Math. Program., Ser. A (2011) 130:249–294
DOI 10.1007/s10107-009-0334-1

FULL LENGTH PAPER

Branching in branch-and-price: a generic scheme

François Vanderbeck

Received: 24 November 2005 / Accepted: 12 November 2009 / Published online: 5 January 2010

Abstract Developing a branching scheme that is compatible with the column generation procedure can be challenging. Application specific and generic schemes have been proposed in the literature, but they have their drawbacks. One generic scheme is to implement standard branching in the space of the compact formulation to which the Dantzig-Wolfe reformulation was applied. However, in the presence of multiple identical subsystems, the mapping to the original variable space typically induces symmetries. An alternative, in an application specific context, can be to expand the compact formulation to offer a wider choice of branching variables. Other existing generic schemes for use in branch-and-price imply modifications to the pricing problem. This is a concern because the pricing oracle on which the method relies might become obsolete beyond the root node. This paper presents a generic branching scheme in which the pricing oracle of the root node remains of use after branching (assuming that the pricing oracle can handle bounds on the subproblem variables). The scheme does not require the use of an extended formulation of the original problem. It proceeds by recursively partitioning the subproblem solution set. Branching constraints are enforced in the pricing problem instead of being dualized via Lagrangian relaxation, and the pricing problem is solved by a limited number of calls to the pricing oracle. This generic scheme builds on previously proposed approaches and unifies them. We illustrate its use on the cutting stock and bin packing problems. This is the first branch-and-price algorithm capable of solving such problems to integrality without modifying the subproblem or expanding its variable space.

Keywords Integer programming · Dantzig–Wolfe reformulation · Branch-and-price

Mathematics Subject Classification (2000) 90

F. Vanderbeck (✉)
University Bordeaux 1 (Institut de Mathématiques) & Inria Bordeaux Sud-Ouest,
33405 Talence Cedex, France
e-mail: fv@math.u-bordeaux1.fr

1 Introduction

Many mixed integer programming problems have a decomposable structure, which makes them well suited for Dantzig–Wolfe reformulation and for which Branch-and-price can be a competitive solution approach. However, branching can be challenging. One needs a scheme that not only has good properties in terms of leading to integrality of the solution and yielding dual bound improvements, but that is also "compatible" with column generation. 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 competitiveness of 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 explored in full detail to date, because, for many practical applications, branching is straightforward. In particular, branching can be implemented in a standard way in the space of the compact formulation to which the Dantzig–Wolfe reformulation was applied and this implementation is competitive in many situations. Villeneuve et al. [26] 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 typically induces symmetries. As an alternative, specialized branching rules for use in Branch-and-price have been developed. Ryan and Foster [14] proposed a scheme for applications that can be reformulated as set partitioning problems, a generalization of which was developed by Vanderbeck [20]. However, both of these specialized schemes may result in structural modifications to the pricing problem. Another line of branching schemes reported in the literature can be understood as implicitly using an extended ("original") formulation and branch on the new variables of the reformulation (as in Belov et al. [3] or Valério de Carvalho [17]). Such approach is application specific and requires to a pricing problem solver that works in this expanded variable space.

The scheme proposed in this paper builds on and unifies previously proposed generic approaches, while avoiding their drawbacks. It can be seen as special case of the implementation of the scheme of [20] in which fractional solutions are cut off by bounding the number of columns selected from specific subsets. This particular implementation permits the use of the original pricing problem oracle after branching, as does the scheme of [26]. The new scheme can also be understood as a refinement of the scheme of [26]: the branching constraints of the new scheme are shown to implicitly fix bounds on the variables of the original formulation, while [26] implements explicit bounds on the original variables. This refinement allows us to avoid the symmetry drawback of the scheme [26] in the case of multiple identical subproblems, as we will show. In the new scheme, as in that of [26], branching constraints can be enforced directly in the subproblem if the pricing oracle can handle bounds on the subproblem variables. Since branching constraints are not dualized (i.e., not placed in the master as in [20]), they induce better improvements of dual bounds. Finally, when the master is a set partitioning

problem, the proposed scheme resembles that of [14], but with a different pricing procedure.

A motivation for this work is the development of a generic code for branch-and-price. Previously existing framework for implementing branch-and-price such as Abacus [16], BCP [11], G12 [13], or Minto [15] are tool boxes that leave it to the user to implement an application specific branching scheme. Other existing codes have been developed for specific classes of applications (such as the vehicle routing problem and its variants); examples include Gencol [6] and Maestro [5]. In these references, the branching issue has been seen so far as a barrier to the automation of branch-and-price. The branching scheme of this paper allows the overcoming of this barrier. The proposed scheme is valid for any column generation application; the only requirement is a pricing oracle that can handle upper and lower bounds on the subproblem variables. The scheme requires no input from the user since the same pricing problem solver can be used after branching. Hence, the scheme leads to a possible black box implementation of branch-and-price. We are currently working on a prototype of a generic branch-and-price implementation, called *BaPCod* [24].

The presentation given herein is not limited to the principles underlying the new scheme. As the implementation of the proposed scheme is not trivial, the paper makes specific proposals regarding the separation of fractional solutions and the pricing procedure after branching. An analysis of the scheme properties allows us to show that the worst case complexity of the enumeration is no worse than when branching in the original variable space.

The paper is organized as follows. Section 2 reviews the Dantzig–Wolfe reformulation principle. Section 3 provides an overview of the new scheme, explaining its principle and introducing the features needed for its implementation. A formal presentation follows for the special case of a binary integer program in Sects. 4–10, detailing a mapping from the Dantzig–Wolfe reformulation to the original problem variable space; presenting how to branch at the root node and beyond, and how column generation is implemented after branching; giving the expression of the resulting dual bound and showing how preprocessing yields simplifications, specifically in the special case where the master is a set partitioning problem. In Sect. 11, we briefly explain how the scheme can be extended to a general mixed integer program. Section 12 presents computational experiments on cutting stock and bin packing problems. The conclusion summarizes the scheme, its contributions, and its limitations.

2 The Dantzig–Wolfe approach

Let us introduce our notation and briefly review the Dantzig–Wolfe approach: the decomposition of a mixed integer program, its reformulation, the column generation procedure, and Lagrangian duality results. To simplify the presentation, we assume a pure integer program (IP) whose variables are bounded. The extension to the unbounded case is presented in [20], while the extension to the mixed integer case is presented in [25]. We consider a well structured IP whose constraint matrix is of the form

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

where A^r and B^r, for $r = 1, \dots, R$, are rational matrices. I.e., there are R diagonal blocks and the problem can be formulated as

$$Z^P = \min \sum_{r=1}^{R} c^r \, x^r \tag{2}$$

$$[\mathrm{P}] \quad \sum_{r=1}^{R} A^r \, x^r \geq a \tag{3}$$

$$B^r \, x^r \geq b^r \quad \text{for } r = 1, \dots, R \tag{4}$$

$$l^r \leq x^r \leq u^r \quad \text{for } r = 1, \dots, R \tag{5}$$

$$x^r \in \mathbb{Z}^n \quad \text{for } r = 1, \dots, R \tag{6}$$

where c^r, a, and b^r are rational vectors of appropriate dimension and l^r (resp. u^r) are lower (resp. upper) bound vectors. Let P_{LP} denote its linear programming (LP) relaxation and Z^P_{LP} its optimal LP value. A subsystem

$$X^r \equiv \{x \in \mathbb{Z}^n : B^r \, x \geq b^r, l^r \leq x \leq u^r\} \tag{7}$$

can be associated with each block r. Let X^r_{LP} be the polyhedron associated to its LP relaxation.

Throughout the paper, we shall distinguish two cases. Either we have *non-identical subsystems*: A^r and the data of subsystems X^r depend on r. Or the *subsystems are identical*: A^r and the data of subsystems X^r do not depend on r. The combined case could also occur for which matrix B is made of non-identical blocks, each of which decomposes into identical sub-blocks: an example would be a Vehicle Routing Problem (VRP) with different vehicle types and several vehicles of each type.

The Dantzig–Wolfe reformulation of problem $[P]$ can be introduced in several ways. For this paper, we adopt the discretization approach of [25]. Let G^r be an enumerated set of generators (a terminology introduced in [25]) for subsystem X^r. Since we assume a bounded and pure integer program, G^r is simply the enumerated set of all discrete integer solutions of X^r, i.e., $X^r = \{x^g\}_{g \in G^r}$, where x^g is the solution vector associated to generator g, and X^r can be reformulated as

$$X^r \equiv \left\{ x = \sum_{g \in G^r} x^g \lambda_g : \sum_{g \in G^r} \lambda_g = 1, \lambda_g \in \{0, 1\} \; \forall g \in G^r \right\}.$$

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

$$
\begin{aligned}
Z^{\mathrm{DM}} &= \min \sum_{r=1}^{R} \sum_{g \in G^r} c^r\, x^g\, \lambda_g^r \\
[DM] \quad & \sum_{r=1}^{R} \sum_{g \in G^r} A^r\, x^g\, \lambda_g^r \geq a \qquad (8) \\
& \sum_{g \in G^r} \lambda_g^r = 1 \quad \text{for } r = 1, \ldots, R \\
& \lambda_g^r \in \{0, 1\} \quad \text{for } r = 1, \ldots, R,\ g \in G^r .
\end{aligned}
$$

Thus, in our notation, g denotes the generator (g is also used as an index referring to the generator), x^g denotes its characteristic vector, $(c^r\ x^g,\ A^r\ x^g)$ is the associated column vector in formulation [DM], and λ_g^r is the associated decision variable. For simplicity, we shall commonly refer to g as a column in the sequel. Let $\mathrm{DM}_{\mathrm{LP}}$ denote the LP relaxation of [DM] and $Z_{\mathrm{LP}}^{\mathrm{DM}}$ be its optimal LP value.

In the case where all subsystems are identical, i.e., $A^r = A$, $B^r = B$, $c^r = c$, $X^r = X$ and $G^r = G$ for $r = 1, \ldots, R$, one can aggregate λ_g^r variables using:

$$
\nu_g = \sum_{r=1}^{R} \lambda_g^r . \qquad (9)
$$

Then, the Dantzig–Wolfe reformulation takes the form:

$$
\begin{aligned}
Z^M &= \min \sum_{g \in G} c\, x^g\, \nu_g && (10) \\
[\mathrm{M}] \quad & \sum_{g \in G} A\, x^g\, \nu_g \geq a && (11) \\
& \sum_{g \in G} \nu_g = R && (12) \\
& \nu_g \in \mathbb{N} \quad \forall g \in G && (13)
\end{aligned}
$$

Let M_{LP} denote its LP relaxation and Z_{LP}^M be its optimal LP value. Observe that the aggregation (9) removes the symmetry in r that is present in the original formulation [P] (where a permutation of the r indexing in vector x gives rise to an alternative representation of the same solution) or in [DM]. We assume the equality convexity constraint (12) although considering convexity constraints of the form $L \leq \sum_{g \in G} \nu_g \leq U$ is a straightforward extension.

The enumeration of set G (or G^r in the case of non-identical subsystems) 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], given in (8), as the disaggregated master used in the case of non-identical subsystems, while [M] is the commonly used form of the master in the case of identical subsystems.

Thus, solving the LP relaxation of [M] by column generation proceeds as follows. (The LP solution of [DM] is analogous.) The master LP is initialized with a subset of (possibly artificial) columns. This restricted program is solved to LP optimality. Let π be the dual solution vector associated with constraints (11). Then, a *pricing problem* is solved:

$$\zeta(\pi) := \min\{(c - \pi A)x : \underbrace{B\,x \geq b, l \leq x \leq u, x \in \mathbb{Z}^n}_{x \in X}\}. \tag{14}$$

when its solution, $x^g := x^*$, defines a negative reduced cost column, the column $(c\,x^g, A\,x^g)$ and associated v_g variable are added to the master and the procedure reiterates. Otherwise, the current LP solution has been proven optimal.

The principal assumption underlying the Dantzig–Wolfe approach is that (14) is a problem that can be solved rather efficiently, compared to [P]. Throughout this paper, we assume that a solver is available for (14) that requires reasonable computing time (although typically not polynomial time). It can be a specialized combinatorial algorithm or one might even use a general purpose commercial MIP solver. We refer to this solver as the *oracle*. Observe that we have included bounds on the variables in the definition of the pricing problem, as our branching scheme proceeds by amending these bounds. In some applications, the bounded version of the pricing problem can be harder (complexity wise) than the unbounded case. On the other hand, considering a bounded pricing problem often yields a stronger dual bound Z^M_{LP} or $Z^{\mathrm{DM}}_{\mathrm{LP}}$ (see [23]).

At each iteration of the column generation procedure, a dual bound can be computed from the pricing problem solutions: dualizing (11) in [M] gives rise to the Lagrangian dual bound

$$\theta(\pi) := \pi\, a + R\, \zeta(\pi), \tag{15}$$

where $\zeta(\pi)$ is defined by (14). These bounds, $\theta(\pi)$, are computed for each dual solution, π, to the restricted master LP. They converge (although not monotonically) towards the optimal value of the master LP. The latter is known to be equivalent to the Lagrangian dual that results from dualizing constraints (3) in [P] (see [9]), i.e.,

$$Z^M_{\mathrm{LP}} = \max_{\pi \geq 0} \theta(\pi) = \min\left\{ \sum_r c\,x^r : \sum_r A\,x^r \geq a,\ x^r \in \mathrm{conv}(X^r)\ \forall r \right\}, \tag{16}$$

where the third form is the Lagrangian bi-dual (see [4]), reminding us that the Dantzig–Wolfe approach produces a bound equal to the LP solution over a polyhedron where the subproblem is "convexified". Thus, the comparison of the LP and IP values of the above formulations is $Z^P_{\mathrm{LP}} \leq Z^M_{\mathrm{LP}} = Z^{\mathrm{DM}}_{\mathrm{LP}} \leq Z^M = Z^{\mathrm{DM}} = Z^P$. When $\mathrm{conv}(X^r) \neq X^r_{\mathrm{LP}}$, the master gives rise to an LP bound that can be better than that of [P].

Thus, Dantzig–Wolfe reformulation allows a solution approach that exploits the availability of an efficient specialized oracle for a subproblem; it avoids symmetry in r in the case of identical subsystems; and it often leads to better quality dual bounds. To apply a branch-and-bound approach based on the Dantzig–Wolfe reformulation, one needs to embed a column generation procedure into a branch-and-bound algorithm. The combined algorithm is known as *branch-and-price* [2]. This raises several issues: (i) the choice of branching on variables of the original formulation or those of the reformulation, or branching on constraints, and the equivalence that may or may not exist between these approaches; (ii) where to enforce the branching constraint (in the master or in the subproblem) and the impact on the strength of resulting Lagrangian dual bound after branching; (iii) potential structural modifications to the pricing problem that may only be compatible with specific solution methods or may make it much harder to perform pricing.

Branching directly on individual variables v_g (resp. λ_g^r), using disjunctive constraints $v_g \leq \lfloor v \rfloor$ or $v_g \geq \lceil v \rceil$, was tested, for instance, by Refs. [12,18]. The resulting branch-and-bound tree is unbalanced (constraint $v_g \leq \lfloor v \rfloor$ is weak) and it combines badly with column generation (in the branch $v_g \leq \lfloor v \rfloor$, one must avoid regenerating the specific column $g \in G$ either by adding constraints to the pricing problem or by looking for the next best subproblem solution). The alternative is to branch on constraints.

A natural combination of variables v_g (resp. λ_g^r) on which to branch is that expressing the value of the variables of the original formulation [P]: mapping v (resp. λ) solutions into x solutions and implementing a branching scheme based on disjunctive constraints for the original variables. In the case of non-identical subsystems, this scheme will suffice to enforce integrality since the mapping

$$x^r = \sum_{g \in G^r} x^g \, \lambda_g^r, \tag{17}$$

defines a unique projection to the original variable space and disjunctive branching constraints of the form $\sum_{g \in G^r} x_i^g \, \lambda_g^r \leq \lfloor \alpha \rfloor$ or $\sum_{g \in G^r} x_i^g \, \lambda_g^r \geq \lceil \alpha \rceil$ in [$\mathrm{DM_{LP}}$] are therefore equivalent to enforcing

$$x_i^r \leq \lfloor \alpha \rfloor \quad \text{or} \quad x_i^r \geq \lceil \alpha \rceil \,, \tag{18}$$

in the original problem for each subproblem r and component i that would have fractional value $\alpha \notin \mathbb{Z}$ (see [2]). Branching can then be enforced directly in the pricing problem: one simply needs to reset a component bound in the subproblem.

