme, its contribution and extensions.

# 1 Dantzig-Wolfe reformulation

Consider a well structured mixed integer program (MIP) whose constraint matrix is of the form

$$\begin{pmatrix} A^1 & A^2 & \dots & A^T \\ B^1 & 0 & \dots & 0 \\ 0 & B^2 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & B^T \end{pmatrix},$$

where  $A^t$  and  $B^t$  are rational matrices. I.e., there are  $T$  diagonal blocks indexed by  $t = 1, \dots, T$  and the problem can be formulated as

$$Z^P = \min \sum_t c^t x^t \tag{1}$$

$$[\text{P}] \quad \sum_t A^t x^t \geq a \tag{2}$$

$$B^t x^t \geq b^t \quad \forall t = 1, \dots, T \tag{3}$$

$$l^t \leq x^t \leq u^t \quad \forall t = 1, \dots, T$$

$$x_I^t \in \mathbb{N}^n \quad \forall t = 1, \dots, T \tag{4}$$

$$x_C^t \in \mathbb{R}^r \quad \forall t = 1, \dots, T$$

where  $c^t$ ,  $a$ , and  $b^t$  are rational vectors of appropriate dimension,  $x_I^t$  (resp.  $x_C^t$ ) denotes the integer (resp. continuous) components of variable vector  $x^t$ , and  $l^t$  (resp.  $u^t$ ) are lower (resp. upper) bound vectors. Thus, a subsystem

$$X^t \equiv \{x \in \mathbb{N}^n : B^t x \geq b^t, l^t \leq x \leq u^t\} \tag{5}$$

can be associated with each block  $t$ .

Throughout the paper, we shall distinguish two cases :

**case 1 (different subsystems):**  $A^t$  and subsystems  $X^t$  input data depend on  $t$ . Examples of applications are capacitated multi-item lot-sizing problems (where a different subsystem can be associated with each item), capacitated facility location problems or generalized assignment problems (where a subsystem is associated with each facility).

**case 2 (identical subsystems):**  $A^t$  and subsystems  $X^t$  input data do not depend on  $t$ . Examples are vehicle routing problems (VRP), where a subsystem is associated with

each vehicle and there are  $T$  identical vehicles; the cutting stock problem (CSP), where a subsystem is associated with each identical stock piece; its special case, the bin packing problem, where a subsystem is associated with each identical bin; or the graph coloring problem where a subsystem (defining a stable set) is associated with each color.

Of course, the combined case could also occur for which matrix  $B$  is made of non-identical blocks, each of which decomposes into identical blocks: an example would be a VRP with different vehicle types and several vehicles of each type.

The Dantzig-Wolfe reformulation of a MIP of the form [P] is presented in [18]. Here, to simplify the presentation, we shall assume that  $X^k$  is a pure integer program. Moreover, we consider it to be a bounded polyhedron (the extension to the unbounded case is presented in [15]). Let  $Q^t$  be an enumerated set of generators (a terminology introduced in [18]) for subsystem  $X^t$ . Since we assumed a bounded and pure integer program,  $Q^t$  is simply the enumerated set of all discrete integer solutions of  $X^t$ , i.e.  $X^t = \{x^q\}_{q \in Q^t}$ , and  $X^t$  can be reformulated as

$$X^t \equiv \left\{ x = \sum_{q \in Q^t} x^q \lambda_q : \sum_{q \in Q^t} \lambda_q = 1, \lambda_q \in \{0, 1\} \forall q \in Q^t \right\}$$

Then, the Dantzig-Wolfe reformulation principle refers to the application of this variable change to [P], which gives rise to reformulation

$$Z^{DM} = \min \sum_t \sum_{q \in Q^t} c^t x^q \lambda_q^t \tag{6}$$

$$[\text{DM}] \quad \sum_t \sum_{q \in Q^t} A^t x^q \lambda_q^t \geq a \tag{7}$$

$$\begin{aligned} \sum_{q \in Q} \lambda_q^t &= 1 & \forall t = 1, \dots, T \\ \lambda_q^t &\in \{0, 1\} & \forall t = 1, \dots, T, q \in Q^t \end{aligned}$$

In case 2, all subsystems are identical, i.e.  $A^t = A$ ,  $B^t = B$ ,  $c^t = c$ ,  $X^t = X$  and  $Q^t = Q$  for  $t = 1, \dots, T$  and one can aggregate  $\lambda_q^t$  variables using

$$\lambda_q = \sum_{t=1}^T \lambda_q^t. \tag{8}$$

Then, Dantzig-Wolfe reformulation takes the form

$$Z^M = \min \sum_{q \in Q} c x^q \lambda_q \quad (9)$$

$$[\text{M}] \quad \sum_{q \in Q} A x^q \lambda_q \geq a \quad (10)$$

$$\sum_{q \in Q} \lambda_q = T \quad (11)$$

$$\lambda_q \in \mathbb{N} \quad \forall q \in Q \quad (12)$$

Observe that the Dantzig-Wolfe reformulation does not exhibit the symmetry in  $t$  that is present in the original formulation [P] (where a permutation of the  $t$  indexing in vector  $x$  gives rise to an alternative representation of the same solution).

The enumeration of set  $Q$  (or  $Q^t$  in case 1) is only theoretical. In practice, the solution of the Dantzig-Wolfe reformulation is handled through dynamic generation of its variables and associated columns in the course of the optimization, a procedure known as *column generation*. In this context the reformulation is called the *master* program: we refer to [DM] as the disaggregated master used in case 1, while [M] is the commonly used form of the master in case 2.

Let us briefly summarize the procedure for solving the LP relaxation of [M] by column generation (that for [DM] is analogous). The master LP is initialized with a subset of (possibly artificial) columns. This restricted program is solved to LP optimality. Let  $\pi$  is the dual solution vector associated to constraints (10). Then, a *pricing problem* is solved:

$$\zeta(\pi) = \min\{(c - \pi A)x : \underbrace{B x \geq b, l \leq x \leq u, x \in \mathbb{N}^n}_{x \in X}\} \quad (13)$$

When its solution,  $x^q := x^*$ , defines a negative reduced cost column, the column  $A x^q$  and associated  $\lambda_q$  variable are added to the master and the procedure reiterates. Otherwise, the current LP solution is proved optimal. We assume that (13) is a problem that can be solved rather efficiently, compared to [P], and that a solver is available for it. We refer to the solver as the *oracle*.

At each iteration of the column generation procedure a dual bound can be computed from the pricing problem solutions: dualizing (10) in [M] gives rise to Lagrangian dual bound

$$\theta(\pi) = \pi a + T \zeta(\pi). \quad (14)$$

These bounds converge (although not monotonically) towards the value of the master LP which is known to be equivalent to the Lagrangian dual that results from dualizing constraints (2) in [P] (see [7]), i.e.

$$Z_{LP}^M = \max_{\pi \geq 0} \theta(\pi) = \min \left\{ \sum_t c x^t : \sum_t A x^t \geq a, x^t \in \text{conv}(X^t) \forall t \right\}. \quad (15)$$

Thus, the comparison of the LP and IP values of the above formulation is  $Z_{LP}^P \leq Z_{LP}^M \leq Z^M = Z^P$ . When  $\text{conv}(X^t) \neq X_{LP}^t$ , as in the above mentioned applications, the master gives rise to a LP bound better than that of [P].

In summary, Dantzig-Wolfe reformulation allows a solution approach that exploits the availability of an efficient oracle for a subproblem; it avoids a symmetry in  $t$  in case 2; and it often leads to better quality dual bounds. However developing a solution approach based on the Dantzig-Wolfe reformulation requires embedding a column generation procedure into a branch-and-bound algorithm. The combined algorithm is known as *branch-and-price* [1].

## 2 Enforcing integrality

To apply a branch-and-bound approach based on the Dantzig-Wolfe reformulation, one needs a branching scheme that enforces integrality in [DM] (resp. [M]). Branching directly on individual variables  $\lambda_q^t$  (resp.  $\lambda_q$ ), whose current value is  $v \notin \{0, 1\}$  (resp.  $v \notin \mathbb{N}$ ), using disjunctive constraints  $\lambda_q^t \leq \lfloor v \rfloor$  or  $\lambda_q^t \geq \lceil v \rceil$  (resp.  $\lambda_q \leq \lfloor v \rfloor$  or  $\lambda_q \geq \lceil v \rceil$ ) is not efficient. It leads to a unbalanced branch-and-bound tree: branching constraint  $\lambda_q^t \leq \lfloor v \rfloor$  (resp.  $\lambda_q \leq \lfloor v \rfloor$ ) is typically much less restrictive than  $\lambda_q^t \geq \lceil v \rceil$  (resp.  $\lambda_q \geq \lceil v \rceil$ ). Moreover, it combines badly with the column generation procedure: the pricing problem must be amended to avoid regenerating the specific column  $q \in Q^t$  (resp.  $q \in Q$ ) on which branching occurred. The alternative is to branch on constraints.

A natural combination of variables  $\lambda$  on which to branch is that expressing the value of the variables of the original formulation [P]: mapping  $\lambda$  solutions into  $x$  solutions and implementing a branching scheme based on disjunctive constraints for the original variables. In case 1, this scheme will suffice to enforce integrality. The mapping is given by

$$x^t = \sum_{q \in Q^t} x^q \lambda_q^t \quad (16)$$

and disjunctive branching constraints take the form

$$\sum_{q \in Q^t} x_i^q \lambda_q^t \leq \lfloor v \rfloor \quad \text{or} \quad \sum_{q \in Q^t} x_i^q \lambda_q^t \geq \lceil v \rceil \quad (17)$$

in [DM]. In fact, such disjunction can be enforced directly in the pricing problem instead of the master: one simply needs to set component bound

$$x_i^t \leq \lfloor v \rfloor \quad \text{or} \quad x_i^t \geq \lceil v \rceil \quad (18)$$

in the subproblem which does not modify its structure.

However, in case 2, when one is working with reformulation [M], the disaggregated original variable values  $x^t$  are not available through a uniquely defined mapping. Instead, what can be enforced in [M] is the integrality of aggregate variables

$$x = \sum_{t=1}^T x^t . \quad (19)$$

The mapping defining  $x$  from  $\lambda$  is

$$x = \sum_{q \in Q} x^q \lambda_q . \quad (20)$$

and disjunctive branching constraints take the form

$$\sum_{q \in Q} x_i^q \lambda_q \leq \lfloor v \rfloor \quad \text{or} \quad \sum_{q \in Q} x_i^q \lambda_q \geq \lceil v \rceil \quad (21)$$

in [M]. Observe that branching constraints on aggregate variables cannot be enforced directly in the pricing problem (except in special case such as  $\sum_{q \in Q} x_i^q \lambda_q \leq 0$ ). However, master constraints of the form (21) do not modify the structure of the pricing problem: in the objective of the pricing problem, variable  $x_i$  simply sees its cost coefficient augmented by the dual variables associated with the branching constraints concerning component  $i$ .

There are applications of the case 2 type in which it is sufficient to enforce integrality of the aggregate variables  $x$  to obtain (at least implicitly) an integer solution to [P] and [M]. Consider the example of the VRP in a symmetric graph  $G = (V, E)$ . A compact formulation is

$$\min \left\{ \sum_e c_e x_e : \sum_{e \in \delta(0)} x_e = 2T, \sum_{e \in \delta(i)} x_e = 2 \forall i \neq 0, \sum_{e \in \delta(S)} x_e \geq 2r(S) \forall S \subseteq V^0, x \in \{0, 1\}^{|E|} \right\} \quad (22)$$

where  $T$  is the number of vehicles, node 0 is the depot,  $\delta(S)$  is the set of edges with one endpoint in  $S$ ,  $V^0 = V \setminus \{0\}$  and  $r(S)$  is the minimum number of vehicles needed to satisfy demands of customers in  $S$ . Let  $Q$  be the set of feasible routes and  $(x_e^q)_{e \in E}$  the edge vector defining those routes. Then, the master can be formulated as

$$\min \left\{ \sum_{q \in Q} \left( \sum_e c_e x_e^q \right) \lambda_q : \sum_{q \in Q} \left( \sum_{e \in \delta(i)} x_e^q \right) \lambda_q = 2 \forall i, \sum_{q \in Q} \lambda_q \leq V, \lambda_q \in \{0, 1\} \forall q \in Q \right\}. \quad (23)$$

Enforcing the integrality of aggregate edge use,  $x_e = \sum_{q \in Q} x_e^q \lambda_q$ , suffices to obtain an integer solution for [P] and [M]:

**Proposition 1** *A solution to the LP relaxation of (23) for which  $\sum_{q \in Q} x_e^q \lambda_q \in \{0, 1\} \forall e$ , must have  $\lambda_q \in \{0, 1\} \forall q \in Q$ .*

**Proof:** Assume there exists  $r$  with  $0 < \lambda_r < 1$ , while  $x_e = \sum_{q \in Q} x_e^q \lambda_q \in \{0, 1\} \forall e$ . Let  $\tilde{e}$  be the first edge in route  $r$  and  $\tilde{Q} = \{q \in Q : \lambda_q > 0 \text{ and } x_{\tilde{e}}^q = 1\}$ . Let  $i$  be the first node beyond which route  $r$  differs from some other routes in  $\tilde{Q}$ . Let  $e_1$  and  $e_2$  be respectively the edge before and after  $i$  in route  $r$ . Let  $Q^1 = \{q \in Q : \lambda_q > 0 \text{ and } x_{e_1}^q = 1\} \ni r$  and  $Q^2 = \{q \in Q : \lambda_q > 0 \text{ and } x_{e_2}^q = 1\} \ni r$ . Our hypothesis implies that  $x_{e_1} = \sum_{q \in Q^1} \lambda_q = 1$  and  $x_{e_2} = \sum_{q \in Q^2} \lambda_q = 1$ . By construction  $(Q^1 \setminus Q^2) \neq \emptyset$  as it contains routes of  $\tilde{Q}$  that differ from  $r$  beyond node  $i$ . Therefore, the degree of node  $i$  is greater than 2, a contradiction. ■

In practice, branching on individual edge variables  $x_e$  yields an unbalanced tree (the branch  $x_e = 0$  being less restrictive than  $x_e = 1$ ). It is more efficient to branch on sets of edges: for instance, [9] uses disjunctive branching constraints

$$\sum_{e \in \delta(S)} x_e \leq \lfloor v \rfloor \quad \text{or} \quad \sum_{e \in \delta(S)} x_e \geq \lceil v \rceil \quad (24)$$

for small subsets of nodes  $S$  (say those requiring 1 or 2 vehicles). These branching constraints (24) set in the master yield no modifications to the structure of the pricing problem.

