Experiments on Vanderbeck's generic Branch-and-Price scheme

von
Marcel Schmickerath
Masterarbeit in Mathematik

vorgelegt der

Fakultät für Mathematik, Informatik und Naturwissenschaften
der Rheinisch-Westfälischen Technischen Hochschule Aachen
angefertigt am Lehrstuhl für

Operations Research

1. Gutachten: Prof. Dr. Marco Lübbecke
2. Gutachten: Prof. Dr. Arie M.C.A. Koster

December 3, 2012

Contents

Abstract This master thesis based on [1] will develop and represent a generic branch-and-price scheme for column generation based on the Dantzig-Wolfe decomposition for mixed integer programs (MIP). In general finding a generic scheme for column generation could be difficult. Often they have their disadvantages or they are specialized to a restricted number of problem classes.

The most significant result of this scheme is that there is no need to expand the variable space or modify the pricing oracle (except setting some bounds on the variables, if the pricing problems are splitted like in [1]) to branch for an optimal (mixed) integer solution. Moreover it considers identical subproblems and avoids symmetry in the B&P tree. Therefore it will partition the subproblem solution set and take a different way for branching constraints instead of the usual disjunctive branching.

Also interesting is the comparison between this generic scheme and the already known Ryan&Foster branching on *set partitioning* instances.

1 Introduction

The main characteristic of this master thesis is the implementation and the experiments with Vanderbeck's generic branch and price scheme for column generation.

While the theory of polyheda and branch-and-bound could be found in [3], the one of column generation could be found in e.g. [2] and will be repeated in a shorter form to represent the theory of branch-and-price.

Although there are other branching schemes for column generation found in the litareture, they have often their disadvantages or are specialized to different classes of problems.

The scheme represented and generalized in a few ways here was developed by Vanderbeck in 2005 and it was said to have many positive aspects like easy handling in the sense of not modifying the pricing or variables and a bounded size of the corresponding B&P tree. In this master thesis the effeciency of this scheme will be proved and compared to other branching schemes. Moreover it will be raised up and represent as a branching scheme for mixed integer programs with different kinds of blocks and some ideas for not in [1] mentioned or specified sizes like the priority or the analysis of the MIP case.

This master thesis is built as follows:

Chapter 2 gives an introduction to the theory of column generation, repeating some basic subjects. For more information see e.g. [11], [2] or [3].

Chapter 3 represents Vanderbeck's generic branch-and-price scheme described in [1]. In this paper the examples and the scheme is explained on binary programs with one aggregated block type. Here we pick up the instructions in the 11th chapter called "Extensions" to raise up the scheme to a branching scheme for mixed integer programs with different block types. So here it is represented directly for MIPs. Only the analysis of the size of the B&P tree and the examples in the MIP case are not mentioned in [1].

Chapter 4 represents the comparison between the Ryan&Foster branching (described in [14]) and this generic scheme on *set partitioning* problems like it is done in [1] chapter 10 ("The Set partitioning special case").

Chapter 5.1 includes some notes for the implementation of the scheme, giving some new ideas and catching a few problems which are not mentioned in [1].

In the section of the computational results (section 5.2) we compare our scheme to others on different instances with different options and structure, e.g. discretization, convexification, aggregation and instances without aggregation.

The conclusion is given in chapter 6.

And finally, the literature and the German summary.

2 Column Generation

Many known optimazation problems can be formulated as an integer linear program (ILP) or more generally as a Mixed Integer Linear Program (MIP), e.g. the *knapsack*, *binpacking*, *set partition*, *cuttingstock*, *maxflow*, *stable set* and other graph theoretical problems. (see e.g. [3])

For further notation and introduction we say a MIP is in general given as:

$$\begin{aligned}
\min \quad & c^T x && && (1)\\
s.t. \quad & Ax \geq b && && (2)\\
& Bx \geq d && && (3)\\
& u \geq x \geq l && && (4)\\
& x \geq 0 && && (5)\\
& x \in \mathbb{R}^n && && (6)\\
& x_i \in \mathbb{Z}, && i \in Z && (7)
\end{aligned}$$

where $A \in \mathbb{Q}^{m \times n},\ B \in \mathbb{Q}^{k \times n},\ c, l, u \in \mathbb{Q}^n,\ b \in \mathbb{Q}^m,\ d \in \mathbb{Q}^k$ and $Z \subseteq \{1, \ldots, n\}$ an index set for integer variables. Moreover $l, u \in \{-\infty, \infty\}$ is a possible value for the variable bounds.

In the case where $Z = \{1, \ldots, n\}$ we call the program an integer linear program (ILP). If $l = 0$ and $u = 1$ we call it a binary program (BP).

If $Z = \emptyset$ it's simply called a linear program (LP). The **relaxed LP** or **relaxation** of a MIP is the LP one gets by omitting the integer constraints.

For later notation we call the variables x **original variables**.

For such problems one can use a solver like SCIP (which is implemented in C), using for a LP the so called simplex algorithm (see [13]) or the branch-and-bound algorithm combined with useful and possibly specialised heuristics for a MIP. (see e.g. [3])

2.1 Dantzig-Wolfe Decomposition

For another approach for solving a MIP we first introduce the Dantzig-Wolfe decomposition, which yields an extended formulation of the problem. This approach will reformulate the original problem and can therefore respect special structures.

First we assume the following structure of our MIP:
The Polyhedron

$$X = \{x \in \mathbb{R}^n \mid Bx \geq d,\ l \leq x \leq u,\ x_i \in \mathbb{Z}\ \forall i \in Z\} \tag{8}$$

contains only points over which optimization becomes rather easy relative to the remaining problem. E.g. if the problem is a *binpacking* then X can be a set of feasible *knapsack* refills. By *cuttingstock* e.g. feasible patterns. (see [2])

The constraints $Ax \geq b$ are unchanged. So now our MIP takes the shorter form

$$\min \quad c^T x$$

$$s.t. \quad Ax \geq b$$

$$x \in X$$

The Dantzig-Wolfe decomposition will now choose another representation of this shorter program. There are two possible ways for that, the so called convexification or the discretization approach.

2.1.1 Convexification

This approach based on the following theorem:

Theorem 1 (Minkowski-Weyl)**.** *A set $X \subseteq \mathbb{R}^n$ is a polyhedron if and only if there exist finite sets $\{x^p\}_{p\in P},\ \{x^r\}_{r\in R} \subset \mathbb{R}^n$ s.t.*

$$X = conv(\{x^p\}_{p\in P}) \ + \ cone(\{x^r\}_{r\in R})$$

*P stands for the **extreme points** and R represents the set of **extreme rays**.*
*$G = P \cup R$ is called the set of **generators** for X.*

Proof. see [4] □

The theorem means that every $x \in X$ can be written as

$$x = \sum_{p\in P} \lambda_p x^p \ + \sum_{r\in R} \lambda_r x^r,$$
$$\sum_{p\in P} \lambda_p = 1,$$
$$\lambda \in \mathbb{R}_+^{|G|}$$

The constraint $\sum_{p\in P} \lambda_p = 1$ is called the convexity constraint.

By omitting the above constraint $x \in X$ in our MIP and replacing each x by the representation of Minkowski-Weyl, we get the following **master problem** where the

variables are λ called **master variables**:

$$\begin{aligned}
\min \quad & \sum_{p\in P} c'_p\lambda_p \; + \sum_{r\in R} c'_r\lambda_r && (9)\\
s.t. \quad & \sum_{p\in P} a'_p\lambda_p \; + \sum_{r\in R} a'_r\lambda_r \geq b && (10)\\
& \sum_{p\in P} \lambda_p = \; 1 && (11)\\
& x = \sum_{p\in P} \lambda_p x^p \; + \sum_{r\in R} \lambda_r x^r && (12)\\
& \lambda \in \mathbb{R}_+^{|G|} && (13)\\
& x_i \in \mathbb{Z} \qquad\qquad \forall i \in Z && (14)
\end{aligned}$$

where $c'_p = c^T x_p,\ c'_r = c^T x_r,\ a'_p = Ax_p,\ a'_r = Ax_r \ \ \forall p \in P,\ r \in R.$

2.1.2 Discretization

Another approach is the so called discretization, where the master variables are chosen to be integer.

At first we consider the case of a given ILP.

By corollary 2.7 in [7] (based on [16] theorem 6.1) we get similar to the convexification approach

$$\begin{aligned}
x &= \sum_{p\in P} \lambda_p x^p \; + \sum_{r\in R} \lambda_r x^r,\\
&\sum_{p\in P} \lambda_p = 1.
\end{aligned}$$

Notice that the set of rays R keeps the same set than in the convexification approach, but now P enumerates all integer points, so it becomes larger than in the convexification approach. On the other hand, the master variables are now integer by adding the constraint

$$\lambda \in \mathbb{Z}_+^{|G|}.$$

One advantage of this approach is that there is no need to enforce integrality on the

original variables x anymore. So the new master problem is

$$\begin{aligned}
\min \quad & \sum_{p\in P} c'_p\lambda_p \; + \sum_{r\in R} c'_r\lambda_r && (15)\\
s.t. \quad & \sum_{p\in P} a'_p\lambda_p \; + \sum_{r\in R} a'_r\lambda_r \geq b && (16)\\
& \sum_{p\in P} \lambda_p = \; 1 && (17)\\
& \lambda \in \mathbb{Z}_+^{|G|} && (18)
\end{aligned}$$

The same approach from another point of view using the set of generators for G and a bounded set X.

Let $\{x^g\}_{g\in G}$ be the enumeration of all integer points in X. Then we can write the master problem as

$$\begin{aligned}
\min \quad & \sum_{g\in G} c'_g\lambda_g && (19)\\
s.t. \quad & \sum_{g\in G} a'_g\lambda_g \geq b && (20)\\
& \sum_{g\in G} \lambda_g = \; 1 && (21)\\
& \lambda_g \in \{0,1\} \qquad \forall g \in G && (22)
\end{aligned}$$

where $c'_g = c^T x^g,\ a'_g = Ax^g \ \ \forall g \in G$.

For an ILP there exist many primal heuristics based on the enumeration of all integer points in X. E.g. column selection heuristics use the integrality of the master variables and the convexity constraint. (see [5] and [17] for more information and other primal heuristics)

Now take a look at a given MIP.

So let X for now be bounded. This is only for a shorter and more simple notation. If X is not bounded then one has to add the variables corresponding to the rays, like seen above.

Here the extended formulation becomes a bit more complicated, as one has to respect the projection for each fixed continuous original variable. (see [18])

Let us first introduce a new notation for the original variables in the given MIP. For $x \in X$ let $x_{int} = (x_i)_{i\in Z}$ be the vector of the original variables which have to be integer and analogous x_{con} the vector of the original variables which are continuous.

For $x_{int} \in \mathbb{Z}^{|Z|}$ let $X(x_{int})$ represent the set of x_{con} in X depending on x_{int}. Then for

a MIP we get the following specialized sets

$$\begin{aligned} G_{pr} &:= \{x_{int} \in \mathbb{Z}^{|Z|} \mid \exists x_{con} \in X(x_{int})\} \\ P(x_{int}) &:= \{x_{con} \in \mathbb{R}^{n-|Z|} \mid x_{con} \text{ extreme point of } X(x_{int})\} \\ G := P &:= \{(x_{con}, x_{int}) \mid x_{int} \in G_{pr},\ x_{con} \in P(x_{int})\} \\ G(x_{int}) &:= \{g = (x^g_{con}, x^g_{int}) \mid x^g_{int} = x_{int} \in G_{pr},\ x^g_{con} \in P(x_{int})\} \end{aligned}$$

Now we apply the discretization on the x_{int} variables and the convexification on the continuous variables x_{con}.

Then the extended formulation of the MIP is given as

$$\min \quad \sum_{g \in G} c'_g \lambda_g \tag{23}$$

$$s.t. \quad \sum_{g \in G} a'_g \lambda_g \geq b \tag{24}$$

$$\sum_{g \in G} \lambda_g = 1 \tag{25}$$

$$\sum_{g \in G(x_{int})} \lambda_g \in \mathbb{Z}_+ \qquad \forall x_{int} \in G_{pr} \tag{26}$$

Constraint (25) now correspond to (21) resp. (17) the convexity constraint. Here the extreme points consist of an integer part x_{int} and the extreme points $x_{cons} \in X(x_{int})$.

Constraint (26) does not enforce integrality on the master variables directly, like (22) resp. (18). Enforcing integrality here directly can lead to loosing the optimality, while integrality on the sum over $g \in G(x_{int})$ for each x_{int} respects the integrality on the integer components x_{int}, as the distributivity rule holds.

2.2 Aggregation

Sometimes X has a special structure, e.g. a so called bordered block diagonal structure. (see [13] for *vehicle routing* or assume a *binpacking* problem with different types of bins resp. vehicle)

$$X = X^1 \times X^2 \times \cdots \times X^K$$

If some of these blocks (e.g. types of bins) are identical, they can be aggregated while using the discretization approach. We call it the aggregation of identical blocks. (see [5],[6])

This strategy will get a strong part of our branching scheme.

From now let our MIP be given as follows

$$z^P := \min \quad \sum_{k=1}^{K} c^k x^k \tag{27}$$

$$s.t. \quad \sum_{k=1}^{K} A^k x^k \geq b \tag{28}$$

$$B^k x^k \geq d^k \qquad \forall k = 1 \dots K \tag{29}$$

$$u^k \geq x^k \geq l^k \qquad \forall k = 1 \dots K \tag{30}$$

$$x^k \geq 0 \qquad \forall k = 1 \dots K \tag{31}$$

$$x^k \in \mathbb{R}^{n_k} \qquad \forall k = 1 \dots K \tag{32}$$

$$x_i^k \in \mathbb{Z}, \quad i \in Z^k \qquad \forall k = 1 \dots K, \tag{33}$$

where we have K blocks and an index set Z^k for integer variables in block k.

Notice: here the number of variables is given as $\sum_{k=1}^{K} n_k$.

Now there are K subproblems given as

$$X^k = \{x \in \mathbb{R}^n |\ B^k x \geq d^k,\ l^k \leq x \leq u^k,\ x \geq 0,\ x_i \in \mathbb{Z}\ \forall i \in Z^k\} \tag{34}$$

We say two blocks i and j are **identical**, if $c^i = c^j,\ A^i = A^j$ and $X^i = X^j$.

Identical blocks can now easily be aggregated, i.e. the corresponding master variables can be summed up. So let U^k denote the number of identical blocks of the type k and K' the number of different types of blocks, i.e. $K = \sum_{k=1}^{K'} U^k$, then our master problem takes the form

$$\min \quad \sum_{k=1}^{K'} \sum_{g \in G^k} c_{kg} \lambda_{kg} \tag{35}$$

$$s.t. \quad \sum_{k=1}^{K'} \sum_{g \in G^k} a_{kg} \lambda_{kg} \geq b \tag{36}$$

$$\sum_{g \in G^k} \lambda_{kg} = U^k \qquad \forall k = 1 \dots K' \tag{37}$$

$$\sum_{g \in G^k(x_{int})} \lambda_{kg} \in \mathbb{Z}_+ \quad \forall x_{int} \in G^k_{pr} \qquad \forall k = 1 \dots K' \tag{38}$$

2.3 Column Generation

In both approaches of the Dantzig-Wolfe decomposition the variable space grows exponentially large. Even while aggregation in the discretization approach is applied, the number of variables still becomes too large for practice.

Moreover, not every generator or extreme point or ray of the given problem is known.

Therefore column generation keeps the number small by only adding new master variables if they are important to get optimality. For this procedure we consider the pricing problem in the next subsection.

As solving the master problem directly is not possible in practice, choosing not the full number of variables and relaxing integrality on the master variables yields the **restricted master relaxation (RMP)**

$$z_{LP}^{RMP} = \min \quad \sum_{k=1}^{K'} \sum_{g \in H^k} c_{kg} \lambda_{kg} \tag{39}$$

$$s.t. \quad \sum_{k=1}^{K'} \sum_{g \in H^k} a_{kg} \lambda_{kg} \geq b \tag{40}$$

$$\sum_{g \in H^k} \lambda_{kg} = U^k \qquad \forall k = 1 \dots K' \tag{41}$$

$$\sum_{g \in H^k(y)} \lambda_{kg} \geq 0 \quad \forall y \in H_{pr}^k \qquad \forall k = 1 \dots K', \tag{42}$$

where $H \subseteq G$. $H = G$ yields the master relaxation.

The choice of the beginning set of variables could be important to get a feasible RMP. Sometimes even $\lambda = 0$ is not a feasible solution. Therefore one can use the Farkas lemma or approach, which can be found in e.g. [11], to get a feasible RMP (if the MIP is feasible).

2.4 Pricing

Of course a solution of the RMP does not need to satisfy optimality. Maybe one needs more master variables to get a better value.