However, in the case of identical subsystems, one is working with reformulation [M]. Then, the disaggregated original variable values, x^r, are not available through a uniquely defined mapping. Note that using [DM] even in the case of identical subsystems would allow one to branch on the original variables but it has an obvious symmetry drawback due to the artificial re-introduction of the r indices on identical

subsystems. With $[M]$, one can only enforce the integrality of aggregate variables

$$x = \sum_{r=1}^{R} x^r = \sum_{g \in G} x^g \; \nu_g, \tag{19}$$

using disjunctive branching constraints of the form

$$\sum_{g \in G} x_i^g \; \nu_g \leq \lfloor \alpha \rfloor \quad \text{or} \quad \sum_{g \in G} x_i^g \; \nu_g \geq \lceil \alpha \rceil, \tag{20}$$

for components i that have fractional value $\alpha \notin \mathbb{Z}$. But branching constraints on aggregate variables are typically not sufficient to eliminate all fractional solutions, and they cannot in general be enforced directly in the pricing problem. In the literature, alternative "original formulations" have sometimes been considered to allow branching on the aggregate value of original variables. Such reformulations typically imply an expanded variable space which can even be of pseudo-polynomial size [3,17].

Other forms of branching-on-constraint strategies have been developed to handle the case of identical subsystems. When the master takes the form of a set partitioning problem, i.e.,

$$\min \left\{ \sum_{g \in G} c \, x^g \; \nu_g : \sum_{g \in G} x_i^g \; \nu_g = 1 \; \forall i, \; \sum_{g \in G} \nu_g = R, \; \nu_g \in \{0,1\} \; \forall g \in G \right\} \tag{21}$$

with $x^g \in \{0,1\}^n$, integrality can be enforced using Ryan and Foster's scheme [14]. For any fractional solution, $\tilde{\nu} \notin \{0,1\}^{|G|}$, there exists a pair of items (i, j) such that the fractional solution $\tilde{\nu}$ can be separated using the disjunction $\sum_{g : x_i^g = x_j^g = 1} \nu_g \leq 0$ or $\sum_{g : x_i^g = x_j^g = 1} \nu_g \geq 1$. In the first branch, as no column that has both $x_i^g = 1$ and $x_j^g = 1$ 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 $x_i^g = x_j^g = 1$, the pricing problem is augmented with constraint $x_i = x_j$. The modifications to the pricing problem may change its structure.

Vanderbeck [20] 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 $\hat{G} \subset G$ and enforcing $\sum_{g \in \hat{G}} \nu_g \in \mathbb{Z}$ through disjunctive branching constraints of the form:

$$\sum_{g \in \hat{G}} \nu_g \leq L - 1 \quad \text{or} \quad \sum_{g \in \hat{G}} \nu_g \geq L \tag{22}$$

for $L \in \mathbb{N}$. In practice, sets $\hat{G}$ are defined as subsets of columns whose vector x^g satisfy prescribed bounds on some of its components: $\hat{G} = G \cap \{sx \geq l\}$, where $l \in \mathbb{Z}^n$ is a vector of bounds and $s \in \{-1, 1\}^n$ defines the sign of each component bound. Then the pricing problem must be modified to account for the dual variable

associated with branching constraint (22) for the columns of $\hat{G}$ and not for those of $G \backslash \hat{G}$.

3 Overview of the proposed scheme

The branching scheme of this paper addresses the case of identical subsystems: the master program is $[M]$ given by (10–13) and branching on the aggregate value of the original variables does not suffice to achieve integrality. The complete generic scheme consists in using first branching disjunctions of the form (20) to achieve integrality of the aggregate value of the "original" variables, and then to apply the scheme proposed herein to finalise the elimination of fractional solution if needed. This complete scheme can serve as a default branching for a generic branch-and-price solver: an informed user might want to use application specific branching rules (such as branching on constraints) before using the default scheme. Applying the new scheme to the case of non-identical subsystems amounts to reproducing the standard branching scheme (18).

The idea underlying the new scheme is to return to the non-identical subsystem situation, by making use of a progressive *subproblem differentiation*, introducing new subsystems dynamically. Using formulation [DM] in the case of identical subsystems amounts to an "a priori" subproblem differentiation, but it induces symmetry. Instead, one can differentiate the subsystem progressively in the course of branching. Viewing the aggregate convexity constraint (12) as a "special ordered set" (SOS), one can branch by progressively partitioning subsystem X, or equivalently the generator set G, and enforcing separate convexity constraints on each subset. A natural scheme is to partition subsystem X by imposing disjunctive constraints on the integer subproblem variables $x_i \in \mathbb{Z}$: a generator subset, $\hat{G} \subseteq G$, which we call a "*column class*", is defined in terms of bound restrictions on some components of x^g. Then, one can do the pricing over $\hat{G}$ using the oracle for (14). To allow us to price columns from each individual column class, $\hat{G}$, independently, we need to implement further branching by partitioning previously defined subsystems, i.e. a nested partition scheme is required.

The proposed scheme can be seen as a specific implementation of the scheme of Ref. [20]: instead of selecting sets, $\hat{G}$, arbitrarily (as illustrated on the left part of Fig. 1), they are defined so as to form a nested partition of the pricing problem solution set (as illustrated on the right part of Fig. 1). Moreover, instead of modifying the pricing problem by introducing indicator variables associated with each column class $\hat{G}$, we solve separate pricing subproblems. Thus, branching constraints are enforced directly in the pricing problem through this enumeration procedure.

To guarantee that the number of newly introduced subproblems is bounded by R, the number of diagonal blocks in (1), we use only branching constraints that enforce lower bound on column classes:

$$\sum_{g \in \hat{G}} v_g \geq L, \tag{23}$$

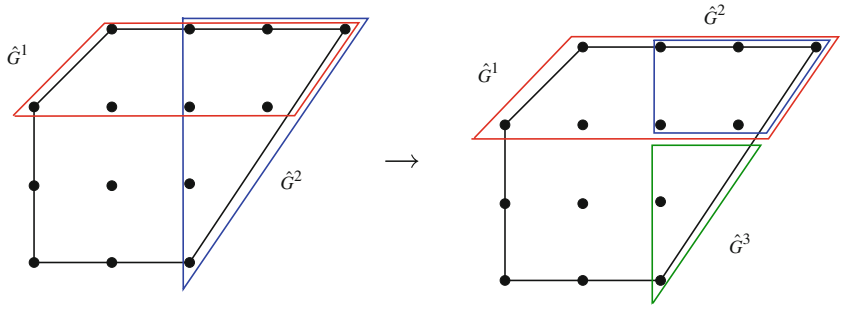

Fig. 1 Partitioning of the pricing problem solution set: using the scheme of Ref. [20] on the *left*, and the nested partition of the new scheme on the *right*

with $L \geq 1$. By the pigeon hole principle, there cannot be more than R such constraints that are active. This is implemented using a non binary enumeration tree: the branch $\sum_{g\in\hat{G}} \nu_g \leq U$ is replaced by enumerating ways in which $\sum_{g\in G\setminus\hat{G}} \nu_g \geq R - U$. We show that each branching constraint of the form (23) induces progress in reaching primal integrality that can be measured in the compact space of the original variables: each branching constraint implicitly fixes a bound on at least one original variable. To establish such equivalence we consider the projection of the master solution in the original formulation. This shows that integrality of the solution must be achieved after adding a polynomial number of branching constraints (assuming fixed bounds on the original variables), even though the number of column classes on which one could branch is exponential.

The dynamic introduction of differentiated pricing subproblems is similar to what is done in the scheme of Villeneuve et al. [26] in its dynamic implementation where pricing subproblems are introduced as needed. Our proposal goes further by providing a practical way of iteratively refining the column class definitions to separate fractional solutions. But, more importantly, our scheme differs by the fact that branching constraints concern groups of subproblems instead of one at the time. Column classes, $\hat{G}$, on which we branch expand over several subsets G^r of formulation [DM] given in (8) but with no explicit reference to any r index. The link between master and original solution spaces established through our projection tool is only implicit (there is no one-to-one correspondence) and it must remain so to avoid symmetry.

Our proposal is completed by a practical strategy to handle the multiple pricing subproblems associated with the various column classes defined at a given branch-and-price node. As column classes are nested, one can organize the enumeration of the associated pricing subproblem in a tree. Then, using appropriate tree search strategies, one can truncate the enumeration of pricing subproblems as soon as a negative reduced cost column is found and still get a dual bound on the best reduced cost. Finally, we show that preprocessing yields significant simplifications to the generic scheme and, in particular, in the special case of a set partitioning problem. The Ryan and Foster scheme is shown to be closely related to the form taken by our scheme when the master is a set partitioning problem.

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

1. Let $\Lambda = \{g : \tilde{\nu}_g > 0\}$.
2. Sort the columns $g \in \Lambda$ in lexicographic decreasing order of vectors x^g.
3. Initialize $\tilde{x}_i^r = 0$ for all i, r; let $z = 0$ and $r = 1$.
4. For each $g \in \Lambda$ in lexicographic order, do
   while $(\tilde{\nu}_g > 0)$, do
   let $\tilde{\lambda}_g^r = \min\{\tilde{\nu}_g, r - z\}$;
   for all $i = 1, \ldots, n$, do $\tilde{x}_i^r = \tilde{x}_i^r + x_i^g \, \tilde{\lambda}_g^r$;
   $\tilde{\nu}_g = \tilde{\nu}_g - \tilde{\lambda}_g^r$;
   $z = z + \tilde{\lambda}_g^r$;
   if $(z == r)$ then $r = r + 1$;

To simplify the detailed presentation, we assume from now on that the subsystem involves binary variables only:

$$X = \{x \in \{0, 1\}^n : \; B\, x \geq b\}. \tag{24}$$

Then, *column classes*, $\hat{G}$, are defined as subsets of columns, g, whose indicator vector, x^g, has some components fixed to zero or one. For instance, $S = < x_1, \overline{x}_2, x_3 >$ 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 class of columns is $G(S) = \{g \in G : x_1^g + \overline{x}_2^g + x_3^g = 3\}$, where $\overline{x}_2^g = 1 - x_2^g$. Note that this model can correspond to a practical strategy that consists in branching on the binary variables resulting from a 0–1 transformation of an integer program. Extensions to the general mixed integer case are outlined in Sect. 11.

4 A mapping that preserves integrality

Each solution ν to $[M]$ (resp. λ to $[M_{\mathrm{LP}}]$) can be transformed into a solution x to $[P]$ (resp. $[P_{\mathrm{LP}}]$). One can disaggregate the solution ν to $[M]$ into a solution $\{\lambda^r\}_{r=1,\ldots,R}$ to [DM] and then use transformation (17) to get R solution vectors $x^r \in [0, 1]^n$ for $[P]$. The procedure is not unique. A possible disaggregation for an LP solution is: $\lambda_g^r = \frac{\nu_g}{R} \quad \forall r = 1, \ldots, R, \, g \in G$. But it can yield a fractional solution x, even when ν happens to be integer. An alternative disaggregation that preserves integrality can be defined recursively as follows. Assume an ordering of the generator set, defined by a precedence operator: $g_1 \prec g_2$ if g_1 precedes g_2 in the list of generators G. Then, let

$$\lambda_g^r = \min \left\{ 1, \nu_g - \sum_{\rho=1}^{r-1} \lambda_g^\rho, \left(r - \sum_{\gamma : \gamma \prec g} \nu_\gamma \right)^+ \right\} \quad \forall r = 1, \ldots, R, \, g \in G. \tag{25}$$

An integer solution ν decomposes into an integer solution λ: $\lambda_g^r = 1$ if $\sum_{\gamma \prec g} \nu_\gamma \leq r - 1$ and $\sum_{\gamma \preceq g} \nu_\gamma \geq r$. The resulting mapping procedure into the X space is given in Table 1 and illustrated in Example 1. Each ordering yields a valid mapping, however a lexicographic ordering provides a better chance to generate an integer solution.

Example 1 We shall use the following numerical example to illustrate the developments to come. The constraints of $[M]$ are defined by

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

Assume $R = 5$ identical subsystems and a set G of feasible columns given below, as well as a master LP solution $\tilde{\nu}_g$:

$\tilde{\nu}_g$	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

Using the mapping of Table 1, we get a solution to the original formulation $\{\tilde{x}^r\}_{r=1,\ldots,5}$:

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

□

In the sequel, we refer to a figurative representation of a master solution. We define a "*strip*" associated with a column or a column class as follows:

Definition 1 Consider a master solution $\tilde{\nu}$ where columns are sorted lexicographically. Let us represent the solution geometrically by a rectangle with height n and width R. In this rectangle, each column, g, defines a "*strip*" of "*width*" $\tilde{\nu}_g$ at a specific position in the sequence of sorted columns. More generally, any column class $\hat{G}$ of consecutive columns in lexicographic order defines a "$\hat{G}$-*strip*" of "*width*" $\tilde{\nu}(\hat{G}) = \sum_{g \in \hat{G}} \tilde{\nu}_g$.

The mapping of master solution $\tilde{\nu}$ into an $\tilde{x}$ solution requires slicing the rectangle of width R to partition it into R sub-*strips* of width 1, each of which shall be associated with an index r. Each column g that is involved in the r^{th} *strip* contributes to the definition of $\tilde{x}^r$ proportionally to its value $\tilde{\lambda}^r_g$ which represents its width in the r^{th} *strip*: see (17). In Table 1, z stands for the current position in the G-*strip* of width R.

The mapping of Table 1 was designed to preserve integrality. Inversely, the resulting $\tilde{x}$ solution can only be binary if $\tilde{\nu}$ is integer:

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

$$\tilde{x}^r_i \in \{0, 1\} \ \ \forall i, r \iff \tilde{\nu}_g \in \mathbb{N} \ \ \forall g$$

Proof It is trivial to note that if $\tilde{\nu}_g \in \mathbb{N}\ \forall g$, then $\tilde{x}_i^r \in \{0,1\}\ \forall i, r$. Let us prove the reverse implication by contradiction. Assuming $F = \{g : \tilde{\nu}_g - \lfloor \tilde{\nu}_g \rfloor > 0\} \neq \emptyset$, we show that $\tilde{x}_i^r \notin \{0,1\}$ for some pair (i,r). Let g_1 be the first fractional column of F in lexicographic order. Let $r = \text{argmax}\{\rho : \lambda_{g_1}^{\rho} > 0\}$ be the last index of a vector $\tilde{x}^r$ to the definition of which g_1 contributes. Observe that $0 < \lambda_{g_1}^r = (\tilde{\nu}_{g_1} - \lfloor \tilde{\nu}_{g_1} \rfloor) < 1$. Hence, there exists another column $g_2 :\ g_1 \prec g_2$ with $0 < \lambda_{g_2}^r < 1$. Let i be the first component in which g_1 and g_2 differ: $x_i^{g_1} = 1$ while $x_i^{g_2} = 0$. Then, $0 < \lambda_{g_1}^r < \tilde{x}_i^r \leq 1 - \lambda_{g_2}^r < 1$. □

Thus, when the subsystems are binary problems, checking the integrality of the solution $\tilde{\nu}$ in $[M_{\text{LP}}]$ is equivalent to checking the integrality of the associated solution $\tilde{x}$ in $[P_{\text{LP}}]$. However, when some of the $\tilde{x}$ variables are general integer, the mapped solution can be integer while $\tilde{\nu}$ is not.

5 Separation of a fractional solution at the root node

Branching is implemented by bounding the *"value"* of *"column classes"* that are defined by *"component bound sequences"*.

Definition 2 A "component bound sequence", S, is an ordered set of bounding restrictions on the components of the subproblem solution vector, x. For the binary case, bounding restrictions amount to fixing components to zero or one:

$$S = < \ddot{x}_{[1]}, \ddot{x}_{[2]}, \ldots, \ddot{x}_{[|S|]} >, \tag{26}$$

where $[p]$ is the index of the component bound in the p^{th} position in the sequence, and the notation $\ddot{x}_{[p]}$ stands for either the case in which the variable is fixed to one ($\ddot{x}_{[p]} = x_{[p]}$) or fixed to zero ($\ddot{x}_{[p]} = \overline{x}_{[p]}$). The "last" component of the sequence is the component x_i defined by index $i = [|S|]$. The "column class" associated with S is defined as

$$G(S) = \left\{ g \in G : \sum_{i \in S} \ddot{x}_i^g = |S| \right\}$$

where the notation $i \in S$ is a short cut for saying that sequence S includes a bound on component x_i, also denoted as $x_i \in S$ or $\overline{x}_i \in S$. The "value" of the class is the cumulative value of its columns in the current master solution, $\tilde{\nu}$:

$$\tilde{\nu}(S) = \tilde{\nu}(G(S)) = \sum_{g \in G(S)} \tilde{\nu}_g.$$

The column class value is its "width" in the geometric solution representation of Definition 1.