In other applications of the case 2 type, branching constraints (21) are not sufficient to eliminate all fractional solutions. Take, for instance, the cutting stock problem. The standard column generation reformulation is

$$\min \left\{ \sum_{q \in Q} \lambda_q : \sum_{q \in Q} x_i^q \lambda_q = d_i \forall i, \lambda_q \in \mathbb{N} \forall q \in Q \right\} \quad (25)$$

where  $Q$  is the set of feasible cutting patterns, i.e. solutions to a knapsack problem

$$X = \{x \in \mathbb{N}^n : \sum_i w_i x_i \leq W, x_i \leq d_i \forall i\} \quad (26)$$

where  $w_i$  and  $d_i$  denote respectively the width and the demand for item  $i$ , while  $W$  is the stock piece width. Observe that enforcing integer value for aggregate variables  $x_i = \sum_q x_i^q \lambda_q$  does not allow to cut off a fractional master solution as  $x_i = d_i \in \mathbb{N}$  in any LP solution to the master.

However, alternative CSP formulations are known that allow branching to integrality using only constraints of the form (21). They require working in an expanded variable space. Valerio [13] observed that solving (26) by dynamic programming yields cutting patterns that can be represented by a path in a graph where nodes are associated with knapsack capacity consumption and arcs are associated with items whose width is equal to the difference between head and tail nodes. If  $(x_{uv}^q)$  is an arc indicator vector defining such path, the CSP can be formulated as

$$\min\left\{\sum_{q \in Q} \lambda_q : \sum_{q \in Q} \left(\sum_{uv: v-u=w_i} x_{uv}^q\right) \lambda_q = d_i \forall i, \lambda_q \in \mathbb{N} \forall q \in Q\right\}. \quad (27)$$

A master solution for which aggregate arc use,  $x_{uv} = \sum_{q \in Q} x_{uv}^q \lambda_q$ , are integer defines an integer solution for [P]. Indeed, the flow decomposition theorem says that integer path flows can be recovered from integer arc flows. This approach requires using a dynamic program for pricing and it introduces a symmetry since a given solution to [P] typically admits different representations into path flows. To avoid such symmetry, Belov et al. [2] formulates the CSP as a vehicle routing problem. A cutting pattern is a tour selecting items while obeying precedence constraints (that break symmetry) and the knapsack capacity. Then, branching on arcs suffices to cut-off all fractional solutions but the approach requires using a resource constrained shortest path pricing problem. Both of these alternative formulations involve a pseudo-polynomial number of variables for the original formulation (each of which might need branching on).

Observe from the above discussion that branching constraints can either go in the master program (resulting in modification to the pricing problem objective function) or in the pricing problem with modification to the associated polyhedron. The structure of the pricing problem is not modified if the reduced cost modification amounts to simply amending the cost coefficient of some existing variables or the polyhedron modification

boils down to amending bounds on variables. Whether branching constraint go in the master or in the subproblem is important for the strength of the dual bound (15) after branching. Constraints that go in the master are dualized (hence we call them *soft* branching constraints). While if they can be implemented in the pricing problem whose polyhedron is convexified, they yield a better improvement in the dual bound (hence we call them *hard* branching constraints).

Let us summarize what can be achieved through branching in [DM] (resp. [M]) on the variables of the original formulation [P] (whose value is known implicitly).

**Observation 1 (Branching on variables of the original formulation)**

- *In case 1, an integer solution to [P] can be obtained using formulation [DM] and enforcing branching through restrictions of the form (18) in the pricing problem. Such branching constraints are hard but do not amend the structure of the pricing problem.*
- *In case 2, one can only achieve the integrality of aggregate solution (19) to [P] using formulation [M] and branching constraint of the form (21) that must in general be implemented as soft branching constraint in the master but do not yield modifications to the structure of the pricing problem.*

Observe that our examples of case 2 applications for which enforcing integrality of the aggregate solution suffices admit a formulation [P] that does not make use the  $t$  indexing (for the VRP, [P] takes the form (22); for the CSP, see [2] and [13]).

Villeneuve et al. [19] say that branching can always be implemented by returning to the variable space of the original formulation and without modifying the structure of the pricing problem. In fact, what their paper proposes is equivalent to using formulation [DM] in all cases, even in case 2, when subsystems are identical. The obvious drawback is that working with [DM] when subsystem are identical induces a symmetry in  $t$ : when branching enforces a constraint on a given  $x^t$  component, the LP solution can often be reorganized by permuting the  $t$  indices to get around it. In fact, even when subsystems are different but closely related (take for instance the CSP with non-identical stock pieces), such symmetry arises and the above branching scheme for case 1 can be inefficient (see [15]).

Thus, in case 2, when what is truly needed is the integrality of disaggregated solution  $(x^t)_{t=1,\dots,T}$ , other form of branching-on-constraints strategies must be developed. When the master takes the form of a set partitioning problem, i.e.,

$$\min\left\{\sum_{q \in Q} c x^q \lambda_q : \sum_{q \in Q} x^q \lambda_q = 1, \sum_{q \in Q} \lambda_q = T, \lambda_q \in \{0, 1\} \forall q \in Q\right\} \quad (28)$$

with  $x^q \in \{0, 1\}^n$ , integrality can be enforced using Ryan-and-Foster's scheme [10]. For any fractional solution,  $\tilde{\lambda} \notin \{0, 1\}^{|Q|}$ , there exists a pair of items  $(i, j)$  such that  $\tilde{\lambda}$  can be separated using disjunction

$$\sum_{q: x_i^q = x_j^q = 1} \lambda_q \leq 0 \quad \text{or} \quad \sum_{q: x_i^q = x_j^q = 1} \lambda_q \geq 1. \quad (29)$$

These constraints can be implemented directly in the pricing problem (as *hard* branching constraints): In the first branch, as no column that have both  $x_i^q = 1$  and  $x_j^q = 1$  can be used, the pricing problem is augmented with constraint  $x_i + x_j \leq 1$ . In the second branch, as items  $i$  and  $j$  are entirely covered by columns with  $x_i^q = x_j^q = 1$ , the pricing problem is augmented with constraint  $x_i = x_j$ . Columns that are not solution to the modified pricing problem can be deleted. The modification to the pricing problem may change its structure or not. In the example of the graph coloring problem, the stable set problem structure is conserved when adding an edge (to enforce  $x_i + x_j \leq 1$ ) or contracting two nodes (to enforce  $x_i = x_j$ )<sup>1</sup>.

Vanderbeck [15] proposes a scheme that generalizes that of Ryan-and-Foster to master programs not restricted to set partitioning problems. It consists in considering progressively more specific subsets  $\hat{Q} \subset Q$  and enforcing  $\sum_{q \in \hat{Q}} \lambda_q \in \mathbb{N}$  through disjunctive branching constraint of the form:

$$\sum_{q \in \hat{Q}} \lambda_q \leq \lfloor v \rfloor \quad \text{or} \quad \sum_{q \in \hat{Q}} \lambda_q \geq \lceil v \rceil. \quad (30)$$

In practice, sets  $\hat{Q}$  are defined as subsets of columns whose vector  $x^q$  satisfy prescribed component bounds. Then, the modification to the pricing problem consists in introducing an indicator variable,  $\delta_{\hat{Q}}$ , that takes value 1 if the subproblem solution satisfies all component bounds defining  $\hat{Q}$  and zero otherwise. Its cost is the dual variable associated

---

<sup>1</sup>This branching scheme is presented in [5] as the method of choice to build a tree enumerating all the solutions to the coloring problem: leaf nodes define cliques where each node requires its own color.

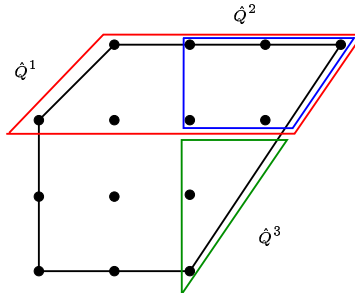


Figure 1: A nested partition of the pricing problem polyhedron

with branching constraint (30). When set  $\hat{Q}$  is defined by a single component bound on a binary variable of  $X$ , no modification is induced in the pricing problem. However, when  $\delta_{\hat{Q}}$  is not a linear expression of the pricing problem variables, this scheme results in modifications to its structure. Branching sets  $\hat{Q}$  that induce few modifications to the pricing problem are used in priority as they are also the ones that lead to the more balanced branch-and-bound tree. Thus, this scheme generally relies on *soft* branching and may result in significant subproblem modifications.

In this paper, we develop a branching scheme for [M] (i.e., for case 2) that can be implemented using the oracle of the original pricing problem. It uses *hard* branching constraints and does not induce a symmetry in  $t$ . It can be seen as a specific implementation of the scheme of [15] whereby constraint (11) is treated as a special order set: integrality is progressively achieved by partitioning set  $Q$  (see Figure 1 for illustration) and enforcing separate cardinality constraint of the form (11) on each subset of columns. Of course the scheme also suits case 1 or a hybrid of case 1 and 2.

In the sequel, to simplify the presentation, we assume a convexity constraint of the form (11). Extension to the case of convexity constraints of the form  $L \leq \sum_{q \in Q} \lambda_q \leq U$  is straightforward. We also make use of implicit bounds on aggregate variables:

$$l_i \leq x_i \leq u_i \quad i = 1, \dots, n \quad (31)$$

where  $x_i = \sum_{t=1}^T x_i^t$  and  $0 \leq l_i \leq u_i \leq T$ . Furthermore, we consider that the subsystem involves binary variables only, i.e.

$$X = \{x \in \{0, 1\}^n : Bx \geq b\} \quad (32)$$

Assuming a binary subproblem is restrictive in the sense that the integer pricing problem must be bounded (to be transformed in a 0-1 problem) and that we assume an “efficient” solver for the binary form of the pricing problem. Extensions of the scheme beyond these assumptions are suggested in footnotes.

### 3 A mapping that preserve integrality

Each solution  $\lambda$  to [M] (resp.  $[M_{LP}]$ ) can be transformed into a solution to [P] (resp.  $[P_{LP}]$ ). This requires a disaggregation of the solution  $\lambda$  into  $T$  solution vectors  $x^t$ . This disaggregation is not unique. The transformation of Table 1 makes use of a lexicographic ordering of the columns  $q \in Q$  that yields a specific disaggregation. Any permutation in the indexing of the subproblem variables gives rise to an alternative disaggregation.

Table 1: Transforming a solution to [M] (resp.  $[M_{LP}]$ ) into a solution for [P] (resp.  $[P_{LP}]$ )

1. Let  $\Lambda = \{q : \lambda_q > 0\}$ . When  $\lambda$  is obtained as an extreme LP solution to the master program augmented with branching constraints,  $|\Lambda|$  is bounded by the number of constraints in the master (including branching constraints).
2. Sort the columns  $q$  of  $\Lambda$  in lexicographic order of their associated vector  $x^q$ .
3. Initialize  $x_i^t = 0$  for all  $i, t$ ; let  $z = 0$  and  $t = 1$  ( $z$  stands for  $\sum \lambda_q$  for the columns  $q$  treated so far, while  $t$  is the index of the vector  $x^t$  being currently filled).
4. For each  $q \in \Lambda$  in lexicographic order, do  
while  $(\lambda_q > 0)$  { let  $v = \min\{\lambda_q, t - z\}$ ; for all  $i = 1, \dots, n$ , do  $x_i^{t+} = x_i^q v$ ;  $\lambda_{q-} = v$ ;  $z += v$ ; if  $(z = t)$ ,  $t += 1$  }.