To guarantee optimality one has to solve the so called pricing problem for each block k, given as

$$\alpha^k(\pi) = \min\{(c^k - \pi^k A^k)^T x^k \mid x^k \in X^k\}, \tag{43}$$

where π^k are the dual variables corresponding to the constraints (40). (see [19])

If $\alpha^k(\pi) \geq 0 \ \forall \ k = 1 \dots K'$ the current solution is optimal. Otherwise the pricing problem yields a new master variable to add into the master relaxation.

Notice that for each block k a pricing problem is solved, i.e. they should be small subproblems. E.g. a *knapsack* problem could be solved using dynamic programming in pseudo polynomial time. (See [13])

Moreover a negative minimum α^k has not to be computed exactly, i.e. adding one variable with to negative reduce costs (even a higher than the minimum) can yield up to a better master solution.

One advantage of column generation is that the pricing problems are rather easier to solve than the whole MIP. E.g. it is possible to use a heuristic to solve possible knapsack constraints, or just solving the subproblem by using another MIP.

For later notation we assume that there is one generic pricing solver, i.e. solving the pricing problems by e.g. another MIP. We will call this solver an **oracle**.

2.5 Lagrange Dual Bound

By solving the pricing problem one can obtain a dual bound for the master relaxation value z_{LP}^M and hence also for the MIP. (see [1],[18])

Let $\pi \geq 0$ denote the dual variable obtaining by dualizing constraint (36), then we get the Lagrange dual bound

$$L(\pi) := \pi b + \sum_{k=1}^{K'} U^k \alpha^k(\pi) \tag{44}$$

The best lower bound is therefore

$$\max_{\pi \geq 0} L(\pi) \tag{45}$$

and is also the optimum of the master LP, i.e. z_{LP}^M. Although the Lagrangian dual bound converges against the optimal solution of the master LP, it is not a monotonically convergence. (see [15])

2.6 Branch-and-Price

For now we can solve the master LP to optimality while using the oracle for the pricing problems. Of course if the current solution is optimal, it does not have to satisfy the integrality constraints for the original variables. While this check is easily done in the convexification approach and also in the discretization approach without aggregation, it is not this easy and also not unique while applying aggregation. Therefore we will discuss a mapping in section 3.1.

So we have to branch. There are several possible ways for that.

2.6.1 Branching on original variables

(see [1],[7])

This approach for branching is quite similar to the one in the branch-and-bound algorithm and is applied in several branching schemes like pseudocost branching, strong branching or reliability branching.

First assume the easy case using the convexification approach. In the master LP we relaxe constraints (14) and hence also (12) can be skipped, as they are only for the transcription of the master variables and the original variables to enforce integrality by (14).

Assume the current master LP solution $\dot{x}$ satisfies $r := \dot{x}_i \notin \mathbb{Z}$ for at least one $i \in Z$. Otherwise the current solution is optimal and we do not need to branch.

Then we get the two intuitive branching decisions by using the **disjunctive branching constraints**

$$x_i \leq \lfloor r \rfloor$$

$$x_i \geq \lceil r \rceil$$

leading two successor nodes in the B&P tree.

This constraints now can be enforced e.g. in two ways which Vanderbeck calls the hard and the soft branching.

The **hard branching** enforces the two branching constraints directly in the pricing problem. But this can make solving the pricing much harder, because the constraints are also convexified. So this can be a contradiction for the choice of easy subproblems for the oracle to solve.

The **soft branching** includes the two branching constraints in the master problem. But this can modify the reduced costs, because the constraints are also dualized.

Now take a look at the discretization approach. Without aggregation of identical blocks, it can be done like in the convexification case. But while identical blocks are aggregated branching will only be applied in one block. So this block has to be isolated from the others and we loose the advantage of aggregation. Otherwise the disaggregated variables can still be fractional while the aggregated variables are integer. (see the mapping in section 3.1)

Therefore branching on original variables while using aggregation is not efficient at all.

2.6.2 Branching on master variables

Instead of branching on the original variables it is possible to branch on the master variables while using the discretization approach. This approach can also be seen quite

similar to the usual branch-and-bound algorithm as we enforce integrality on the master variables λ themselves.

So assume for the current master LP solution $\dot{\lambda}$ holds $r := \dot{\lambda}_g \notin \mathbb{Z}_+$ for one $g \in H$ and first assume no aggregation of identical blocks. Then the disjunctive branching constraints because of (18) are

$$\lambda \leq \lfloor r \rfloor$$

$$\lambda \geq \lceil r \rceil$$

Now assume applying aggregation of identical blocks and for the current master LP solution $\dot{\lambda}$ holds $r := \sum_{g \in G^k(x_{int})} \dot{\lambda}_{kg} \notin \mathbb{Z}_+$ for one aggregated block k and one $x_{int} \in G^k_{pr}$. Then the disjunctive branching constraints because of (42) are

$$\sum_{g \in G^k(x_{int})} \lambda_{kg} \leq \lfloor r \rfloor$$

$$\sum_{g \in G^k(x_{int})} \lambda_{kg} \geq \lceil r \rceil$$

Of course this branching scheme is also intuitive like branching on the original variables and it respects the aggregation while using discretization. But it has two disadvantages. (see [7])

As mentioned above, the number of master variables can become exponentially large. So branching on each of these variables yields an extremely large B&P tree. Moreover this tree will become unbalanced, because making one decision on one variable of an exponential number of variables is only a weak decision. So branching on the original variables is stronger and even the size of the B&P tree does not depend on an exponentially large number of master variables to branch on.

Another disadvantage is that while bounding one master variable its corresponding point or ray has to be forbidden in the pricing problem. Otherwise it could be generated again and yield the same fractional master LP solution after pricing. So it will not represent a feasible branching rule, as the old solution is not forbidden by it. How this could be done can be read in [20] 11.2.4. Notice: While this is easily done for a BP, it becomes rather more complicated for a MIP.

Just mention one more point for branching on the master variables. Also interesting is the fact, that fractional master variables still can yield integer original variables. So enforcing integrality on this variables could be a too hard tool. E.g.

$$\dot{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 0.5 \begin{pmatrix} 1 \\ 2 \end{pmatrix} + 0.5 \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

2.6.3 Other branching schemes

Other branching rules like pseudocost branching, strong branching or reliability branching can be found in [10]. A specialized branching scheme called Ryan&Foster will be found in section 4.1 restricted on instances called *set partitioning* or raised up to *set cover*. And a first generalization from this by Vanderbeck can be found in [21].

2.7 Pseudocosts

One special way for branching is the so called pseudocost-branching from branch-and-bound and can e.g. be applied while branching on the original variables. These costs are a value to keep up how good the branch on one component, i.e. original variable, was. So they are computed by following the branch-and-bound tree and can help at the next decision for a fractional variable one wants to branch on. (see [10])

Let for each branching decision where branching is applied on x_i denote

$$\begin{aligned} f_i^- &:= x_i - \lfloor x_i \rfloor, & f_i^+ &:= \lceil x_i \rceil - x_i \\ \Delta_i^- &:= z^{LP}(\lfloor x_i \rfloor) - z_{old}^{LP}, & \Delta_i^+ &:= z^{LP}(\lceil x_i \rceil) - z_{old}^{LP} \end{aligned}$$

where z_{old}^{LP} is the optimal LP value of the current parent node and $z^{LP}(\lfloor x_i \rfloor)$ (resp. $z^{LP}(\lfloor x_i \rfloor)$) the LP values of the child node whith branching constraints $x_i \leq \lfloor x_i \rfloor$ (resp. $x_i \geq \lceil x_i \rceil$).

Then the **objective gain per unit change in variable** x_i is given as

$$\zeta_i^+ := \frac{\Delta_i^+}{f_i^+},\ \zeta_i^- := \frac{\Delta_i^-}{f_i^-}$$

Let ν_i^+ be the number of current branching decisions where $x_i \leq \lfloor x_i \rfloor$ was applied (resp. ν_i^-) and σ_i^+ the sum of its objective gain per unit (resp. σ_i^-).

Then **the upwards (resp. downwards) pseudocost** are given as

$$\Psi_i^+ := \frac{\sigma_i^+}{\nu_i^+},\ \Psi_i^- := \frac{\sigma_i^-}{\nu_i^-} \tag{46}$$

Now we consider two possible ways to work with these costs. Assume in the current node of the branch-and-bound tree denotes J the index set of original variables where $x_j,\ j \in J$ does not satisfy the integrality condition. Now one can branch on one of these components. The pseudocosts can help by this decision by choosing the component i^* for branching with

$$i^* = \arg\max_{j \in J} \{(1-\mu)\min\{f^+\Psi_i^+,\ f^-\Psi_i^-\} + \mu \max\{f^+\Psi_i^+,\ f^-\Psi_i^-\}\},$$

where $0 \leq \mu \leq 1$ is a parameter for the convex combination. (in [10] e.g. $\mu = \frac{1}{6}$)
Or the maximum over $f^+\Psi_i^+,\ f^-\Psi_i^-$.

Of course the pseudocosts are at the beginning, i.e. before branching was applied on the specific component, uninitialized. Therefore one can choose the pure pseudocost branching with initial $\Psi_i^+ = \Psi_i^- = 1$ or the strong branching where a pseudo-tree is calculated, i.e. calculate the B&P tree to a given depth and reject it after the calculation of the pseudocosts. Also these costs can have a reliability depending on the depth of the current tree. (see [10] for more information)

2.8 SCIP & GCG

The branching scheme is implemented in GCG (Generic Column Generation, by Gamrath 2010 [7], [8] and currently maintained by Bergner, Gamrath, and Puchert) environment written in C.

GCG is an extension of SCIP (Solving Constraint Integer Programs, by Achterberg et al [9] at the Zuse Institute Berlin) to get an efficient branch-and-price solver. Moreover SCIP is known as one of the fastest non-commercial MIP solvers based on branch-and-bound. (see [22])

Both, SCIP and GCG contains a lot of useful heuristics or other plugins like cuts, presolver or branching rules. (see e.g. [7], [5])

3 The generic Branch-and-Price scheme

In this section we want to repeat the generic branching scheme of Vanderbeck (see [1]), but given in a more general case, like it is mentioned there. The binary case for a BLP is explained in [1], also given some examples.

We assume the discretization approach with aggregation of identical blocks. Otherwise while using the convexification approach the generic branching scheme will not work, as we will see later in the analysis.

First for a better understanding we give a short overview on the whole branching scheme.

The program sequence plan is given in figure 1 and the pseudocode with the name of submethods is given in algorithm 1.

Like in section 2.6 explained we start at first solving the RMP and get the current master LP solution λ. We start pricing and compute the reduced costs. If they are negative, we add the corresponding master variable and generator to the RMP resp. H and start solving the RMP again. We repeat this procedure until we do not receive negative reduced costs from the oracle, i.e. λ is optimal for the master LP.

So let for now λ be optimal for our master LP.

Now we transform the solution of the extended formulation to a solution of our original problem. In other words, we map λ into x. How this can be done while using aggregation will be explained in section 3.1.

Having the corresponding original solution it is rather simple to check whether all integrality constraints are satisfied by the current solution. If so, the current solution is the searched optimal (mixed) integer solution of the MIP.

Assume there are still some integrality constraints violated. Otherwise we do not need any branching rule.

This is the point where the generic scheme comes into play. As in every branching scheme the current fractional solution has to be forbidden. Vanderbeck's generic scheme does so by first separating the fractional component by a computed bound sequence S. How this is done and what this sequence exactly means can be found in section 3.2.

The computed sequence S then induces the branching constraints, which are calculated in the next step of the scheme. How this could be done is also explained in section 3.2.1, as for deeper steps in the B&P tree information about previous bound sequences and nodes are needed for further branching. This is explained in section 3.2.2. While the analysis and the rules for branching schemes are directly following in section 3.3.

While computing the successor nodes depending on S, as Vanderbeck does not apply the usual disjunctive branching, it is rather important to keep the B&P tree nearly small, i.e. the question is: has every node to be generated in the tree? This is what the preprocessing decides. See for that section 3.4.

Moreover Vanderbeck further splitted up the pricing problems to get stronger dual

bounds. This is touched in section 3.5. For now we first keep the pricing discribed in section 2.4 in mind, as it also holds.

Now one has to select a new node in the B&P tree to branch further as it is done in all branching schemes. The corresponding branching constraint is added to the RMP and the so modified RMP is solved again. So the cycle is closed and the scheme starts from this point again.

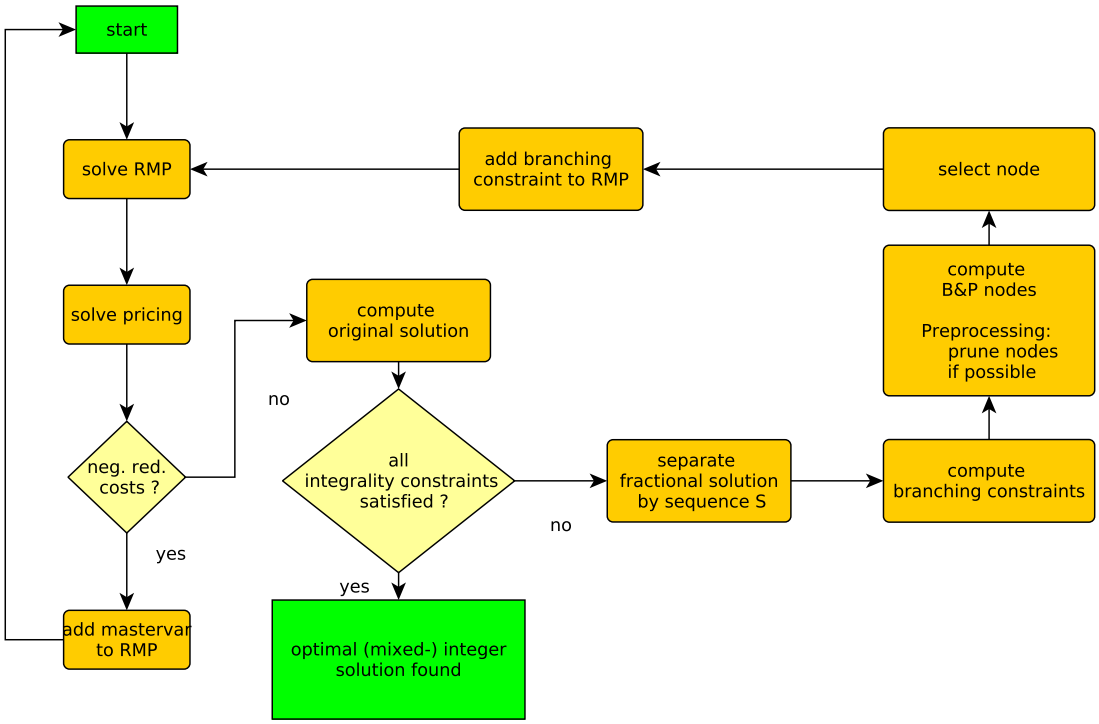


Figure 1: Program sequence plan for Vanderbeck's generic branching scheme.

Algorithm 1 Overview of the scheme

a) solve RMP to get current solution λ
b) call $SolvePricing(\lambda,\ H,\ S)$, if neg. red. costs add generators to H and go to a)
c) call $Map\lambda ToX(\lambda)$ to get an original solution for P resp. relaxation of P
while x is not feasible for P **do**
  d) call $Separate(\lambda,\ H)$ to get component bound sequence S
  g) call $CreateChildNodes(\lambda,\ S,\ H)$
  h) call $Preprocess(N)$
  i) choose a B&P node N
  j) solve N to get new current solution λ
  k) call $SolvePricing(\lambda,\ H,\ S)$, if neg. red. costs add generators to H and go to j)
  l) call $Map\lambda ToX(\lambda)$ to get an original solution for P resp. relaxation of P
end while
return z_{LP}^{RMP} resp. x

From now on we assume that λ is an optimal solution for the master LP corresponding to the current node in the B&P tree.

We will now introduce a representation of this solution as a rectangle in the following way:

Definition 1. *A master solution λ can be represented by K' tables or **rectangles**, where each table for the block k has **height** $n_k + 1$ and **width** U^k. A column in this table is given by the master variable λ_{kg} and the corresponding generator $g \in H^k$ in a specific order (first lexicographic order and later the so called induced lexicographic order). Such a column is called a **strip** of **width** λ_{kg}. A set of generators H'^k then defines a H'^k**-strip** of **width** $\lambda(H'^k) := \sum_{g \in H'^k} \lambda_{kg}$.*

An example is given in the next section (see example 1) and a deeper illustration follows at the analysis of the mapping.

3.1 Map λ to x

In this section we want to present the mapping from [1] to compute from a RMP solution λ while using aggregation an original solution x.

As the mapping is already known while there are no identical aggregated blocks, because of (12), the aggregated case is not that simple.

As already mentioned a fractional master solution can yield an integer original solution for P. But the other case can also appear. Consider the case where λ is integer, then $\lambda_{k'g} = \frac{\lambda_{kg}}{U^k}$ is a feasible solution for all $k' \in \{k'_1, \ldots, k'_{U^k}\}$ corresponding to block k and does not need to be integer and therefore x can also be fractional.