At the root node, the separation of a fractional solution, $\tilde{\nu}$, to $[M_{\text{LP}}]$ proceeds as follows. Identify a "column class", $G(S)$, whose value, $\tilde{\nu}(S)$, is fractional, and impose

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

1. Let $F = \{g : \tilde{\nu}_g - \lfloor \tilde{\nu}_g \rfloor > 0\}$, $I = \{1, \ldots, n\}$, $S = <>$, *record*$= \emptyset$.
2. Separate(F, I, S, *record*)
   (a) *Check whether the current set of columns has fractional columns:* If $F = \emptyset$, return.
   (b) *Compute values of aggregate variables* $(\alpha_i)_{i \in I}$:
   For all $i \in I$, let $\alpha_i = \sum_{g \in F} x_i^g \, \tilde{\nu}_g$.
   (c) *Detect fractional* α_i *if any:*
   Found = false;
   For all $i \in I$, do
   if $(f = \alpha_i - \lfloor \alpha_i \rfloor > 0)$
   add the pair $(< S, x_i >, f)$ to *record*;
   Found = true;
   If (Found), return.
   (d) *Identify discriminating components:* $J = \{i \in I : \; 0 < \alpha_i < \tilde{\nu}(F)\}$.
   (e) *Partition according to the component with highest branching priority:*
   Let $i^* = \text{argmax}_{j \in J}\{\text{priority}_j\}$;
   Separate $(F(< S, x_{i^*} >), J \backslash \{i^*\}, < S, x_{i^*} >, \textit{record})$;
   Separate $(F(< S, \overline{x}_{i^*} >), J \backslash \{i^*\}, < S, \overline{x}_{i^*} >, \textit{record})$;
   return;
3. Select a branching sequence S in *record* according to branching priorities on its last component.

disjunctive constraints to enforce its integrality. The proof of Proposition 1 provides a possible selection of S defined by the first i components of the characteristic vector, x^{g_1}, of the first fractional column, g_1. However, the implementation of the branching scheme is sensitive to the size of the chosen sequence S: a smaller size S shall induce fewer branch-and-bound nodes and a more balanced branch-and-bound tree. Hence, we seek to implement separation based on sequences, S, with few component bounds.

Reference [20] presents two separation procedures, one that yields a minimal size subset of component bounds (using an enumeration scheme that has exponential worst case complexity) and another that achieves the best complexity in the worst case. The procedure proposed here, in Table 2, is a compromise between these two: it partially relies on enumeration in search for a minimal size component sequence, S, while guaranteeing a polynomial worst case complexity. Its use is illustrated in Example 2. Amongst component sequences with the same size that are candidates for branching, we select the sequence S whose "last component" has the highest branching priority (assuming branching priorities and directions are defined for the "original" variables of the subproblem). We shall indeed show later that branching on $G(S)$ is equivalent to implicitly fixing the last component of S in the original formulation. One could give more emphasis on selecting a last component of higher priority by allowing the size of S to increase. In Table 2, $F = \{g : \tilde{\nu}_g - \lfloor \tilde{\nu}_g \rfloor > 0\}$ denotes the set of fractional columns; $\tilde{\nu}(F) = \sum_{g \in F} \tilde{\nu}_g$ denotes its value; $F(S) = F \cap G(S)$ is the set of fractional columns satisfying component bounds in S; *record* is the list of identified sequences S with fractional value $f = \tilde{\nu}(F(S)) \notin \mathbb{N}$ on which we could branch; and $priority_i$ denotes the branching priority level associated with component x_i of subproblem solution vector $x \in \{0, 1\}^n$.

Example 2 (Example 1 continued) Consider the fractional master solution $\tilde{\nu}$ provided in Example 1. It yields

$$\tilde{\nu}(F(x_i)) = \tilde{\nu}(F(\overline{x}_i)) = 1 \quad \text{for } i = 1, \ldots, 4.$$

Assume that subproblem variables are indexed in order of non-increasing priority. Hence, we make recursive calls to the routine Separate of Table 2 to split further the column class defined by $S =< x_1 >$ and $< \overline{x}_1 >$. In the first call to Separate, we get $\tilde{\nu}(F(x_1, x_i)) = \tilde{\nu}(F(x_1, \overline{x}_i)) = \frac{1}{2}$ for $i = 2, 3$ and pick the highest priority set: $S =< x_1, x_2 >$ or $S =< x_1, \overline{x}_2 >$ depending on priority in branching directions. On the other hand, in the second call, further splitting the column class defined by $S =< \overline{x}_1 >$ yields the branching sequence $S =< \overline{x}_1, x_2 >$ or $S =< \overline{x}_1, \overline{x}_2 >$. □

Observe that $|F| \leq m$, where m is the number of constraints in the master formulation at the current branch-and-bound node (assuming that $\tilde{\nu}$ is obtained as an extreme LP solution to the master program). Each 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 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 $O(n^2|F|^2)$. One could reduce the complexity to $O(n|F|\log|F|)$ by applying the recursive call to only one side of the partition in Step (f), selecting the subset of fractional columns with the smallest value.

Given a component bound sequence S whose associated class has fractional value $\tilde{\nu}(S) \notin \mathbb{N}$ for the current master solution $\tilde{\nu}$, one implements branching by adding a disjunctive constraint that eliminates this fractional solution. A natural scheme would be to use a binary disjunction

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

However, the branching constraint $\nu(S) \leq \lfloor \tilde{\nu}(S) \rfloor$ of the first branch is typically weaker than that of the second branch (leading to an unbalanced tree). A stronger disjunctive constraint can be derived by using a non binary branching tree, as illustrated in Example 3 below. A general presentation of the branching scheme follows the example. An important remark concerning the proposed scheme is that the solution sets associated with the descendant nodes are not necessarily disjoint (we do not have a partition of the master solution space). We prove however that the scheme yields a valid separation: Proposition 2 shows that the fractional solution is eliminated and no integer solution is lost. The finiteness of the scheme shall be proved in Sect. 6: we show that the size of the branch-and-price tree is bounded (in particular, we give a polynomial bound on the depth of the tree).

Example 3 (Example 2 continued) Assume that set $S =< x_1, x_2 >$ was selected for branching and that one uses a binary disjunction: $\nu(x_1, x_2) \leq 1$ or $\nu(x_1, x_2) \geq 2$. In

the first branch, one may have the solution:

$\tilde{\nu}_g$	0	0	1	$\frac{3}{4}$	0	0	1	1	0	$\frac{1}{2}$	$\frac{1}{4}$	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

(this fractional solution can easily be seen to satisfy the master constraints of Example 1). An alternative scheme can be derived considering the following intuitive argument. The branching constraint $\nu(x_1, x_2) \leq 1$ force the LP solver to use fewer columns from class $G(x_1, x_2)$ compared to the current solution, $\tilde{\nu}$. As the total number of columns is fixed, $\nu(G) = R$, this can only be achieved by increasing the number of columns selected from $G \backslash G(x_1, x_2) = G(x_1, \overline{x}_2) \cup G(\overline{x}_1)$. Hence, from an IP perspective, we can define two new branches: either $\nu(x_1, \overline{x}_2) \geq 2$ (assuming implicitly that $\nu(x_1) \geq 3$), or $\nu(\overline{x}_1) \geq 3$ (which implies $\nu(x_1) \leq 2$). In total, we define 3 descendant nodes (defined by a branching constraint) as follows:

$$\begin{aligned} \text{Node}(1) &\equiv \nu(\overline{x}_1) \geq 3, \\ \text{Node}(2) &\equiv \nu(x_1, \overline{x}_2) \geq 2, \\ \text{Node}(3) &\equiv \nu(x_1, x_2) \geq 2. \end{aligned}$$

Observe that the above solution is not feasible in any of Node(1), Node(2), or Node(3), which illustrates that this non-binary scheme can be stronger. However, the solution spaces associated with each node are not disjoint: for instance, solutions with $\nu(x_1, x_2) \geq 2$ can also arise in node 2, if $\nu(x_1) \geq 4$, or, in node 1, if $\nu(x_1, \overline{x}_2) = 0$. Nevertheless, as we see in this example, the identical solutions that could arise at several tree nodes would be solutions that deviated largely from the current solution and hence would probably yield a significant increase in the dual bound. □

The general scheme is characterized by a generic definition of successor nodes:

Definition 3 Let $\tilde{\nu}$ be the master LP solution at the current node and S be the "component bound sequence" (see Definition 2) that has been selected for branching. For $p = 1, \ldots, |S| - 1$, let S^p be the subsequence composed of the first p component bounds of S and $L^p = \tilde{\nu}(S^p) \in \mathbb{N}$. Let $L^{|S|} = \lceil \tilde{\nu}(S) \rceil$ be the round-up value of column class $G(S)$, while $S^0 = <>$ and $L^0 = \tilde{\nu}(G) = R$. Let $G^p = G(S^p)$ for $p = 0, \ldots, |S|$. Observe that $L^0 \geq L^1 \geq L^2 \geq \cdots \geq L^{|S|}$ are nonincreasing integer values, while $G = G^0 \supseteq G^1 \supseteq G^2 \supseteq \ldots \supseteq G^{|S|} = G(S)$. Let $\overline{G}^p = G^{p-1} \backslash G^p = G(S^{p-1}, \bar{\ddot{x}}_{[p]})$ for $p = 1, \ldots, |S|$, where $\bar{\ddot{x}}_{[p]} = 1 - \ddot{x}_{[p]}$ is the complement of the p^{th} component bound of S. Then, the $|S| + 1$ "successor nodes" are defined as follows:

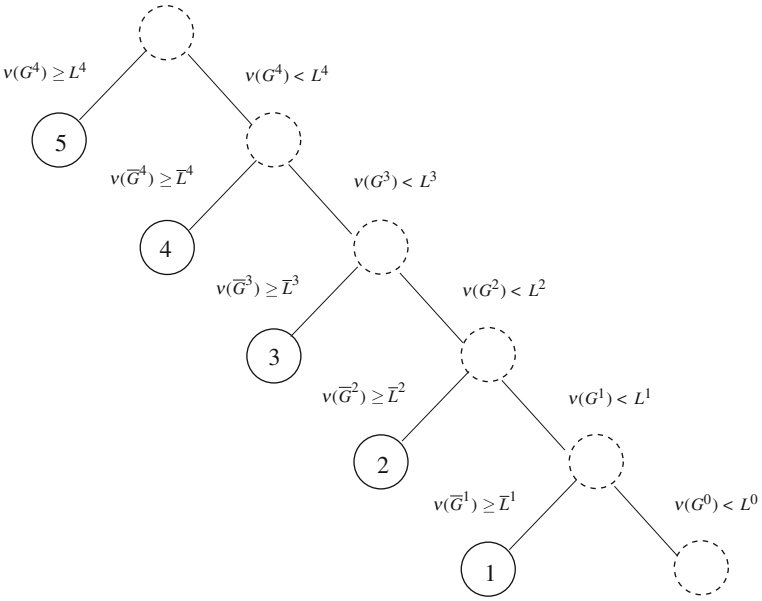

Fig. 2 Example of implicit branching tree when $|S| = 4$: $\overline{G}^k = G^{k-1} \backslash G^k$ and $\overline{L}^k = L^{k-1} - L^k + 1$

for $p = 1, \ldots, |S|$, the branching constraint defining node p is:

$$\text{Node}(p) \equiv v(G^{p-1} \backslash G^p) \geq L^{p-1} - L^p + 1; \tag{27}$$

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

$$\text{Node}(|S| + 1) \equiv v(G^{|S|}) \geq L^{|S|}. \tag{28}$$

Figure 2 illustrates branching at a node for which the branching sequence, S, involves 4 component bounds: $|S| = 4$, yielding 5 descendant nodes. The node numbers refer to the associated node definitions (27–28). The definition of the *strip* (see Definition 1) associated with each nested column class could, for instance, take the form illustrated in Fig. 3: vertical lines define the boundaries of the *strips* associated with nested column classes and their complement. The disjunctive branching constraint can be understood as follows: in an integer solution either the value of class G^4 is increased to the next integer value, i.e., $v(G^4) \geq L^4$ (the width of the G^4-*strip* in Fig. 3 must increase), or it is rounded down, i.e., $v(G^4) < L^4$ (the width of the G^4-*strip* decreases) and some other *strip* of Fig. 3 must have an increased value, i.e., either $v(\overline{G}^4) = v(G^3 \backslash G^4) \geq L^3 - L^4 + 1$, or $v(G^3) < L^3$, or both. Similarly, the case $v(G^3) < L^3$ decomposes recursively into $v(G^2 \backslash G^3) \geq L^2 - L^3 + 1$, or $v(G^1 \backslash G^2) \geq L^1 - L^2 + 1$, or $v(Q \backslash G^1) \geq L^0 - L^1 + 1$, which are the different ways in which the value of the complementary class $G \backslash G^3$ can increase its value in an integer solution.

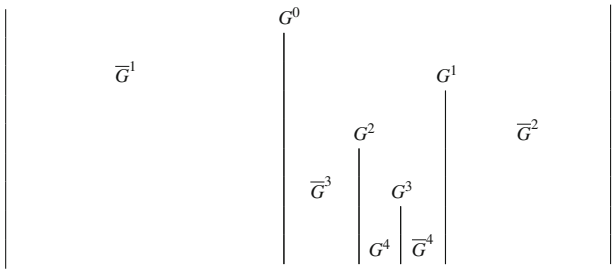

Fig. 3 Example of a partitioning of columns into nested classes when $|S| = 4$: each *strip* bounded by *vertical lines* has a width that represents the value $\nu(\hat{G})$ of the associated class $\hat{G}$

The dotted nodes that appear in Fig. 2 are not explicitly defined in our scheme, i.e., constraints $\nu(G^p) < L^p$ on the branches leading to the dotted nodes are not strictly enforced. The subtrees hanging from those branches include solutions that do not satisfy these constraints because our scheme does not define a partition of the solution space. 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, $\tilde{\nu}$, that are not represented in any of the nodes $l = p, \ldots, |S| + 1$ are those for which $\tilde{\nu}(G^p) < L^p$.*

Proof We show this by induction. For $p = |S|+1$, the result is trivial. Let us derive the result for any $p \leq |S|$, assuming the result holds for $p+1$. Take an integer solution, $\tilde{\nu}$, that does not satisfy any of the branching constraints defining node $l = p, \ldots, |S|+1$. By the induction hypothesis, we must have $\tilde{\nu}(G^{p+1}) < L^{p+1}$ and hence $\tilde{\nu}(G^{p+1}) \leq L^{p+1} - 1$. Moreover, as $\tilde{\nu}$ must violate the branching constraint defining Node(p), we have $\tilde{\nu}(G^p \backslash G^{p+1}) \leq L^p - L^{p+1}$. As $\nu(G^p) = \nu(G^p \backslash G^{p+1}) + \nu(G^{p+1})$, the two previous inequalities imply $\tilde{\nu}(G^p) \leq L^p - 1 < L^p$. □

Although our branching can yield up to $n + 1$ branches, Lemma 1 shows that nodes $1, \ldots, p - 1$ only serve the purpose of enumerating integer solution in which $\nu(G^p) < L^p$. This shall be exploited in Sects. 6 and 9 to show how the scheme simplifies in practice.

Let us now show that the branching scheme yields a valid separation that cuts off the current fractional solution and eliminates no integer solution.

Proposition 2 *The descendant nodes, Node$(1), \ldots,$ Node$(|S|+1)$, of Node(0) define a valid separation scheme, i.e.,*

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

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

in class $G(S^{|S|-1})$ shall use either at least $L^{|S|}$ columns in class $G(S^{|S|})$ or use at least $L^{|S|-1} - L^{|S|} + 1$ in the complementary class, $G^{|S|-1} \backslash G^{|S|}$, or both. Other integer solutions use strictly less than $L^{|S|-1}$ columns in class $G^{|S|-1}$. In the latter case, the complementary class, $G \backslash G^{|S|-1}$, must hold a least $R - L^{|S|-1} + 1$ columns, i.e., one more than in the current solution $\tilde{\nu}$. The complementary class, $G \backslash G^{|S|-1}$, can be partitioned into $G^{p-1} \backslash G^p$ for $p = 1, \ldots, |S| - 1$: $G \backslash G^{|S|-1} = \cup_{p=1,\ldots,|S|-1}(G^{p-1} \backslash G^p)$ and these classes are disjoint. Thus, the case where the value of the complementary set, $G \backslash G^{|S|-1}$, must increase by one unit can be decomposed into $|S| - 1$ subcases depending on which subset of the partition sees its value increased by one unit compared to the current solution $\tilde{\nu}$. These subcases are defined by Node(1) to Node($|S| - 1$). □

6 Maintaining a nested separation scheme beyond the root node

Assume that the current branch-and-price node is defined by branching constraints, $\nu(S^k) \geq L^k$, for $k = 1, \ldots, K$, where the associated classes, $G^k = G(S^k)$, define a nested partition of G. We show how to eliminate a current fractional master solution, $\tilde{\nu}$, while maintaining a nested partition of G. Next, we observe that, in practice, many of the successor nodes of Definition 3 can be eliminated by a dominance rule. Then, we derive a key property of our scheme: each branching constraint implies primal progress in reaching integrality that can be measured in the space of the original variables. We begin by making several observations that characterize basic properties of a branching scheme based on a nested partition of the column set.