**Example 1** *We shall use the following numerical example to illustrate the developments to come. The master constraint are defined by*

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 2 \end{pmatrix} \quad \text{and} \quad a = \begin{pmatrix} 5 \\ 5 \\ 10 \end{pmatrix}$$

Upper bounds on aggregate variables  $x_i = \sum^t x_i^t$  are  $u = (5, 5, 5, 5)$  and  $T = 5$ . The table below gives the assumed set  $Q$  of feasible columns along with their respective weight  $\tilde{\lambda}_q$  in what is assumed to be the master LP solution:

$\tilde{\lambda}_q$	0	$\frac{1}{2}$	1	$\frac{1}{2}$	0	0	1	1	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	5
$x_1$	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	3
$x_2$	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	3
$x_3$	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	2
$x_4$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	2

Using the mapping of Table 1, we get a solution to the original formulation

$$x^{t=1} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ \frac{1}{2} \end{pmatrix}, x^{t=2} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}, x^{t=3} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, x^{t=4} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}, x^{t=5} = \begin{pmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{pmatrix}.$$

■

The inverse mapping, from [P] to [M] exists for integer solutions but typically not for fractional LP solutions since the two LP polyhedron may not be equivalent (typically some LP solutions to  $[P_{LP}]$  are not feasible for  $[M_{LP}]$ ). Given an integer solution  $x$  to [P], one can define an integer solution  $\lambda$  to [M] as follows. Initialize  $\lambda = 0$ . For each  $t = 1, \dots, T$ , identify  $q \in Q$  such as  $x^q = x^t \in \{0, 1\}^n$  and set  $\lambda_q = \lambda_q + 1$ .

The mapping of Table 1 preserves integrality:

**Proposition 2** *If  $x$  is a solution generated from a solution  $\lambda$  to  $[M_{LP}]$  using the procedure of Table 1, then*

$$x_i^t \in \{0, 1\} \quad \forall i, t \iff \lambda_q \in \mathbb{N} \quad \forall q$$

**Proof:** If  $\lambda_q \in \mathbb{N} \quad \forall q$ , obviously  $x_i^t \in \{0, 1\} \forall i, t$ . Now let us prove the reverse implication by contradiction: assuming  $F = \{q : \lambda_q - \lfloor \lambda_q \rfloor > 0\} \neq \emptyset$ , we show that  $x_i^t \notin \{0, 1\}$  for some pair  $i, t$ . Let  $q_1$  be the first fractional column of  $F$  in lexicographic order. Let  $t$  be the last index of a vector  $x^t$  to the definition of which  $q_1$  contributes:  $0 < v(q_1) = \lambda_{q_1} - \lfloor \lambda_{q_1} \rfloor < 1$  in the last iterate of Step 4 of Table 1 for column  $q_1$ . Let  $q_2$  be the next column in lexicographic order with positive weight, i.e.  $\lambda_{q_2} > 0$  but  $q_2$  might not belong to  $F$ . As  $v(q_1) < 1$ , column  $q_2$  also contributes to the definition of  $x^t$ :  $v(q_2) > 0$  and  $v(q_1) + v(q_2) \leq 1$ . Columns  $q_1$  and  $q_2$  being different, let  $i$  be the smallest index of a

binary component in which they differ. Due to the lexicographic ordering,  $x_i^{q_1} = 1$  while  $x_i^{q_2} = 0$ . Then,  $0 < v(q_1) < x_i^t \leq 1 - v(q_2) < 1$ .  $\blacksquare$

Proposition 2 shows that it is equivalent to check and enforce the integrality of the solution  $\lambda$  in  $[M_{LP}]$  or that of the associated solution  $x$  in  $[P_{LP}]$ .<sup>2</sup>

## 4 Separation of a fractional solution at the root node

To enforce integrality of master solutions, we gradually specify cardinality restrictions on subclasses of columns,  $\hat{Q} \subseteq Q$ , that together define a nested partition of  $Q$ . Subclasses  $\hat{Q}$  are defined by fixing the value of some binary variables  $x_i$ . For instance,  $S = \langle x_1, \bar{x}_2, x_3 \rangle$  denotes a sequence of component bounds in which  $x_1$  is fixed to one,  $x_2$  to zero and  $x_3$  to one. The associated subclass of columns is  $Q(S) = \{q \in Q : x_1^q + \bar{x}_2^q + x_3^q = 3\}$  where  $\bar{x}_2^q = 1 - x_2^q$ . In general terms, a sequence  $S$  of component bounds is denoted by

$$S = \langle \tilde{x}_{[1]}, \tilde{x}_{[2]}, \dots, \tilde{x}_{[|S|]} \rangle$$

where  $[p]$  denotes the index of the component bound in the  $p^{\text{th}}$  position in the sequence,  $\tilde{x}_{[p]}$  is a notation to encompass both cases of a fixing to one ( $\tilde{x}_{[p]} = x_{[p]}$ ) or to zero ( $\tilde{x}_{[p]} = \bar{x}_{[p]}$ ), and

$$Q(S) = \{q \in Q : \sum_{i \in S} \tilde{x}_i^q = |S|\}$$

where the notation  $i \in S$  is a short cut for saying that sequence  $S$  includes a bound on component  $i$ , also denoted as  $\tilde{x}_i \in S$ .<sup>3</sup>

At the root node, the separation of a fractional solution to  $[M_{LP}]$  proceeds as follows. Let  $\tilde{\lambda}$  be the fractional master solution. Then, identify a sequence  $S$  of component bounds such as  $\tilde{\lambda}(S) \notin \mathbb{N}$ , where

$$\lambda(S) = \sum_{q \in Q(S)} \lambda_q.$$

---

<sup>2</sup>Note that the procedure of Table 1 easily extends to a general mixed integer program (the lexicographic ordering remains well defined in the presence of non-binary integer variables or continuous variables). However, the equivalence property of Proposition 2 does not carry on for the non-binary case: then, a solution  $x$  derived from  $\lambda$  can be integer although  $\lambda$  is not; and insisting on having integer values of  $\lambda$  is a sufficient but not necessary condition to achieve integrality for [P]. Thus, the mapping of Table 1 becomes a useful tool to check integrality in the non-binary case.

<sup>3</sup>To generalize the scheme to the case of a non-binary integer program, one simply needs to consider column subsets defined by integer bounds on the integer variables, e.g.  $x_i \leq v$  or  $x_i \geq v + 1$ .

There must exist such a sequence  $S$ : the proof of Proposition 1, in particular, indicates that the first  $i$  components of the first fractional column  $q_1$  can provide one. However, the implementation of the branching scheme is sensitive to the the cardinality of the chosen sequence  $S$ . As we shall see, a smaller cardinality  $S$  induces fewer branch-and-bound nodes and a more balanced branch-and-bound tree. Hence, we are keen to implement separation based on low cardinality component sequences  $S$ .

Reference [15] presents two separation procedures, one that achieves the best complexity in the worst case and another that yields a minimal cardinality subset of component bounds (using an enumeration scheme that has exponential worst case complexity). The procedure proposed here, in Table 2, is a compromise between these two: it partially relies on enumeration in search for a minimal cardinality component sequence  $S$  while guaranteeing a polynomial worst case complexity. In Table 2,  $F = \{q : \tilde{\lambda}_q - \lfloor \tilde{\lambda}_q \rfloor > 0\}$  denotes the set of fractional columns,  $\tilde{\lambda}(F) = \sum_{q \in F} \tilde{\lambda}_q$  its weight,  $F(S) = F \cap Q(S)$  is the set of fractional columns satisfying component bounds in  $S$ , and *record* is the list of identified sequences  $S$  with fractional weight  $f = \tilde{\lambda}(F(S)) \notin \mathbb{N}$  on which we could branch.

**Example 2 (example 1 continued)** Consider the following fractional master solution  $\tilde{\lambda}$ :

$$\tilde{\lambda}: \begin{array}{c|cccccccccccc|c} \tilde{\lambda}_q & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 1 & 1 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 5 \\ \hline x_1 & 1 & \mathbf{1} & 1 & \mathbf{1} & 1 & 1 & 1 & 0 & 0 & 0 & \mathbf{0} & 0 & \mathbf{0} & 0 & 0 & 3 \\ x_2 & 1 & \mathbf{1} & 1 & \mathbf{0} & 0 & 0 & 0 & 1 & 1 & 1 & \mathbf{1} & 0 & \mathbf{0} & 0 & 0 & 3 \\ x_3 & 1 & \mathbf{0} & 0 & \mathbf{1} & 1 & 0 & 0 & 1 & 1 & 0 & \mathbf{0} & 1 & \mathbf{1} & 0 & 0 & 2 \\ x_4 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 1 & 0 & 1 & 0 & 1 & \mathbf{0} & 1 & \mathbf{0} & 1 & 0 & 2 \end{array}$$

It yields

$$\lambda(F(x_i)) = \lambda(F(\bar{x}_i)) = 1 \text{ for } i = 1, \dots, 4.$$

Assume that subproblem variable are indexed in order of non-increasing priority. Hence, we make recursive calls to *Separate* to split further column class defined by  $S = \langle x_1 \rangle$  and  $\langle \bar{x}_1 \rangle$ . We get  $\lambda(F(x_1, x_i)) = \lambda(F(x_1, \bar{x}_i)) = \frac{1}{2}$  for  $i = 2, 3$  and pick highest priority set  $S = \langle x_1, x_2 \rangle$ . On the other hand, further splitting column class defined by  $S = \langle \bar{x}_1 \rangle$  yields branching set  $S = \langle \bar{x}_1, x_2 \rangle$ . ■

Observe that  $|F| \leq m$  where  $m$  is the number of constraints in the master formulation at the current B-a-B node (which will grow when branching constraints are added). Each

Table 2: Separation of a fractional master solution,  $\tilde{\lambda}$ , at the root node

1. Let  $F = \{q : \tilde{\lambda}_q - \lfloor \tilde{\lambda}_q \rfloor > 0\}$ ,  $I = \{1, \dots, n\}$ ,  $S = \langle \rangle$ , record =  $\emptyset$ .
2. Separate( $F$ ,  $I$ ,  $S$ , record)
  - (a) If  $F = \emptyset$ , return.
  - (b) If  $f = \tilde{\lambda}(F) - \lfloor \tilde{\lambda}(F) \rfloor > 0$ , record.add( $S$ ,  $f$ ) and return.
  - (c) *Compute aggregate values*  $(v_i)_{i \in I}$ : Let  $v = 0$ .  
For all  $i \in I$  and  $q \in F$ ,  $v_i += x_i^q \tilde{\lambda}_q$ .
  - (d) *Detect fractional  $v_i$  if any*: Found = false; for all  $i \in I$ , do {  
if  $(f = v_i - \lfloor v_i \rfloor > 0)$  { record.add( $\langle S, x_i \rangle$ ,  $f$ ); Found = true.} }
  - (e) If Found, return.
  - (f) *Identify discriminating components*:  $J = \{i \in I : 0 < v_i < \tilde{\lambda}(F)\}$ .
  - (g) *Partition according to the component with highest branching priority*:  
Let  $i^* = \operatorname{argmax}_{j \in J} \{\text{priority}_j\}$ ;  
Separate( $F(\langle S, x_{i^*} \rangle)$ ,  $J \setminus \{i^*\}$ ,  $\langle S, x_{i^*} \rangle$ , record);  
Separate( $F(\langle S, \bar{x}_{i^*} \rangle)$ ,  $J \setminus \{i^*\}$ ,  $\langle S, \bar{x}_{i^*} \rangle$ , record);  
return;
3. Select a branching set  $S$  in record according to branching priority.

call to subroutine Separate( $F$ ,  $I$ ,  $S$ , record) of Table 2 requires  $O(n|F|)$  operations where  $n$  is the number of binary components. Routine Separate is called recursively. The depth of the tree of recursive calls is bounded by  $n$ , the number of components. The number of leaf nodes in the tree of recursive calls is bounded by  $|F|$ , since the fractional column set is partitioned at each stage and we only explore non-empty subsets. Therefore, the overall complexity is roughly  $O(n^2 * |F|^2)$ . One could get a  $O(n * |F| \log |F|)$  complexity by applying the recursive call to only one side of the partition in Step (g), selecting the subset of fractional columns with the smallest cardinality. <sup>4</sup>

---

<sup>4</sup>In the case of convexity constraint of the form  $L \leq \sum_{q \in Q} \lambda_q \leq U$ , Step (b) can return an empty set  $S$ , and branching starts by fixing the number of columns used in the solution. The extension to the non-binary case, goes as follows: in step (c), for each integer variable  $x_i$ , fractional columns are sorted by decreasing value of  $x_i^q$ . Then, applying the mapping of Section 3 restricted to component  $i$  may allow

Once the component bound sequence  $S$  is selected, we partition the master solution space. A natural scheme would be to use disjunctive constraint

$$\lambda(S) \leq \lfloor \tilde{\lambda}(S) \rfloor \quad \text{or} \quad \lambda(S) \geq \lceil \tilde{\lambda}(S) \rceil \quad (33)$$

where  $\tilde{\lambda}(S) = \sum_{q \in Q(S)} \tilde{\lambda}_q \notin \mathbb{N}$ . However, the branching constraint  $\lambda(S) \leq \lfloor \tilde{\lambda}(S) \rfloor$  of the first branch is typically weaker than that of the second branch (leading to an unbalanced tree). This drawback can be corrected in part by using a non binary branching tree, as illustrated in Example 3 below. The general presentation of the branching scheme follows. We then prove that the scheme yields a valid separation (Proposition 3 shows that no integer solutions are omitted) and that it is finite (Proposition 4 shows that some progress is made in reaching integrality at each separation, hence the size of the branch-and-price tree is bounded).