Moreover undoing the aggregation, called **disaggregation**, is not unique.

Example 1. Let $K' = 1,\ K = U^1 = 2,\ n_1 = 4$ and the current solution λ be given as

λ_g	0.5	1	0	0.5	0	0
x_1	4	4	3	2	1	1
x_2	4	2	1	0	2	4
x_3	4	2	1	0	2	1
x_4	4	2	1	0	2	1

A possible original solution could be obtained by simply summing up the components times their master variable λ_g and splitting it up by the number of identical blocks $U^1 = 2$

$$x^1 = \frac{0.5}{2}\begin{pmatrix}4\\4\\4\\4\end{pmatrix} + \frac{1}{2}\begin{pmatrix}4\\2\\2\\2\end{pmatrix} + \frac{0.5}{2}\begin{pmatrix}2\\0\\0\\0\end{pmatrix} = \begin{pmatrix}3.5\\2\\2\\2\end{pmatrix},\ x^2 = x^1$$

Notice that this solution is not integer.

But we could disaggregate the master variables in another way. A deeper look at the first and the fourth column of the rectangle shows, that these two master variables sum up to 1. So why not sum them up and leave the sum of the other columns for the second components?

$$x^1 = 0.5\begin{pmatrix}4\\4\\4\\4\end{pmatrix} + 0.5\begin{pmatrix}2\\0\\0\\0\end{pmatrix} = \begin{pmatrix}3\\2\\2\\2\end{pmatrix},\ x^2 = \begin{pmatrix}4\\2\\2\\2\end{pmatrix}$$

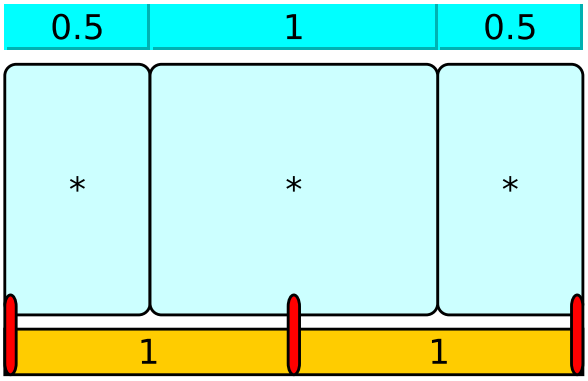
This yields another original solution. Moreover this one is integer.

So we need a fixed mapping that **preserves integrality**, i.e. one concrete mapping that computes an integer original solution if it can be obtained by the current integral master LP solution.

The idea of the mapping from [1] is quite similar to the second variant in the example, but a bit more efficient as it does not need to search two or more strips which sum up their widths to one. It just blocks the rectangle.

Example 2 (continued)**.** Let λ be given like in the last example.

By counting only the strips with positive widths, the rectangle can be illustrated as follows:



The blue boxes represent the strips and the yellow boxes at the bottom are the width of the disaggregated master variables, which is because of (25) given as 1.

The red marks note where a new component starts and ends. So what is in the first two marks sums up (with respect to the width, i.e. weight) to x^1 and the other part to x^2.

$$x^1 = 0.5 \begin{pmatrix} 4 \\ 4 \\ 4 \\ 4 \end{pmatrix} + 0.5 \begin{pmatrix} 4 \\ 2 \\ 2 \\ 2 \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \\ 3 \\ 3 \end{pmatrix}, \; x^2 = 0.5 \begin{pmatrix} 4 \\ 2 \\ 2 \\ 2 \end{pmatrix} + 0.5 \begin{pmatrix} 2 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

As the width of the second strip is splitted up to compute the last half of x^1 and the first half of x^2 we call the two component vectors x^1, x^2 **floating**.

In words:
To get a unique mapping we first define an order (e.g. lexicographically decreasing) on the set of generators and start the calculation in the left side of the rectangle. Beginning with $r = 1$ by adding each generator multiplies with a factor which corresponds to the width of the strip with respect to the not yet filled disaggregated master variables. In particular: If the width w e.g. of the first strip is smaller than 1, the corresponding generator is multiplied with w and will be summed up with another one until the sum of the widths is 1 and we get x^1. If the width w e.g. of the first strip is greater than 1 the corresponding generator is multiplied with 1 and is therefore x^1 itself. Then set $r = r + 1$ and multiply the generator with $w - 1$ and go on for x^2. If a component ends up in a strip with width w and factor t, e.g. for $r = 1$, the factor for the first generator in the sum for x^2 is $\min\{1, w - t\}$, i.e. the distance to the next disaggregated master variable.

So for the factor are the last three cases to regard. Formally is this factor given as

d

$$\mu_{k'_r g} := \min\ \{1,\ \lambda_{kg} - \sum_{p=1}^{r-1} \mu_{k'_p g},\ (r - \sum_{g':\ g'<g} \lambda_{kg'})^+\}\ \ \forall r = 1 \dots U^k,\ g \in H^k,$$

where $(r - \sum_{g':\ g'<g} \lambda_{kg'})^+ := \max\ \{0,\ (r - \sum_{g':\ g'<g} \lambda_{kg'})\}$

This value is the factor resp. the weight the actual strip has to sum up in the rth component.

In more detail: 1 is the width of the component r defined by (25). $\lambda_{kg} - \sum_{p=1}^{r-1} \mu_{k'_p g}$ is the width which is left from λ after the first $(r-1)$th components were computed and $(r - \sum_{g':\ g'<g} \lambda_{kg'})^+$ is the width left for component r. Components computed by at least one strip with a factor different from the width of the strip are called **floating** components. Notice that components which are floating can be integer, like in the last example seen. But under construction of the mapping, a component which is not floating is integral. So floating components are the critical ones.

By this knowledge we can develop a mapping which is given in algorithm 2. Here r counts the current component to compute x^r, $\mu_{k'_r g}$ is the factor satisfying the above formula and z keeps up the distance to the next full disaggregated master variable.

Let us prove the property of the factor by an easy induction on r:

Lemma 1. *In each step of algorithm 2 holds*

$$\mu_{k'_r g} = \min\ \{1,\ \lambda_{kg} - \sum_{p=1}^{r-1} \mu_{k'_p g},\ (r - \sum_{g':\ g'<g} \lambda_{kg'})^+\}$$

Proof. Apply induction on $r \in \mathbb{N}$

(first induction)

Let be $r = 1$ and as initialized is $z = 0$. So it is $r - z = 1$, $\sum_{p=1}^{r-1} \mu_{k'_p g} = 0$ and $\sum_{g':\ g'<g} \lambda_{kg'} = 0$ as we start in the left side of the rectangle. Therefore it holds

$$\mu_{k'_r g} = \min\ \{\lambda_{kg},\ 1\} = \min\ \{1,\ \lambda_{kg} - \sum_{p=1}^{r-1} \mu_{k'_p g},\ (r - \sum_{g':\ g'<g} \lambda_{kg'})^+\}$$

Assume the minimum is not one. Otherwise we are finished.

We replace $\lambda_{kg} = \lambda_{kg} - \mu_{k'_r g}$ like in f) and $z = z + \lambda_{kg}$ like in g). As the assumption holds it is $\lambda_{kg} = 0$ and therefore the algorithm increases g.

So it is $\sum_{g':\ g'<g} \lambda_{kg'} = \lambda_{kg'}$ the previous factor and hence

$$\mu_{k'_r g} = \min\ \{\lambda_{kg},\ r - \lambda_{kg'}\} = \min\ \{1,\ \lambda_{kg} - \sum_{p=1}^{r-1} \mu_{k'_p g},\ (r - \sum_{g':\ g'<g} \lambda_{kg'})^+\}$$

Now repeat the last step until it is $z = r$.

(induction requirement)
Let for one $r - 1$ be

$$\mu_{k'_l g} = \min \{1,\ \lambda_{kg} - \sum_{p=1}^{l-1} \mu_{k'_p g},\ (l - \sum_{g':\ g'<g} \lambda_{kg'})^+\} \quad \forall\ l = 1, \ldots, r-1$$

(induction step)
$r - 1 \mapsto r$:
Because of h) and the construction of the algorithm it is $z = r - 1$. Now the proof is quite similar to the first induction. □

In other words, the mapping gives a partition of each table into U^k strips of width 1. r is the number of the current strip and z denotes the actual position in the strip of width U^k.

In [1] it is shown that by using this mapping integrality on the aggregated master variables is equivalent to the integrality on the original variables if the given problem is a BLP. But remember the last example: in the case of an ILP (or even MIP) a fractional λ can still yield an integer x. So the proof only holds in one direction and this fact is intuitively clear as all entries and factors are integral. This mapping is a good tool to check whether the current master LP solution satisfies all the integrality constraints. So assume the given problem is a MIP, then transform the master solution into an original solution and check the integrality constraints for the not continuous components. So the mapping can be obtained in the MIP case, while ignoring the continuous components. Which are ignored in every generator from now on.

Algorithm 2 MapλTox

for $k = 1 \dots K'$ **do**
  a) let $\Lambda = \{g \in H^k |\ \lambda_{kg} > 0\}$
  b) sort $g \in \Lambda$ lexicographical decreasing by x^{kg}
  c) set $x_i^{k'_j} = 0 \ \ \forall j = 1 \dots U^k,\ i = 1 \dots n_k,\ z = 0,\ r = 1$
  for $g \in \Lambda$ in lexical order **do**
    while $\lambda_{kg} > 0$ **do**
      d) set $\mu_{k'_r g} = \min\{\lambda_{kg},\ r - z\}$
      for $i = 1 \dots n_k$ **do**
        e) $x_i^{k'_r} = x_i^{k'_r} \ + \ x_i^{kg} \mu_{k'_r g}$
      end for
      f) $\lambda_{kg} = \lambda_{kg} - \mu_{k'_r g}$
      g) $z = z \ + \ \mu_{k'_r g}$
      if $z = r$ **then**
        h) $r = r \ + \ 1$
      end if
    end while
  end for
end for
return x

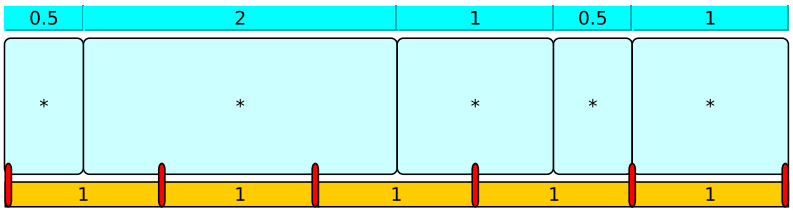
This mapping is not only a useful tool it is also the motivation of the generic branching scheme and therefore the idea of the scheme is quite similar to this mapping.

As seen only floating components can end up in fractional original variables. So if it is possible to avoid floating components, i.e. guarantee that each factor $\mu_{k'_r g}$ is integer, then clearly the obtained original solution is integer too. Motivate this on an easy example:

Example 3. Let the given problem be a MIP, $K' = 1,\ K = U^1 = 5,\ n_1 = 4$ and the current solution λ be given in lexicographic order as

λ_g	0.5	2	1	0.5	1	0
x_1	4	4	4	3	2	0
x_2	1	1	0	1	1	0
x_3	1	0	0	1	0	0
x_4	1	0	0	0	1	1

By counting only the strips with positive width, the rectangle can be illustrated as follows:



From the mapping we obtain the original fractional solution

$$x^1 = \begin{pmatrix} 4 \\ 1 \\ 0.5 \\ 0.5 \end{pmatrix}, \; x^2 = \begin{pmatrix} 4 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \; x^3 = \begin{pmatrix} 4 \\ 0.5 \\ 0 \\ 0 \end{pmatrix}, \; x^4 = \begin{pmatrix} 3.5 \\ 0.5 \\ 0.5 \\ 0 \end{pmatrix}, \; x^5 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

The floating components are x^1, x^3, x^4.

Let's say we first detected x^1 to be floating. Why not try to avoid this floating? This is exactly what the branching scheme will do. We split up the rectangle between the first and second strip, because the component is floating across these two. Now there are two cases to avoid the floating property: x^1 could be full in the left part of the rectangle or in the right part of it.

The first one could be obtained by setting the master variable corresponding to the first strip on value one. The other one by setting it to zero. As this method is similar to the branching on the master variable and also not efficient like it, the branching scheme keeps up a stronger method. Vanderbeck does not only bound single master variables, he bounds a class of them.

Putting x^1 to the left part of the rectangle could also be done by e.g. grouping the last 5 strips to a so called column class and decrease the bound on the whole size of this combined strip by one, i.e.

$$\lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 \leq 4$$

And in the other part with

$$\lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 \geq 5$$

In this example this choice for the component was not really the best one, as one of the other two components would split up the rectangle more balanced.

So what we now will do in the next sections is to show why this branching works, how such a component can be detected, which one is the best of them and how can the partition of the column classes or the B&P tree becomes more balanced.

3.2 Separation of a fractional solution at...

After checking the current solution and detecting it is not an optimal solution with integrality on the corresponding original variables, one have to branch. Here we consider two cases for branching. Branching in the root (i.e. first node) in the corresponding branch-and-price tree and branching in a node which is not the root node. The main difference of these two cases is the knowledge of the previous calculated sequence called component bound sequence S.

Definition 2. *A **component bound sequence** S for a block k is an ordered set of triples $\dot{x} = (i,\ \gamma,\ \alpha)$ where $i \in n_k$ defines the component or original variable on which the bound will be applied, $\gamma \in \{\leq,\ >\}$ gives the sense of the bound and $\alpha \in \mathbb{R}$ defines the value of the bound.*
*The pth element of S will be denoted with $\dot{x}_{[p]}$ and defines a bound on the i_pth component. Each S defines a **associated Column class** on the current set of generators H given by*

$$H(S) := \{g \in H \mid x^g \text{ satisfies } S\}$$

*The **value of a column class associated with** S is given by*

$$\lambda(S) := \lambda(H(S)) := \sum_{g \in H(S)} \lambda_g$$

In the definition the two senses for γ has to be disjunctive, as we will then partition H by S. The $H(S)$-strip has then the width $\lambda(S)$.

Now take a look at the last example:

Example 4 (continued)**.** Let the given problem be a MIP, $K' = 1,\ K = U^1 = 5,\ n_1 = 4$ and the current solution λ be given in lexicographic order as

λ_g	0.5	2	1	0.5	1	0
x_1	4	4	4	3	2	0
x_2	1	1	0	1	1	0
x_3	1	0	0	1	0	0
x_4	1	0	0	0	1	1

As seen the floating components are $x^1,\ x^3$ and x^4.

First let a component bound sequence S be given as

$$S = \langle (1, >, 3),\ (2, >, 0) \rangle$$

The correspond $H(S)$-strip is given by the first two strips and has width $\lambda(S) = 2.5$. The right part of the rectangle is then the strip defined by $H \setminus H(S)$ and has width $2.5 = 5 - 2.5$.

λ_g	0.5	2	1	0.5	1	0
x_1	4	4	4	3	2	0
x_2	1	1	0	1	1	0
x_3	1	0	0	1	0	0
x_4	1	0	0	0	1	1

So the branching can be applied by e.g. setting the two branching constraints

$$\lambda(H(S)) \geq 3$$

to cover the floating component in the left part or

$$\lambda(H(S)) \leq 2 \quad \Leftrightarrow \quad \lambda(H \setminus H(S)) \geq 3 = 5 - 2$$

to cover it in the right part.

A shorter bound sequence could be

$$S = \langle (1, >, 3) \rangle$$

This would cover x^4 in the left resp. right part of the rectangle and defines a partition with three strips on the left and on the right side.

The example shows, if we can find a component bound sequence separating a fractional component, then it defines the branching constraints.

3.2.1 ... the root node

In the root node there are at first no information about previous component bound sequences. This information will in any other node than the root node get important as we will see in the next subsection. But how will we get a component bound sequence S which induces our branching constraints?

First we notice that strips with an integral width are not that important to find a floating component, because every such strip covers at least one component. So floating can only appear if there are strips with a fractional width. Otherwise the mapping would compute an integer original solution x as seen.

So we only look at strips with a positive fractional width. Here we know that at least one component is floating, since the current original solution is fractional. But it can also sum up to an integral value. So we have to check this.

For each block k let be

$$F = \{g \in H^k | \ \lambda_{kg} - \lfloor \lambda_{kg} \rfloor > 0\},$$

$S = \langle\rangle,\ I = \{1 \dots n_k\}$ and

$$\alpha_i := \sum_{g \in F} x_i^{kg} \lambda_{kg}$$

the **component value** for component $i \in I$.

If one of the component values α_i is fractional, then we have found the component which we want to separate and add the splitting bound sequence $(i, \leq, b)$ to our component bound sequence S. The value b to split up the rectangle can be chosen as the median over all entries in the ith entry of all strips in F. This yields a more balanced partition.