Observation 1 *In the nested collection of component sets and associated bounds,* $\{(S^k, L^k)\}_{k=1,\ldots,K}$*, that defines a branch-and-price node, no two* (S^k, L^k) *are identical.*

Indeed, at the stage where a branching constraint, $\nu(S) \geq L$, was added, it had to cut off the current fractional master solution, $\tilde{\nu}$, and therefore, either $S \neq S^k$ held for all previously defined S^k, or else $S = S^k$ and $L = \lceil \tilde{\nu}(S^k) \rceil > L^k$ for some previously defined branching constraint k.

Observation 2 *At a given branch-and-price node, the branching constraints define a collection of nested column classes. Therefore, the inclusion relations between the classes can be represented in a tree where ascendancy represents inclusion.*

Example 4 Let $n = 4$ and $R = 10$. Assume that the current branching constraints are:

$$\nu(x_2, x_3, x_4) \geq 1\,, \tag{29}$$
$$\nu(x_2, x_3, x_4, \overline{x}_1) \geq 1, \tag{30}$$
$$\nu(x_2, \overline{x}_3, x_1) \geq 1, \tag{31}$$
$$\nu(\overline{x}_2, x_1) \geq 4, \tag{32}$$
$$\nu(\overline{x}_2, x_1, x_4) \geq 2, \tag{33}$$
$$\nu(\overline{x}_2, \overline{x}_1) \geq 3, \tag{34}$$
$$\nu(\overline{x}_2, \overline{x}_1, x_3, x_4) \geq 1. \tag{35}$$

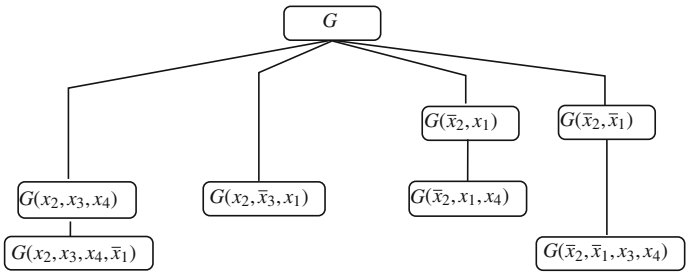

Fig. 4 Tree of column classes for Example 4

The associated tree of column classes is given in Fig. 4. □

Definition 4 Given a collection $\{G^k\}_{k=1,\ldots,K}$ of nested column classes, we define the associated "tree of column classes" as follows. The root node is associated with G. The leaf nodes represent classes G^l that have no subclass in the collection, i.e., $G^j \not\subset G^l$ for all $j = 1, \ldots, K$ with $j \neq l$. The node associated with G^l hangs from that associated with G^j, if $G^l \subset G^j$ and there is no k such that $G^l \subset G^k \subset G^j$. In that case, G^l is a "direct successor" of G^j and G^j a "direct predecessor" of G^l. More generally, we say that class G^l is a "predecessor" of G^j if $G^l \supset G^j$, and G^l is a "successor" of G^j if $G^l \subset G^j$. The set of column classes that are direct successors of G^l is denoted $dsucc(l)$; the direct predecessor is denoted $dpred(l)$; the set of all predecessors (resp. successors) of G^l is denoted $pred(l)$ (resp. $succ(l)$).

Observation 3 *At a given branch-and-price node, the branching constraints induce a specific ordering of the columns.*

Because the collection of column classes $\{G^k\}_{k=1,\ldots,K}$ defines a nested partition of G, all the associated component bound sequences $S^k, k = 1, \ldots, K$, must start with a bound on the same component (as illustrated in Example 4): $G^l \subset G^j \Leftrightarrow S^j \subset S^l \Leftrightarrow \ddot{x}_{[p]}(S^l) = \ddot{x}_{[p]}(S^j)$ for $p = 1, \ldots, |S^j|$. The order induced by the collection of component bound sequences, $\{S^k\}_{k=1,\ldots,K}$, consists in sorting columns primarily according to the components in the order in which they appear in the sequences defining the column classes to which the columns belong. Beyond the component order prescribed by component sequences S^k, one uses a standard lexicographic order. Table 3 presents a comparison operator that implicitly defines the ordering induced by a nested collection of branching sequences. Its parameters are defined as follows: C denotes a set of nested branching sequences, S denotes the sequence of component bounds already tested, I denotes the set of components that have not been fixed by a component bound so far, and p denotes the next position in branching sequences of C that needs to be considered. $C(S)$ denotes the subset of sequences starting with S, i.e., $C(S) = \{S^k \in C : S \subseteq S^k\}$. Column g_1 precedes column g_2 in the induced order if the operator $g_1 \prec (C, S, I, p) g_2$ returns true for $C = \{S^k\}_{k=1,\ldots K}$, $S = <>$, $I = \{1, \ldots, n\}$, and $p = 1$. Here is an example of its application:

Example 5 (Example 4 continued) Assume that the node is defined by branching constraints (29–35) and that the current fractional master solution is

Table 3 Definition of the operator $g_1 \prec (C, S, I, p)\ g_2$

1.	If $C = \emptyset$, return $x_I^{g_1} \prec x_I^{g_2}$ where x_I^g stands for the characteristic vector of column g restricted to its components $i \in I$ and $x^a \prec x^b$ is the standard lexicographic operator that returns true if x^a is before x^b in lexicographic order and false otherwise.
2.	Let i be the component such that $\ddot{x}_{[p]} = \ddot{x}_i$ in all $S^k \in C$.
3.	If $x_i^{g1} = x_i^{g2} = 1$, return $g_1 \prec (C(< S, x_i >), < S, x_i >, I\backslash\{i\}, p+1)\ g_2$;
4.	If $x_i^{g1} = x_i^{g2} = 0$, return $g_1 \prec (C(< S, \overline{x}_i >), < S, \overline{x}_i >, I\backslash\{i\}, p+1)\ g_2$;
5.	return true if $(x_i^{g1} > x_i^{g2})$ and false otherwise.

g	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\tilde{\nu}_g$	0	$\frac{3}{4}$	$\frac{1}{4}$	1	1	1	1	1	0	0	1	1	$\frac{3}{2}$	$\frac{1}{2}$	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

The ordering induced by the nested branching sequences is

g	8	1	9		2	3	10	11		4	6	5	7		12	13	14	15
$\tilde{\nu}_g$	1	0	0		$\frac{3}{4}$	$\frac{1}{4}$	0	1		1	1	1	1		1	$\frac{3}{2}$	$\frac{1}{2}$	0
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

Definition 5 Given a nested collection of component bound sequences $\{S^k\}_{k=1,\ldots,K}$, the so-called "induced lexicographic order" (ILO) is the ordering defined by sorting the columns, $g \in G$, using the operator of Table 3 called with parameters $C = \{S^k\}_{k=1,\ldots K}$, $S =<>$, $I = \{1, \ldots, n\}$, and $p = 1$.

In the sequel, we always assume that columns are sorted using the ILO. To simplify the notation, the precedence relation $g_1 \prec g_2$ is now meant to say g_1 precedes g_2 in the ILO. Furthermore, any reference to the mapping procedure of Table 1 assumes an ILO sorting of the columns.

The procedure to separate a fractional master solution, $\tilde{\nu}$, can now be easily described. To achieve a nested partition of the column set, separation must be done either by further partitioning a class G^k of the current collection $\{G^k\}_{k=1,\ldots,K}$ or its complementary class, $\overline{G}^k$, or by resetting the bound on an existing class, G^k. Intuitively, a candidate component sequence on which to branch is identified by taking the set of fractional columns down the tree of column classes and checking the value of column classes for fractionality, as illustrated in Fig. 5. Three cases may arise as outlined in Table 4.

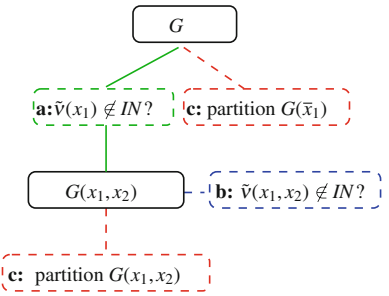

Fig. 5 Testing the fractionality of column subsets in node(3) of Example 3 to identify a branching set

Table 4 Cases arising when separating a fractional master solution assuming a current collection of column classes $\{G^k\}_{k=1,\ldots,K}$

case a: An intermediate class, $\hat{G}$, with $G^l \subset \hat{G} \subset G^j$ for some j and $l \in \{1, \ldots, k\}$ is identified as having fractional value; then one can branch on the associated sequence $\hat{S}$.
case b: An existing class, G^l, has fractional value; then one can branch on S^l which implies resetting the associated L^l bound.
case c: An existing leaf class or the complementary class to any intermediate class, is partitioned further by applying the procedure Separate of Table 2 to the fractional columns of this class; this yields a fractional value class on which to branch.

Formally, separation proceeds as follows. Let i be the first component present in all previous branching sequences $S^k, k = 1, \ldots, K$, i.e., $\ddot{x}_{[1]}$ is either x_i or $\bar{x}_i$ in all S^k. Let $F = \{g : \tilde{\nu}_g - \lfloor \tilde{\nu}_g \rfloor > 0\} \neq \emptyset$ and $\tilde{\nu}(x_i) = \sum_{q \in F : x_i^g = 1} (\tilde{\nu}_g - \lfloor \tilde{\nu}_g \rfloor)$. If $\tilde{\nu}(x_i) \notin \mathbb{N}$, we can branch on $S =< x_i >$. Otherwise, we recursively explore both sets of fractional columns $F(< x_i >)$, $F(< \bar{x}_i >)$ (if not empty) in search of 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 5. As above, C stands for the set of component bound sequences associated with branching constraints defining the current node, S is the component bound sequence defining the current column class, p denotes the next position to be considered in the sequences of C, $C(S)$ is the subset of sequences starting with S, I is the set of components that are candidate for further partitioning of the current column class, and *record* gathers the set of possible branching sequences. Its use is illustrated in Example 6.

Example 6 (Example 5 continued) It is more convenient to follow the separation procedure in the second table of Example 5 that represents the master LP solution sorted in ILO. Note that $\tilde{\nu}(F) = 3$. The future branching sequence, S, must start with x_2 or $\bar{x}_2$. Partitioning according to component 2 splits the set of fractional columns, with $\tilde{\nu}(F(x_2)) = 1$ and $\tilde{\nu}(F(\bar{x}_2)) = 2$. Component 3 does not allow us to partition $F(x_2)$ further since all the fractional columns of $G(x_2)$ have $\bar{x}_3 = 1$; neither does component 1. Thus, $\tilde{\nu}(F(x_2, \bar{x}_3, x_1)) = 1$ must be further split using routine Separate which returns $S^1 =< x_2, \bar{x}_3, x_1, x_4 >$. On the other hand, set $F(\bar{x}_2)$ cannot be partitioned further with component 1, but component 3 yields a fractional value subset. Thus, another possible branching sequence is $S^2 =< \bar{x}_2, \bar{x}_1, x_3 >$. One of these two branching sequences is selected according to branching priorities.

Table 5 Separation of a fractional master solution at a node other than the root

1. Let $F = \{g : \tilde{v}_g - \lfloor \tilde{v}_g \rfloor > 0\}$, $I = \{1, \ldots, n\}$, $S = <>$, *record*$= \emptyset$, $p = 1$, and $C = \{S^k\}_{k=1,\ldots K}$.
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 $x_{[p]} = \ddot{x}_i$ in all $S^k \in C$.
   (c) Let $\alpha_i = \sum_{g \in F} x_i^g \tilde{v}_g$ and $f = \alpha_i - \lfloor \alpha_i \rfloor$.
   (d) *Check whether class* $G(< S, x_i >)$ *has fractional value:* If $f > 0$, add the pair $(< S, x_i >, f)$ to *record* and return.
   (e) *Recursively explore the first subclass of columns,* $G(< S, x_i >)$*:* If $\alpha_i > 0$, Explore($C(< S, x_i >)$, $p + 1$, $F(< S, x_i >)$, $I \backslash \{i\}$, $< S, x_i >$, *record*).
   (f) *Recursively explore the second subclass of columns,* $G(< S, \overline{x}_i >)$*:* If $\alpha_i < \tilde{v}(F)$,
   Explore($C(< S, \overline{x}_i >)$, $p + 1$, $F(< S, \overline{x}_i >)$, $I \backslash \{i\}$, $< S, \overline{x}_i >$, *record*).
3. Select a branching sequence S in *record* according to branching priorities on its last component.

Let us consider the complexity of the separation procedure of Table 5. 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 of the procedure is $O(n^2\,|F|^2)$ as it was at the root node. Here too, one can reduce the complexity by exploring only the smallest value subset of fractional columns (selecting step (e) or (f)). However, exploring both sides often allows us to identify a branching sequence S with fewer component bounds (as illustrated in Example 5) or higher branching priority.

In practice, the branching 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 or not. Case b of Table 4 often consists in resetting the bound of a class whose parent class defines an active branching constraint. Exploiting Lemma 1 in these situations yields a restriction in the number of successor nodes to only two nodes. More generally,

Proposition 3 *The set of descendant nodes defined in* (27–28) *can be restricted to only nodes:* $k, \ldots, |S| + 1$*, where* S *is the sequence chosen for branching to eliminate the current fractional solution,* $\tilde{v}$*, and* k *is the size of the largest subsequence* S^k *of* S *for which a previously defined branching constraint* $v(G^k) \geq L^k$ *is active, i.e.,* $L^k = \tilde{v}(G^k)$.

The proof is a straightforward application of Lemma 1: if constraint $v(G^k) \geq L^k$ is enforced, no integer solution exists with $v(G^k) < L^k$, and hence, there is no need to enumerate descendant nodes whose only purpose was to consider integer solutions with $v(G^k) < L^k$. Moreover, one can further prune the enumeration tree by dominance:

Observation 4 *A branch-and-price node, N_1, defined by a branching constraint $\nu(G^k) \geq L^k$ needs not be generated if it has an ancestor node, N_0, whose direct successor, N_2, is defined by the same constraint.*

Indeed, in such case, integer solutions satisfying all branching constraints defined at node N_0 and constraint $\nu(G^k) \geq L^k$ are already enumerated in node N_2. Thus, Proposition 3 and Observation 4 explain why in practice the branch-and-bound tree can be close to be binary.

To conclude this section, we analyse the impact of our branching constraints in the variable space of the original formulation. We begin by an informal illustration. Returning to Example 3, we indicate the progress in reaching primal integrality that is made implicitly with each new branching constraint.

Example 7 (Example 3 continued) Assume a fractional solution $\tilde{\nu}$ and associated column class values $\tilde{\nu}(x_1, x_2)$, $\tilde{\nu}(x_1, \overline{x}_2)$ and $\tilde{\nu}(\overline{x}_1)$ given by:

$\tilde{\nu}_g$	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)										$(\overline{x}_1)$				
	(x_1, x_2)			$(x_1, \overline{x}_2)$											
	1.5			1.5							2				

Let $\tilde{x}^r$ for $r = 1, \ldots, 5$ be the associated solution obtained from the mapping procedure of Table 1. Observe that for $r = 1$ vector $\tilde{x}^r$ is entirely determined from columns of class $G(x_1, x_2)$; for $r = 3$, from columns of class $G(x_1, \overline{x}_2)$; for $r = 4$ and 5 from columns of class $G(\overline{x}_1)$; while for $r = 2$, $\tilde{x}^r$ is partially derived from columns of class $G(x_1, x_2)$ and partially from columns of class $G(x_1, \overline{x}_2)$. Now let us see the consequence of branching constraints on the projected solutions $\tilde{x}^r$. Branching induces modification of the widths of the "strips" (see Definition 1) associated with the different column classes:

- In Node(1), $\nu(\overline{x}_1) \geq 3$ implies that three $\tilde{x}^r$ vectors resulting from the mapping of Table 1 after branching will be derived from the columns of class $G(\overline{x}_1)$ and hence $\tilde{x}^r_1 = 0$ for at least 3 r-indices (while before branching we had $\tilde{x}^r_1 = 0$ for only 2 r-indices). We say that component x^r_1 is now "covered" (or fixed) for 3 r-indices.
- In Node(3), $\nu(x_1, x_2) \geq 2 \Rightarrow \tilde{x}^r_1 = \tilde{x}^r_2 = 1$ for at least 2 r-indices (while before branching we had $\tilde{x}^r_1 = \tilde{x}^r_2 = 1$ for a single index r). We say that x^r_1 and x^r_2 are now "covered" for 2 r-indices.
- In Node(2), we would expect that constraint $\nu(x_1, \overline{x}_2) \geq 2$ implies $\tilde{x}^r_1 = 1$ and $\tilde{x}^r_2 = 0$ for at least 2 r-indices (while at the ancestor node $\tilde{x}^r_1 = 1$ and $\tilde{x}^r_2 = 0$ for a single index r). But class $G(x_1, x_2)$ that precedes class $G(x_1, \overline{x}_2)$ in the lexicographic order can take a fractional value. If it does so, the "strip" of width 2 associated with class $G(x_1, \overline{x}_2)$ would not coincide exactly with two x^r vectors in the mapped solution. We say that the $G(x_1, \overline{x}_2)$-*strip* is " floating" over 2 r-indices.