**Example 3 (example 2 continued)** *Assume set  $S = \langle x_1, x_2 \rangle$  was selected for branching. If we use a binary branching scheme, enforcing*

$$\lambda(x_1, x_2) \leq 1 \quad \text{or} \quad \lambda(x_1, x_2) \geq 2 ,$$

*we may have solution*

$\tilde{\lambda}_q$	0	0	1	$\frac{3}{4}$	0	0	1	1	0	$\frac{1}{2}$	$\frac{1}{4}$	0	$\frac{1}{2}$	0	0	5
$x_1$	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	$2 + \frac{3}{4}$
$x_2$	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	$2 + \frac{3}{4}$
$x_3$	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	$2 + \frac{1}{4}$
$x_4$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	$2 + \frac{1}{4}$

*in node  $\lambda(x_1, x_2) \leq 1$  (it can easily be seen to satisfy the master constraints). Instead, we decompose the case where  $\lambda(x_1, x_2) \leq 1$  as follows. Compared to the current solution,  $\tilde{\lambda}$ , we want to use fewer columns from among  $Q(x_1, x_2)$ . For this to happen, as the total number of selected columns is fixed to  $T$ , we must increase the number of columns selected from  $Q \setminus Q(x_1, x_2)$ . Because  $Q \setminus Q(x_1, x_2) = Q(x_1, \bar{x}_2) \cup Q(\bar{x}_1)$ , increasing the*

---

to identify a component bound that defines a fractional weight subset of columns. In step (g), one must also select a bound value for  $x_{i^*}$  (a balanced partition of the column set can be achieved by selecting the median value of those encountered in fractional columns) and one must leave component  $i^*$  as a potential component for further separation in the recursive calls to Separate. Then, Step (b) checks whether the subclass induced by the median value bound on  $i^*$  has fractional weight.

number columns selected from the complementary set decomposes into either  $\lambda(x_1, \bar{x}_2) \geq 2$  (assuming implicitly that  $\lambda(x_1) \geq 3$ ), or  $\lambda(\bar{x}_1) \geq 3$  (which implies  $\lambda(x_1) \leq 2$ ). Thus, we define 3 descendant nodes (by adding branching constraints) as follows:

$$\text{Node}(1) \equiv \lambda(\bar{x}_1) \geq 3 ,$$

$$\text{Node}(2) \equiv \lambda(x_1, \bar{x}_2) \geq 2 ,$$

$$\text{Node}(3) \equiv \lambda(x_1, x_2) \geq 2 .$$

Observe that the above solution is not feasible in any of Node(1), Node(2), or Node(3). ■

The general scheme goes as follows. For  $p = 1, \dots, |S| - 1$ , let  $S^p$  be the subsequence made of the first  $p$  component bounds of  $S$  and  $L^p = \tilde{\lambda}(S^p)$ . Let  $L^{|S|} = \lceil \tilde{\lambda}(S) \rceil$  be the round-up of the current weight of column class  $Q(S)$ , while  $S^0 = \langle \rangle$  and  $L^0 = \tilde{\lambda}(Q) = T$ . Thus,  $L^0 \geq L^1 \geq L^2 \geq \dots \geq L^{|S|}$  are decreasing integer values. Let  $Q^p = Q(S^p)$  and  $\bar{Q}^p = Q^{p-1} \setminus Q^p = Q(S^{p-1}, \bar{x}_{[p]})$  for  $p = 1, \dots, |S|$ . Then, we generate  $|S| + 1$  successor nodes as follows. For  $p = 1, \dots, |S|$ , the branching constraint defining node  $p$  is:

$$\text{Node}(p) \equiv \lambda(Q^{p-1} \setminus Q^p) \geq L^{p-1} - L^p + 1 . \quad (34)$$

While node  $|S| + 1$  is defined by:

$$\text{Node}(|S| + 1) \equiv \lambda(Q^{|S|}) \geq L^{|S|} . \quad (35)$$

Figure 2 illustrates branching at a node for which the separation set  $S$  involves 4 component bounds,  $|S| = 4$ , yielding 5 descendant nodes. The node number are referring to the associated node definitions (34-35). The definition of nested column classes is illustrated in Figure 3. The disjunctive branching constraint can be understood as follows: in an integer solution either the cardinality of class  $Q^4$  is increased to the next integer value, i.e.  $\lambda(Q^4) \geq L^4$  (intuitively, the width of slice  $Q^4$  increases in Figure 3), or it is rounded down, i.e.  $\lambda(Q^4) < L^4$  (the width of slice  $Q^4$  decreases; then, some other slice of Figure 3 must have an increased width as formalized hereafter). In any integer solution with  $\lambda(Q^4) < L^4$ , either the cardinality of the complementary class increases,  $\lambda(\bar{Q}^4) = \lambda(Q^3 \setminus Q^4) \geq L^3 - L^4 + 1$ , or the cardinality of the super-class decreases,  $\lambda(Q^3) < L^3$  or both. Similarly, the case  $\lambda(Q^3) < L^3$  decomposes recursively into  $\lambda(Q^2 \setminus Q^3) \geq L^2 - L^3 + 1$ , or  $\lambda(Q^1 \setminus Q^2) \geq L^1 - L^2 + 1$ , or  $\lambda(Q \setminus Q^1) \geq L^0 - L^1 + 1$  which are the different ways in which the cardinality of the complementary class  $Q \setminus Q^3$  can increase.



In Figure 2, the dotted nodes are not explicitly defined. The constraint  $\lambda(Q^p) < L^p$  on the branch leading to the dotted nodes are not strictly enforced, i.e. the subtree hanging from that branch hold solutions that do not satisfy this constraint. However, all integer solutions that do not belong to the left subtree satisfy the constraint on the right branch:

**Lemma 1** *The only integer solutions that are not represented in any of the nodes  $l = p, \dots, |S| + 1$  are those for which  $\lambda(Q^p) < L^p$ .*

**Proof:** Let us show the result by induction. For  $p = |S| + 1$ , the result is trivial. Now let  $p \leq |S|$ . Assume the result is correct for  $p + 1$ , let us show how to derive it for  $p$ . If  $\lambda$  is not solution in any of the nodes  $l = p, \dots, |S| + 1$ , by the induction hypothesis we must have

$$\lambda(Q^{p+1}) \leq L^{p+1} - 1$$

and  $\lambda$  must violate the branching constraint defining  $\text{Node}(p)$ , i.e.

$$\lambda(Q^p \setminus Q^{p+1}) \leq L^p - L^{p+1}$$

As  $\lambda(Q^p) = \lambda(Q^p \setminus Q^{p+1}) + \lambda(Q^{p+1})$ , the two above constraints imply  $\lambda(Q^p) \leq L^p - 1$ . ■

Thus, although with this branching scheme a node can have up to  $n + 1$  descendants<sup>5</sup>, nodes  $1, \dots, p - 1$  only serve the purpose of enumerating integer solution in which  $\lambda(Q^p) < L^p$ . This shall be exploited in Sections 7 and 8 to show how the scheme simplifies in practice. Note that the associated solution set are not disjoint but they yield a valid partition.

**Proposition 3** *The descendant branch-and-bound nodes  $\text{Node}(1), \dots, \text{Node}(|S| + 1)$  of  $\text{Node}(0)$  define a valid separation scheme, i.e.,*

- (i) *the fractional solution  $\tilde{\lambda}$  of  $\text{Node}(0)$  is not feasible in any of the descendant nodes;*
- (ii) *any integer solution to  $\text{Node}(0)$  is feasible in at least one of the descendant nodes.*

**Proof:** By construction, the current solution  $\tilde{\lambda}$  violates some of the branching constraints in each descendant node. It remains to make sure that no integer solution have been lost in the process. All integer solutions use  $T$  columns in total (counting multiple copies of the same column). Clearly, integer solutions that use at least  $L^{|S|-1}$  columns in class

---

<sup>5</sup>In applications where the master convexity constraint  $\sum_{q \in Q} \lambda_q = T$  is replaced by  $L \leq \sum_{q \in Q} \lambda_q \leq U$ , node  $\lambda(Q^0) < L^0$  of Figure 2 should be explicitly defined. Thus, branching on a set  $S$  yields  $|S| + 2$  descendant nodes.

$Q(S^{|S|-1})$  shall use either at least  $L^{|S|}$  columns in class  $Q(S^{|S|})$  or at least  $L^{|S|-1} - L^{|S|} + 1$  in the complementary sub-class  $Q^{|S|-1} \setminus Q^{|S|}$ , or both. Other integer solutions use strictly less than  $L^{|S|-1}$  columns in class  $Q^{|S|-1}$ . As the total number of columns is fixed to  $T$ , this implies that the complementary class,  $Q \setminus Q^{|S|-1}$  must hold a least  $T - L^{|S|-1} + 1$  columns, i.e., one more than in the current solution  $\tilde{\lambda}$ . The complementary class,  $Q \setminus Q^{|S|-1}$ , can be partitioned into  $Q^{p-1} \setminus Q^p$  for  $p = 1, \dots, |S| - 1$ , i.e.,  $Q \setminus Q^{|S|-1} = \cup_{p=1, \dots, |S|-1} (Q^{p-1} \setminus Q^p)$  and these sub-classes are disjoint. Thus, the case where the cardinality of the complementary set,  $Q \setminus Q^{|S|-1}$ , must increase by one unit can be decomposed into  $|S| - 1$  sub-cases depending on which of the subset of the partition sees its cardinality increased by one unit compared to the current solution  $\tilde{\lambda}$ . These sub-cases are defined by Node(1) to Node( $|S| - 1$ ). ■

Example 3 did illustrate that this scheme is stronger than (33). In fact, one can prove that each branching results in fixing some  $x_i^t$  component for the solution  $x$  obtained by the procedure of Table 1. Let us first illustrate this by a numerical example.

**Example 4 (example 3 continued)** Assume a fractional solution represented below with the associated weight for column subclasses  $Q(x_1, x_2)$ ,  $Q(x_1, \bar{x}_2)$  and  $Q(\bar{x}_1)$ . The framed values point to column component that define the boundaries of column classes.

$\tilde{\lambda}_q$	0	$\frac{1}{2}$	1	$\frac{1}{2}$	0	0	1	1	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0
$x_1$	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
$x_2$	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
$x_3$	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
$x_4$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
	$(x_1)$						$(\bar{x}_1)$								
	$(x_1, x_2)$			$(x_1, \bar{x}_2)$											
	1.5			1.5			2								

After branching, the size of the slices (i.e., the weight of the associated column classes) are changed: For Node(1),  $\lambda(\bar{x}_1) \geq 3 \Rightarrow x_1^t = 0$  for  $t = 3$ . For Node(2),  $\lambda(x_1, \bar{x}_2) \geq 2 \Rightarrow x_2^t = 0$  for  $t = 2$ . For Node(3),  $\lambda(x_1, x_2) \geq 2 \Rightarrow x_2^t = 1$  for  $t = 2$ . ■

**Proposition 4** After branching, some new  $x_i^t$  are fixed to zero or one in the  $x$  solution obtained by the procedure of Table 1 (using a lexicographic ordering induced by the order in which components appear in the sequences that define branching constraints). Hence, the number of leaf nodes in the branch-and-price tree is at most  $nT$ .

**Proof:** For nodes  $p = 1, \dots, |S| - 1$ , the branching constraint induces an increased cardinality on column class  $\overline{Q}^p = Q^{p-1} \setminus Q^p = Q(S^{p-1}, \overline{x}_{[p]})$ . Say  $\overline{x}_{[p]} = x_{[p]}$  (resp.  $\overline{x}_{[p]} = \overline{x}_{[p]}$ ), then component  $x_{[p]}^t$  is set to one (resp. to zero) for a given index  $t$  that, in the current solution, was associated with the first (resp. the last) vector from among the all vectors  $x^t$  associated with column class  $Q^p$ . The designated component  $x_i^t$  had not been fixed in this way before since the current solution violates that setting. For node  $|S|$  (resp.  $|S| + 1$ ),  $x_{[|S|]}^t$  is fixed to zero (resp. to one) for the index  $t$  associated with the  $L^{|S|}$ <sup>th</sup> vector  $x^t$  associated with column class  $Q(S)$  while it was fractional in the current solution. In each case, the specific index  $t$  for which a  $x_i^t$  is fixed may change after further branching, but the number of  $x_i^t$  components that are fixed can only increase in descendant nodes. ■

Further down the branch-and-price tree, separation of fractional solutions must be implemented in such a way as to preserve a nested partition of the space of subproblem solutions. Indeed, as we show in Section 5, solving the master after branching can be done efficiently when the column sub-classes defined by branching constraints are either disjoint or nested. To achieve a nested partition of the column set, separation must be done by further partitioning either  $Q(S^p)$  or its complementary class  $\overline{Q}(S^p)$  for some  $S^p \subseteq S$  of some previously used branching set  $S$ . This procedure shall be detailed in Section 7.