But it can happen that all the values are integer. (see e.g. the first bound sequence in example 13)

So we have to split up the rectangle further, without any concrete knowledge which component is floating. Therefore we assume a given priority for every $i \in I$ and discuss this in section 5.1.3. For now we assume the priority is lexicographic decreasing with respeect to i. Then we split up by the first component and add $(1, \leq, b)$ to the sequence S and repeat the calculation with the strips in F which satisfies S. This will yield a nested partition of the column classes which is explained later. Of course could the sense for the bound also be $>$. So the other part of the rectangle has to be explored too. As the strips defined by the current S gets smaller in at least one of them has to be a fractional component value. Otherwise the entries all sum up to integer values and therefore would the original solution has been (mixed) integer. As seen the so obtained bound sequence does not have to be unique. Therefore we collect all possible sequences in a set, say *record*. Later we need a possible small S, i.e. we choose the smallest bound sequence in *record* for our S. As we also want to respect the priorities of the components, we first choose the sequence in *record* which has the highest priority for the last bound in it and then choose the smallest one of all these with the same priority. As we first select our bounds for the sequences with the highest priority, this selection respects the priority the most and keeps S small too.

The algorithm for this separation is given in algorithm 3 and works exactly like already explained above.

For separating a fractional solution in block k call $Separate(\lambda, F, I, S, record)$, where $S = \langle\rangle,\ I = \{1 \dots n_k\}$ and $record = \emptyset$.

Step a) is only a breakpoint for the recursion call in i) resp. j). In b) are the component values computed and c) for checking them on integrality. If there is a fractional one, the corresponding bound is in e) added to the current sequence S and then to *record* the set of all component bound sequences, while in e) we detect that this sequence is a feasible, i.e. separating one and we can abort the algorithm at f). If there is no fractional component value, then we compute in g) the index set $J \subseteq I$. J only contains the critical indices. In particular it is $0 \leq \alpha_i \leq \lambda(F)\ \forall\ i \in I$ as directly known from the definition. So Indices in J are those indices which are candidates for a new bound in

the current sequence, as bounding them can yield a real partition of the column classes. Step h) is to choose the index with the highest priority like mentioned above and in i) and j) are the recursive calls to the algorithm for splitting up further getting two sets of sequence bounds, which are put together in k).

After applying the algorithm, choose one of the component bound sequences S in *record* with the highest branching priority on its last component and the shortest size.

The shortcut $F(S)$ stands for $F \cap H(S)$. $median_i$ is the median in component i over all fractional columns. (See section 5.1.1 for details, because the choice of this value could be important) Notice that like mentioned in [1] i^* stays here in I, because of the non-binary value the component can later be choosed again with another component bound value. But in some cases it has to be removed too. (See therefore also section 5.1)

Algorithm 3 Separate($\lambda,\ F,\ I,\ S,\ record,\ k$)

if $F = \emptyset$ *or* $I = \emptyset$ **then**
  a) **return** *record*
end if
b) $foundFractionalCompValue := FALSE$
for $i \in I$ **do**
  c) $\alpha_i := \sum_{g \in F} x_i^{kg} \lambda_{kg}$
  d) $f := \alpha_i - \lfloor \alpha_i \rfloor$
  if $f > 0 \ \ i \in Z^k$ **then**
    e) $record := record \cup \{(\langle S, (i, \leq, median_i)\}, f)\rangle$
    f) $foundFractionalCompValue := TRUE$
  end if
end for
if $foundFractionalCompValue$ **then**
  g) **return** *record*
end if
h) $J := \{i \in I |\ 0 < \alpha_i < \lambda(F)\}$
i) $i^* := \arg\max_{j \in J}\{priority_j\}$
j) $record1 := Separate(\lambda,\ F(\langle S, (i^*, \leq, median_{i^*})\rangle),\ J,$
  $\langle S, (i^*, \leq, median_{i^*})\rangle,\ record,\ k)$
k) $record2 := Separate(\lambda,\ F(\langle S, (i^*, >, median_{i^*})\rangle),\ J,$
  $\langle S, (i^*, >, median_{i^*})\rangle,\ record,\ k)$
l) $record := record \cup record1 \cup record2$
m) **return** record

As in each branching step in the B&P we have to separate the current fractional

solution, the complexity of the procedure should be polynomial or pseudo polynomial and indeed it is.

Notice that in each iteration of *Separate* are $|F||I| = |F|n_k$ operations, because of the for-loops. Moreover the definition of J yields component bound sequences which really splits up the set of generators. The number of recursive calls is bounded too. By definition of J we decrease the index set if in F, the current set of fractional strips, one component $i \in I$ is bounded, i.e. all entries in the generator correspond to x_i, then it is $\alpha_i = \lambda(F)$ and therefore this index is not anymore in J. So let $v_{ik} = u_i^k - l_i^k$ denote the number of different entries in the generators in H and V_k denote the maximum over all $v_{ik},\ i = 1 \dots n_k$. (Given a BLP it is $V_k = 1$). Then there are at most $2V_k n_k$ recursive calls. In each iteration F will be decreased by at least one, because in each step we further split up the column classes and we do not further explore any part with empty F as we return for that in a). So there are $|F|$ possible strips with fractional width in which the recursive call ends up. This all yields the complexity of $O(V_k n_k^2 |F|^2)$.

By choosing only one of the two recursive calls i) or j), e.g. the one where $F(\langle S, \dot{x}_{[i^*]} \rangle)$ is the smallest, we can observe a complexity of $O(V_k n_k |F| log(|F|))$.

Remember that the size of F is bounded by the number of constraints in the master-problem, because λ is the optimal LP solution.

The construction of S yields $\lambda(S)$ to be fractional, therefore there are the branching constraints

$$\lambda(S) \leq \lfloor \lambda(S) \rfloor \quad or \quad \lambda(S) \geq \lceil \lambda(S) \rceil$$

by using the usual disjunctive branching constraints.

But the first constraint is in general weaker than the second and yields an unbalanced branch-and-price tree. (See Example 1)

Example 5 (continued)**.** Keep up the example 1 with λ given as

λ_g	0.5	2	1	0.5	1	0
x_1	4	4	4	3	2	0
x_2	1	1	0	1	1	0
x_3	1	0	0	1	0	0
x_4	1	0	0	0	1	1

and

$$S = \langle (1, >, 3),\ (3, >, 0) \rangle$$

Now the branching constraint for the first successor node $NODE(1)$ can be

$$NODE(1) \equiv \lambda(H(S)) \geq 1$$

and the weaker constraint for the second successor node

$$NODE(2) \equiv \lambda(H(S)) \leq 0$$

So in $NODE(2)$ we only know that one master variable is fixed to zero. As in the explanation of branching on the master variables in section 2.6.2 mentioned this is a weak decision and the B&P tree can get unbalanced.

Ignoring the fact that the partition of the column classes, i.e. the rectangle, is already unbalanced (therefore the B&P tree can be unbalanced too) and that we can observe better partition by choosing another component bound sequence S, we can split up the bigger right part and get a more balanced tree.

We can define a nested partition induced by S by only taking the first bound in the sequence and invert it to get

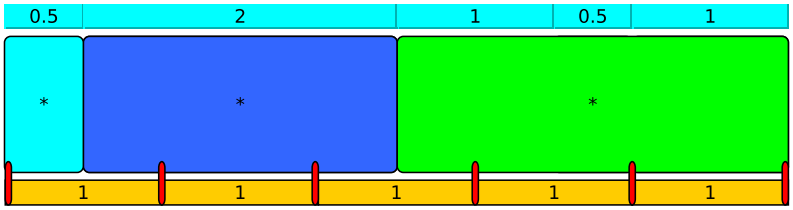
$$S^1 = \langle (1, \leq, 3) \rangle$$

Now take the first two bounds of S and invert the second to get

$$S^2 = \langle (1, >, 3),\ (3, \leq, 0) \rangle$$

And for further notation let be $S^3 = S$.

Now take a look at the illustration.



S^1 induces the green part of the partition. S^2 the dark blue part and S^3 the bright blue part. Now we define three branching nodes by

$$NODE(1) \equiv \lambda(H(S^3)) \geq 1$$
$$NODE(2) \equiv \lambda(H(S^2)) \geq 3$$
$$NODE(3) \equiv \lambda(H(S^1)) \geq 3$$

As $NODE(1)$ still contains a weak decision, because of the unbalanced partition, the old big decision is now splitted up in $NODE(2)$ and $NODE(3)$.

In particular: $NODE(1)$ will as already mentioned cover the floating component x^1. But by applying $NODE(2)$ or $NODE(3)$ we now get more information as we get one

component more covered in the dark blue or in the green part. Later in the analysis of the scheme we will see that this branching is a stronger step towards integrality if the blue and green column classes satisfying a specialized property.

Hence giving up the disjunction, i.e. the disjunctive partition of the solution space, using a non-binary branching tree yields a more balanced one.

The following definition generalizes the last idea:

Definition 3. *Given a current master LP solution λ and a component bound sequence S at a current B&P node. For each block k on which the branching scheme is applied by S define $|S|+1$ successor nodes by*

$$NODE(p) \equiv \lambda(G^{p-1} - G^p) \geq L^{p-1} - L^p + 1, \quad p = 1 \dots |S| \tag{47}$$

$$NODE(|S|+1) \equiv \lambda(G^{|S|}) \geq L^{|S|}, \tag{48}$$

where S^p is the subsequence of the first p component bounds in S,

$$L^p := \lambda(S^p) \quad \forall\ p = 1, \dots, |S| - 1,$$
$$L^{|S|} := \lceil \lambda(S) \rceil,$$

$S^0 := \langle\rangle$, $L^0 := \lambda(H^k) = U^k$ *and* $G^p := H^k(S^p)$ *for* $p = 0 \dots |S|$.

Notice that $L^p \in \mathbb{N}$, $p = 0 \dots |S| - 1$ (otherwise S could been smaller which is a contradiction to our smallest bound sequence S) and $L^{|S|}$ fractional,
$H^k(S) = G^{|S|} \subseteq G^{|S|-1} \subseteq \dots \subseteq G^1 \subseteq H^k = G^0$ and $L^{|S|} \leq L^{|S|-1} \leq \dots \leq L^1 \leq L^0$ per construction of G^p and S.

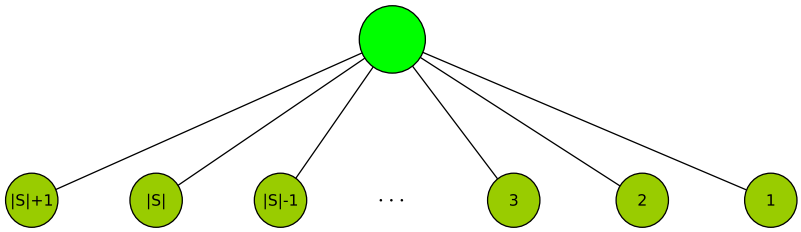


Figure 2: Illustration of one branching step for a fixed block k.

The order of the nodes is deterministic as the following lemma tells. Because any feasible integral solution which is not in at least one of the nodes $NODE(|S|+1), \dots, NODE(t)$ has to be in at least one $NODE(p)$, $p < t$. More in detail it tells us that in this branching no integral (or in the MIP case no mixed integral) solution will be lost.

Lemma 2 ([1])**.** *If a feasible integral (or mixed integral) solution λ is not represent in a Node(l) for at least one $l \in \{p, \ldots, |S|+1\}$ then it satisfies $\lambda(G^{p-1}) < L^{p-1}$.*

Proof. [1]

We show the result by induction on p.

(first induction)

Let $p = |S| + 1$.

Then λ not represent by the $NODE(|S|+1)$ implies per definition of this node

$$\lambda(G^{|S|}) < L^{|S|}.$$

(induction requirement)

For one $p \in \mathbb{N}$ let the result holds for $p+1$

(induction step)

$p \mapsto p-1$:

Let λ not be represent in $NODE(l)$ for $l = p, \ldots, |S|+1$ and observe $p \leq |S|$, otherwise holds the first induction.

Then by induction requirement the solution satisfies the constraint

$$\lambda(G^p) < L^p$$

and hence

$$\lambda(G^p) \leq L^p - 1,$$

because of the integrality on L^p and the convexity constraint.

Moreover, because λ is not represent in the

$$NODE(p) \equiv \lambda(G^{p-1} - G^p) \geq L^{p-1} - L^p + 1,$$

the solution satisfies

$$\lambda(G^{p-1} - G^p) \leq L^{p-1} - L^p.$$

The width of the strips is additive, i.e.

$$\lambda(G^{p-1}) = \lambda(G^{p-1} - G^p) + \lambda(G^p).$$

Hence the solution satisfies

$$\lambda(G^{p-1}) = \lambda(G^{p-1} - G^p) + \lambda(G^p) \leq L^{p-1} - L^p + \lambda(G^p) < L^{p-1} - L^p + L^p = L^{p-1}.$$

□

This observes that the solution of the MIP is in at most one of the sucessor nodes, because: if a mixed integral solution λ is not represent in the

$$NODE(1), \ldots, NODE(|S|+1)$$

then it satisfies

$$\lambda(H^k) = \lambda(G^0) < L^0 = U^k$$

which is a violation of the convexity constraint.

The current fractional solution is not feasible in every sucessor node, because in every node the corresponding strip $G(S^p)$ must have a different width. Because of the rounding this result is clear for $NODE(|S|+1)$. In the other nodes the addition of 1 increases the width of the $(G^{p-1} - G^p)$-strip.

So we see that the generic branching scheme is a valid branching if it ends up after a finite number of steps in an optimal integer solution. This will be the result of the analysis in section 3.3. But for now we close this subsection with the example from [23]:

Example 6. Given a BLP, let be $K' = 1,\ U^1 = 5,\ n_1 = 4$ with λ given as

λ_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

The algorithm *Separate* now computes in the first iteration

$$\alpha_i = 1 \quad \forall\ i = 1, \ldots, 4$$

So keep up the bound $(1, >, 0)$

Then in the corresponding recursive call it is

$$\alpha_2 = \frac{1}{2}$$

So the algorithm terminates with the component bound sequence

$$S = \langle (1, >, 0),\ (2, >, 0) \rangle$$

Now calculating the bounds yields up to

$$L^0 = 5,\ L^1 = 3,\ L^2 = \lceil 1.5 \rceil = 2$$

This induces the three successor nodes

$$\begin{aligned} NODE(1) \equiv & \quad \lambda(\langle(1,\leq,0)\rangle) \geq 5-3+1=3 \\ NODE(2) \equiv & \quad \lambda(\langle(1,>,0),(2,\leq,0)\rangle) \geq 3-2+1=2 \\ NODE(3) \equiv & \quad \lambda(\langle(1,>,0),(2,>,0)\rangle) \geq 2 \end{aligned}$$

By taking a deeper look at each node, we see that in each node one component is implicit fixed. In particular we bound in $NODE(1)$ $x_1^r = 0$, in $NODE(2)$ $x_2^r = 0$ and in $NODE(3)$ $x_2^r = 1$ for some unknown r correspond to each node. But as the width of the corresponding strip is bound by an integer size, this component will at least become integral when the width really is fixed to an integer value, e.g. when the compliment column class is bounded to $U^k - L$, where L is the bound in the strip. This in a more general view is the idea why the generic branching scheme will find an integer solution in the end.

3.2.2 ... a node after the root node

Now consider the case of a node in the branch-and-price tree which is not the root node. Therefore let

$$\lambda(S^p) \geq L^p,\ G^p = H^k(S^p), \quad p = 1 \dots P$$

denote all the P branching constraints in the current node correspond to block k.

Notice that we assume that no pair of this constraints are identical. Otherwise the fractional solution was not cut off by the branching constraint. Even when the component bound sequences are identical the bound value is greater than the previous one. This case is mostly possible and in such a case some of the constraints are redundant and can be removed by a preprocessing. (see subsection 3.4)

Before we explain how to separate at a deeper node in the B&P tree, the question is: can we even just call separate again to get another useful component bound sequence S without any respect to the bound sequences calculated in the ancestor nodes?

And the answer is: No.

Example 7. Given a BLP with $K' = 1,\ U^1 = 3,\ n_1 = 4$ and λ be given in lexicographic order as

λ_g	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
x_1	1	1	0	0	0	0
x_2	1	1	1	1	0	0
x_3	1	0	1	0	1	0
x_4	0	0	0	0	0	1

Assume we are at depth 1 in the B&P tree and the current node is defined by the branching constraint

$$\lambda(\langle(1,>,0)\rangle) \geq 1$$

The separate method computes

$$\alpha_1 = 1,\ \alpha_2 = 2, \alpha_3 = \frac{3}{2},\ \alpha_4 = \frac{1}{2}$$

So the new component bound sequence with no respect to the previous one is given as

$$S = \langle(3,>,0)\rangle$$

with $\lambda(S) = \frac{3}{2}$.

Therefore we get the two successor nodes defined by

$$NODE(1) \equiv \lambda((3,\leq,0)) \geq 2$$

$$NODE(2) \equiv \lambda((3,>,0)) \geq 2$$

Remember: the idea of the branching scheme was to get the bounded component in the mapped original solution integer. So in this example we expect the $3rd$ component in at least one component vector would be integer by applying these branching constraints.