A more formal presentation requires the definition of what we call a "*frame*" for each branching constraint $\nu(S) \geq L$. Following up on Definition 1, the branching constraint can be seen intuitively as imposing a minimum width L to a $G(S)$-*strip* (remember that columns of $G(S)$ are consecutive in the ILO sorting). This *strip* can be later subdivided if branching constraints are introduced on subsets of class $G(S)$. Nevertheless, the branching constraint $\nu(S) \geq L$ already fixes part of the x solution obtained through the mapping of Table 1: the contribution of the columns of the *strip* to the definition of x^r vectors is fixed for all $i \in S$. Hence, we refine the concept of a *strip* by introducing a *frame* that can be understood as a *strip* on a specific selection of consecutive lines $i \in S$ in the table representing a master solution in ILO.

Definition 6 Let S be a component bound sequence and $L \in \mathbb{N}$ be the associated width. Then, the "(S, L)-*strip*" relative to a master solution $\tilde{\nu}$ is made of the first columns of $G(S)$ up to value L in the table representing the master solution in ILO: it is a $G(S)$-*strip* of width L. It is precisely defined by an interval of cumulative values of columns in the ILO sequence:

$$V(S, L) = \left[\sum_{g:\, g \prec G(S)} \tilde{\nu}_g, \sum_{g:\, g \prec G(S)} \tilde{\nu}_g + L \right).$$

The interval is closed on the left and open on the right. The notation $g \prec G(S)$ says that column g is prior to those of $G(S)$ in the ILO.

An "(S, L)-*frame*" is the restriction of an "(S, L)-*strip*" to a selection of consecutive lines $i \in S$. We say that a component x_i^r of the $\tilde{x}$ solution obtained through the procedure of Table 1 is "covered" by the (S, L)-*frame* if $i \in S$ and $(r-1) \in V(S, L)$ (in other words, the *strip* of width 1 that is associated with $\tilde{x}^r$ begins in the (S, L)-*strip*). Thus, an (S, L)-*frame* covers $L\,|S|$ components x_i^r.

Example 8 (Examples 4 and 5 continued) The table below illustrates the *frames* associated with branching constraints (29–35) for a different master solution, $\tilde{\nu}$, represented in ILO. It shows only the columns with $\tilde{\nu}_g > 0$ and duplicates the columns that lie over several slices (associated with different index r). The bold face x_i^g components are those contributing to the definition of x_i^r components that are "covered" by these *frames*. On the bottom of the table are represented the boundaries of the R *strips* of width 1 associated with the $\tilde{x}^r$ vectors.

g	8	9	2	2	3	11	4	6	5	7	12	13	13	14
$\tilde{\lambda}_g^r$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$	1	1	1	1	1	1	$\frac{1}{2}$	$\frac{1}{2}$
x_2	**1**	1	**1**	**1**	**1**	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
x_3	**1**	1	**0**	**0**	**0**	0 x_1	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**
x_4	**1**	0 x_1	**1**	**1**	**1**	0 x_4	**1**	**1**	0	0	x_3 **1**	1	1	0
x_1	**0**	0 x_4	1	1	0	0 x_3	1	0	1	0	x_4 **1**	0	0	1
	$r=1$	$r=2$		$r=3$			$r=4$	$r=5$	$r=6$	$r=7$	$r=8$	$r=9$	$r=10$	

Components x_i^r are covered by the $(< x_2, x_3, x_4 >, 1)$-*frame* for $r = 1$ and $i \in \{2, 3, 4\}$; there are covered by the $(< x_2, x_3, x_4, \overline{x}_1 >, 1)$-*frame* for $r = 1$ and $i \in \{2, 3, 4, 1\}$, by the $(< x_2, \overline{x}_3, x_1 >, 1)$-*frame* for $r = 3$ and $i \in \{2, 3, 1\}$, by the $(< \overline{x}_2, x_1 >, 4)$-*frame* for $r \in \{4, 5, 6, 7\}$ and $i \in \{2, 1\}$ by the $(< \overline{x}_2, x_1, x_4 >, 2)$-*frame* for $r \in \{4, 5\}$ and $i \in \{2, 1, 4\}$ by the $(< \overline{x}_2, \overline{x}_1 >, 3)$-*frame* for $r \in \{8, 9, 10\}$ and $i \in \{2, 1\}$, and by the $(< \overline{x}_2, \overline{x}_1, x_3, x_4 >, 1)$-*frame* for $r = 8$ and $i \in \{2, 1, 3, 4\}$. Although the (S, L)-*frames* are not disjoint, the (S, L)-*strips* are nested. A component can be covered by multiple *frames*.

To formalize the impact of branching constraint, $v(S) \geq L$, on the $\tilde{x}$ solution obtained through the mapping of a master solution, $\tilde{\nu}$, note that:

Observation 5 *If the columns preceding those defining an (S, L)-frame in the ILO have a total value that is integer, i.e., if $v = \sum_{g:\, g \prec G(S)} \tilde{\nu}_g \in \mathbb{N}$, then interval $V(S, L)$ has integer extreme points and each component x_i^r "covered" by the (S, L)-frame has integer value in the $\tilde{x}$ solution obtained through the mapping of Table 1. More precisely, $\tilde{x}_i^r = 1$ if $x_i \in S$ and $\tilde{x}_i^r = 0$ if $\overline{x}_i \in S \; \forall i \in S$, $r = v + 1, \ldots, v + L$.*

If $v \in \mathbb{N}$, the (S, L)-*frame* boundaries define *strips* that encompass exactly L consecutive x^r vectors.

Thus, the (S, L)-*frame* associated with branching constraint $v(S) \geq L$ can be understood as defining a set of x_i^r components that are potentially fixed to an integer value in the mapped x solution. However, the *frame* is "floating" around, i.e., the specific r-indices for which this fixing shall occur may change. The boundaries of the *frame* fall precisely in between r-indices only when ILO preceding columns sum up to an integer value. Allowing such "floating" is a way to avoid the symmetry in r. Nevertheless, the integrality of the associated x solution shall be achieved at some stage:

Proposition 4 *Consider a branch-and-price node defined by a nested collection of component sets and associated bounds $\{(S^k, L^k)\}_{k=1,\ldots,K}$ and a master solution $\tilde{\nu}$. If, in the $\tilde{x}$ solution derived through the mapping of $\tilde{\nu}$ as defined in Table 1,* **every** *x_i^r component* **is "covered"** *by some (S^k, L^k)-frame for $i = 1, \ldots, n$ and $r = 1, \ldots, R$, then the $\tilde{x}$ solution obtained through the mapping of Table 1 is integer.*

Proof We prove that if the $\tilde{x}$ solution derived through the mapping of Table 1 is fractional, there exists an x_i^r component that is not covered by any (S, L)-*frame*. By Proposition 1, if the mapped $\tilde{x}$ solution is fractional, then the master solution $\tilde{\nu}$ must be fractional. Let g_1 be the first column in the ILO with fractional value: $0 < \tilde{\lambda}_{g_1}^r < 1$ for $r = \sum_{g \prec g_1} \tilde{\nu}_g + \lceil \tilde{\nu}_{g_1} \rceil$. Let g_2 be the first column in ILO such that $g_1 \prec g_2$ and $\tilde{\lambda}_{g_2}^r > 0$. Let i be the first index in the index ordering underlying the ILO for which $x_i^{g_1} \neq x_i^{g_2}$. Then, $\tilde{x}_i^r \notin \{0, 1\}$. Assume by contradiction that component x_i^r is covered by an (S, L)-*frame* associated with some branching constraint of the nested collection defining the current node. By Definition 6, this means that $i \in S$ and $(r - 1) \in V(S, L)$. Let i be the p^{th} component for which a bound is defined in sequence S: $S =< \ddot{x}_{[1]}, \ldots, \ddot{x}_{[|S|]} >, \ddot{x}_{[p]} = x_i$ or $\overline{x}_i$, and $p \leq |S|$. Now observe that by definition of g_1, $\sum_{g \prec g_1} \tilde{\nu}_g \in \mathbb{N}$. Thus, by definition of r, column g_1 is the

first column of the r^{th}-*strip*. I.e., column g_1 is found at position $(r-1)$ in the ILO: $\sum_{g \prec g_1} \tilde{\nu}_g \leq r-1$ and $\sum_{g \prec g_1} \tilde{\nu}_g + \tilde{\nu}_{g_1} > r-1$. Therefore, if x_i^r is covered by an (S, L)-*frame*, we must have $g_1 \in G(S)$. This implies that the first p component bounds of S are satisfied by the characteristic vector x^{g_1}. But, by definition of i, $x_i^{g_1} \neq x_i^{g_2}$. As $i \in S$, $g_2 \notin G(S)$. Because $g_1 \in G(S)$ is the first column with fractional value in the ILO and $g_2 \notin G(S)$ is the first column with positive value following g_1 in the ILO, the (S, L)-*frame* must have its right boundary at value $\sum_{g \prec g_2} \nu_g \notin \mathbb{N}$. This is a contradiction. Indeed, the right boundary of the (S, L)-*frame* should have been at an integer value since its left boundary is at an integer value and $L \in \mathbb{N}$. Hence, no (S, L)-*frame* can cover x_i^r. □

Although a component x_i^r can be covered by several *frames*, the nested definition of branching constraints guarantees that each (S^k, L^k)-*frame* that covers an x_i^r imposes the same bound on x_i^r. More specifically:

Observation 6 *When the $\tilde{x}$ solution derived through the mapping of Table 1 is integer, each x_i^r component that is covered by an (S, L)-frame has a value in $\tilde{x}$ that is prescribed by the associated component bound in S.*

This observation results from Proposition 1 and Observation 5. Indeed, $\tilde{x}$ being integer implies that $\sum_{g:\, g \prec G(S)} \tilde{\nu}_g \in \mathbb{N}$, for all (S, L)-*frames*.

To guarantee that progress is made in reaching primal integrality in the x-space with each new branching constraint, we need to show that each *frame* yields coverage of at least one "new" x_i^r component. This property is only implicit as we cannot exhibit a fixed pair (i, r) for which x_i^r is covered by the new branching constraint.

Definition 7 The "representative" of an (S, L)-*frame* is the covered x_i^r component such that i is the last component of S in the sense of Definition 2, and r is the largest index such as $(r-1) \in V(S, L)$, where $V(S, L)$ is the interval of Definition 6.

In other words, the "representative" of an (S, L)-*frame* is the "bottom-right" component from amongst all the x_i^r that are covered by this *frame*. We can show that a given x_i^r component can be the representative of only one *frame*:

Proposition 5 *Consider a branch-and-price node defined by a nested collection of component sets and associated bounds, $\{(S^k, L^k)\}_{k=1,\ldots,K}$, and any feasible solution, $\tilde{\nu}$, to the master LP at that node. In the associated $\tilde{x}$ solution, obtained through the mapping procedure of Table 1, a given x_i^r component can be the "representative" of at most one (S^k, L^k)-frame.*

Proof Assume by contradiction that component x_i^r is the representative of two distinct *frames*: the (S^1, L^1) and the (S^2, L^2)-*frame*. The fact that component x_i^r is covered by both *frames* implies that $(r-1) \in V(S^1, L^1) \cap V(S^2, L^2)$. Take the column g with $\tilde{\nu}_g > 0$ that lies at width $(r-1)$ in the G-*strip* of columns sequenced in ILO, i.e., $\sum_{\gamma \prec g} \tilde{\nu}_\gamma \leq r-1$ and $\sum_{\gamma \prec g} \tilde{\nu}_\gamma + \tilde{\nu}_g > r-1$. As $(r-1) \in V(S^1, L^1) \cap V(S^2, L^2)$, we must have $g \in G(S^1) \cap G(S^2)$. Hence, the component bounds defining S^1 and S^2 must be set to the value found in x^g. Moreover, as i is the last component of both S^1 and S^2, in the sense of Definition 2, we must have $|S^1| = |S^2|$, and thus $S^1 = S^2$.

Now, as the $G(S^1)$-*strip* and the $G(S^2)$-*strip* start at the same point in the ILO table, and as r is the furthest right index for both the (S^1, L^1) and the (S^2, L^2)-*frame*, we must have $L^1 = L^2$. Thus, both *frames* must be identical in contradiction to our assumption. □

Thus, although *frames* may overlap, Proposition 5 gives us a way to show that the number of covered x_i^r components increases in each branch. Intuitively, each (S, L)-*frame* is unique and has its own "representative". Hence, each *frame* can be given credit for ensuring the covering of at least its "bottom-right" component. Observation 6 says that an x_i^r component that is covered is implicitly fixed to a binary value. In that sense, *each branching constraint implicitly fixes at least one* x_i^r *component to its integer value in the mapped solution obtained through the procedure of Table 1*. Building on this intuition, we can show formally that the scheme ends up with an integer solution after adding at most $n\ R$ branching constraints. What is more, even though the branching tree is not binary, we can show that the number of leaf nodes is bounded by 2^{nR}.

Proposition 6 *With the proposed branching scheme, the depth of the branch-and-price tree is bounded by* $n\ R$ *and the number of leaf nodes is bounded by* 2^{nR}.

Proof Each branching constraint defining a node is different (see Observation 1) and defines its own (S, L)-*frame* relative to the current master LP solution that covers $L\ |S|$ components of the $\tilde{x}$ solution obtained through the procedure of Table 1 (see Definition 6), including its "bottom-right representative". By Proposition 5, any x_i^r component can be the representative of at most one *frame*. Hence, by the pigeon hole principle, each x_i^r component must be "covered" after at most $n\ R$ branches. Once each x_i^r component is covered, the mapped $\tilde{x}$ solution is integer (by Proposition 4) and so is the associated master solution $\tilde{\nu}$ (by Proposition 1). Thus, we have shown that after at most $n\ R$ branches, we get an integer solution. Now, Observation 6 tells us that, in this integer solution, each x_i^r component takes the value that is prescribed by the S associated with the covering *frame*: $\tilde{x}_i^r = 1$ if $x_i \in S$ and $\tilde{x}_i^r = 0$ if $\overline{x}_i \in S$. Thus, there are 2 possible values for each of the $n\ R$ components and hence at most $2^{n\ R}$ leaf nodes in the branch-and-price tree. □

7 Solving the master after branching

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

$$\min \sum_{g \in G} c\, x^g\, \nu_g \tag{36}$$

$$[\text{M(node)}] \quad \sum_{g \in G} A\, x^g\, \nu_g \geq a \qquad (\pi) \tag{37}$$

$$\sum_{g \in G} \nu_g = R \qquad (\sigma_0) \tag{38}$$

$$\sum_{g \in G^k} \nu_g \geq L^k \qquad (\sigma_k) \qquad \forall k = 1, \ldots, K \tag{39}$$

$$\nu_g \geq 0 \quad \forall g \in G \tag{40}$$

where constraints $\sum_{g \in G^k} \nu_g \geq L^k$ are branching constraints and (π, σ) denotes the dual solution.

As underlined in Observation 1, each branching constraint is different. However, the collection $\{(S^k, L^k)\}_{k=1,\ldots,K}$ could include (S^1, L^1) and (S^2, L^2) with $S^1 = S^2$ but $L^1 < L^2$ (such (S^2, L^2) can result from case b of Table 4). Then, branching constraint $\nu(S^2) \geq L^2$ clearly dominates $\nu(S^1) \geq L^1$ and the latter can be deleted from the formulation. Hence, in the sequel we assume that each branch-and-price node is defined by the nested collection of component sets and associated bounds, $\{(S^k, L^k)\}_{k=1,\ldots,K}$ where no two of the associated branching constraints, $\nu(S^k) \geq L^k$, concern the same column class, $G^k = G(S^k)$. Even then, some branching constraints may be dominated by others:

Definition 8 We say that branching constraint $\nu(S^k) \geq L^k$ is "redundant" if

$$\sum_{l \in dsucc(k)} L^l \geq L^k \tag{41}$$

(using the notation of Definition 4). The "marginal lower bound" imposed on a column class G^k by branching constraint $\nu(S^k) \geq L^k$ is defined as

$$\overline{L}^k = L^k - \sum_{l \in dsucc(k)} L^l.$$

It is strictly positive if and only if the branching constraint is non-redundant.

In example 4, illustrated in Fig. 4, branching constraint (29) is redundant: its marginal lower bound is zero.