## 5 Solving the master after branching

At a given branch-and-price node, the master LP takes the form:

$$\min \sum_{q \in Q} c x^q \lambda_q \quad (36)$$

$$[\text{M(node)}] \quad \sum_{q \in Q} A x^q \lambda_q \geq a \quad (\pi) \quad (37)$$

$$\sum_{q \in Q} \lambda_q = T \quad (\sigma_0) \quad (38)$$

$$\sum_{q \in Q^k} \lambda_q \geq L^k \quad (\sigma_k) \quad \forall k = 1, \dots, K \quad (39)$$

$$\lambda_q \geq 0 \quad \forall q \in Q \quad (40)$$

where  $(\pi, \sigma)$  denotes the associated dual solution and constraints  $\sum_{q \in Q^k} \lambda_q \geq L^k$  are branching constraints. Each class  $Q^k$  is defined by a component bound sequence  $S^k$ , i.e.  $Q^k = Q(S^k)$ . As mentioned above, we assume that the branching constraints have been introduced in such a way that the collection  $Q^k$  for  $k = 1, \dots, K$  define a nested partition

of  $Q$ . Hence, they can be represented by a tree. The root node is associated with  $Q$ . The leaf nodes represent classes  $Q^k$  that have no subclasses in the collection. The node associated with  $Q^i$  hangs from that associated with class  $Q^j$  if  $Q^i \subset Q^j$  and there is no  $k$  such that  $Q^i \subset Q^k \subset Q^j$ .

**Example 5** Let  $n = 4$  and  $T = 10$ . Assume that the current set of branching constraints is:

$$\lambda(x_2, x_3, x_4) \geq 1, \quad (41)$$

$$\lambda(x_2, x_3, x_4, \bar{x}_1) \geq 1, \quad (42)$$

$$\lambda(x_2, \bar{x}_3, x_1) \geq 1, \quad (43)$$

$$\lambda(\bar{x}_2, x_1) \geq 4, \quad (44)$$

$$\lambda(\bar{x}_2, x_1, x_4) \geq 2, \quad (45)$$

$$\lambda(\bar{x}_2, \bar{x}_1) \geq 3, \quad (46)$$

$$\lambda(\bar{x}_2, \bar{x}_1, x_3, x_4) \geq 1. \quad (47)$$

The associated tree of column classes is given in Figure 4. ■

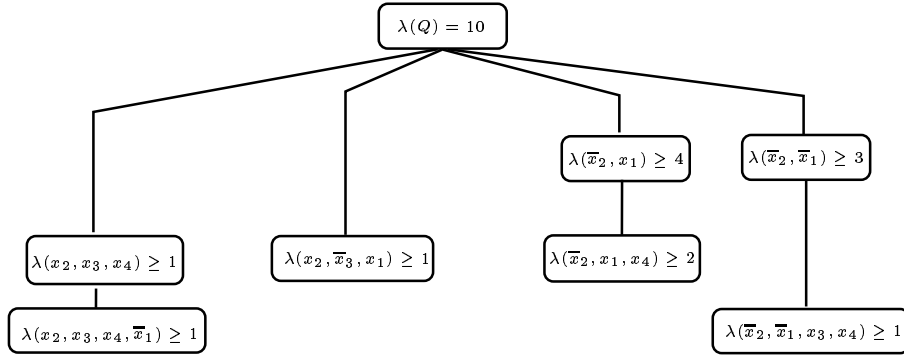


Figure 4: Tree of nested column classes

Some branching constraints can be eliminated as they cannot be active.

**Definition 1** A branching constraint  $\lambda(Q^k) \geq L^k$  is redundant if

$$\sum_{i \in \text{succ}(k)} L^i \geq L^k \quad (48)$$

where  $\text{succ}(k)$  is the set of branching constraints associated with direct successors of class  $Q^k$  in the current tree of column classes.

In example 5, branching constraint (41) is redundant. Observe that redundant branching constraints will remain so at descendant nodes. Hence, in the sequel, we assume that redundant branching constraints have been eliminated from the set of branching constraint defining the current branch-and-price node. Note that class  $Q$  itself can define a redundant cardinality constraint  $\sum_{q \in Q} \lambda_q \geq T$  (when the lower bounds in its sub-classes sum up to  $T$ ). However, in this case, we still associate the root of the column class tree with  $Q$ .

**Proposition 5** *The number of non-redundant branching constraints at a branch-and-price node is at most  $T$ .*

**Proof:** Once *redundant* branching constraints have been eliminated, there remains a collection of column classes  $Q^k$  for  $k = 1, \dots, K$  for which  $\sum_{i \in \text{succ}(k)} L^i < L^k$ . Thus, each *non-redundant* class  $k$  is associated with a strictly positive marginal lower bound  $\bar{L}^k = L^k - \sum_{i \in \text{succ}(k)} L^i > 0$ :  $\bar{L}^k$  columns are reserved for class  $Q^k$  that are not in any currently defined sub-classes of  $Q^k$ . As there are at most  $T$  columns in total, they can be at most  $T$  classes with strictly positive marginal lower bound. ■

Let us now see how the master program [M(node)] can be solved by column generation. For each *non-redundant* class  $Q^k$  and associated component bound sequence  $S^k$ ,  $k = 1, \dots, K$ , we define a pricing problem:

$$[SP^k] \quad \zeta^k(\pi) = \min\{(c - \pi A)x : x \in X^k\}, \quad (49)$$

where  $X^k = \{x \in \{0, 1\}^n : Bx \geq b, \tilde{x}_{[i]} = 1 \forall i \in S^k\}$ . It can be solved with the oracle provided for pricing problem (13): additional constraints consists only in fixing the value of some binary variables. If pricing problem  $[SP^k]$  is infeasible (i.e.  $X^k = \emptyset$ ), we assume  $\zeta^k(\pi) = \infty$ . The column generation procedure can be carried out by solving each pricing problem (49) for  $k = 1, \dots, K$  and returning columns that have negative reduced costs. If none can be found, the current master LP value is optimal.

Observe that pricing subproblems (49) are not independent. A relaxation of a pricing problem over  $Q^i$  is to solve it over a super-class  $Q^j \supset Q^i$ , i.e.

**Observation 2** *If  $Q^i \subset Q^j$  (i.e.  $S^i \supset S^j$ ),  $\zeta^i(\pi) \geq \zeta^j(\pi)$ .*

If the solution to this relaxation is feasible for the original pricing problem, then it is optimal for it, i.e.

**Observation 3** *If  $x^j = \operatorname{argmin} \zeta^j(\pi)$  and  $x^j \in Q^i \subset Q^j$ , then  $x^j = \operatorname{argmin} \zeta^i(\pi)$ .*

However, it is to be observed that if a *non-redundant* branching constraint,  $k$ , is active, the associated dual reward will be positive, i.e.  $\sigma_k > 0$ . This in turn induces a reduction of the reward  $\pi_i$  associated with components  $\tilde{x}_i$  present in sequence  $S^k$ . Hence, it shall make it less likely that a solution  $x^k \in Q^k$  defines a solution for a more constrained subproblem  $Q^j \subset Q^k$ .

To exploit the relationship that exists between pricing subproblems, column generation can proceed by enumerating subproblems along a breadth first search exploration of the tree of column classes associated to *non-redundant* branching constraints. Then, one can interrupt the procedure as soon as a negative reduced cost column is found and yet have a dual bound on the reduced cost of unsolved subproblems. The proposed procedure is given in Table 3, where  $\operatorname{pred}(k)$  is the set of branching constraints associated with super classes of  $Q^k$ .

Table 3: Column generation procedure

1. Let  $D$  be the depth of the tree of column classes associated to *non-redundant* branching constraints and  $d = 0$ .
2. If  $d > D$ , Stop.
3. For each pricing problem  $SP^k$  at depth  $d$ , do
  - (a) If  $x^{\operatorname{pred}(k)} \in Q^k$ , continue ( $SP^k$  already solved);
  - (b) If  $\zeta^{\operatorname{pred}(k)} - \sum_{j \in \operatorname{pred}(k)} \sigma_j - \sigma_k \geq 0$ , continue ( $SP^k$  shall not yield a negative reduced cost column);
  - (c) Compute  $\zeta^k$  and  $x^k$ ;
  - (d) If  $\zeta^k - \sum_{j \in \operatorname{pred}(k)} \sigma_j - \sigma_k < 0$ , return  $x^k$  and Stop.
  - (e) If  $x^k$  is feasible for any successor of  $SP^k$ ,  $i \in \operatorname{succ}(k)$ , record it as solution for  $SP^i$ . If, moreover,  $\zeta^k - \sum_{j \in \operatorname{pred}(i)} \sigma_j - \sigma_i < 0$ , return  $x^k$  and Stop.
4. Let  $d += 1$  and goto step 2.

As the procedure stops at the first identified negative reduced cost column, the order in which subproblems are treated is important.

**Remark 1** *In the procedure of Table 3, a heuristic rule to select the next sub-problem  $SP^k$  to be solved from among those of the same depth is to give priority to that with the largest lower bound  $L^k$  and, among ties, to select the one that has not been solved for the largest number of iterations.*

The intuition is: (i) there are more columns to generate among class with larger cardinality  $L^k$ ; (ii) their weight is larger in the dual bound computation (see Section 6) and hence it is important to get their exact reduced cost. Then, using a cyclic scanning of the subproblem is natural given that dual solution adjust for the recently added columns.

## 6 Lagrangian dual bounds

At each iteration of the column generation procedure, a valid dual bound on the master value can be obtained by Lagrangian relaxation. Dualizing constraints (37) yields

$$\pi a + \min \left\{ \sum_{q \in Q} (c - \pi A) x^q \lambda_q : \sum_{q \in Q} \lambda_q = T; \sum_{q \in Q^k} \lambda_q \geq L^k \quad k = 1, \dots, K; \lambda_q \geq 0, q \in Q \right\}. \quad (50)$$

Because column classes are nested, this problem is trivial.

**Proposition 6** *The solution of problem (50) is*

$$\theta(\pi) = \pi a + \sum_{k=0}^K \bar{L}^k \zeta^k(\pi) \quad (51)$$

where  $Q^0 = Q$ ,  $L^0 = T$ , and  $\bar{L}^k$  are marginal lower bound, i.e.  $\bar{L}^k = L^k - \sum_{i \in \text{succ}(k)} L^i$ .

**Proof:** The result derived from Observation 2 (that implies  $\sum_{q \in Q^k} \lambda_q^* = L^k \quad \forall k$  in an optimal solution  $\lambda^*$  to (50)). ■

Observe that a Lagrangian dual bound can be computed even if the column generation procedure falls short of solving every pricing subproblems  $SP^k$  defined by (49). Replacing  $\zeta^k(\pi)$  by a dual bound on  $\zeta^k(\pi)$  in (51) yields a valid dual bound for  $[M(\text{node})]$ . Observation 2 points naturally to the pricing subproblem dual bounds obtained when solving hyper-classes. Thus, provided we use a breadth first search enumeration of the tree pricing subproblems, an approximation of  $\theta(\pi)$  can easily be computed at any stage of the

column generation procedure:  $\tilde{\theta}(\pi) = \pi a + \sum_{k \in G} \bar{L}_G^k \zeta^k(\pi)$ , where  $G$  is the set of solved subproblems and  $\bar{L}_G^k = L^k - \sum_{i \in \text{succ}(k) \cap G} L^i$ . To this end, we might want to solve problem  $SP^0$  even if class  $Q$  is *redundant* (i.e. when  $\bar{L}^0 = 0$ ). Alternatively, one needs to solve all subproblems  $SP^j$  for  $j \in \text{succ}(\text{root})$  to compute a Lagrangian bound when  $\bar{L}^0 = 0$ .

An important property of our branching scheme is that branching constraints are enforced in the subproblem, which yields stronger dual bound than when there are enforced in the master and hence dualized:

**Proposition 7** *Upon completion of the column generation procedure, the dual bound (51) yields the value of the Lagrangian dual problem whose primal formulation is*

$$\min \left\{ \sum_t c x^t : \sum_t A x^t \geq a, x^t \in \text{conv}(X^k) \text{ for } t = t^{k-1}, \dots, t^k - 1 \text{ and } k = 0, \dots, K \right\} \quad (52)$$

where  $t^{-1} = 1$ , and  $t^k = t^{k-1} + \bar{L}^k$  for  $k = 0, \dots, K$ .

**Proof:** Applying the Dantzig-Wolfe reformulation principle to

$$\min \left\{ \sum_t c x^t : \sum_t A x^t \geq a, x^t \in X^k \text{ for } t = t^{k-1}, \dots, t^k - 1 \text{ and } k = 0, \dots, K \right\}$$

one gets

$$\begin{aligned} \min \left\{ \sum_{k=0}^K \sum_{t=t^{k-1}}^{t^k-1} \sum_{q \in Q^k} c x^q \lambda_q^t : \sum_{k=0}^K \sum_{t=t^{k-1}}^{t^k-1} \sum_{q \in Q^k} A x^q \lambda_q^t \geq a; \right. \\ \left. \sum_{q \in Q^k} \lambda_q^t = 1 \quad \forall k, t = t^{k-1}, \dots, t^k - 1; \lambda_q^t \geq 0 \quad \forall t, q \right\}. \end{aligned} \quad (53)$$

By Geoffrion [7], (53) has LP value equal to that of (52). Note that our master program has the same LP solution as (53). Indeed, by aggregating  $\lambda_q^t$  variables in (53), defining  $\lambda_q = \sum_t \lambda_q^t$ , and by regrouping convexity constraints  $\sum_{q \in Q^k} \lambda_q^t = 1$  of nested sub-classes  $Q^k$ , one gets formulation [M(node)].  $\blacksquare$

To see that enforcing branching constraint in the subproblem can yield stronger dual bound, one just needs to compare (52) to the bound given by (15) when master constraint  $\sum_t A x^t \geq a$  include the branching constraints and  $X^t = X$  (as defined in (32)) for all  $t = 1, \dots, T$ .