Assume we select $NODE(2)$ to be the next node in the B&P tree to expand further. ($NODE(1)$ can be seen quite similar)

A new master LP solution λ can be

λ_g	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{1}{3}$
x_1	1	1	0	0	0	0
x_2	1	1	1	1	0	0
x_3	1	0	1	0	1	0
x_4	0	0	0	0	0	1

Now apply the mapping from section 3.1 and we get

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

We see the first component is integer in at least one of these vectors as we expected by our first branching step. But we also see the 3rd component is fractional in all of them. This is a contradiction to the idea of the branching scheme. In other words: the last step is more or less useless.

The example shows that we need to respect component bound sequences resp. the partition of the column classes done at the ancestor nodes. Otherwise the branching scheme may not terminate after a finite number of branching steps or if it does it could have made a lot of useless steps.

To keep up our idea to get closer to an integral solution we have to keep the partition of the column classes as a nested one. This means that the column classes we define in each step of the branching scheme are disjunct or subsets. But they do not have to overlap each other. Then the subset property keeps up old already covered components and the integral solution becomes closer in each branching step as the analysis will show in section 3.3.

So the question is how to respect the previous partitions? For that we first define the column class tree which keeps the relationship between them.

Definition 4. *Given the column classes $G^p = H^k(S^p)$ for $p = 1 \dots P$ for the current block k in the current branch-and-price node. The* ***associated tree of column classes for block*** *k is defined as follows.*

The ***root*** *is associated with H^k. Each* ***leaf node*** *corresponds to a class G^p with no subclass, i.e. there is no subset $G^l \subset G^p$, $l \neq p$. A node associated with G^p is* ***adjacent*** *to a node correspond to class G^j if and only if $G^p \subset G^j$ and there is no subclass between them, i.e. there is no G^i with $G^p \subset G^i \subset G^j$ (treecondition). In this case G^p is a* ***direct successor*** *of G^j and resp. G^j a* ***direct predecessor*** *of G^p. In general if there could be a subclass between them we call them only* ***sucessor*** *and* ***predecessor****.*

The ***set of all direct successors*** *of a node associated to G^p is denoted by $dsucc(p)$, the* ***set of direct predecessors*** *by $dpred(p)$, the* ***set of successors*** *by $succ(p)$ and the* ***set of predecessors*** *by $pred(p)$.*

If this tree definition really is a tree by our partition of the column classes, we call this partition a **nested partition**.

We illustrate the definition of the tree of the column classes by an example:

Example 8. Let $K' = 1,\ U^1 = 12$ and $n_1 = 4$. The current branching constraints are

$$\lambda(\langle(2,\leq,3),(3,\leq,0),(4,\leq,1)\rangle) \geq 1 \quad (49)$$
$$\lambda(\langle(2,\leq,3),(3,\leq,0),(4,\leq,1),(1,>,0)\rangle) \geq 1 \quad (50)$$
$$\lambda(\langle(2,\leq,3),(3,\leq,0),(4,>,1),(1,>,0)\rangle) \geq 4 \quad (51)$$
$$\lambda(\langle(2,\leq,3),(3,\leq,0),(4,>,2)\rangle) \geq 3 \quad (52)$$
$$\lambda(\langle(2,\leq,3),(3,>,0),(1,\leq,0)\rangle) \geq 1 \quad (53)$$
$$\lambda(\langle(2,>,3),(1,\leq,0)\rangle) \geq 1 \quad (54)$$
$$\lambda(\langle(2,>,3),(1,\leq,0),(4,\leq,1)\rangle) \geq 1 \quad (55)$$
$$\lambda(\langle(2,>,3),(1,>,0)\rangle) \geq 1 \quad (56)$$
$$\lambda(\langle(2,>,3),(1,>,0),(3,\leq,1),(4,\leq,1)\rangle) \geq 1 \quad (57)$$
$$\lambda(\langle(2,>,3)\rangle) \geq 2 \quad (58)$$

The tree of column classes is then given in figure 3.

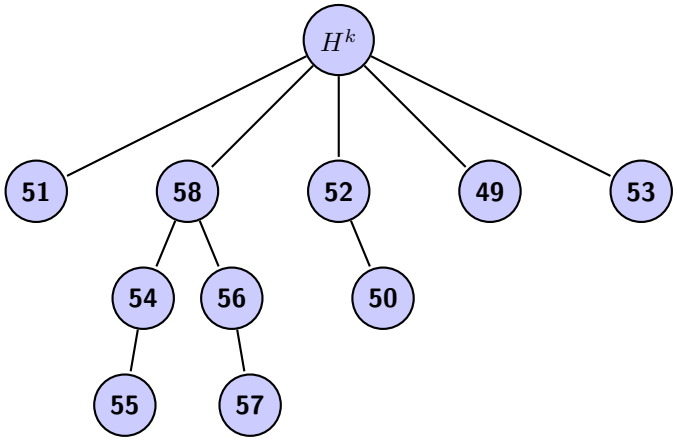


Figure 3: Example for a column class tree for a fixed block k. The number of the nodes corresponds to the number of the associated branching constraint.

As we see, the condition for each column class to not overlap another column class really defines a tree. This tree is induced by the set of the branching constraints resp. the partition of the column classes and the tree itself induces an order of these column classes, e.g. by using a depth first search from left to right and enumerate each node of the column class tree after all child nodes are explored. This enumeration defines then the order of the strips in our rectangle and we will call this order the induced lexicographic order.

Applying this order on the column classes of our example yields the following sequence for the corresponding branching constraints:

$$51,\ 55,\ 54,\ 57,\ 56,\ 58,\ 50,\ 49,\ 52,\ 53$$

The idea of the induced lexicographic order on the strips is to look up whether the strip is in one of these column classes starting at the left. This respects the previous partition of the column classes and as we apply the depth first search, the property of the subset of each column class holds. So then the new sort of the rectangle keeps strips in the same column class together.

Then if there are two strips in the same column class they can be sort lexicographic.

So the tree of the column classes defines an order of the strips which respects the previous partitions. (Notice: the lexicographic order is not that important. One can also define another sort at the beginning and then get the induced sort of it by the tree of the column classes)

But compute this tree and then sort by it can get expensive. So do we need to compute this tree in practice?

Take a look at the associated component bound sequences S^p for each block k. Because they partition the whole column class H^k in a nested way by component bounds each S^k starts with a bound on the same component. From the definition of the associated columns class tree one gets: every component bound sequence S^p associated to a node of the same subtree with the root in depth d has the same first d component bounds. Because $G^p \subset G^j \Leftrightarrow S^j \subset S^p$ which means that the two component bound sequences bounds the same $|S^j|$ components.

Given by this tree or by the nested partition we define a new order on the strips. First it should represent the partition and then the order is set to be lexicographically. The so called **induced lexicographic order** (ILO) is defined on two columns g, h by calling $ILO(g, h, C, S, I, p, k)$ where C denotes the set of branching sequences, i.e. initial $C = \{S^1 \dots S^P\}$, S the component bounds which are already tested, i.e initial $S = \langle\rangle$, I the set of components which are not yet fixed, i.e. initial $I = \{1 \dots n_k\}$ and p the current position in the bound sequences, i.e. initial $p = 1$. (see algorithm 4)

The shortcut $C(S)$ denotes the subset of C which have the first bounds like S, i.e.

$C(S) = \{S^p \in C |\ S \subseteq S^p\}$.

Algorithm 4 ILO($g,\ h,\ C,\ S,\ I,\ p$)

if $C = \emptyset$ **then**
  if $x_I^{kg} < x_I^{kh}$ in lexicographic order **then**
    a) **return** $TRUE$
  end if
  b) **return** $FALSE$
end if
c) Find component i which is fixed in the pth entries in all $S' \in C$
Let $x_{[p]} = (i, \gamma, \alpha) \in S^1$ the pth comp. bound in first (reference) sequence $S^1 \in C$
if x_i^{kg} *and* x_i^{kh} satisfies $X_{[p]}$ **then**
  d) **return** $ILO(g, h, C(\langle S, (i, \gamma, \alpha)\rangle), \langle S, (i, \gamma, \alpha)\rangle, I \setminus \{i\}, p+1, k)$
end if
if x_i^{kg} *and* x_i^{kh} (both) do not satisfy $X_{[p]}$ **then**
  e) Let $\beta \in \{\leq, >\} \setminus \{\gamma\}$
  f) **return** $ILO(g, h, C(\langle S, (i, \beta, \alpha)\rangle), \langle S, (i, \beta, \alpha)\rangle, I \setminus \{i\}, p+1, k)$
end if
g) **return** $x_i^{kg} > x_i^{kh}$

Let us apply this ordering on an example

Example 9. Let $K' = 1,\ U^1 = 12$ and $n_1 = 4$, the branching constraints given as 49 - 58 (without 51, 52, 57 to small it down) and the current master LP solution given as

λ_g	0.2	0.3	0.5	1	1	0.3	1.7	2	1	0.5	1	1.5	1
x_1	4	4	3	2	1	1	0	2	0	5	3	0	3
x_2	3	2	1	0	2	4	5	0	2	1	6	0	0
x_3	3	2	1	0	2	1	1	3	0	1	2	1	4
x_4	3	2	1	0	2	1	1	1	1	0	0	1	2

The induced lexicographic order with respect to the column class tree is then given as:

λ_g	1	0.3	1.7	0.5	0.2	0.3	0.5	1	2	1	1.5	1	1
x_1	3	1	0	5	4	4	3	3	2	1	0	2	0
x_2	6	4	5	1	3	2	1	0	0	2	0	0	2
x_3	2	1	1	1	3	2	1	4	3	2	1	0	0
x_4	0	1	1	0	3	2	1	2	1	2	1	0	1

The reason why the ILO is so important shows the comparison between the ILO and the lexicographic order.

The lexicographic order here is given as

λ_g	0.5	0.2	0.3	1	0.5	1	2	1	0.3	1	1.7	1	1.5
x_1	5	4	4	3	3	3	2	2	1	1	0	0	0
x_2	1	3	2	6	1	0	0	0	4	2	5	2	0
x_3	1	3	2	2	1	4	3	0	1	2	1	0	1
x_4	0	3	2	0	1	2	1	0	1	2	1	1	1

Just following the method *Separate* would yield e.g.

$$\alpha_1 = 0.5 \cdot 5 + 0.2 \cdot 4 + 0.3 \cdot 4 + 0.5 \cdot 3 + 0.3 \cdot 1 + 1.7 \cdot 0 + 1.5 \cdot 0 = 6.3$$

which is fractional. So the component bound sequence could be e.g.

$$S = \langle (1, >, 0) \rangle$$

and correspond to the branching constraints

$$\lambda(\langle (1, >, 0) \rangle) \geq 8$$

and

$$\lambda(\langle (1, \leq, 0) \rangle) \geq 10 - 8 + 1 = 3.$$

This partition of the column classes destroys the partition which was previous calculated and is represented in the ILO. As seen this is important to reach integrality. (see also section 3.3)

So the ILO respects the previous partition of the column classes and helps to avoid an overlap of two or more component bound sequences, i.e. column classes. But as the last example shows we need to tell the information not only to the mapping but also to the separate method.

How do we use this order to branch further?

A deep look at the associated tree of column classes shows that there are three possible cases by a separation of a fractional solution.

a) An intermediate node, i.e. an intermediate subclass, has fractional value, then branch on the associated component bound sequence.

b) An already existing node, i.e. an old subclass, has fractional value, then branching yields to reset the bound in the corresponding branching consraints.

c) A new leaf node, i.e. another subclass, has fractional value, then branching will further partition one existing subclass.

So the new separate method has to explore for this three cases.
The algorithm which do so is given in algorithm 5.
So analogous to the case at the root node we call the separate function
$Explore(C,\ p,\ F,\ I,\ S,\ record,\ k)$ with the initial values

$$C = \{S^1 \dots S^P\},\ I = \{1 \dots n_k\},\ S = \langle\rangle,\ record = \emptyset,\ p = 1$$

and $F = \{g \mid \mu_{kg} - \lfloor \mu_{kg} \rfloor > 0\}$.

Now in each iteration of the method we decrease C the set of all previous defined component bound sequences by following the next bound in the sequences in step b) and split up the set C in f) and g).

If we get a new leaf node in the corresponding tree of column classes we end up in a), while the other two cases appear in d). (The value b_i needs some special properties which are explained in section 5.1.1. It should split up the column classes with respect to the ILO but it should yield a real partition too.) In particular if the current component bound sequence is equal to one of those in the current C then we end up here in the same node, otherwise we get a new node between two nodes in the tree.

At the end the new component sequence can be chosen from *record* the set of all possible separating component bound sequences, like it is done after the method *Separate* at the root node of the B&P tree.

The complexity of the separation method *Explore* is in $O(V_k n_k^2 |F|^2)$ like the one of *Separate*.

In particular: one call of *Explore* needs $O(|F|)$ operations without *Separate* which needs $O(n_k|F|)$ operations in each recursive call. Like in the analysis of *Separate* the number of recursive calls is bounded in $O(n_k|F|)$, since there are just $|F|$ possible partitions.

The complexity can also be reduced to $O(V_k n_k |F| log(|F|))$ by choosing only one of the two recursive calls f) or g) in each step, e.g. the one with the smallest F.

Algorithm 5 Explore($\lambda,\ p,\ F,\ I,\ S,\ record,\ k$)

if $C = \emptyset$ **then**
  a) **return** $Separate(F, I, S, record, k)$
end if
b) Find component i which is fixed in the pth entries in all $S' \in C$
Let $x_{[p]} = (i, \gamma, median_i) \in S^1$ the pth comp. bound in first (reference) sequence $S^1 \in C$
c) Let $\alpha_i = \sum_{g \in F} x_i^{kg} \lambda_{kg},\ f = \alpha_i - \lfloor \alpha_i \rfloor$
if $f > 0 \ \ i \in Z^k$ **then**
  d) $record = record \cup \{(\langle S, (i, \leq, b_i)\rangle, f)\}$
  e) **return** $record$
end if
if $\alpha_i > 0$ **then**
  f) $record1 = Explore(C(\langle S, (i, >, median_i)\rangle)), p+1, F(\langle S(i, >, median_i)\rangle)),$
  $I \setminus \{i\}, \langle S, (i, >, median_i)\rangle, record, k)$
end if
if $\alpha_i < \lambda(F)$ **then**
  g) $record2 = Explore(C(\langle S, (i, \leq, median_i)\rangle)), p+1, F(\langle S(i, \leq, median_i)\rangle)),$
  $I \setminus \{i\}, \langle S, (i, \leq, median_i)\rangle, record, k)$
end if
h) $record = record1 \cup record2$
i) **return** $record$

After separation of the fractional solution, the branching constraints will be computed like in section 3.2.1 and therefore the properties of a branching rule (except terminating which is shown in section 3.3 shown) also hold.

Before we show the termination of the scheme, we first answer the question: do we need all $|S|+1$ successor nodes in each branching step?

Proposition 1 ([1])**.** *The descendant nodes can be restricted to only* $l, \ldots, |S|+1$ *nodes, where* S *is the current component bound sequence and*

$$l = \max_{p=1,\ldots,P:\ \lambda(G^p)=L^p} \{|S^p|\}$$

Proof. [1] Let l be like in the proposition.

As the lemma 2 holds, we know that if the constraint $\lambda(g^l) \geq L^l$ was already calculated and by choosing this node, i.e. a path in the branch-and-price tree along this node associated with this constraint, we already cut off integer solution for which $\lambda(g^l) < L^l$ holds.

Therefore at computing the new descendant nodes we do not need the constraints, i.e. nodes, which deserve integer solution with $\lambda(g^l) < L^l$. □

An easy way for pruning is the **prune by dominance**, i.e. a node N needs not to be generated if one of the ancestor nodes has a successor node defined by the same branching constraint. Notice that this result only holds for branching constraints depend on the same block k. As the next section shows, prune by dominance and the proposition keeps the whole branch-and-price tree small. In the case of a *set partitioning* it is almost binary, which will be shown in section 4.

Other pruning aspects like prune by infeasibility or decrease the size of P is shown in section 3.4, the preprocessing.