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 constraints defining the current branch-and-price node. These simplifications allow us to derive a useful property:

Proposition 7 *The number of non-redundant branching constraints at a branch-and-price node is at most R; otherwise, the node problem is infeasible.*

Proof Assume a branch-and-price node defined by the collection $\{(S^k, L^k)\}_{k=1,\ldots,K}$. The non-redundant branching constraints can be equivalently reformulated as $\nu(C^k) \geq \overline{L}^k$ for $k = 1, \ldots, K$, where class $C^k \subseteq G$ is defined as, $C^k := (G^k \backslash (\cup_{i \in dsucc(k)} G^i))$, and $\overline{L}^k$ is the marginal lower bound which according to Definition 8 must be strictly positive, i.e., $\overline{L}^k \geq 1$ for $k = 1, \ldots, K$. Observe that classes $\{C^k\}_{k=1,\ldots,K}$ are disjoint (because classes $\{G^k\}_{k=1,\ldots,K}$ are nested and all different). Master constraint (12) implies $\nu(G) \leq R$. Hence, the system $\nu(C^k) \geq \overline{L}^k \geq 1$ for $k = 1, \ldots, K$, $\nu(G) \leq R$, is infeasible if $K > R$ (this statement formalizes what is known as the "pigeon hole principle"). □

After elimination of redundant branching constraints, the master program, [M(node)], is solved by column generation using a pricing procedure that relies only on the oracle of the root node. To each non-redundant branching constraint, $\nu(S^k) \geq L^k, k = 1, \ldots, K$, we associate a subproblem, $[SP^k]$, to price the columns of G^k:

$$[SP^k] \quad \zeta^k(\pi) := \min\{(c - \pi A)x : x \in X^k\}, \tag{42}$$

where $X^k := \{x \in \{0,1\}^n : \; B\,x \geq b, \ddot{x}_i = 1 \; \forall i \in S^k\}$. While $[SP^0]$ is defined by (14), or equivalently (42) with $X^0 := X$. Each of these pricing subproblems can be solved with the oracle provided for pricing problem (14): additional constraints consist only in fixing the value of some binary variables. If pricing problem $[SP^k]$ is infeasible (i.e., $X^k = \emptyset$), let $\zeta^k(\pi) = \infty$. The column generation procedure can be carried out by solving each pricing problem (42), for $k = 0, \ldots, K$, and returning columns that have negative reduced costs, i.e., returning $x^k := argmin\ \zeta^k(\pi)$, if $\zeta^k(\pi) - \sum_{l \in pred(k)} \sigma_l < 0$ (using the notation of Definition 4). If none can be found, the current master LP value is optimal.

Observe however that pricing subproblems (42) are not independent. Solving pricing problem l over a class $G^l \supset G^j$ is solving a relaxation of problem j since the objective functions of both problems are identical:

Observation 7 *If $G^l \supset G^j$ (i.e., $S^l \subset S^j$), $\zeta^l(\pi) \leq \zeta^j(\pi)$ (i.e., $\zeta^l(\pi)$ defines a dual bound for $[SP^j]$).*

If the solution to $[SP^l]$, seen as a relaxation of $[SP^j]$, is feasible for $[SP^j]$, then it is optimal for it:

Observation 8 *If $x^l := argmin\ \zeta^l(\pi)$ and $x^l \in G^j \subset G^l$, then $x^l = argmin\ \zeta^j(\pi)$.*

To exploit the relationships that exist between pricing subproblems, $[SP^k]$, column generation can proceed by enumerating subproblems following a breadth first search in the associated tree of column classes of Definition 4. Then, one can interrupt the procedure as soon as a negative reduced cost column is found and still have a dual bound on the reduced cost of unsolved subproblems. Indeed, for the unsolved pricing subproblems, a dual bound is provided by the value of any of their predecessor in the column class tree that has been solved. Such a procedure is outlined in Table 6. As the procedure stops at the first identified negative reduced cost column, the order in which subproblems are treated is important (we use a heuristic rule to break ties in the sorting of step 1, giving priority to subproblem SP^k with the largest $\overline{L}^k \sigma_k$).

8 Lagrangian dual bounds

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

Table 6 The column generation procedure. (Notations $pred(k), dpred(k), dsucc(k)$ are from Definition 4; x^k denotes the solution of subproblem $[SP^k]$, i.e., $x^k = argmin\ \zeta^k(\pi)$.)

1. Sort subproblems, SP^k, following the partial order defined by the precedence relation between the associated column classes in the tree representation of Definition 4.
2. For each pricing subproblem SP^k in the sorted list, do
   (a) If $x^{dpred(k)} \in G^k$, continue ($x^{dpred(k)}$ *also solves* SP^k).
   (b) If $\zeta^{dpred(k)}(\pi) - \sum_{j \in pred(k)} \sigma_j - \sigma_k \geq 0$, continue ($SP^k$ *shall not yield a negative reduced cost column*).
   (c) Compute $\zeta^k(\pi)$ and x^k.
   (d) If $\zeta^k(\pi) - \sum_{j \in pred(k)} \sigma_j - \sigma_k < 0$, return x^k and Stop (x^k *yields a negative reduced cost column*).
   (e) If x^k is feasible for any SP^l with $l \in dsucc(k)$, let $x^l = x^k$. If, moreover, $\zeta^k(\pi) - \sum_{j \in pred(l)} \sigma_j - \sigma_l < 0$, return x^l and Stop (x^l *yields a negative reduced cost column*).

$$\pi\ a + \min \left\{ \sum_{g \in G} (c - \pi\ A)\ x^g\ v_g : \sum_{g \in G} v_g = R; \right.$$
$$\left. \sum_{g \in G^k} v_g \geq L^k\ ,\ k = 1, \ldots, K;\ v_g \geq 0, g \in G \right\}. \tag{43}$$

Because column classes are nested, this problem admits a closed form solution.

Proposition 8 *The solution of problem* (43) *is*

$$\theta(\pi) := \pi\ a + \sum_{k=0}^{K} \overline{L}^k \zeta^k(\pi) \tag{44}$$

where $G^0 = G$, $L^0 = R$, *and* $\overline{L}^k$ *are the marginal lower bounds of Definition 8.*

Proof The result follows from Observation 7 that implies $\sum_{g \in G^k} v_g^* = L^k\ \forall k$ in an optimal solution v^* to (43). More specifically, $v_g^* = \overline{L}^k$ for $g \in G^k$ such that $x^g = argmin\ \zeta^k(\pi)$. □

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 (42). Replacing $\zeta^k(\pi)$ by a dual bound, $\overline{\zeta}^k(\pi)$, on SP^k in (44) yields a valid dual bound for $[M(\text{node})]$. Observation 7 points naturally to a pricing subproblem dual bound, $\overline{\zeta}^k(\pi) = \zeta^l(\pi)$, obtained when pricing on a predecessor class, $l \in pred(k)$. Thus, provided that we enumerate the pricing subproblems following a breadth first search in the associated tree of column classes, an approximation of $\theta(\pi)$ can easily be computed

at any stage of the column generation procedure: $\hat{\theta}(\pi) := \pi\ a + \sum_{k \in G} \overline{L}^k_G \zeta^k(\pi)$, where G is the set of solved subproblems and $\overline{L}^k_G = L^k - \sum_{l \in dsucc(k) \cap G} L^l$.

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

Proposition 9 *Upon completion of the column generation procedure, the dual bound (44) yields the value of a Lagrangian dual problem whose primal formulation is*

$$\min \left\{ \sum_r cx^r : \sum_r Ax^r \geq a,\ x^r \in conv(X^k)\ for\ r = \rho^{k-1}, \ldots, \rho^k - 1,\ k = 0, \ldots, K \right\} \tag{45}$$

where $\rho^{-1} = 1$, *and* $\rho^k = \rho^{k-1} + \overline{L}^k$ *for* $k = 0, \ldots, K$.

Proof Note that formulation $[M(\text{node})]$ given in (36–40) is equivalent to the disaggregated master:

$$\min \left\{ \sum_{k=0}^{K} \sum_{r=\rho^{k-1}}^{\rho^k - 1} \sum_{g \in G^k} c\ x^g\ \lambda^r_g : \sum_{k=0}^{K} \sum_{r=\rho^{k-1}}^{\rho^k - 1} \sum_{q \in G^k} A\ x^g\ \lambda^r_g \geq a; \right.$$
$$\left. \sum_{g \in G^k} \lambda^r_g = 1 \quad \forall k, r = \rho^{k-1}, \ldots, \rho^k - 1;\ \lambda^r_g \geq 0\ \forall r, q \right\}. \tag{46}$$

By Geoffrion [9], (46) has an optimal LP value equal to that of (45). □

Observation 9 *The Lagrangian dual problem value given by* (45) *is tighter than the bound given by* (16) *for which the dualized branching constraints are included in* $\sum_r A\ x^r \geq a$.

9 Preprocessing at a branch-and-price node

The first preprocessing operation at a node consists in deleting redundant branching constraints: keep at most one constraint per class G^k (that with the largest L^k); then delete constraints with no strictly positive marginal lower bounds, i.e., with $\overline{L}^k \leq 0$. The remaining constraints must satisfy $\sum_{k=1}^{K} \overline{L}^k \leq R$, otherwise the node is pruned by infeasibility. For the rest of this section, we include in the collection of branching constraints, the master convexity constraint (12) with index $k = 0$, letting $G^0 = G$, S^0 be the empty sequence, and $\overline{L}^0 = R - \sum_{k \in dsucc(0)} L^k$.

Observe that constraints defined by the collection $\{(S^k, L^k)\}_{k=0,\ldots,K}$ induce bounds on the value of aggregate subproblem variables defined in (19): $(\sum_{k: S^k \ni x_i} \overline{L}^k)$ and $(\sum_{k: S^k \not\ni x_i} \overline{L}^k)$ define respectively a lower and an upper bound on the value of aggregate variable x_i. One must check that these bounds are compatible with the bounds

induced by constraints (3) and (5). The latter are denoted by gl_i and gu_i (for global lower and upper bounds):

$$gl_i \leq \sum_r x_i^r \leq gu_i \quad i = 1, \ldots, n. \tag{47}$$

The current node master problem $[M(\text{node})]$ is infeasible if

$$\sum_{k:S^k \ni x_i} \overline{L}^k > gu_i \quad \text{or} \tag{48}$$

$$\sum_{k:S^k \not\ni \overline{x}_i} \overline{L}^k < gl_i \tag{49}$$

for some i.

On the other hand, if the branching constraints imply that the global upper bound for component i is reached, i.e., if

$$\sum_{k:S^k \ni x_i} \overline{L}^k = gu_i, \tag{50}$$

then the classes where component i is not fixed, $\{k : i \notin S^k\}$, can be augmented with component bound restriction $x_i = 0$. This restriction takes the form of implied branching constraints: given a component i satisfying (50), we define a new branching constraint,

$$\nu(S^k, \overline{x}_i) \geq L^k - \sum_{l \in dsucc(k):S^l \ni x_i} L^l, \tag{51}$$

for all $k : S^k \not\ni i$ (the notation $dsucc(.)$ is from Definition 4). Observe that constraint (51) makes branching constraint k redundant and hence it simply replaces it. Also note that this operation preserves the nested partition property. Indeed, once all branching constraints $k : S^k \not\ni i$ are processed in this way, all the redefined branching sequences S^k for $k = 1, \ldots, K$ include a component bound on i (i.e., either x_i or $\overline{x}_i$). 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.

Reciprocally, if the global lower bound on a component i can only be met by setting $x_i^g = 1$ in all columns of classes, $k : S^k \not\ni i$, i.e., if $\sum_{k:S^k \not\ni \overline{x}_i} \overline{L}^k = gl_i$, then, branching constraints indexed by $k : S^k \not\ni i$ can be reset as

$$\nu(S^k, x_i) \geq L^k - \sum_{l \in dsucc(k):S^l \ni \overline{x}_i} L^l \tag{52}$$

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 G is redundant, i.e., when $\sum_{k\in dsucc(root)} L^k = R$, columns in $G\setminus\cup_{k\in dsucc(root)} G^k$ can be deleted from the formulation. Such preprocessing can be generalized to any class k such that

$$\sum_{l\in dsucc(k)} L^l = U^k, \tag{53}$$

where U^k is a valid upper bound on class k value. Then, columns of $G^k\setminus\cup_{l\in dsucc(k)} G^l$ can be deleted. The upper bound U^k can be set for instance to

$$\min\left\{R - \sum_{l:l\neq k, l\notin succ(k)} \overline{L}_l,\ \min_{i:x_i\in S^k}\left\{gu_i - \sum_{l:l\neq k, l\notin succ(k), S^l\ni x_i} \overline{L}_l\right\},\right.$$
$$\left.\min_{i:\overline{x}_i\in S^k}\left\{R - \sum_{l:l\neq k, l\notin succ(k), S^l\ni \overline{x}_i} \overline{L}_l - gl_i\right\}\right\}.$$

Note that, as $L^k \leq U^k$, (53) implies that $\overline{L}^k \leq 0$, i.e., class G^k is associated with a redundant branching constraint. Hence, in our pricing procedure, we will not generate columns from class $G^k\setminus\cup_{l\in dsucc(k)} G^l$, but we shall only consider its active subclasses, G^l for $l \in succ(k)$.

10 The set partitioning special case

When the master program is a set partitioning problem of the form (21), preprocessing is particularly effective as global bounds (47) are tight: $gl_i = gu_i = 1\ \forall i$. Example 9 illustrates this on the bin packing problem.

Example 9 The bin packing problem takes the form (21) where G is the set of solutions to a binary knapsack problem:

$$X = \left\{x \in \{0,1\}^n : \sum_i w_i x_i \leq W\right\}. \tag{54}$$

Let $n = 5$, $w = (6, 4, 3, 2, 7)$, $W = 10$ and $R = 3$. The generators are:

g	1	2	3	4	5	6	7	8	9	10	11	12	13	14
x_1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
x_2	1	0	0	1	1	1	1	0	0	0	0	0	0	0
x_3	0	1	0	1	1	0	0	1	1	1	0	0	0	0
x_4	0	0	1	1	0	1	0	1	0	0	1	1	0	0
x_5	0	0	0	0	0	0	0	0	1	0	1	0	1	0

The master LP solution is $\frac{7}{3}$ with $\tilde{\nu}_1 = \frac{2}{3}, \tilde{\nu}_3 = \tilde{\nu}_4 = \frac{1}{3}, \tilde{\nu}_9 = \frac{2}{3}, \tilde{\nu}_{11} = \frac{1}{3}, \tilde{\nu}_{14} = \frac{2}{3}$. Then, branching on $S =< x_1, x_2 >$ yields 3 nodes:

$$\begin{aligned} \text{Node}(1) &\equiv \nu(\overline{x}_1) \geq 3; \\ \text{Node}(2) &\equiv \nu(x_1, \overline{x}_2) \geq 1; \\ \text{Node}(3) &\equiv \nu(x_1, x_2) \geq 1. \end{aligned}$$

Preprocessing allows us to prune Node(1) by infeasibility (by (49) for $i = 1$). 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 $G(x_1, x_2)$, we can add an implied branching constraint $\nu(\overline{x}_1) \geq R - 1 = 2$. Similarly item 2 is entirely covered by class $G(x_1, x_2)$. Thus, the branching constraints are redefined as $\nu(x_2, x_1) \geq 1$ and $\nu(\overline{x}_2, \overline{x}_1) \geq 2$. Then, class G becomes redundant and columns 2 to 7 can be deleted. Now, the master solution at Node(3) is $\frac{5}{2}$ with $\tilde{\nu}_1 = 1, \tilde{\nu}_8 = \tilde{\nu}_9 = \tilde{\nu}_{11} = \tilde{\nu}_{14} = \frac{1}{2}$. Assume that for separation we choose $S =< \overline{x}_2, \overline{x}_1, x_3, x_4 >$. Although $|S| = 4$, only three successor nodes need to be defined according to Proposition 3. Indeed, Node (3.1) and (3.2) would only aim at enumerating solutions where $\nu(\overline{x}_2, \overline{x}_1) < 2$ which contradicts the previous constraint $\nu(\overline{x}_2, \overline{x}_1) \geq 2$. Hence, there remain

$$\text{Node}(3.3) \begin{cases} \nu(x_2, x_1) \geq 1 \\ \nu(\overline{x}_2, \overline{x}_1) \geq 2 \\ \nu(\overline{x}_2, \overline{x}_1, \overline{x}_3) \geq 2 \end{cases};$$

$$\text{Node}(3.4) \begin{cases} \nu(x_2, x_1) \geq 1 \\ \nu(\overline{x}_2, \overline{x}_1) \geq 2 \\ \nu(\overline{x}_2, \overline{x}_1, x_3, \overline{x}_4) \geq 1 \end{cases};$$

$$\text{Node}(3.5) \begin{cases} \nu(x_2, x_1) \geq 1 \\ \nu(\overline{x}_2, \overline{x}_1) \geq 2 \\ \nu(\overline{x}_2, \overline{x}_1, x_3, x_4) \geq 1 \end{cases}.$$

In Node(3.3), the second constraint is made redundant by the third. Applying (52), the constraints become $\nu(x_3, x_2, x_1) \geq 1$ and $\nu(\overline{x}_3, \overline{x}_2, \overline{x}_1) \geq 2$. The former constraint makes the problem infeasible, as $G(x_3, x_2, x_1) = \emptyset$. In Node (3.4), preprocessing leads to redefining the branching constraints as $\nu(\overline{x}_3, x_2, x_1) \geq 1$, $\nu(\overline{x}_3, \overline{x}_2, \overline{x}_1) \geq 1$ and $\nu(x_3, \overline{x}_2, \overline{x}_1, \overline{x}_4) \geq 1$. While, in Node (3.5), they take the form $\nu(\overline{x}_4, \overline{x}_3, x_2, x_1) \geq 1$, $\nu(\overline{x}_4, \overline{x}_3, \overline{x}_2, \overline{x}_1) \geq 1$ and $\nu(x_4, x_3, \overline{x}_2, \overline{x}_1) \geq 1$. Then, Node (3.4) and Node (3.5) have both an integer master LP solution.