## 7 Enforcing a nested separation scheme

At a branch-and-price node other than the root, the choice of a component bound sequence  $S$  on which to branch is restricted to those yielding a nested partition of the

subproblem solution space. If the current set of branching constraint defines a nested partition of  $Q$ , all previous branching sequences  $S^k$ ,  $k = 1, \dots, K$ , start with a bound on the same component (as in Example 5). The beginning of the sequence  $S$  on which we branch must follow this pattern. Let us say that this first component is  $i$ , i.e.,  $\tilde{x}_{[1]}$  is either  $x_i$  or  $\bar{x}_i$  in all  $S^k$ . Using the notation of Section 4, let  $\tilde{\lambda}$  be the master LP at the current node,  $F = \{q : \tilde{\lambda}_q - \lfloor \tilde{\lambda}_q \rfloor > 0\} \neq \emptyset$  and  $\tilde{\lambda}(\tilde{x}_i) = \sum_{q \in F: \tilde{x}_i^q = 1} (\tilde{\lambda}_q - \lfloor \tilde{\lambda}_q \rfloor)$ . If  $\tilde{\lambda}(x_i) \notin \mathbb{N}$ , we can branch on  $S = \langle x_i \rangle$ . Otherwise, we recursively explore both sets of fractional columns  $F(\langle x_i \rangle)$ ,  $F(\langle \bar{x}_i \rangle)$  (if not empty) in search for a set  $S$  on which to branch. When no more previous branching sequence dictates the next component, we call the subroutine Separate of Table 2. The complete separation procedure is presented in Table 4. Therein,  $C$  stands for the set of sequences associated to *non-redundant* branching constraints,  $p$  denotes the current position in the sequence and  $C(S)$  the subset of sequences starting with  $S$ . Its use is illustrated in Example 6.

Table 4: Separation of a fractional master solution at a node other than the root

1. Let  $F = \{q : \tilde{\lambda}_q - \lfloor \tilde{\lambda}_q \rfloor > 0\}$ ,  $I = \{1, \dots, n\}$ ,  $S = \langle \rangle$ , record =  $\emptyset$ ,  $p = 1$ , and  $C = \{S^k\}_{k=1, \dots, K}$  the set of *non-redundant* branching constraints.
2. Explore( $C$ ,  $p$ ,  $F$ ,  $I$ ,  $S$ , record)
  - (a) If  $C = \emptyset$ , return Separate( $F$ ,  $I$ ,  $S$ , record).
  - (b) Let  $i$  be the component such that  $x_{[p]} = \tilde{x}_i$  in all  $S^k \in C$ . Set  $I = I \setminus \{i\}$ .
  - (c) Let  $v_i = \sum_{q \in F} x_i^q \tilde{\lambda}_q$ .
  - (d) If  $f = v_i - \lfloor v_i \rfloor > 0$  (*column class  $Q(\langle S, x_i \rangle)$  has fractional weight*), record.add( $\langle S, x_i \rangle$ ,  $f$ ) and return.
  - (e) If  $v_i > 0$ , Explore( $C(\langle S, x_i \rangle)$ ,  $p + 1$ ,  $F(\langle S, x_i \rangle)$ ,  $\langle S, x_i \rangle$ , record).
  - (f) If  $v_i < \tilde{\lambda}(F)$ , Explore( $C(\langle S, \bar{x}_i \rangle)$ ,  $p + 1$ ,  $F(\langle S, \bar{x}_i \rangle)$ ,  $\langle S, \bar{x}_i \rangle$ , record).
3. Select a record according to branching priority.

**Example 6 (example 5 continued)** Assume that the node is defined by branching constraints (42-47) and the fractional solution is

$\tilde{\lambda}_q$	1	0	1		$\frac{3}{4}$	$\frac{1}{4}$	0	1		1	1	1	1		1	$\frac{3}{2}$	$\frac{1}{2}$	0	10
$x_2$	1	1	1		1	1	1	1		0	0	0	0		0	0	0	0	
$x_3$	1	1	1		0	0	0	0	$x_1$	1	1	1	1		0	0	0	0	
$x_4$	1	0	0	$x_1$	1	1	0	0	$x_4$	1	1	0	0	$x_3$	1	1	0	0	
$x_1$	0	1	0	$x_4$	1	0	1	0	$x_3$	1	0	1	0	$x_4$	1	0	1	0	

with  $\tilde{\lambda}(F) = 3$ . Sequence  $S$  must start with  $x_2$  or  $\bar{x}_2$ . Partitionning according to component 2 split the set of fractional columns, with  $\tilde{\lambda}(F(x_2)) = 1$  and  $\tilde{\lambda}(F(\bar{x}_2)) = 2$ . Component 3 does not allow to partition  $F(x_2)$  further since all the weight is on  $\bar{x}_3$  neither does component 1. Thus  $\tilde{\lambda}(F(x_2, \bar{x}_3, x_1)) = 1$  must be further split using routine Separate which returns  $S^1 = \langle x_2, \bar{x}_3, x_1, x_4 \rangle$ . On the other hand, set  $F(\bar{x}_2)$  cannot be partitionned further with component 1, but component 3 yields a fractional weight subset. Thus, another possible branching set is  $S^2 = \langle \bar{x}_2, \bar{x}_1, x_3 \rangle$ . One of these two branching sets is selected according to branching priorities. ■

Let us consider the complexity of the separation procedure of Table 4. The routine Explore is called recursively and, on each leaf of the tree of recursive calls to Explore, routine Separate is called recursively. However, the overall depth of the tree of recursive calls is bounded by the number of components,  $n$ , and the overall number of leaf nodes in the tree of recursive calls to Explore and Separate is bounded by  $|F|$ , since the fractional column set is partitioned at each stage and we only explore non-empty subsets. The work done in a call to Explore is in  $O(|F|)$  while that in a call to Separate is  $O(n|F|)$ . Hence, the overall complexity the procedure is roughly  $O(n^2 * |F|^2)$  as it was at the root node. Here too, one can get a lower complexity by exploring only the smallest cardinality subset of fractional columns (selecting step (e) or (f)). However, exploring both sides often allows to identify a branching set  $S$  with fewer component bounds as illustrated in Example 6.

The branching sequences  $S$  resulting from the above procedure come in three kinds: either it is obtained by resetting the bound of an inactive branching constraint (case a) or by further splitting an existing class – adding new component bound restrictions to an existing sequence  $S^k$  which can either be a leaf node class in the tree of column classes (case b) or not (case c). Once, the sequence  $S$  is identified for separation, we generate successor nodes as presented in Section 4. However, in view of Lemma 1, we do not need

to generate the successor nodes  $1, \dots, k-1$ , that only serve the purpose of enumerating integer solutions in which  $\lambda(Q^k) < L^k$ , when a previous branching constraint states the opposite, i.e.  $\lambda(Q^k) \geq L^k$ .

**Proposition 8** *When branching is needed, the set of descendant nodes defined in (34-35) can be restricted to only nodes  $k, \dots, |S| + 1$ , where  $S$  is the sequence chosen to separate the current fractional solution  $\tilde{\lambda}$  and  $k$  is the size of the largest subsequence  $S^k$  of  $S$  for which a previously defined branching constraint  $\lambda(Q^k) \geq L^k$  is active, i.e.  $L^k = \tilde{\lambda}(Q^k)$ .*

The proof is a straightforward application of Lemma 1.

In practice, this observation often leads to generating only two branches since the separation sequence,  $S$ , is in most cases obtained by adding a single component bound to a previously defined set  $S^k$  associated with an active branching constraint, whether  $S^k$  is associated with a leaf node class (in case b) or not (in case c). In case a, resetting the bound of a class whose parent class defines an active branching constraint also yields only two branches. Thus, the branch-and-bound tree is often close to be a binary tree. Moreover, some successor nodes can typically be pruned by infeasibility as shown next.

## 8 Preprocessing at branch-and-price node

The first preprocessing operation at a node consists in deleting redundant branching constraints  $j$  such that  $L^j \leq \sum_{i \in succ(j)} L^i$ . As we have just seen, further branching can only increase the lower bound enforced on a subset  $Q^j$ . Hence, redundant branching constraints shall not become active at a descendant node. They can be deleted from the node definition. Then, there remains only branching constraints,  $k$ , with strictly positive marginal lower bound  $\bar{L}^k = L^k - \sum_{i \in succ(k)} L^i > 0$  that must verify  $\sum_{k=1}^K \bar{L}^k \leq T$ .

Branching constraints induce a lower bound on the aggregate value of subproblem variables. The current node problem is infeasible if such induced lower bounds exceed global upper bounds (31), i.e., if

$$\sum_{k: S^k \ni x_i} \bar{L}^k > u_i \text{ for some } i.$$

Similarly, the current node problem is infeasible if the the global lower bound (31) cannot

be met for some component  $i$ , i.e. if

$$T - \sum_{k:S^k \ni \bar{x}_i} \bar{L}^k < l_i .$$

On the other hand, if  $\sum_{k:S^k \ni x_i} \bar{L}^k = u_i$ , we must have  $x_i^q = 0$  in all columns specifically generated for class  $k \in \bar{K}(x_i) = \{k : S^k \not\ni x_i\}$ , i.e., in the notation of Proposition 7,  $x_i^t = 0$  for  $t = t^{k-1}, \dots, t^k - 1$ ,  $k \in \bar{K}(x_i)$ . Thus, an *implied branching constraint* can be added of the form

$$\lambda(S^k, \bar{x}_i) \geq \bar{L}^k \tag{54}$$

for all  $k \in \bar{K}(x_i)$ . Observe that constraint (54) makes branching constraint  $k$  *redundant* and hence it simply replaces it.

Once all branching constraints  $k \in \bar{K}(x_i)$  are processed in this way, all branching sequences  $S^k$  for  $k = 1, \dots, K$  include  $\tilde{x}_i$  (i.e either  $x_i$  or  $\bar{x}_i$ ). Thus, this component can be brought in the first position in all sequences and this guarantees that the branching constraints still define a nested partition of the solution space. Similarly, if  $T - \sum_{k:S^k \ni \bar{x}_i} \bar{L}^k = l_i$ , for some component  $i$ , then, for all branching constraints  $k \in \bar{K}(\bar{x}_i) = \{k : S^k \not\ni \bar{x}_i\}$ ,  $x_i$  can be added to their component bound sequence  $S^k$  and this component is placed in first position in all sequences. We refer to such preprocessing as *further specification of column classes*.

Another form of preprocessing consists in *deleting columns* that are no longer needed. Note that when class  $Q$  is *redundant*, i.e. when  $\sum_{k \in \text{succ}(\text{root})} L^k = T$ , columns in  $Q \setminus \cup_{k \in \text{succ}(\text{root})} Q^k$  can be deleted from the formulation. Such preprocessing can be generalized to any *redundant* class  $j$  for which

$$\sum_{k \in \text{succ}(j)} L^k = U^j$$

where  $U^j$  is a valid upper bound on class  $j$  cardinality. For instance,  $U^j$  can be set as

$$\min \left\{ T - \sum_{k \notin \text{subtree}(j)} \bar{L}_k, \min_{i: x_i \in S^j} \left\{ u_i - \sum_{k \notin \text{subtree}(j): S^k \ni x_i} \bar{L}_k \right\}, \min_{i: \bar{x}_i \in S^j} \left\{ T - \sum_{k \notin \text{subtree}(j): S^k \ni \bar{x}_i} \bar{L}_k - l_i \right\} \right\},$$

where  $\text{subtree}(j)$  stands for the set of *non-redundant* branching constraints that are associated with subclasses of  $Q^j$ .

## 9 The set partitioning special case

When the master program is a set partitioning problem of the form (28), preprocessing is particularly effective as global bounds are tight:  $l_i = u_i = 1 \forall i$ . Example 7 illustrates this on the bin packing problem.