3.3 Depth of the branch-and-price tree

This section contains the analysis of the width and depth of the B&P tree using the generic scheme.
For a BLP we can directly adopt the analysis in [1] and get:

Theorem 2. *Let* $n := \sum_{k=1\ldots K}\{n_k\},\ K = U := \sum_{k=1\ldots K'}\{U^k\}$ *and the given problem is a BLP.*
The depth of the branch-and-price tree on a BLP is bounded by nU *and the number of leaf nodes is bounded by* 2^{nU}*.*

For the understanding of the proof of this theorem and to generalize it to a MIP we need more theory about the variable space and therefore the following definition:

Definition 5. *Let block k be fixed.*

*Given a component bound sequence S, the associated width $L \in \mathbb{N}$ and the current master solution λ ordered in the ILO, then we call a (S, L)-**strip** the strip consisting of columns of the $G(S)$-strip from the first one up to the value L. So it is a $G(S)$-strip of width L. Adding all strips of the rectangle in ILO beginning at the first column on the left, the (S, L)-strip for block k is described by the interval*

$$V(S, L) = [\sum_{g<G(S)} \lambda_{kg} \ , \ \sum_{g<G(S)} \lambda_{kg} \ + \ L)$$

*A restriction of a (S, L)-strip to the lines which are bounded by S is called a (S, L)-**frame**.*

*For λ the current master solution let x be the mapped solution using the mapping in section 3.1. For an aggregated block k we call $x_i^{k'_r}$ **covered** by the (S, L)-frame if $i \in S$ and $r - 1 \in V(S, L)$. We call the component **overcovered** by the (S, L)-frame if it is covered by it and the corresponding component is fixed in S, i.e. $x_i^{gk} = x_i^{hk} \ \forall x^{hk}$, x^{gk} satisfying S.*

In words is a (S, L)-strip the set of strips consisting of the first L strips of the $G(S)$-strip. A (S, L)-frame is then a (S, L)-strip containing only components which are bounded by the bounds in the component bound sequence S. A component covered by a (S, L)-frame means that the strip correspond to $x^{k'_r}$ begins in the (S, L)-strip and overcovered means it is not just bounded but also fixed.

Given a BLP then a component is covered if and only if it is overcovered, because $i = 0$ in S or $i = 1$ in S.

Specially in the binary case covering also means fixing one component implicitly and one (S, L)-frame covers $L|S|$ components of $x^{k'_r}$.

Example 10 (continued)**.** Like in the last example the current solution in ILO is given as

λ_g	1	0.3	1.7	0.5	0.2	0.3	0.5	1	2	1	1.5	1	1
x_1	3	1	0	5	4	4	3	3	2	1	0	2	0
x_2	6	4	5	1	3	2	1	0	0	2	0	0	2
x_3	2	1	1	1	3	2	1	4	3	2	1	0	0
x_4	0	1	1	0	3	2	1	2	1	2	1	0	1
r	1	2	3	4	4	4	5	6	8	9	10	11	12

Now take a look at each frame:

The frame $(\langle(2,>,3),(1,>,0)\rangle,1)$ covers the components x_2^1 and x_1^1. Because the components are fixed they are overcovered too.

The frame $(\langle(2,>,3)\rangle,2)$ covers the components x_2^1 and x_2^2. But they are not overcovered.

The frame $(\langle(2,>,3),(1,\leq,0),(4,\leq,1)\rangle,1)$ covers the components x_2^3, x_1^3 and x_4^3. And they are all overcovered too.

The frame $(\langle(2,>,3),(1,\leq,0)\rangle,1)$ covers and overcovers the components x_2^3, x_1^3 too.

The frame $(\langle(2,\leq,3),(3,>,0),(1,\leq,0)\rangle,1)$ covers the components x_2^{10}, x_3^{10} and x_1^{10}. They are overcovered too.

The frame $(\langle(2,\leq,3),(3,\leq,0),(4,\leq,1),(1,>,0)\rangle,1)$ covers and overcovers the components x_2^{11}, x_3^{11}, x_4^{11} and x_1^{11}. But the frame $(\langle(2,\leq,3),(3,\leq,0),(4,\leq,1)\rangle,1)$ covers the components x_2^{11}, x_3^{11} and x_4^{11} too. But only x_3^{11} is overcovered.

We see on x_2^1 that it is possible for a component to be covered by two or more different frames.

Notice that a component which is overcovered can loose this property again. E.g. after pricing a new master variable with the generator $(3,5,2,0)^T$ could be added to the master. Then its possible that x_1^1 is still overcovered but x_2^1 not.

Another aspect is that x_2^2 is covered but floating. It defines the value $0.3\cdot4+0.7\cdot5=4.7$ which is fractional. If this component is overcovered, e.g. its fixed to 5 then this value becomes integer. $(0.3\cdot5+0.7\cdot5=(0.3+0.7)\cdot5=1\cdot5=5)$.

Overcovering is an usefull tool to guarantee this integrality, but even a covered but not overcovered component can get an integral value, e.g. $0.3\cdot3+0.7\cdot33=24$.

If the border of the interval $V(S,L)$ is integral, i.e. $\sum_{g<G(S)}\lambda_{kg}\in\mathbb{N}$, then each component which is overcovered by the corresponding (S,L)-frame is integer by using the mapping in the first subsection. Notice that the index r of the potentially fixed component is **floating**, i.e. the r is not fixed in this sense and can be changed while branching. This makes the branching scheme independent from the component and the corresponding component vector directly and as the covered component is not specific one avoids the symmetry while specializing one of them in each branching step.

So each branching constraint $\lambda(S)\geq L$ defines a set of components which are covered and potentially overcovered.

Proposition 2 ([1])**.** *Let the current B&P node be defined by the component bound sequences $S^1,\ldots,S^P$ and the corresponding bounds $L^1,\ldots,L^P$. Let λ be the current master solution and x the associated solution obtained by the mapping in section 3.1. For a BLP holds: If for every block k each component $x_i^{k'_r}$ is covered by a (S^p,L^p)-frame, $r=1\ldots U^k,\ i=1\ldots n_k, i\in Z^k$, then x is integer (resp. mixed integer).*

Proof. see [1] observation 5. □

The last proposition holds intuitively for a MIP if covering is replaced by overcovering the component.

As seen in the example it is possible for one component to be covered by more than one (S, L)-frame. But the bound on it is the same, which guarantees the following intuitive proposition:

Proposition 3. *Let x be integer (resp. mixed integer). If $x_i^{k'_r}$ is covered by a (S, L)-frame, the value is bounded by S.*

Proof. see [1] □

The idea of this analysis is to show that in each step of the branching scheme a new component is covered or overcovered. Then the scheme will at least find an integer (resp. mixed integer) solution x. For this approach we need to specify the covered component, by the following definition:

Definition 6. *Let $x_i^{k'_r}$ be covered by a (S, L)-frame and i be the last component in S which belongs to a covered component and let r be the largest index with $r-1 \in V(S, L)$. Then $x_i^{k'_r}$ is called the **representative** of a (S, L)-frame.*

So a representative of a frame is the last covered component, i.e. it is on the right bottom.

The definition is unique in the following sense:

Lemma 3. *Let the current node of the B&P tree be given by the constraints associated with $(S^1, L^1), \ldots, (S^P, L^P)$, λ a feasible solution and x defined by the mapping in section 3.1. Then a component $x_i^{k'_r}$ can only be the representative of at most one (S^p, L^p)- frame.*

Proof. see [1] proposition 5. □

Now we show:

Proposition 4. *The number of covered component increases for all blocks k by each branching step using the purposed scheme until all components are covered.*

Proof. A (S, L)-frame is unique and even its representative is not a representative of another frame. This representative is covered, so each frame covers at least one different component for a block k. Each covered component is bounded by the associated S. So defining a new branching constraint by the scheme defines a new bound sequence S which yields a new frame with a new covered representative. If all components are covered, new component bound sequences S set new bounds to the components until they are overcovered. □

Now we can give the proof of the theorem 2 which gives us a bound on the size of the B&P tree for a BLP:

Proof. (see [1]) First let the block k be fixed.

We know that each of the branching constraints defining the current node is different resp. the set of branching constraints containing to block k. Each constraint or frame has a unique representative. So after $n_k U^k$ branching steps every component is covered, because the number of covered components increases at each step. And so the solution yield by the mapping from the first section is integer (resp. mixed integer). So the depth of the B&P tree while only branching on block k is bounded by $n_k U^k$. Now repeat this for each block. This yields the depth of

$$\sum_{k=1\ldots K'} n_k U^k \leq \sum_{k=1\ldots K'} n_k \cdot \sum_{k=1\ldots K'} U^k =: nU.$$

In the MIP case the components which correspond to continuous variables can be ignored in the frame and covering theory. So denotes C^k the number of continuous variables given in block k, then the bound can be small down to

$$\sum_{k=1\ldots K'} (n_k - C^k)U^k \leq \sum_{k=1\ldots K'} \{n_k - C^k\} \cdot \sum_{k=1\ldots K'} U^k =: (n - C)U$$

Each component is bounded by the associated component bound sequence correspond to the covering frame. This yields in the binary case at most 2^{nU} or $2^{(n-C)U}$ leaf nodes, because each covered component is bounded by one or zero. □

The following example shows that the theorem for a BLP does not hold for a MIP.

Example 11. $K' = 1,\ n_1 = 2,\ U^1 = 4,\ Z = \{1, 2\}$, i.e. $x \in \mathbb{Z}$ and the current solution:

λ_g	0.2	0.3	0.5	1	1	0.3	0.7
x_1	4	4	3	2	1	1	0
x_2	3	2	1	0	2	1	1

The current branching constraints are associated by :

$$S^1 = \langle (x_1, \geq, 2),\ (x_2, \geq, 1)\rangle, \quad L^1 = 1$$
$$S^2 = \langle (x_1, \geq, 2),\ (x_2, <, 1)\rangle, \quad L^2 = 1$$
$$S^3 = \langle (x_1, <, 2),\ (x_2, \geq, 2)\rangle, \quad L^3 = 1$$
$$S^4 = \langle (x_1, <, 2),\ (x_2, <, 2)\rangle, \quad L^4 = 1$$

Each component is covered by at least one frame.

But e.g. x_1^1 is not integer using the mapping $Map\lambda ToX$:

$$x_1^1 = 0,2 \cdot 4 + 0.3 \cdot 4 + 0.5 \cdot 3 = 3.5$$

So at least one more branching step is needed.

Moreover (S^1, L^1) and (S^2, L^2) (resp. (S^3, L^3) and (S^4, L^4)) can replace the maybe previous given redundant branching constraint associated by:

$$S^5 = \langle (x_1, \geq, 2) \rangle, \quad L^5 = 2$$

and this makes redundant:

$$S^6 = \langle (x_1, \geq, 2) \rangle, \quad L^6 = 1$$

resp.

$$S^7 = \langle (x_1, <, 2) \rangle, \quad L^7 = 2$$

and this makes redundant:

$$S^8 = \langle (x_1, <, 2) \rangle, \quad L^8 = 1$$

So the depth of the B&P tree is at least $8 + 1 > 8 = nU$

In the case of a MIP we get then:

Theorem 3. *Given a MIP and the original variables are bounded by l and u like it is mentioned in section 2.*

Let

$$L := \max_i \{l_i\},\ O := \max_i \{u_i\},\ n := \sum_{k=1} K' n_k$$

and $U := \sum_{k=1 \ldots K'} U^k$.

Then the depth of the branch-and-price tree is bounded by $nUK'(O-L)$ and the number of leaf nodes is bounded by $(O - L + 1)^{nUK'(O-L)}$

Proof. As in the BLP case seen, every component is covered after a depth of at most $n_k U^k$ while branching only in block k. But to guarantee integrality we need every component to be overcovered. It means that the components in each strip have to be fixed too. So in the worst case each strip has to be splitted up into smaller strips, where the component is fixed to a value between L and O. There are $O - L + 1$ different

numbers. So counting the current strip there are at most $O - L$ further branching steps needed to overcover the components. Because of the pigeon hole principle there are at most U^k nonredundant branching constraints. (see section 3.4 proposition 5) So there are at most U^k strips in all nodes. Discovering for each strip then yields up to at most $U^k(O - L)$ further branching steps. Therefore the depth to overcover all components in block k is at most

$$n_k U^k + U^k(O - L) = U^k(n_k + O - L).$$

Summing up yield the bound for the whole depth of the tree as

$$\sum_{k=1}^{K'} U^k(n_k + O - L) \leq (\sum_{k=1}^{K'} U^k) \cdot (\sum_{k=1} K'(n_k + O - L)) = nUK'(O - L).$$

For each component in S are at most $(O - L + 1)$ possible values. So there are at most $(O - L + 1)^{nUK'(O-L)}$ leaf nodes.

Like above if C denotes the number of continous variables, the values can be decreased to $(n - C)UK'(O - L)$ resp. $(O - L + 1)^{(n-C)UK'(O-L)}$ □

3.4 Preprocessing at a node

Before solving the master LP with the new branching constraints one can decrease the number of the constraints P, or prune the node before solving the RMP again. One can speed up a bit by some preprocessing aspects.

Example 12. Given a BLP, let be $K' = 1,\ U^1 = 2,\ n_1 = 4, S = \langle\rangle$, the current B&P node N is defined by the branching constraints

$$\lambda(\langle(1, >, 0)\rangle) \geq 2 \tag{59}$$
$$\lambda(\langle(1, > 0), (2, > 0)\rangle) \geq 1 \tag{60}$$

and the current master LP solution λ is given as

λ_g	0.2	0.8	0.5	0	0.5	0	0	0
x_1	1	1	1	1	1	0	0	0
x_2	1	1	0	0	0	1	1	0
x_3	1	0	1	1	0	1	0	1
x_4	1	0	1	0	0	0	0	0

As the first bound is applied on component 1 *Explore* computes

$$\alpha_1 = 1,$$

which is integral. So store the bound $(1, >, 0)$ in the component bound sequence S. The next bound is on component 2, therefore compute

$$\alpha_2 = 1,$$

which is integral too. As the two sets of fractional column classes in $H^1(\langle(1, >, 0), (2, \gamma, 0)\rangle)$ for $\gamma \in \{\leq, >\}$ are not empty add e.g. $(2, \leq, 0)$ to S. Now there are no previous bounds and *Separate* detects

$$\alpha_3 = 0.5$$

So it is e.g.

$$S = \langle(1, >, 0), (2, \leq, 0), (3, >, 0)\rangle,\ \lambda(S) = 0.5$$

Then the successor nodes are

$$NODE(1) \equiv \lambda(\langle(1, \leq, 0)\rangle) \geq 1 \tag{61}$$
$$NODE(2) \equiv \lambda(\langle(1, >, 0), (2, >, 0)\rangle) \geq 2 \tag{62}$$
$$NODE(3) \equiv \lambda(\langle(1, >, 0), (2, \leq, 0), (3, \leq, 0)\rangle) \geq 1 \tag{63}$$
$$NODE(4) \equiv \lambda(\langle(1, >, 0), (2, \leq, 0), (3, >, 0)\rangle) \geq 1 \tag{64}$$

Now take a look at the succesor nodes.
$NODE(1)$ is pruned by infeasibility because of (59). In particular: Hence the width of the column class $H^1((1, >, 0))$ is already bounded to U^1 by constraint (59), the master variables in the $H^1((1, \leq, 0))$-strip can be fixed to zero and the whole column class can be cut off from the generator set.
$NODE(2)$ has a branching constraint which dominates the constraint (60).
$NODE(3)$ and $NODE(4)$ have a branching constraint which combined with (60) makes the constraint (59) redundant.

Now assume we are in $NODE(4)$.
Let the current master LP solution λ be given as

λ_g	0.2	0.8	0.5	0.5	0	0	0	0
x_1	1	1	1	1	1	0	0	0
x_2	1	1	0	0	0	1	1	0
x_3	1	0	1	1	0	1	0	1
x_4	1	0	1	0	0	0	0	0

Let the new component bound sequence be

$$S = \langle (1, >, 0), (2, \leq, 0), (3, >, 0), (4, >, 0) \rangle, \ \lambda(S) = 0.5$$

Then the successor nodes are

$$NODE'(1) \equiv \lambda(\langle (1, \leq, 0) \rangle) \geq 1 \tag{65}$$
$$NODE'(2) \equiv \lambda(\langle (1, >, 0), (2, >, 0) \rangle) \geq 2 \tag{66}$$
$$NODE'(3) \equiv \lambda(\langle (1, >, 0), (2, \leq, 0), (3, \leq, 0) \rangle) \geq 1 \tag{67}$$
$$NODE'(4) \equiv \lambda(\langle (1, >, 0), (2, \leq, 0), (3, >, 0), (4, \leq, 0) \rangle) \geq 1 \tag{68}$$
$$NODE'(5) \equiv \lambda(\langle (1, >, 0), (2, \leq, 0), (3, >, 0), (4, >, 0) \rangle) \geq 1 \tag{69}$$

As $NODE'(1)$ is again pruned by infeasibility it is also pruned by dominance, like $NODE'(2)$ and $NODE'(3)$ too, because we know an ancestor node of the B&P tree here N (needs not to be a direct ancestor node) which defines successor nodes with the same three branching constraints (61), (62) and (63).
Now the constraint (64) is redundant in $NODE'(4)$ and $NODE'(5)$.