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

Proposition 10 *When the master program is a set partitioning problem of the form (21), our branching scheme has the following properties (after preprocessing):*

(i) $L^k = 1$ *for all branching constraints k such that* S^k *includes a "setting-to-one bound", i.e., for all* $k : S^k \ni x_i$ *for some* i.

(ii) *The tree of column classes has depth 1, with* $K - 1$ *classes that involve at least one "setting-to-one bound" (each of these classes has its own set of "setting-to-one bounds") and one class, say* G^1, *that involves only "setting-to-zero*

bounds"; more specifically, $G^1 = G(\{\overline{x}_i : x_i \in S^k \text{for some } k \in \{2, \ldots, K\}\})$ *with* $L^1 = R - \sum_{k=2}^{K} L^k$.

(iii) $\sum_{k=1}^{K} \overline{L}^k = R$ *and hence columns of* $G \backslash (\cup_k G^k)$ *can be deleted.*

(iv) *A fractional solution* $\tilde{\nu}$ *satisfies all branching constraints at equality, i.e.,* $\tilde{\nu}(G^k) = L^k$ *for all* k. *Hence, eliminating it requires further splitting an existing column class.*

(v) *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.*

(vi) *When further splitting a class that involves "setting-to-zero-bounds" only (*G^1 *in our notation), identifying a set* S *on which to branch requires at most two "setting-to-one-bounds" and there are at most three successor nodes.*

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

(ii) A class that involves a setting-to-one-bound cannot have a non-redundant predecessor class whose definition also involves setting-to-one-bounds, because both would have bound $L^k = 1$. If $x_i \in S^k$, then, as $L^k = g u_i = 1$, $\overline{x}_i$ is added to all other class definitions. In particular adding bounds $\overline{x}_i$ to class G yields G^1. By construction, G^1 is the only class that does not involves a setting-to-one-bound and G^1 has no successor.

(iii) is a direct consequence of (ii).

(iv) In a fractional solution to the master, all column classes have integer value because L^k is both a lower bound and an upper bound on the class (even for class G^1).

(v) Assume $\exists k : x_i \in S^k$, $\tilde{\nu}(G^k) = 1$, and $|F^k| > 1$ where $F^k = \{g \in G^k : \tilde{\nu}_g - \lfloor \tilde{\nu}_g \rfloor > 0\}$. As all columns in F^k are different, $\exists j : j \notin S^k$ such that $0 < \tilde{\nu}(S^k, x_j) < 1$. Proposition 3 implies that only two successor nodes need to be defined.

(vi) Say $\tilde{\nu}(G^1) = L^1 > 0$. Let $I(G^k) = \{i : \sum_{g \in G^k} x_i^g \, (\tilde{\nu}_g - \lfloor \tilde{\nu}_g \rfloor) > 0\}$. Consider two cases: either $I(G^1) \cap I(G^k) = \emptyset$ for all $k = 2, \ldots, K$ or not. In the latter case, one could branch as in case (iv). But one can also split G^1: a single setting-to-one-bound, x_i, suffices to define a set $S =< S^1, x_i >$ on which to branch, as one can simply select some component i in $I(G^1) \cap I(G^k)$ for some k. In the former case where $I(G^1) \cap I(G^k) = \emptyset$ for all $k = 2, \ldots, K$, we have $\tilde{\nu}(S^1, x_i) = 1$ for any $i \in I(G^1)$. But, as all columns are different, $\exists j \in I(G^1), j \neq i : 0 < \tilde{\nu}(S^1, x_i, x_j) < 1$. Choose $S =< S^1, x_i, x_j >$ for such pair (i, j). By Proposition 3, descendant nodes $1, \ldots, |S| - 2$ need not be generated as they are enumerating solutions where $\nu(G^1) < L^1$. □

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. However, the depth of our tree is $O(n\ R)$ while that of Ryan and Foster's scheme is $O(n^2)$ (note that $R \leq n$). Columns can be deleted after branching in both schemes. The main difference therefore is that the pricing problem is solved

by enumerating the active subproblems in our scheme while, under Ryan and Foster's scheme, a modified subproblem is solved that can become intractable because of the extra constraints resulting from branching. Moreover, our intermediate Lagrangian bounds (44) 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 10 and informally stated in Observation 11.

Example 10 Assume that we are at a branch-and-bound node defined under Ryan and Foster's scheme by branching constraints

$$\nu(x_1, x_2) \geq 1, \tag{55}$$
$$\nu(x_2, x_3) \geq 1, \tag{56}$$
$$\nu(x_4, x_5) \leq 0. \tag{57}$$

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 $\nu(x_4, \overline{x}_5) \geq 1$ or symmetrically, $\nu(\overline{x}_4, x_5) \geq 1$. Under our scheme the "equivalent" node problem could be defined by branching constraints:

$$\nu(x_1, x_2, x_3, \overline{x}_4) \geq 1 \tag{58}$$
$$\nu(\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4, \overline{x}_5) \geq 1 \tag{59}$$
$$\nu(\overline{x}_1, \overline{x}_2, \overline{x}_3, \overline{x}_4) \geq R - 2 \tag{60}$$

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

$$X^1 = X \cap \{x : x_1 = x_2 = x_3 = 1, x_4 = 0\},$$
$$X^2 = X \cap \{x : x_1 = x_2 = x_3 = 0, x_4 = 1, x_5 = 0\},$$
$$X^3 = X \cap \{x : x_1 = x_2 = x_3 = x_4 = 0\}.$$

Note that $X^k \subseteq \hat{X}$ for $k = 1, 2, 3$.

Observation 10 *Under Ryan and Foster's scheme, intermediate dual bounds at a branch-and-price node take the form*

$$\theta^{RF}(\pi) = \pi\, a + R\hat{\zeta}(\pi) \tag{61}$$

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

$$\min \left\{ \sum_r c\, x^r : \sum_r A\, x^r \geq a, x^r \in conv(\hat{X}) \text{ for } r = 1, \ldots, R \right\}, \tag{62}$$

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

Observation 11 *Assuming an "equivalent" set of branching constraints ("equivalence" is only loosely defined by way of Example 10), intermediate bounds (44) dominate (61) and the Lagrangian dual bound (45) dominates (62).*

Indeed, for all X^k used in the expression of (45), we have $X^k \subseteq \hat{X}$ (X^0 needs not be considered since Proposition 10 says that class G is redundant) and therefore $\hat{\zeta}(\pi) \leq \zeta^k(\pi)$ and $conv(X^k) \subseteq conv(\hat{X})$ for all k.

11 Extensions

The above presentation of our branching scheme assumes a pure binary subproblem, and a master convexity constraint of the form $\sum_{g \in G} \nu_g = R$. Extensions are possible beyond these assumptions.

First consider the case where subsystems are general mixed integer programs. Then, the mapping of Table 1 becomes a useful tool to check whether a master solution implicitly defines an integer solution for the original formulation: the mapped solution $\tilde{x}$ could define an integer solution even though $\tilde{\nu}$ was fractional. The lexicographic ordering remains well defined in the presence of non-binary integer variables or continuous variables. Column classes are defined by integer bounds on the integer variables, e.g., $x_i \leq \lfloor \alpha \rfloor$ or $x_i \geq \lceil \alpha \rceil$. In the mixed integer case, it is indeed sufficient to impose integrality condition on integer subproblem variables (see [25]). A "component bound sequence" S, is a sorted list of triplets including the index of the integer component, the sense of the inequality (upper or lower bound), and the value of the bound. The extension of routine "Separate" of Table 2 implies choosing a bound value for a fractional mapped component x_i (a balanced partition of the column set can be achieved by selecting the median value of those encountered in fractional columns); moreover, one must leave the index i as a potential component for further separation in the recursive calls to "Separate". For the preprocessing of Sect. 9, the lower and upper bounds on aggregate variables take the form $(\sum_k \underline{b}_i^k \overline{L}^k)$ and $(\sum_k \overline{b}_i^k \overline{L}^k)$, where $\underline{b}_i^k$ and $\overline{b}_i^k$ define respectively the value of the lower and upper bounds on x_i in column class k. When the class defines no component bound on x_i, the default bound values (5) are used.

In the case of convexity constraints of the form $L \leq \sum_{g \in G} \nu_g \leq U$, one needs to check whether $\nu(G)$ is fractional. If so, branching starts by fixing the number of columns used in the solution. Then, in Definition 3, the branching scheme based on component sequence S must include an extra node defined by a branching constraint of the form $\nu(G) \leq L^0 - 1$, which yields $|S|+2$ descendant nodes in total. The separation routine of Table 2 shall then return class G as the branching set as long as $\nu(G) \notin \mathbb{N}$.

Standard disjunctive branching on class G implicitly serves the purpose of enumerating different possible values for R. As Proposition 2 holds for each of those R, the result extends to the case of convexity constraints of the form $L \leq \sum_{g \in G} \nu_g \leq U$.

12 Numerical testing

The proposed branching scheme is implemented in our generic branch-and-price code prototype, called *BaPCod* [24]. The current implementation is relatively basic: it does not include all the preprocessing features of Sect. 9 and does not exploit computation done at previous branch-and-price nodes (column class tree, ILO sorting, and the like are computed from scratch). Computing times also include important overhead for several validity checks such as previous existence of generated columns. Tests on the cutting stock problem (CSP) and its special case, the bin packing problem (BPP) have been carried out on a PC bi-pro dual-core Intel(R) Xeon(R) X5460 3.16 GHz with 16GB of RAM under Scientific Linux 5.0. This experimentation illustrates the fact that our proposal is not just a theoretic scheme, but one that behaves relatively well in the computational practice. In particular, our results show that (i) our non-binary branching tree does not grow too large in practice (as predicted by our theoretical arguments) when compare to a more "standard" binary scheme, (ii) that the routines used for the separation of fractional solutions are not too cumbersome (although our rough implementation becomes time consuming on large problems), and (iii) that the increase in the number of calls to the pricing oracles is significant, but the resulting increase in overall time spent in the oracle does not grow in the same proportion since these extra subproblems are easier (this is not reflected in our numerical results because the reported time spent in the pricing procedure includes overhead for managing the tree of subproblems).

To benchmark our results, we compare them with the closest alternative branching scheme that could be used for the CSP and the BPP. Of course, the efficiency of a branching scheme is very sensitive to parametrization (branching priorities and directions). We did not attempt to optimize the parametrization, but we use the same framework for all comparisons. Moreover, we do not make any use of primal heuristics and rely only on the branching scheme to produce integer solutions. A full blown comparison/competition between the branching schemes proposed in the literature for these problems would require competitive implementations using the same framework, parameter tuning and combination with primal heuristics. This is beyond the scope of this paper. Our numerical experiment only aims at demonstrating the practical viability of our theoretical proposal.

The standard column generation formulation for the bin packing problem is given in Example 9. Its generalization to cutting stock problem involves integer right-hand-sides $d_i \in \mathbb{N}$ for the master constraint and a bounded integer knapsack as pricing problem:

$$\max \left\{ \sum_i \pi_i x_i \, : \, \sum_i w_i x_i \leq W \, ; \, 0 \leq x_i \leq u_i \ \forall i, x \in \mathbb{N}^n \right\} \tag{63}$$

where $u_i = \min\{d_i, \left\lfloor \frac{W}{w_i} \right\rfloor\}$. The subproblem can be transformed into a binary problem with class bound [22]:

$$\max \left\{ \sum_i \sum_{j=1}^{n_i} \pi_i \, 2^j \, x_{ij} : \sum_i \sum_{j=1}^{n_i} w_i \, 2^j \, x_{ij} \le W, \ \sum_{j=1}^{n_i} 2^j x_{ij} \le u_i \ \forall i, \ x_{ij} \in \{0,1\} \forall i j \right\} \tag{64}$$

where $x_{ij} = 1$ *iff* the binary component of multiplicity 2^j associated with item i is selected, and $n_i = \lfloor \log_2 u_i \rfloor + 1$. If one uses the latter extended formulation, one can branch using standard binary disjunctive branching constraint of the form

$$\sum_{g \in G} x_{ij}^g \, v_g \le \lfloor \alpha \rfloor \qquad \text{or} \qquad \sum_{g \in G} x_{ij}^g \, v_g \ge \lceil \alpha \rceil. \tag{65}$$

Constraints (65) enforce the integrality of the values of aggregate variables x_{ij}. They induce modified item costs, $\pi_{ij} \neq \pi_i \, 2^j$. Problem (64) can be solved using the branch-and-bound algorithm of Ref. [22] that can handle item costs $\pi_{ij} \neq \pi_i \, 2^j$ (the algorithm is a generalization of the standard branch-and-bound approach of Horowitz and Sahni [10] for the 0-1 knapsack).

We compare the branching scheme of this paper to branching rule (65) that was initially tested in Ref. [19]. In theory, this rule alone does not suffice to obtain an integer solution; but experimentally it does for some instances. When it does not, [19] completes it by more complex rules of the generic scheme of Ref. [20], but these lead to pricing problem modifications. Here, we shall simply stop the algorithm when no more branching constraints of the form (65) can be found even though the solution might still be fractional. The size of the branch-and-price tree obtained so far defines a lower bound on what would be the size of the complete tree. When (65) suffices to achieve integrality, our comparative results show that the size of branch-and-price tree obtained with the newly proposed scheme is of the same order of magnitude. The same remark holds for computing times.

Table 7 presents our numerical results for the CSP. We compare three branching schemes. (*i*) We use branching rule (65), which we denote AG. Notation AG refers to branching on the aggregate value of the original variables. (*ii*) We apply the scheme of the present paper, setting component bounds on binary variables x_{ij} from the above 0–1 form (64) of the knapsack subproblem, which we denote CS(x_{ij}). Notation CS refers to Component bound Sequence. (*iii*) We apply our new scheme setting component bounds directly on integer variables x_i, which we denote CS(x_i) (we implemented the extension of the branching scheme to the non-binary case as described in Sect. 11). We use the same oracle (the branch-and-bound algorithm of Ref. [22]) in all three cases, setting the knapsack pricing problem in the form (64) although we could use a standard binary or integer knapsack solver when using the scheme of the present paper. The master is initialized with artificial columns (vectors of the canonical basis associated with the item covering constraints) in all cases. Note that, for the CSP, the master formulation does not include convexity constraint (12). Hence, we have

implemented the generalization of our branching scheme by which we first branch on $\nu(G)$ if the latter is fractional. The only feasible branch is the round-up branch, where $\sum_{g\in G} \nu_g \geq R = \lceil Z^M_{\mathrm{LP}} \rceil$. [1]

For our test we use the randomly generated instances of Ref. [19]. There are 6 classes of instances characterized by the item weight distribution and the average item demand $\overline{d}_i$: for class 1 $w \sim U[1,7500]$, $d_i \sim U[1, 100]$, and hence $\overline{d} = 50$; for class 2, $w \sim U[1,5000]$ and $\overline{d} = 50$, for class 3: $w \sim U[1, 2500]$ and $\overline{d} = 50$; for class 4a: $w \sim U[500,5000]$ and $\overline{d} = 50$; for class 4b, $w \sim U[500,5000]$, $d_i \sim U[1,200]$, and hence $\overline{d} = 100$; for class 5 $w \sim U[500,2500]$ and $\overline{d} = 50$. In all cases, the knapsack capacity is set to $W = 10000$. There are 20 instances per class with $n = 50$ items (the data are available on http://hal.inria.fr/inria-00311274/fr/). Class 5, that involves medium size items, is the hardest for branching. For this class, we increase the number of items up to $n = 200$, with $\overline{d} = 50$, to show how the performance of our scheme evolves with the instance size. The associated results are average over 10 random instances for $n \in [60,100]$ and 5 instances for $n > 100$. In all these tests, when selecting variables for branching, priority is given to the component i with largest width weighted by its fractional part and bound value, i.e., $\text{priority}_i = w_i * (x_i - \lfloor x_i \rfloor) * \lceil x_i \rceil$. The branch-and-price tree is searched using depth first priority amongst node with the minimum dual bound.