**Example 7** *Take a bin packing problem on 4 items for which the feasible knapsack solutions are represented by the columns*

$q$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$x_1$	1	1	1	1	1	1	0	0	0	0	0	0	0	0
$x_2$	1	1	0	0	0	0	1	1	1	1	0	0	0	0
$x_3$	0	0	1	1	0	0	1	1	0	0	1	1	0	0
$x_4$	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Let  $T = 2$ . Assume the fractional master solution is  $\tilde{\lambda}_2 = \tilde{\lambda}_3 = \tilde{\lambda}_{10} = \tilde{\lambda}_{11} = \frac{1}{2}$ . Then, branching on  $S = \langle x_1, x_2 \rangle$  yields 3 nodes:

$$\text{Node}(1) \equiv \lambda(\bar{x}_1) \geq 2$$

$$\text{Node}(2) \equiv \lambda(x_1, \bar{x}_2) \geq 1$$

$$\text{Node}(3) \equiv \lambda(x_1, x_2) \geq 1$$

Preprocessing allows to prune Node(1) by infeasibility. Now, let us examine Node(3) (the preprocessing of Node(2) being symmetric). Preprocessing detects that, as item 1 is entirely covered by columns of class  $Q(x_1, x_2)$ , we can add branching constraint  $\lambda(\bar{x}_1) \geq T - 1 = 1$ . Similarly item 2 is entirely covered by class  $Q(x_1, x_2)$ . Thus, the branching constraints are redefined as  $\lambda(x_2, x_1) \geq 1$  and  $\lambda(\bar{x}_2, \bar{x}_1) \geq 1$ . Then, class  $Q$  becomes redundant and columns 3 to 10 can be deleted. Now, assume that the fractional solution at node(3) is  $\tilde{\lambda}_1 = \tilde{\lambda}_2 = \tilde{\lambda}_{11} = \tilde{\lambda}_{12} = \frac{1}{2}$  and that  $S = \langle \bar{x}_2, \bar{x}_1, x_4 \rangle$  is chosen for separation. Although  $|S| = 3$ , only two successor nodes need to be defined. Indeed, according to Lemma 1, Node (3.1) and (3.2) would only aim at enumerating solutions where  $\lambda(\bar{x}_2, \bar{x}_1) < 1$  which contradicts previous constraint  $\lambda(\bar{x}_2, \bar{x}_1) \geq 1$ . Hence, there remains

$$\text{Node}(3.3) \left\{ \begin{array}{l} \lambda(x_2, x_1) \geq 1 \\ \lambda(\bar{x}_2, \bar{x}_1) \geq 1 \\ \lambda(\bar{x}_2, \bar{x}_1, \bar{x}_4) \geq 1 \end{array} \right. ;$$

$$\text{Node}(3.4) \left\{ \begin{array}{l} \lambda(x_2, x_1) \geq 1 \\ \lambda(\bar{x}_2, \bar{x}_1) \geq 1 \\ \lambda(\bar{x}_2, \bar{x}_1, x_4) \geq 1 \end{array} \right. .$$

In Node(3.3) and (3.4) only the first and last constraints are non-redundant. In Node (3.3), preprocessing leads to redefining them as  $\lambda(\bar{x}_4, \bar{x}_2, \bar{x}_1) \geq 1$  and  $\lambda(x_4, x_2, x_1) \geq 1$ . While, in Node (3.4), they take the form  $\lambda(x_4, \bar{x}_2, \bar{x}_1) \geq 1$  and  $\lambda(\bar{x}_4, x_2, x_1) \geq 1$ . ■

Therefore, for set partitioning problems, the branching scheme takes a simpler form:

**Proposition 9** *When the master program is a set partitioning problem of the form (28), our branching scheme has the following properties (after preprocessing):*

(i)  $L^k = 1$  for all branching constraint  $k$  whose defining component sequence,  $S^k$ , includes a “setting-to-one bound”, i.e.  $x_i \in S^k$  for some  $i$ .

(ii) The depth of the tree of column classes is one and class  $Q$  is redundant, i.e., column classes  $Q^k$  define a partition of the column set  $Q$ . Each class  $Q^k$  has its own set of “setting-to-one” bounds,  $x_i$ , while the complementary bound,  $\bar{x}_i$ , is in the definition of all the other classes. One class is defined by “setting-to-zero” bounds only: say  $Q^1 = Q \setminus \cup_{k=2}^K Q^k = Q(\bar{x}_{[1]}, \dots, \bar{x}_{[l]})$  where  $x_{[j]} \in S^k$  for some  $k \in \{2, \dots, K\}$  for all  $j = 1, \dots, l$ . If  $L^1 = T - \sum_{k=2}^K L^k = 0$ ,  $Q^1$  is redundant and therefore omitted.

(iii) A fractional solution  $\tilde{\lambda}$  verifies all branching constraint at equality, i.e.  $\tilde{\lambda}(Q^k) = L^k$  for all  $k$ . Hence, eliminating it requires further splitting an existing column class.

(iv) When further splitting a column class that involves a setting-to-one-bound, only one more setting-to-one-bound is needed to identify a set  $S$  on which to branch and there are only two feasible successor nodes.

(v) When further splitting a class that involves “setting-to-zero” bounds only ( $Q^1$  in our notations), identifying a set  $S$  on which to branch requires at most two setting-to-one-bounds and there are at most three successor nodes. The third node (if any) is defined by a branching constraint of the form

$$\lambda(\bar{x}_{[1]}, \dots, \bar{x}_{[i]}) \geq T - K + 1$$

meaning that component  $[i]$  must be packed along side components  $[1], \dots, [i - 1]$  in one of the column subclasses  $Q^k$  for  $k = 2, \dots, K$ . This case could be partitioned into  $K - 1$  sub-cases according to which subclass  $Q^k$  is chosen.

**Proof:** (i) if  $x_i \in S^k$ , then  $\lambda(Q^k) \leq u_i = 1$ . Thus, the only feasible strictly positive bound is  $L^k = 1$ .

(ii) If  $x_i \in S^k$ , then, as  $L^k = u_i = 1$ ,  $x_i$  cannot be in any other class defining sequence, but  $\bar{x}_i$  can be added to all other class definition, including  $Q$  itself which give rise to class  $Q^1$ . The latter has no descendant nodes as all other classes involve setting-to-one bounds. A

class that involves a setting-to-one-bound cannot have a *non-redundant* super-class whose definition also involves setting-to-one-bounds because both would have bound  $L^k = 1$ .

(iii) In a fractional solution to the master, all column subclasses have integer cardinality because  $L^k$  is both a lower bound and an upper bound on the subclass (even for class  $Q^1$ ).

(iv) Assume  $\exists k : x_i \in S^k$  and  $|\Lambda^k| > 1$  where  $\Lambda^k = \{q \in S^k : \tilde{\lambda}_q - \lfloor \tilde{\lambda}_q \rfloor > 0\}$ . Then,  $\tilde{\lambda}(Q^k) = 1$  but, as all columns in  $\Lambda^k$  are different,  $\exists j : x_j \notin S^k$  such that  $0 < \tilde{\lambda}(S^k, x_j) < 1$ . Proposition 8 implies that only two successor nodes need to be defined.

(v) Say  $\tilde{\lambda}(Q^1) = L^1 > 0$ . Let  $I(Q^k) = \{i : \sum_{q \in Q^k} x_i^q (\tilde{\lambda}_q - \lfloor \tilde{\lambda}_q \rfloor) > 0\}$ . Consider two cases: either  $I(Q^1) \cap I(Q^k) = \emptyset$  for all  $k = 2, \dots, K$  or not. In the latter case, one could branch as in case (iv). But one can also split  $Q^1$ : a single setting-to-one-bound,  $x_i$ , suffices to define a set  $S = \langle S^1, x_i \rangle$  on which to branch as one can simply select some component  $i$  in  $I(Q^1) \cap I(Q^k)$  for some  $k$ . In the former case where  $I(Q^1) \cap I(Q^k) = \emptyset$  for all  $k = 2, \dots, K$ , we have  $\tilde{\lambda}(S^1, x_i) = 1$  for any  $i \in I(Q^1)$ . But, as all columns are different,  $\exists j \in I(Q^1) : 0 < \tilde{\lambda}(S^1, x_i, x_j) < 1$ . Choose  $S = \langle S^1, x_i, x_j \rangle$  for some pair  $(i, j)$ . By Proposition 8, descendant nodes  $1, \dots, |S| - 2$  need not be generated as they are enumerating solutions where  $\lambda(Q^1) < L^1$ . ■

With the above characterization, our branching scheme for set partitioning like problems can be compared to the specialized scheme of Ryan-and-Foster. Our scheme yields 2 or 3 successor nodes while, in Ryan-and-Foster's scheme, the branch-and-price tree is binary. In both schemes, columns are deleted after branching. In our scheme, the pricing problem is solved by enumerating on the active subproblems instead of solving a modified subproblem that can become intractable (because of the extra constraints resulting from branching). Our intermediate Lagrangian bounds (51) can lead to a stronger Lagrangian dual bound than under Ryan-and-Foster's scheme because we impose tighter restrictions on the pricing problem. This is illustrated in Example 8 and formalized in Proposition 10.

**Example 8** *Assume that we are at a branch-and-bound node defined under Ryan-and-Foster's scheme by branching constraints*

$$\lambda(x_1, x_2) \geq 1 \tag{55}$$

$$\lambda(x_2, x_3) \geq 1 \tag{56}$$

$$\lambda(x_4, x_5) \leq 0 \tag{57}$$

Constraints (55-56) are enforced by adding  $x_1 = x_2$  and  $x_2 = x_3$  in the subproblem while constraint (57) amounts to adding  $x_4 + x_5 \leq 1$  in the subproblem. The modified subproblem polyhedron is

$$\hat{X} = X \cap \{x : x_1 = x_2, x_2 = x_3, x_4 + x_5 \leq 1\} .$$

Note that constraint (57) is equivalent to enforcing  $\lambda(x_4, \bar{x}_5) \geq 1$  or symmetrically,  $\lambda(\bar{x}_4, x_5) \geq 1$ . Under our scheme the “equivalent” node problem could be defined by branching constraints:

$$\lambda(x_1, x_2, x_3, \bar{x}_4) \geq 1 \quad (58)$$

$$\lambda(\bar{x}_1, \bar{x}_2, \bar{x}_3, x_4, \bar{x}_5) \geq 1 \quad (59)$$

$$\lambda(\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4) \geq T - 2 \quad (60)$$

(assuming that successive branching sets were  $S^1 = \langle x_1, x_2 \rangle$ ,  $S^2 = \langle x_1, x_2, x_3 \rangle$ , and  $S^3 = \langle \bar{x}_1, \bar{x}_2, \bar{x}_3, x_4, x_5 \rangle$ ). Then, we would have 3 subproblems with respective polyhedron

$$X^1 = X \cap \{x : x_1 = x_2 = x_3 = 1, x_4 = 0\} ,$$

$$X^2 = X \cap \{x : x_1 = x_2 = x_3 = 0, x_4 = 1, x_5 = 0\} ,$$

$$X^3 = X \cap \{x : x_1 = x_2 = x_3 = x_4 = 0\} .$$

■

**Observation 4** Under Ryan-and-Foster’s scheme, intermediate dual bounds at a branch-and-price node take the form

$$\theta^{RF}(\pi) = \pi a + T \hat{\zeta}(\pi) \quad (61)$$

where  $\hat{\zeta}(\pi)$  is the solution of the modified subproblem and the Lagrangian dual bound has value equal to

$$\min\left\{\sum_t c x^t : \sum_t A x^t \geq a, x^t \in \text{conv}(\hat{X}) \text{ for } t = 1, \dots, T\right\}, \quad (62)$$

where  $\hat{X}$  is the polyhedron of the modified subproblem.

**Proposition 10** Assuming an “equivalent” set of branching constraints (as in the illustration of Example 8), intermediate bounds (51) dominate (61) and Lagrangian dual bound (52) dominates (62).

**Proof:** For all  $X^k$  used in the expression of (52), we have  $X^k \subseteq \hat{X}$  and therefore  $\hat{\zeta}(\pi) \leq \zeta^k(\pi)$  for all  $k$ . ■

## 10 Numerical Testing

The proposed branching scheme was partially implemented. None of the features of Sections 8 and 9 – preprocessing, further specification of column classes, deletion of columns – were implemented. Instead of using the column generation procedure described in Table 3 (based on a breadth first search enumeration of subproblems), all pricing subproblems associated with active constraints (38-39) are enumerated. The most negative reduced cost is used to compute dual bounds (14) instead of using (51). Hence, the current implementation amounts to dualizing branching constraints instead of convexifying them in the subproblem.

Tests have been carried on bin packing and cutting stock problems. The standard column generation formulation for the cutting stock problem is (25) where columns are solutions to the knapsack problem (26). The knapsack problem can be transformed in a binary problem with class bound [16]:

$$\max\left\{\sum_i \sum_{j=1}^{n_i} \pi_i 2^j x_{ij} : \sum_i \sum_{j=1}^{n_i} w_i 2^j x_{ij} \leq W, \sum_{j=1}^{n_i} 2^j x_{ij} \leq d_i \forall i, x_{ij} \in \{0, 1\} \forall ij\right\}$$

where  $n_i = \lfloor \log_2 \min\{d_i, \frac{w_i}{W}\} \rfloor + 1$ . Then, branching can be implemented by enforcing the integrality of the aggregate value of these 0-1 variables, using disjunctive constraints

$$\sum_{q \in Q} x_{ij}^q \lambda_q \leq \lfloor v \rfloor \quad \text{or} \quad \sum_{q \in Q} x_{ij}^q \lambda_q \geq \lceil v \rceil . \quad (63)$$

Such scheme does not theoretically allow to cut off every fractional solutions; however, in practice, it does on most instances [14]. In the pricing subproblem, only the cost of items are modified as a result (then,  $\pi_{ij} \neq \pi_i 2^j$ ). The 0-1 knapsack problems are solved using the branch-and-bound algorithm of [16] that can handle binary item cost  $\pi_{ij} \neq \pi_i 2^j$  and class bounds (the algorithm is a generalization of the standard branch-and-bound approach for the 0-1 knapsack of Horowitz and Sahni).

Three branching schemes are compared: that of [14] and the generic scheme of the present paper that is applied either to binary variables  $x_{ij}$  resulting from the 0-1 decomposition or directly to integer variables  $x_i$ . For the latter, we implemented the extension of the branching scheme to the non-binary case as described in footnotes along the paper. Then, the integer knapsack subproblems are also solved using the branch-and-bound algorithm of [16] after 0-1 transformation. In all tests, the master is initialized with a single

artificial column and no primal heuristics are used. We rely only on the branching scheme to produce integer solutions. After solving the root node, the cut  $\sum_{q \in Q} \lambda_q \geq \lceil Z_{LP}^M \rceil$  is added to the master. It plays the role of the convexity constraint  $\sum_{q \in Q} \lambda_q = T$ . Observe that  $T$  is not polynomial in the input size but it is polynomial in the output size (the solution file may involve  $T$  different patterns). We set the incumbent value to  $\lceil Z_{LP}^M \rceil + 1$  for use as a cut-off value (CSP instances whose solution has value larger than  $\lceil Z_{LP}^M \rceil$  are rare).