The example shows that in some cases nodes are not just pruned by dominance or infeasability it can also happen that one or more branching constraints dominate previous defined branching constraints of one or more ancestor node.
We now formalize this dominance in the following definition:

Definition 7. *A branching constraint $\lambda(S^p) \geq L^p$ is redundant if*

$$\sum_{l \in dsucc(p)} L^l \geq L^p$$

*The **marginal lower bound** for class G^p is*

$$Lm^p := L^p - \sum_{l \in dsucc(p)} L^l$$

Directly from the definition follows: For the marginal lower bound holds

$$Lm^p > 0 \ \Leftrightarrow \ \lambda(S^p) \geq L^p \text{ not redundant.}$$

Obviously a redundant branching constraint can be omitted.

Proposition 5 ([1] pigeon hole principle)**.** *The number of branching constraints for a block k that are not redundant at a given B&P node is at most U^k or the current path in the B&P tree is infeasible.*

Proof. Let T be the number of non-redundant constraints at the current node.
It is

$$\lambda(S^p) \geq L^p \;\Leftrightarrow\; \lambda(C^p G^i) \geq L^p - \sum_{l \in dsucc(p)} L^l = Lm^p,$$

where $C^p := G^p - \bigcup_{i \in dsucc(p)}$.

For a non-redundant constraint holds $Lm^p > 0$ and because of integrality $Lm^p \geq 1$. While the partition is nested, i.e. $G^1, \ldots, G^P$ are disjoint, are $C^1, \ldots, C^T$ disjoint too. Therefore we have the disjunctive constraints

$$\lambda(C^p) \geq Lm^p \geq 1,\ p = 1 \ldots T.$$

Summing up the constraints yields

$$\lambda(H^k) \geq \sum_{p=1}^{T} \lambda(C^p) \geq T,$$

but this is a contradiction to $T > U^k$, because from the master problem holds the constraint

$$\lambda(H^k) \leq U^k.$$

□

Another way for pruning is prune by infeasability. E.g. the remaining branching constraints correspond to block k has to satisfy

$$\sum_{p=1}^{P} L^p \leq U^k,$$

because of the convexity constraint.

But also the bounds on the original variables l^k and u^k have to be respected.

For each block k one gets the global bounds

$$gl^k \leq \sum_{r=1}^{U^k} x^{k'_r} \leq gu^k$$

e.g. with $gl^k = U^k l^k$ and $gu^k = U^k u^k$ corresponding to the aggregation and the used mapping.
Because of the nested partition one gets in the BLP case the two constraints for each component i

$$\sum_{p:\ (i,>,0) \in S^p} L^p \leq gu_i^k$$

$$\sum_{p:\ (i,\leq,0)\notin S^p} L^p \geq gl_i^k$$

If one of these is violated the current node is infeasible. (Moreover in [1] the bound can be reseted if one of these constraints is satisfied with equality. See "Further specification and deleting columns".)
In the MIP case this constraints can be extended to

$$\sum_{p:\ (i,>,b)\in S^p} (b+1)L^p \leq gu_i^k$$

$$\sum_{p:\ (i,\leq,b)\notin S^p} (b+1)L^p \geq gl_i^k$$

3.5 Pricing

At a given node in the B&P tree Vanderbeck defines in [1] for each branching constraint a new subproblem:

Definition 8. *Let SP_k^0 ($X_0^k = X^k$) represent the original pricing problem for block k given in section 2.4. For each constraint $\lambda(S^p) \geq L^p$ based on block k a subproblem SP_k^p exists which prices only columns of G^p*

$$[SP_k^p] \quad \zeta_k^p(\pi) := \ \min\{(c^k - \pi^k A^k)x \mid x \in X_k^p\}$$

where $X_k^p := \{x \in \mathbb{R}^{n_k} |\ x \in X^k,\ x$ satisfies $S^p\}$.
If $X_k^p = \emptyset$ then set $\zeta_k^p = \infty$.

Now the column generation procedure can be changed in the following sense. A new column is added to the master if the reduced costs $\zeta_k^p(\pi) - \sum_{l\in pred(p)} \sigma_l$ are negative, where σ_l denotes the dual variable belonging to the branching constraint $\lambda(S^l) \geq L^l$. Like $pred(p)$ was defined, holds $l \in pred(k)$ if $S^l \subset S^p$.

Its easy to see that for $l \in pred(p)$ $[SP_k^l]$ is a relaxation of $[SP_k^p]$ and hence $\zeta_k^l(\pi) \leq \zeta_k^p(\pi)$. Therefore if $x^l := \arg\min \zeta_k^l(\pi) \in G^p$ it is $x^l = \arg\min \zeta_k^p(\pi)$.

Using these facts [1] computes a tree of pricing problems where the relation given in the column classes is respected. Then this tree is solved in the given order until negative reduced costs are detected. (see [1] table 6)

This pricing is not yet implemented. Like it is mentioned in [1], the calculation of the tree is expensive and even there it is solved heuristically. But it is a usefull tool to show in theory how the dual bound corresponds to the branching scheme.

In particular: The pricing problem described in section 2.4 is solved at the root node of the pricing tree. This is enough to get an optimal master LP solution by adding master variables with negative reduced costs. So the knowledge which column class the corresponding strip belongs to is not really neccessary. As we do not get a strong dual bound like in [1] the scheme still works and saves therefore a bit of time.

3.6 The dual bound

In [1] the results for the new pricing methods yields the new tighter Lagrangian dual bound

$$\sum_{k=1\ldots K'} \min \{\sum_r c^k x^{k'_r} \mid \sum_r A^k x^{k'_r} \geq b^k,\ x^{k'_r} \in\ conv(X_k^p),\ r = \rho^{p-1} \ldots \rho^p - 1,\ p = 0 \ldots P\}$$

where $\rho^{-1} = 1,\ \rho^p = \rho^{p-1} + Lm^p,\ p = 0 \ldots P$. (Remember that P also depends on the block k, i.e. $P = P^k$).
(see [1] proposition 9).

4 Set Partitioning

In this section we will compare the branching scheme and the Ryan&Foster branching on *set partitioning* problems like e.g. *binpacking*. So let us first explain these two problems.

Set Partitioning
Given: A set of items $N = \{1, \dots, n\}$ and a set of subsets $\{P_1, \dots, P_m\} \subseteq P(N)$ with costs $c(P_i),\ i = 1, \dots, m$
Question: Find a minimum cost partition for N, i.e. find $I \subseteq \{1, \dots, m\}$ with $\bigcup_{i \in I} P_i = N$ and $P_i \cap P_j = \emptyset,\ i, j \in I,\ i \neq j$ while $\sum_{i \in I} c(P_i)$ is minimal.

A *set partitioning* is given as a BLP as

$$\begin{aligned} \min \quad & c^T x && (70)\\ s.t. \quad & Ax = 1 && (71)\\ & x \in \{0,1\}^n, && (72) \end{aligned}$$

where $A \in \{0,1\}^{m \times n},\ c \in \mathbb{Q}^n$.

We say the master problem is a *set partitioning* if it is given as (with aggregation of identical blocks):

$$\begin{aligned} \min \quad & \sum_{k=1}^{K'} c^k \sum_{g \in G^k} x^{kg} \lambda_{kg} && && (73)\\ s.t. \quad & \sum_{g \in G^k} x_i^{kg} \lambda_{kg} = 1 && \forall i = 1 \dots n_k,\ k = 1 \dots K' && (74)\\ & \sum_{g \in G^k} \lambda_{kg} = U^k && \forall k = 1 \dots K' && (75)\\ & \lambda_{kg} \in \{0,1\} && \forall g \in G^k,\ k = 1 \dots K', && (76) \end{aligned}$$

where $x^{kg} \in \{0,1\}^{n_k}\ \forall k = 1 \dots K'$.

The constraint (74) can be written as

$$W^k \lambda_k = 1 \quad \forall k = 1 \dots K' \tag{77}$$

where W^k is a matrix with entries in $\{0,1\}$ and $\lambda_k = (\lambda_{kg_1}, \dots, \lambda_{kg_{|G|}})^T$.

Many problems can be reduced to a *set partitioning* problem, e.g. the *binpacking* problem.

Binpacking
Given: $N = \{1, \dots, n\}$ a set of items with weight $w_i,\ i = 1, \dots, n$ and W the size of a bin
Question: Find $m \in \mathbb{N}$ and a partition $\{P_1, \dots, P_m\}$ of N into m bins with $\sum_{i \in P_j} w_i \leq b\ \forall j = 1, \dots, m,\ \bigcup_{j=1}^{m} P_j = N,\ P_i \cap P_j = \emptyset,\ i, j \in \{1, \dots, m\},\ i \neq j$ and $m \leq n$ minimal.

A *binpacking* is given as a BLP as

$$\min \quad \sum_{j=1}^{n} y_j \tag{78}$$

$$s.t. \quad \sum_{i=1}^{n} w_i x_{ij} \leq W y_j \qquad \forall\ j = 1, \dots, n \tag{79}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\ i = 1, \dots, n \tag{80}$$

$$y \in \{0,1\}^n \tag{81}$$

$$x \in \{0,1\}^{n \times n}, \tag{82}$$

where x_{ij} say whether item i is in bin j or not and $y_j = 1$ means bin j is used.
In words:
(79) sets y_j to 1 if the bin is used and it contains the constraint for the size W of the bin. Constraint (80) guarantees that every item i is in exactly one bin j.
Now we apply the discretization approach setting

$$X = \{x \in \{0,1\}^n |\ \sum_{i=1}^{n} w_i x_i \leq W\}$$

the set of feasible binpackings, then we get the master problem

$$\min \quad \sum_{g \in G} x^g \lambda_g \tag{83}$$

$$s.t. \quad \sum_{g \in G} x_i^g \lambda_g = 1 \qquad \forall i = 1 \dots n \tag{84}$$

$$\sum_{g \in G} \lambda_g = 1 \tag{85}$$

$$\lambda_g \in \{0,1\} \qquad \forall g \in G \tag{86}$$

where $x^g \in X$.

And this is a *set partitioning* problem like seen above. The generators are feasible knapsack patterns computed by the subproblem e.g. by using dynamic programing for *binpacking*.

4.1 Ryan and Foster

The idea of the Ryan&Foster branching is as follows:

Theorem 4 (Ryan&Foster [12])**.** *For a fractional solution λ exist k, r, s with*

$$0 < \sum_{t:\ W^k_{r,t}=1,\ W^k_{s,t}=1} \lambda_{kg_t} < 1 \tag{87}$$

Proof. Let λ be the current solution of the restricted master LP. And assume the solution violates the integrality condition in block k. Then there is a $\lambda_{kg_o} \notin \{0, 1\}$.

The Problem is a *set partitioning*, i.e. it holds

$$\sum_{i=1}^{n_k} W^k_{ri} \lambda_{kg_i} = 1$$

for each row r of the Matrix W^k.

Now let r be fixed by a row with $W^k_{ro} = 1$. (Such an r exists or the variable would not be needed for the BLP, i.e. can be fixed to an integer value or the BLP is not bounded)

As λ_{kg_o} is fractional there has to be a o' such that $\lambda_{kg_{o'}}$ is fractional too and it holds $W^k_{ro'} = 1$.

Assume there are no redundant constraints anymore. Then each row of the Matrix W^k is different. Therefore it is possible to choose a row s of W^k with $W^k_{so} = 1$ and $W^k_{so'} = 0$.

Then it is

$$1 \ = \ \sum_{i=1}^{n_k} W^k_{ri} \lambda_{kg_i} \ = \ \sum_{i:\ W^k_{ri}=1} \lambda_{kg_i} \ > \ \sum_{i:\ W^k_{ri}=1, W^k_{si}=1} \lambda_{kg_i} \ \geq \ \lambda_{kg_o} \ > \ 0$$

□

So in the branching scheme of Ryan&Foster one looks up the Matrix W^k for a given block k to contain a submatrix looking like $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ corresponding to the generators, say $x_i = \lambda_{kg_i}$, $x_j = \lambda_{kg_j}$.

If so, one defines the two disjunctive successor nodes by the hard branching constraints

$$x_i \;=\; x_j \qquad \text{“same”}$$

with

$$\sum_{t:\ W^k_{it}=1, W^k_{jt}=1} \lambda_{kgt} \geq 1 \;\Leftrightarrow\; \lambda(\langle (i,>,0),(j,>,0)\rangle) \geq 1,$$

which is like replacing the submatrix by $\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$,

and

$$x_i + x_j \leq\; 1 \qquad \text{“differ”}$$

with

$$\sum_{t:\ W^k_{it}=1, W^k_{jt}=1} \lambda_{kgt} \leq 0 \;\Leftrightarrow\; \lambda(\langle (i,>,0),(j,\leq,0)\rangle) \geq 1,$$

which is like replacing the submatrix by $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

Let us give a short example on which the Vanderbeck branching scheme is applied too.

Example 13. Let $K' = 1,\ n_1 = 5,\ U^1 = 3$, the *set partitioning* is given as a *binpacking*, i.e.

$$X^1 = \{x \in \{0,1\}^5 \mid \sum_{i=1\ldots5} w_i x_i \leq W\},$$

here $w = (6,4,3,2,7)$, $W = 10$ and the current master solution λ (given in lexicographic order) is

λ_g	$\frac{2}{3}$	0	$\frac{1}{3}$	$\frac{1}{3}$	0	0	0	0	$\frac{2}{3}$	0	$\frac{1}{3}$	0	0	$\frac{2}{3}$
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 Ryan&Foster branching scheme could now work as follows:
Applying the mapping discribed in the second section yields integer original variables correspond to the first component. But e.g. in the second component is $x^1_2 = \frac{2}{3} \notin \mathbb{Z}$. So one can choose this variable for a branching candidate and look up for another one in the same block with the submatrix $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

Here we can see the first and third column for x_2 and x_1 are of this structure. So one gets the two successor nodes in the B&P tree by:

Same: The hard branching constraint is $x_1 = x_2$ with the constraint $\lambda_{g_1} \geq 1$ in the master.

Differ: The hard branching constraint is $x_1 + x_2 \leq 1$ with the constraint $\lambda_{g_1} \leq 0$ in the master.

In this example is the *same* case only present in one column. This gives an idea that in general the tree could get unbalanced, as there could be more columns in the *differ* case (more combinations with other components possible) and makes the decision *same* weaker than this.

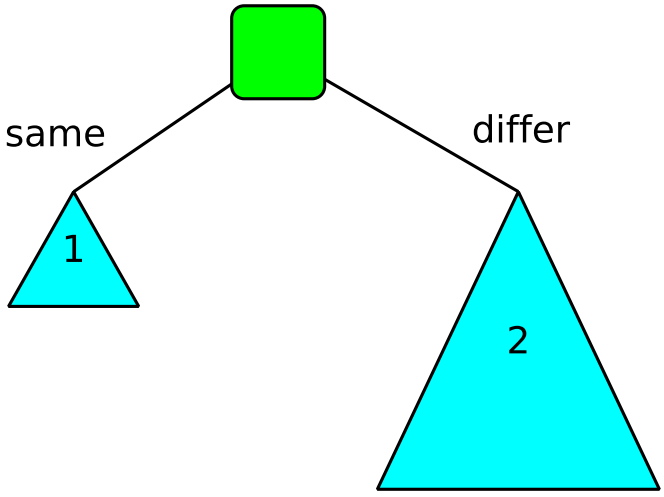


Figure 4: Unbalanced tree for Ryan&Foster branching

4.2 The generic scheme

In [1] the scheme is explained on a *set partitioning* problem with specialized properties given in the following proposition:

Proposition 6 ([1] Prop. 10)**.** *Let the master be a* set partitioning *and the current node be defined by the branching constraints for the corresponding block k denoted like above after the preprocessing.*

Then the generic scheme satisfies the following properties:

a) $(i, >, 0) \in S^p$ *for some* $i \Rightarrow L^p = 1$.

b) the tree of column classes corresponding to the current node for block k *has depth* 1 *and* $P - 1$ *of this classes contains* $(i, >, 0)$ *for at least one* i *and the last class* G^1 *satisfies* $G^1 = \{(i, \leq, 0) \mid (i, >, 0) \in S^p \text{ for some } p \in \{2, \ldots, P\}\}$ *with* $L^1 = U^k - \sum_{p=2}^{P} L^p$.

c) Columns of $G^0 \setminus \bigcup_{p=1}^{P} G^p$ *can be deleted.*

d) Separate another fractional master solution λ *yields in splitting up one further column class.*

e) Splitting up a column class G^p, $p \in \{2, \ldots, P\}$ *yields at most two feasible successor nodes.*

f) Splitting up column class G^1 *yields at most three feasible successor nodes.*

Proof. ([1])

a) Let $(i, >, 0) \in S^p$ and the node correspond to (S^p, L^p) not been pruned in the preprocessing. Then it holds (seen in section preprocessing) $\lambda(S^p) \leq g u_i^k = 1$, as the problem is a *set partitioning*. Therefore it is $0 < L^p = 1$.

b) Because of *a)* there could be no predecessor node which is not redundant for a node with $(i, >, 0) \in S^p$ for some i. The partition yields then a class containing $(i, \leq, 0)$ and ends up in G^1 as it contains no > 0 bound on a component.

c) Clearly by *b)* and the strip with columns in $G^0 \setminus \bigcup_{p=1}^{P} G^p$ has width zero, i.e. they cannot be *used*.

d) Because of the *set partitioning* constraints, the left hand sides L^p are upper and lower bounds, i.e. the constraints are satisfied with equality. In particular: for $(i, >, 0) \in S^p$ this is clear by *a)*. The constraint belong to L^1, i.e. G^1 is also satisfied with equality cause of the convexity constraint and *b)*. Therefore one of the existing column classes has to be splitted up further while branching.

e) Let G^p, $p > 1$ be the column class which is splitted up further while branching and let $(i, >, 0) \in S^p$. Then it is $\lambda(G^p) = 1$ like *d)* and

$$|F^p| = |\{g \in G^p \mid \lambda_{kg} - \lfloor \lambda_{kg} \rfloor\}| \; > 1$$

or it would not be splitted up further. Since all clomuns are different (also in F^p) there is a component j not yet bounded by S^p and defines a strip with fractional value $0 < \lambda(S^p, (j, >, 0)) < 1$. So in this branching step $S = \langle (j, >, 0) \rangle$ is a possible choice and the scheme goes on with at most two successor nodes.

f) Let $I(G^p) = \{i \mid \sum_{g \in G^p} x_i^{kg}(\lambda_{kg} - \lfloor \lambda_{kg} \rfloor) > 0\}$.