Table 7 presents the average results for each class or size. The table reports the branching rule used (*br-rule*), the number of nodes (*nod*) in the branch-and-price tree, its depth (*dep*), the best dual bound (*DB*), the number of calls to the oracle (*#Sp*), the number of columns generated (*#Col*), the time spent in solving the master (*tMAST*) with Xpress-MP LP solver [27], the time spent in the pricing procedure of Table 6 (*tSP*), the time spent for separation in subroutine Explore (*tEX*), the overall time (*tTotal*), and the number of instances solved to integer optimality out of 20, or 10 when $n > 50$, or 5 when $n > 100$ (this number is in bold face when some instances could not be solved to optimality). **Time units are ticks** (1 tick $= \frac{1}{100}$ of a second). The comparison between branching rule CS and AG is somewhat biased by the fact that under AG the hardest problems have not been solved to integer optimality. Nevertheless, it seems to indicate that the size of the AG branch-and-price tree is at least of the same order of magnitude as that with CS(x_i) branching. Comparing CS(x_{ij}) to CS(x_i) shows that the extension of our new scheme to the integer case gives rise to a scheme with good practical performance. The size of the branch-and-price tree varies

[1] There is a technical remark regarding our theoretical result on the size of the branch-and-price tree: we proved that the depth of the tree is polynomial in n and R, but, for the CSP, R is not polynomial in the input size. Thus, the so-called original formulation in its form (2–6) is not compact neither. These considerations remain theoretical. In fact, the number of different cutting patterns used in an optimum solution, which we may denote R', is polynomial in the input size as proven in Ref. [7]. R' is much smaller than R when item demands d_i are large. To satisfy the theoretical need for a polynomial depth of the branch-and-bound tree, we could use an alternative original formulation that is compact and an associated Dantzig–Wolfe reformulation with a polynomial number of columns R', where a column is defined by a cutting pattern and its multiplicity in the solution [21]. However, even if we use the traditional formulation (that of Example 9 with integer right-hand-sides $d_i \in \mathbb{N}$), the depth of the branch-and-price tree will be experimentally in $O(nR')$. There are typically $O(R')$ non-zero variables in master solution and non-zero ν_g have value much larger than 1. Hence, the (S, L)-frames defined through branching tend to have a large width and ensure the covering of several x^r_i components at a time (and implicitly fix them to an integer value).

Table 7 Comparative results for cutting stock problems

inst-size/class	br-rule	nod	dep	DB	#Sp	#col	tMAST	tSP	tEX	tTotal	opt
	AG	45	30	992.3	165	132	27	36		104	**12**
$n = 50$, C1	CS(x_{ij})	33	30	992.3	185	135	13	42	0	91	20
	CS(x_i)	36	35	992.35	153	137	13	37	1	85	20
	AG	343	149	635.5	586	180	419	138		980	**3**
$n = 50$, C2	CS(x_{ij})	148	143	635.5	535	249	67	405	29	929	20
	CS(x_i)	140	137	635.5	356	232	84	202	20	610	20
	AG	911	245	317.05	3217	333	2144	291		6207	**1**
$n = 50$, C3	CS(x_{ij})	277	243	317.05	4778	718	256	3538	92	5672	20
	CS(x_i)	270	260	317.05	3825	742	303	1525	87	3715	20
	AG	93	65	690.9	180	146	76	144		317	**7**
$n = 50$, C4a	CS(x_{ij})	82	79	690.9	221	170	37	251	9	428	20
	CS(x_i)	74	72	690.9	231	169	38	237	7	391	20
	AG	119	86	1381.1	184	148	83	216		426	**14**
$n = 50$, C4b	CS(x_{ij})	114	91	1381.1	229	161	39	309	12	566	20
	CS(x_i)	100	98	1381.1	187	160	38	263	10	459	20
	AG	305	204	372.6	617	277	450	98		960	**0**
$n = 50$, C5	CS(x_{ij})	455	190	372.6	3584	521	300	2911	116	6318	20
	CS(x_i)	260	245	372.6	5060	649	355	2392	82	4460	20
	AG	401	269	450.4	669	340	701	154		1514	**0**
$n = 60$, C5	CS(x_{ij})	251	235	450.4	4075	673	304	5740	105	7879	10
	CS(x_i)	321	308	450.4	7110	811	527	3485	173	7451	10
	AG	436	304	525.8	722	414	952	193		1995	**0**
$n = 70$, C5	CS(x_{ij})	340	296	525.8	6692	811	492	9210	231	13393	10
	CS(x_i)	346	341	525.8	5530	923	662	6598	246	11577	10
	AG	453	312	609.8	801	481	1275	268		2568	**0**
$n = 80$, C5	CS(x_{ij})	378	337	609.8	6504	996	789	14127	315	20452	10
	CS(x_i)	417	389	609.8	15128	1173	1308	10726	396	19166	10
	AG	521	370	681.8	982	558	1695	369		3411	**0**
$n = 90$, C5	CS(x_{ij})	1113	377	681.8	14142	1184	1601	22651	923	51938	10
	CS(x_i)	504	481	681.8	17875	1430	1902	11562	681	26225	10
	AG	560	391	764.3	991	617	2268	454		4309	**0**
$n = 100$, C5	CS(x_{ij})	1946	430	764.3	36212	1335	4245	40968	2589	141981	**8**
	CS(x_i)	496	489	764.3	20084	1637	2230	19401	874	36268	10
$n = 120$, C5	CS(x_i)	605	588	925.4	21925	2123	3767	30854	1452	59691	5
$n = 140$, C5	CS(x_i)	997	777	1095.2	146900	3158	8778	67536	4030	185442	5
$n = 160$, C5	CS(x_i)	1186	912	1252.4	112044	3681	9475	81810	5551	205453	5
$n = 180$, C5	CS(x_i)	1227	1106	1395.6	283865	4451	16215	105542	9658	356423	5
$n = 200$, C5	CS(x_i)	1036	1025	1558.6	111334	4109	14273	94214	11375	305741	5

bold values emphasize cases where all instances could not be solved to optimality

Table 8 Comparative results for bin packing problems

inst-size	br-rule	nod	dep	DB	#Sp	tMAST	tSP	tEX	tTotal	opt
$n = 100$	CS	27	26	15.5	1411	630	113	5	1583	10
$n = 100$	FL	70	69	15.5	7491	1005	310		5367	10
$n = 150$	CS	39	38	23	2857	3220	284	11	7026	10
$n = 150$	FL	104	103	23	19037	4488	1485		25337	10
$n = 200$	CS	51	50	30.5	5045	15197	608	23	26966	10
$n = 200$	FL	148	147	30.5	40079	20871	10272		89749	10
$n = 250$	CS	63	62	38.2	8095	46682	1184	35	79865	10
$n = 250$	FL	185	184	38.2	45507	31996	6106		108008	10
$n = 300$	CS	72	71	46	11090	186470	1765	49	233832	5
$n = 300$	FL	237	236	46	116783	165018	306455		768864	5
$n = 350$	CS	85	84	54.3	17858	588988	3270	72	719174	3
$n = 400$	CS	96	95	61.6	20409	1796555	4393	98	1955008	3

more widely from one instance to the next one under CS(x_{ij}): in the case $n = 100$, the node limit of 10000 was reached for 2 instances out of 10. We tested CS(x_i) on larger instances. The average tree size is not strictly increasing with n because of the variations that arise from one instance to the next.

On pure bin packing instances (when $d_i = 1\ \forall i$), the above branching rules CS(x_{ij}) and CS(x_i) are the same, since $x_i \in \{0, 1\}$ in the knapsack subproblem, while AG does not permit any cutting of fractional solutions. The only branching scheme proposed in the literature for the BPP that does not require any specific oracle is the approach of Ref. [26], but it suffers from a symmetry drawback. Instead, we reformulate the BPP as a Facility Location (FL) problem:

$$\min \left\{ \sum_{i=1}^{n} x_{ii} : \sum_{i=1}^{n} x_{ij} = 1\ \forall j, \sum_{j=1}^{n} w_j x_{ij} \leq W\ x_{ii}, x_{ij} \in \{0, 1\}\ \forall (i, j) \right\},$$

where $x_{ij} = 1$ if item j is in the same bin as item i (it is only defined if $i \leq j$) and $x_{ii} = 1$ if a bin labelled by item i (seen as a facility) is open. The master is set up in the form (8) with n binary knapsack subproblems of decreasing size. Branching is then performed on the value of aggregate variables x_{ij} of this extended formulation. This approach is denoted by FL. To the best of our knowledge, the FL approach has not been tested in previous work reported in the literature. Computational tests are carried out under the same framework as for the CSP (same initialization, no primal heuristics, depth first search amongst nodes with minimum dual bound). Table 8 reports average comparative results on 10 randomly generated instances for $n = 100$ to $n = 250$, 5 randomly generated instances for $n = 300$, and 3 randomly generated instances for $n = 350$. The item width is drawn uniformly in $[500, 2500]$ and $W = 10,000$ (class 5); all items are different. The comparison between the CS scheme and the FL

approach shows that the new scheme yields a significant reduction in tree size, number of subproblems that are solved and computing time.

Conclusion

We have detailed a generic branching scheme for use when applying a Dantzig–Wolfe decomposition approach to a problem with identical subsystems. The pricing problem after branching decomposes into independent subproblems associated with each column class, each of which can be solved with the oracle provided for the pricing problem of the root node. The scheme relies on four novel features: (i) a dynamic nested partition of the generator set, (ii) an implicit grouping of subsytems to avoid individual r-indices that yield symmetry, (iii) an exclusive use of branching constraints that enforce lower bounds on column classes to guarantee a polynomial number of active column classes (the branching disjunction may require more than two branches), and (iv) a tree-search enumeration of active pricing subproblems that may permit pruning or early termination. The proposed branching scheme is shown to have good theoretical properties both in terms of achieving integrality (as each branching constraint amounts to implicitly fixing a bound on some variable of the original formulation) and in terms of improving dual bounds (the Lagrangian dual bounds are shown to be equivalent to solving an LP over a polyhedron where both sets of subproblem constraints and branching constraints are convexified). The worst case size of the branch-and-price tree is not worse than if one had implemented branching in the compact space of the original variables.

Computational testing on cutting stock and bin packing problems seems to indicate that the proposed scheme is a practical way to get to integrality while not modifying the structure of the pricing problem and avoiding symmetry. Observe that every previous approach for these problems required either modification of the pricing problem or the use of an extended variable space (which require in turn a specific pricing problem solver). These computations also illustrate that the generalization of the proposed scheme to the case of a non-binary integer subproblem and generalized master convexity constraints works fine.

Our generic scheme assumes that adding bounds on the subproblem variables does not impair its solution. In some applications the oracle cannot handle bounds on the subproblem variables and fixing some variables may destroy the subproblem structure (as fixing an arbitrary arc in a constrained shortest path subproblem). However, there might be ad-hoc ways to get around this issue by being more prescriptive for the selection of branching set. For instance, the so called "path-splitting" branching rule used for multi-commodity flow in Ref. [1] or for the Vehicle Routing Problems with split delivery in Ref. [8] can be understood as a special case of our scheme where one branches on a subset of binary variables associated with arcs where the sequence of arcs that are fixed to 1 must form an elementary path (this restriction is enforced at the stage of separating a fractional solution). Branching constraints that bound the number of routes that use a common starting path are consistent with the use of a resource constrained shortest path oracle.

Acknowledgments We are very grateful to Ph. Meurdesoif, P. Pesneau, Y. Pochet, E. Uchoa and B. Vignac for their comments on the early versions of this work and to A. Miller for his great help in improving the final form. We express deep thanks to the anonymous referees and the associated editor for their constructive remarks that have had a significant influence on the revision of this paper and for their patience in mentioning typos. A special thank goes to R. Butel, the engineer who set up the platform for our computational tests. A Xpress-MP license was granted to us by Fair Isaac in the context of the Dash's Academic Partner Program.

References

1. Barnhart, C., Hane, C.A., Vance, P.H.: Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. Oper. Res. **48**(2), 318–326 (2000)
2. Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H.: Branch-and-price: column generation for huge integer programs. Oper. Res. **46**, 316–329 (1998)
3. Belov, G., Letchford, A.N., Uchoa, E.: A node-flow model for the 1D stock cutting: Robust branch-cut-and-price. Tech. report RPEP, vol. 5, no. 7. Universidade Federal Fluminense (2005)
4. Briant, O., Lemaréchal, C., Meurdesoif, Ph., Michel, S., Perrot, N., Vanderbeck, F.: Comparison of bundle and classical column generation. Math. Progr. **113**(2), 299–344 (2008)
5. Chabrier, A.: Génération de Colonnes et de Coupes utilisant des sous-problèmes de plus court chemin. Thèse de doctorat, Université d'Angers (2003)
6. Desaulniers, G., Desrosiers, J., Langevin, A., Marcotte, O., Savard, G., Soumis, F., Solomon, M.: GENCOL, a mathematical optimizer. http://www.crt.umontreal.ca/~gencol/gceng.html (1990)
7. Eisenbrand, F., Shmonin, G.: Carathéodory bounds for integer cones. Oper. Res. Lett. **34**(5), 564–568 (2006)
8. Gendreau, M., Dejaxm, P., Feillet, D., Gueguen, C.: Vehicle routing with time windows and split deliveries, Working Paper (2005)
9. Geoffrion, A.: Lagrangian relaxation for integer programming. Math. Prog. St. **2**, 82–114 (1974)
10. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. J. ACM **21**, 277–292 (1974)
11. Ladányi, L., Ralphs, T.K., Trotter, L.E. Jr. : Branch, cut, and price: sequential and parallel. In: Jünger, M., Naddef, D. (eds.) Computational Combinatorial Optimization, pp. 223–269. Springer, New York (1998)
12. Parker, M., Ryan, D.M.: A column generation algorithm for brandwidth packing. Telecommun. Syst. **2**, 185–195 (1994)
13. Puchinger, J., Stuckey, P.J., Wallace, M., Brand, S.: From high-level model to branch-and-price solution in G12. *CPAIOR 2008*, The Fifth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, vol. 5015 of LNCS, pp. 218–232. Springer, New York (2008)
14. Ryan, D.M., Foster, B.A.: An integer programming approach to scheduling. In: Wren, A. (eds.) Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling, pp. 269–280. North-Holland (1981)
15. Savelsbergh, M.W.P., Nemhauser, G.L.: Functional description of MINTO, a mixed INTeger optimizer. Report COC-91-03A, Georgia Institute of Technology, Atlanta (1993)
16. Thienel, S.: ABACUS—a branch-and-cut system. Ph.D. Thesis, Universität zu Köln (1995)
17. Valério de Carvalho, J.: Exact solution of bin packing problems using column generation and branch-and-bound. Ann. Oper. Res **86**, 629–659 (1999)
18. Vance, P.H.: Branch-and-price algorithms for the one-dimensional cutting stock problem. Comput. Optim. Appl. **9**(3), 211–228 (1998)
19. Vanderbeck, F.: Computational study of a column generation algorithm for bin packing and cutting stock problems. Math. Prog. **86**, 565–594 (1999)
20. Vanderbeck, F.: On Dantzig–Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. Oper. Res. **48**, 111–128 (2000)
21. Vanderbeck, F.: Exact algorithm for minimising the number of setups in the one-dimensional cutting stock problem. Oper. Res. **48**, 915–926 (2000)
22. Vanderbeck, F.: Extending Dantzig's bound to the bounded multi-class binary knapsack problem. Math. Prog. **94**(1), 125–136 (2002)

23. Vanderbeck, F.: Implementing mixed integer column generation. In: Desaulniers, G., Desrosiers, J., Solomon, M.M. (eds.) Column Generation, Kluwer, Dordrecht (2005)
24. Vanderbeck, F.: BaPCod—a generic branch-and-price code. depot APP 08-120018-000 IDDN (2008). http://wiki.bordeaux.inria.fr/realopt/
25. Vanderbeck, F., Savelsbergh, M.W.P.: A Generic view of Dantzig–Wolfe decomposition in mixed integer programming. Oper. Res. Let. **34**(3), 296–306 (2006)
26. Villeneuve, D., Desrosiers, J., Lübbecke, M.E., Soumis, F.: On compact formulations for integer programs solved by column generation. Ann. Oper. Res. **139**(1), 375–388 (2005)
27. Xpress-MP: User guide and reference manual, release 18.10. Dash Optim. (2007)