Two sets of 10 instances are used for these comparisons (there are the first two sets of test problems described in Table 1 in [14]). The first set are industrial instances with 7 to 43 items (but demands are small in larger instances); the second set is made of randomly generated instances with 20 items and average demand of 50. In all these tests, when selecting variables for branching, priority is given to the most fractional value weighted by the width of the item. The branch-and-price tree is searched using depth first priority.

Average results over the 10 instances are presented in Table 5. The columns provide the number of nodes (nod) in the branch-and-price tree, its depth (dep), the best IP solution (inc), the best dual bound (Db), the number of pricing problem solved (#Sp), the number of columns generated (#Col), the time spent solving the master (tmast) with Xpress-mp LP solver, the time spent in the pricing problem oracle (tsp), the time spent for separation in subroutine Explore (tex) and the overall time (which includes a lot of overhead for debugging tests such as checking whether a column was previously generated). Time units are ticks ( $\frac{1}{100}$  of seconds) on a PC 1GHz. For each set of test problems, we compare branching on aggregate value of  $x_{ij}$  variables using (63) (AG), branching on sets of  $x_{ij}$  components (CS( $x_{ij}$ )), branching on sets of  $x_i$  components (CS( $x_i$ )), and an hybrid approach defined below (HYB). When using AG branching, the number of instances (out of 10) that were solved to integrality is reported in column (tex) in bold face.

When branching rule AG does not allow to cut-off a fractional solution, the algorithm stops with the cut-off value  $\lceil Z_{LP}^M \rceil + 1$  as incumbent. Then, the counters of the partial resolution are counted in the average. This happened for 4 (respectively 1) out of the 10 industrial (resp. random) instances. (Hence, the comparison between AG and CS is biased by the fact that under AG the hardest problems have not been solved to optimality.) In reference [14], branching rules (63) are completed by more complex rules presented in [15] to achieve integrality but the latter lead to pricing problem modifications. However, the overall scheme was shown to out-perform previously proposed schemes. Note that the

problem	branch	nod	dep	inc	Db	#Sp	#Col	tmast	tsp	tex	time
indust	AG	46	39	77.5	77.1	125	126	36	155	<b>6</b>	473
instanc.	CS( $x_{ij}$ )	59	51	77.1	77.1	150	149	51	264	20	799
$n =$	CS( $x_i$ )	66	47	77.1	77.1	180	171	81	424	25	1407
7 to 43	HYB	53	51	77.1	77.1	145	145	50	249	22	753
random	AG	48	41	217.8	217.7	83	84	15	49	<b>9</b>	234
instanc.	CS( $x_{ij}$ )	54	48	217.7	217.7	91	91	20	85	9	354
$n = 20$	CS( $x_i$ )	59	54	217.7	217.7	96	96	19	104	7	313
$\bar{d} = 50$	HYB	48	43	217.7	217.7	85	85	14	74	6	286

Table 5: Comparative results for Cutting Stock Problems

generic scheme of this paper allows to achieve integrality within roughly the same size of branch-and-bound tree while using only the oracle for the original pricing problem. The time spent in subproblems under CS could be reduced by applying the breadth first search enumeration of subproblems. When working with integer variables  $x_i$ , the mapping of Section 3 sometimes allows to detect integrality while  $\lambda$  is still fractional. We observed that branching mode CS( $x_i$ ) can lead to more unbalanced tree than branching mode CS( $x_{ij}$ ) but this is compensated in part by the early detection of integer solution through the mapping of Section 3. Hence, a hybrid version (HYB) was tested where we branch using CS( $x_{ij}$ ) but test integrality before branching by applying the mapping for variables  $x_i$ .

Bin packing problem instances that are used for numerical tests in the literature are typically randomly generated with 500 to 1000 items with integer width drawn uniformly in interval  $[1, 100]$  or smaller. This yields many identical items that must be aggregated to avoid symmetry. Hence, those instances are truly cutting stock problem with low demands. Our numerical results of Table 6 show that as item demand gets low, branching rule AG does not suffice to achieve integrality on most instances. We generated instances randomly with 100 items whose width is drawn in  $[\frac{W}{4}, \frac{W}{2}]$  or  $[\frac{W}{10}, \frac{W}{2}]$  for a bin capacity  $W = 10000$ . All items have different width. Their demand is generated in  $[1, d_{\max}]$  where  $d_{\max} = 1, 2$ , or  $3$ . For  $d_{\max} = 3$ , AG fails to allow separation at some node for 8 out of 10 instances. For  $d_{\max} = 2$ , AG fails in all cases (and obviously so for  $d_{\max} = 1$ ). In such case, before this paper, one had no other means than using a branching scheme leading to structure modification in the subproblem, as that of Ryan-and-Foster, or working in an extended space of pseudo-polynomial size, as in [13] (which becomes problematic when

$W$  gets large).

problem	br	nod	dep	inc	Db	#Sp	#Col	tmast	tsp	tex	time
$w_i \in [\frac{W}{4}, \frac{W}{2}], d_i \leq 3$	AG	47	31	80.2	79.4	344	341	107	1433	<b>2</b>	2313
$w_i \in [\frac{W}{4}, \frac{W}{2}], d_i \leq 3$	CS	94	74	79.4	79.4	413	402	275	2688	64	7237
$w_i \in [\frac{W}{4}, \frac{W}{2}], d_i \leq 2$	AG	16	14	59.5	58.5	331	331	78	1145	<b>0</b>	1740
$w_i \in [\frac{W}{4}, \frac{W}{2}], d_i \leq 2$	CS	34	32	58.5	58.5	402	402	138	1790	14	2859
$w_i \in [\frac{W}{4}, \frac{W}{2}], d_i = 1$	CS	38	37	39.8	39.8	528	529	356	1042	27	2817
$w_i \in [\frac{W}{10}, \frac{W}{2}], d_i = 1$	CS	65	64	30.1	30.1	1024	1025	882	3581	53	10503

Table 6: Comparative results for random Bin Packing Problems with 100 items and  $W = 10000$  (results are average over 10 instances).

## Conclusion

This paper presents a generic branching scheme for use in branch-and-price. Fractional solutions are cut-off by partitioning the set of columns into nested subclasses. (This can be viewed as implementing branching for the special ordered set (SOS) defined by the convexity constraint of the master reformulation.) Each enumerated branch enforces a lower bound on the number of columns to be selected in a subclass. A subclass is defined by setting restrictive bounds on some variables of the pricing problem. Hence, the pricing problem decomposes into independent subproblems associated with each column subclass, each of which can be solved with the oracle provided for the pricing problem of the root node.

The number of active branching constraints and associated subproblems is bounded by  $T$  (the number of blocks in the original formulation). Moreover, the nested definition of column subclasses leads to a natural tree representation of active subproblems. (Branching constraints at a branch-and-price node yield a branch-and-bound tree for the pricing problem.) By enumerating pricing subproblems in a breadth first search order, one can interrupt the column generation iterate as soon as a negative reduced cost column is found and still get a dual bound. The branching scheme is shown to be strong both in terms of achieving integrality (as each branching constraint amounts to fixing some variable in the original formulation) and in terms of improving dual bounds (the Lagrangian dual bounds

are shown to be equivalent to solving a LP where both sets of subproblem constraints and branching constraints are convexified).

The present scheme unifies previous work on branching in branch-and-price algorithm. It is a specific implementation of the scheme of [15]. It can be seen as an implicit way to implement the scheme of Villeneuve et al. [19]: instead of defining  $T$  subproblems a priori, there are introduced dynamically in the course of branching and they do not get a specific index so as to avoid the symmetry drawback. When faced with a set partitioning problem, the present scheme is close to reproducing the scheme of Ryan-and-Foster, but in an implementation that does not require an oracle for modified subproblems and that can yield better dual bounds.

The paper places the branching issue into context, showing when branching can be trivially implemented using disjunction on the variables of the original formulation and when it cannot. Observe that when our scheme is applied to a problem with non-identical subsystems (case 1), it reproduces the standard scheme of branching on the variables of the original formulation. One must keep in mind that we have focused on default branching that guarantees integrality. In practice, more aggregate branching constraints that yield a more balanced tree shall typically be used first (e.g., using (24) for a VRP), while the default scheme is used to finish-off the search for integer solutions. In particular, in case 1, when the different subsystems involve the same variables (e.g., take a CSP with different stock pieces), our scheme can be applied across subsystems using branching constraint of the form  $\sum_t \sum_{q \in Q^t(S)} \lambda_q^t \geq L$ .

Preliminary computational testing on bin packing and cutting stock problems show that the proposed scheme is a practical way to get to integrality while never modifying the structure of the pricing problem and avoiding symmetry. Observe that every previous approaches for these problems required either to modify the pricing problem or to work in an extended variable space (the 0-1 knapsack with class bounds in [14], or the arc flow variables of a dynamic program for the integer knapsack in [13] or a resource constrained shortest path in [2]). Thus, this is the first scheme able to solve bin packing and cutting stock problems while staying with the original variables and not requiring any specific solver for the sub-problem (for  $CS(x_i)$ , any knapsack solver can be used). These computations also illustrate that the scheme generalizes to the case of a non-binary integer problem. The extension to a mixed integer program shall combine the above with the

specificity of the sufficient integrality condition developed in [18].

## Acknowledgments

We are grateful to Ph. Meurdesoif and P. Pesneau for their comments on the first draft of this paper.

## References

- [1] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, P.H. Vance (1998). Branch-and-Price: Column Generation for Huge Integer Programs. *Oper. Res.*, 46:316-329.
- [2] G. Belov, A.N. Letchford, and E. Uchoa (2005). A Node-Flow Model for the 1D Stock Cutting: Robust Branch-Cut-and-Price. Working paper available at <http://www.lancs.ac.uk/staff/letchfoa/csp.pdf>.
- [3] O. Briant, C. Lemaréchal, Ph. Meurdesoif, S. Michel, N. Perrot and F. Vanderbeck. Comparison of Bundle and Classical Column Generation. Rapport de recherche INRIA, 5453, janvier 2005.
- [4] A. Chabrier, (2003). Génération de Colonnes et de Coupes utilisant des sous-problèmes de plus court chemin. *Thèse de doctorat*, Université d'Angers.
- [5] Claude Berge (1973). Graphs and Hypergraphs, North-Holland Pub. Co.
- [6] G. Desaulniers, J. Desrosiers, A. Langevin, O. Marcotte, G. Savard, F. Soumis and M. Solomon. GENCOL, a mathematical optimizer (<http://www.crt.umontreal.ca/~gencol/gceng.html>).
- [7] A. Geoffrion (1974). Lagrangian relaxation for integer programming. *Math Prog. Study*, 2:82-114.
- [8] L. Ladányi, T.K.Ralphs and L.E.Trotter, Jr (1998). Branch, Cut, and Price: Sequential and Parallel. *Computational Combinatorial Optimization*, M. Junger and D. Naddef (Eds), Springer.

- [9] R. Fukasawa, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa and R.F. Werneck (2004). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Proceedings of IPCO'04, New York, Lecture Notes in Computer Science, Vol.3064 pp.1-15.
- [10] Ryan, D. M. and B. A. Foster (1981). An integer programming approach to scheduling. In A. Wren (editor), *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, North-Holland, Amsterdam, 269-280.
- [11] M.W.P.Savelsbergh and G.L.Nemhauser (1993). *Functional description of MINTO, a Mixed INTEger Optimizer*. Report COC-91-03A, , Georgia Institute of Technology, Atlanta, Georgia.
- [12] S. Thienel (1995). ABACUS - A Branch-And-CUt System. *Doctoral Thesis*, Universität zu Köln.
- [13] J. Valerio de Carvalho (1999). Exact solution of bin packing problems using column generation and branch-and-bound. *Ann. Oper. Res.*, 86:629-659.
- [14] F. Vanderbeck (1999). Computational Study of a Column Generation algorithm for Bin Packing and Cutting Stock Problems. *Math. Prog.*, 86, 565-594.
- [15] F. Vanderbeck (2000). On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Oper. Res.*, 48(1):111-128.
- [16] F. Vanderbeck (2002). Extending Dantzig's Bound to the Bounded Multi-Class Binary Knapsack Problem. *Math. Prog.*, 94(1), pp 125-136.
- [17] F. Vanderbeck (2003). Automated Dantzig-Wolfe re-formulation or how to exploit simultaneously original formulation and column generation re-formulation. *Working Paper*, no U-03.24, MAB, Université Bordeaux 1.
- [18] F. Vanderbeck and M.W.P. Savelsbergh (2005). A Generic View of Dantzig-Wolfe Decomposition in Mixed Integer Programming. To appear in *Oper. Res. Let.*
- [19] Villeneuve D., J. Desrosiers, M.E. Lübbecke and F. Soumis (2005). On Compact Formulations for Integer Programs Solved by Column Generation. , *Ann. Oper. Res.*, 139(1), 375-388.