First assume $I(G^1) \cap I(G^p) \neq \emptyset$, *for some* $p \in \{2 \dots P\}$. Then one can branch like in e) or branch directly on G^1 by setting $S = (i, > 0)$ for any $i \in I(G^1) \cap I(G^p)$.

Now assume the other case, i.e. $I(G^1) \cap I(G^p) = \emptyset, \forall p = 2 \dots P$. Because of the *set partitioning* constraint it is $\lambda(S^1, (i, >, 0)) = 1$ for any $i \in I(G^1)$. Like in e): columns are different so there is a $j \in I(G^1)$, $j \neq i$ and it is $0 < \lambda(S^1, (i, >, 0), (j, > , 0)) < 1$. So $S = \langle S^1, (i, >, 0), (j, >, 0) \rangle$ is a possible choice. As the first $|S| - 2$ nodes are pruned by dominance (see section 3.4). So there are at most 3 successor nodes.

□

An Observation of this proposition is that the B&P tree for *set partitioning* problems is binary except a few nodes with three succcessor nodes.

Example 14 ([1] example 9)**.** Like above let $K' = 1$, $n_1 = 5$, $U^1 = 3$, the *set partitioning* is given as a *binpacking*, i.e.

$$X^1 = \{x \in \{0,1\}^5 \mid \sum_{i=1\dots5} w_i x_i \leq W\},$$

here $w = (6, 4, 3, 2, 7)$, $W = 10$ and the current master solution λ given in lexicographic order is

λ_g	$\frac{2}{3}$	0	$\frac{1}{3}$	$\frac{1}{3}$	0	0	0	0	$\frac{2}{3}$	0	$\frac{1}{3}$	0	0	$\frac{2}{3}$
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

Calling the separation method yields

$$\alpha_i = 1, \quad i = 1, \dots, 4.$$

Because of the *set partitioning* constraint non of the α_i can be fractional in the first call of *Separate*. Now say the first component has the highest priority for branching. So

add $(1, >, 0)$ to $record = \emptyset$ and call up *Separate* by recursion. Then one gets $\alpha_2 = \frac{2}{3}$. So adding this $(2, >, 0)$ to *record* and stop separating yields

$$S = \langle (1, >, 0), (2, >, 0) \rangle$$

and therefore the three successor nodes defined by

$$\begin{aligned} &\lambda(\langle (1, \leq, 0) \rangle) && \geq 3 \\ &\lambda(\langle (1, >, 0), (2, \leq, 0) \rangle) && \geq 1 \\ &\lambda(\langle (1, >, 0), (2, >, 0) \rangle) && \geq 1 \end{aligned}$$

The first node can be pruned by infeasibility, because the *set partitioning* constraint is violated for $i = 1$. (This can e.g. be detected by taking a look at the variable bounds gl_1, see [1])

The depth of the corresponding column class tree is 1.

(Now in [1] the resetting of bounds is applied. Here we go on without this.)

Now continue e.g. with the last node. The new master solution given in ILO is

λ_g	1	0	0	0	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	$\frac{1}{2}$
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

Following the separation method one gets $S = \langle (1, \leq, 0), (2, \leq, 0), (3, >, 0), (4, >, 0) \rangle$. The first two successor nodes are now pruned by dominance and one gets the three successor nodes defined by:

$$\begin{aligned} &\lambda(\langle (1, \leq, 0), (2, \leq, 0), (3, >, 0) \rangle) && \geq 2 \\ &\lambda(\langle (1, \leq, 0), (2, \leq, 0), (3, >, 0), (4, \leq, 0) \rangle) && \geq 2 \\ &\lambda(\langle (1, \leq, 0), (2, \leq, 0), (3, >, 0), , (4, >, 0) \rangle) && \geq 1 \end{aligned}$$

4.3 comparison

The proposition shows that in general the size of the tree of the Ryan&Foster scheme is close to the tree correspond to the generic branching scheme. Nevertheless it is not impossible to get three successor nodes. But as seen in the analysis the depth of the

tree is restricted to $O(nK'U)$ while the tree in the Ryan&Foster branching is bounded by $O(n^2K')$ and it is $U \leq n$. (see [1])

But if $U^k = 1$ it is $U = n$ and moreover the generic scheme looses the efficiency and becomes an enumeration of single strips like trying out.

Let us now take a look at the following example from [1] which shows that the intermediate Langrangian can become a dual bound which is stronger than in Ryan&Foster while using the splitted tighter pricing described in section 3.5:

Example 15 ([1] Example 10)**.** Given a *set partitioning*.

Let be $K' = 1,\ U^1 = R,\ n_1 = n$ and the subproblem is given by X. Further let the current node while following the Ryan&Foster branching be described by the branching constraints

$$\lambda(\langle(1,>,0),(2,>,0)\rangle) \geq 1 \tag{88}$$
$$\lambda(\langle(2,>,0),(3,>,0)\rangle) \geq 1 \tag{89}$$
$$\lambda(\langle(4,>,0),(5,>,0)\rangle) \leq 0 \tag{90}$$

The constraints (88) and (89) are the *same* constraints with the hard constraints $x_1 = x_2$ and $x_2 = x_3$ in the subproblem. While (90) is the *differ* constraint with the hard constraint $x_4 + x_5 \leq 1$ in the subproblem. So in all the subproblem becomes

$$\dot{X} = X \cap \{x|\ x_1 = x_2 = x_3,\ x_4 + x_5 \leq 1\}$$

Now try to reach an equal node while using the generic branching scheme.
Assume our first component bound sequence was

$$S^1 = \langle(1,>,0),(2,>,0)\rangle$$

With respect to 88 we choose the node correspond to the constraint

$$\lambda(\langle(1,>,0),(2,>,0)\rangle) \geq 1 \tag{91}$$

The next component bound sequence could have been

$$S^2 = \langle(1,>,0),(2,>,0),(3,>,0)\rangle$$

With respect to (89) we choose the node correspond to the constraint

$$\lambda(\langle(1,>,0),(2,>,0),(3,>,0)\rangle) \geq 1 \tag{92}$$

Note that (92) dominances the constraint (91). Now say the next component bound sequence is

$$S^3 = \langle (1, \leq, 0), (2, \leq, 0), (3, \leq, 0), (4, >, 0), (5, >, 0) \rangle$$

With respect to (90) we choose the node correspond to the constraint

$$\lambda(\langle (1, \leq, 0), (2, \leq, 0), (3, \leq, 0), (4, >, 0), (5, \leq, 0) \rangle) \geq 1 \tag{93}$$

Now (92) and (93) are the branching constraint in the node corresponding to the node of the Ryan&Foster branching. But now here we can use more information, i.e. we use the property of a *set partitioning* problem 74 and we get

$$\begin{aligned}
&\lambda(\langle (1, >, 0), (2, >, 0), (3, >, 0), (4, \leq, 0) \rangle) && \geq 1 && (94)\\
&\lambda(\langle (1, \leq, 0), (2, \leq, 0), (3, \leq, 0), (4, >, 0), (5, \leq, 0) \rangle) && \geq 1 && (95)\\
&\lambda(\langle (1, \leq, 0), (2, \leq, 0), (3, \leq, 0), (4, \leq, 0) \rangle) && \geq R - 2 && (96)
\end{aligned}$$

Constraint (94) combines (92) and (93) as the last constraint catches already x_4 and (96) is the convexity constraint combined with the already catched components $x_1, \ldots, x_4$.

In the pricingtree defined in section 3.5 we get the subproblems

$$\begin{aligned}
X^1 &= X \cap \{x |\ x_1 = x_2 = x_3 = 1,\ x_4 = 0\}\\
X^2 &= X \cap \{x |\ x_1 = x_2 = x_3 = 0,\ x_4 = 1,\ x_5 = 0\}\\
X^3 &= X \cap \{x |\ x_1 = x_2 = x_3 = x_4 = 0\}
\end{aligned}$$

Now it is

$$X^i \subseteq \dot{X},\ i = 1,\ 2,\ 3$$

and hence is

$$\alpha_{X^i}(\pi) \leq \alpha_{\dot{X}}(\pi),\ i = 1,\ 2,\ 3$$

and therefore is the dual bound for the generic branching scheme tighter than the one of the Ryan&Foster scheme.

Some numerical results are found in section 5.2.

5 Implementation & Testing

As already mentioned the branching scheme is implemented in GCG without the pricing described in [1]. And was tested on different instances. While doing the implementation work and trying to raise the result up to MIPs with more than one block type, a few problems and aspects came out. So this section first mentions the new and maybe different things to [1] and in the second subsection the numerical results.

5.1 Implementation

For an element (i, γ, b) of a component bound sequence S is the set $\{\leq, >\}$ not important in the following sense: In this masterthesis an element $\dot{x}_{[p]} = (i, \gamma, b) \in S$ is given with $\gamma \in \{>, \leq\}$. E.g. in the implementation it is $\gamma \in \{\geq, <\}$ as it is more comfortable since the redundant bound of $(i, \geq, 0)$ is always given.
So let from now on be $\gamma \in \{\geq, <\}$.

5.1.1 Separate with median

Take a deeper look at the function *Explore* or *Separate*.

In [1] is the method *Separate* given for a BLP. There the index i^* is removed from the index set I, but for a MIP it is said to stay there, but this can yield a not terminating function. See e.g. *example* 1 in [1] for a BLP and call up *Separate* without removing of i^*. So it has to be removed in one step. This case is e.g. when in this step the component in the corresponding column class is not only bounded by S but also fixed by it. So then another bound for this component does not make any sense.

This check is implemented while using a special value for b.

In [1] b is mentioned to be the median of the generators ith entry over all fractional strips correspond to F. But as in the analysis of the complexity of the function *Separate* (or *Explore*) in each step the partition of column classes has to be a real partition, i.e. none of the subclasses bounded by $\geq b$ resp. $< b$ is empty. Now consider the case where median is also the mininum, then this assumption is violated because the subclass bounded by $< b$ is of course empty. So in the implementation is not only the median calculated but also the minimum. If these two values are the same, the function called $getMedian(i)$ calculates the average m and returns $\lceil m \rceil$. This value is only equal to the minimum if and only if the entries are all the same. So doing a last check with $\lceil m \rceil = min$. If so, the corresponding component i is already fixed and therefore i is removed from I and the algorithm will look up for another component.

But even if the median defines a partition it can make trouble. Because it can happen that the scheme is not a branching scheme, as the fractional solution still is in at least

one of the successor nodes.

Take a look at the following example.

Example 16. Let $K' = 1,\ n_1 = 4,\ U^1 = 10$ and the current master solution λ given in lexicographic order is

λ_g	0.5	0.2	0.3	1	0.5	0.5	2	1	0.3	1.5	0.7	1	0.5
x_1	5	4	4	3	3	3	2	2	1	1	0	0	0
x_2	1	3	2	6	1	0	0	0	4	2	5	2	0
x_3	1	3	2	2	1	4	3	0	1	2	1	0	1
x_4	0	3	2	0	1	2	1	0	1	2	1	1	1

Moreover assume we are just at the root node, making things a bit easier to show.

Now *Separate* would detect e.g. the fractional value $\alpha_1 = 9.3$ and will separate the first component.

The median over the fractional columns is 3.

Choosing

$$S = \langle (1, \geq, 3) \rangle$$

yields the two branching constraints

$$\lambda(\langle (1, \geq, 3) \rangle) \geq 3$$

and

$$\lambda(\langle (1, <, 3) \rangle) \geq 10 - 3 + 1 = 8.$$

But the current fractional solution still satisfies

$$\lambda(\langle (1, \geq, 3) \rangle) \geq 3.$$

Bounding the first component by 5 or 1 would cut the current solution off.

So the example shows that $\lambda(S)$ has to be fractional like it is in [1] assumed. But just the median can violate this.
Therefore in the implementation the case of a detection of a fractional α_i the bound b for this component is calculated like in algorithm 6.

In words:
Beginning with $b = median$ and increase the bound by one resp. decrease until the width is fractional. This choice for b guarantees that $\lambda(S)$ is fractional so that the branching scheme still can be used and as it starts from the median it tries to yield a mostly balanced partition and therefore also a balanced tree. While beginning with $b = \max$ and decrease it by one until μ is fractional is a method which can help to overcover the component earlier.

Algorithm 6 BoundValue($\lambda, S,\ k$)

a) Set $b = getMedian(i)$
b) compute μ the width of the subclass bounded by $\geq b$
c) Set $j = 0$
while while($\mu \in \mathbb{N}_0$ **do**
    d) $j = j + 1$
    if j odd **then**
        e) $b = b + j$
    else
        f) $b = b - j$
    end if
end while
g) return b

5.1.2 Different blocks

Another question for the implementation was how should different types of blocks be respected in the branching scheme.

Consider two different ways. We will call them the parallel and the serial blocking.

At the **parallel blocking** in each branching step is the branching scheme applied for each type of block. This approach is surely the first and intuitive one.

Example 17. Let $K' = 2$ and the component bound sequences for block 1 be given as $S,\ |S| = s$ and $Q,\ |Q| = q$ for block 2.

By following the scheme we get $s + 1$ branching constraints for block 1 resp. $q + 1$ for block 2. Respecting all combinations of the constraints yields $(s + 1) \cdot (q + 1)$ succesor nodes.

Notice that the $(s + 1) \cdot (q + 1)$ nodes are all in the same depth of the B&P tree and moreover they are weighted even same too, i.e. without exploring one node further they are all in the same priority of the tree.

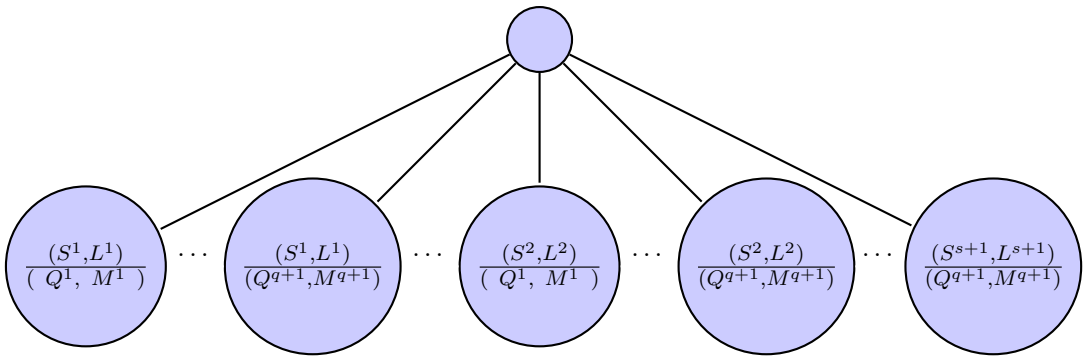


Figure 5: Example for parallel blocking with two block types. (S^p, L^p) means the pth branching constraint for block 1 resp. (Q^r, M^r) for block 2

The example shows that the tree gets a huge width. Also this can decrease the depth of it, all successor nodes "look the same" in the tree without exploring further. By the way: pruning by dominance now means to check not just the current constraint but also the combination of them too, which is more expensive.

In the implementation the other approach is chosen. The **serial blocking**.
Serial blocking means, that in each step of the branching scheme one only branch on one block type a time.

Example 18. Like in the previous example let be $K' = 2$ and the component bound sequences for block 1 be given as S, $|S| = s$ and Q, $|Q| = q$ for block 2.

So we get $(s + 1)$ succesor nodes in the first step and then maybe for each of these successor $(q + 1)$ successor nodes.

Notice that in all we can get at most $(s + 1) \cdot (q + 1)$ nodes too, but they are not all "looking the same", as at branching in the second block we got a feedback for the choice in the first block.

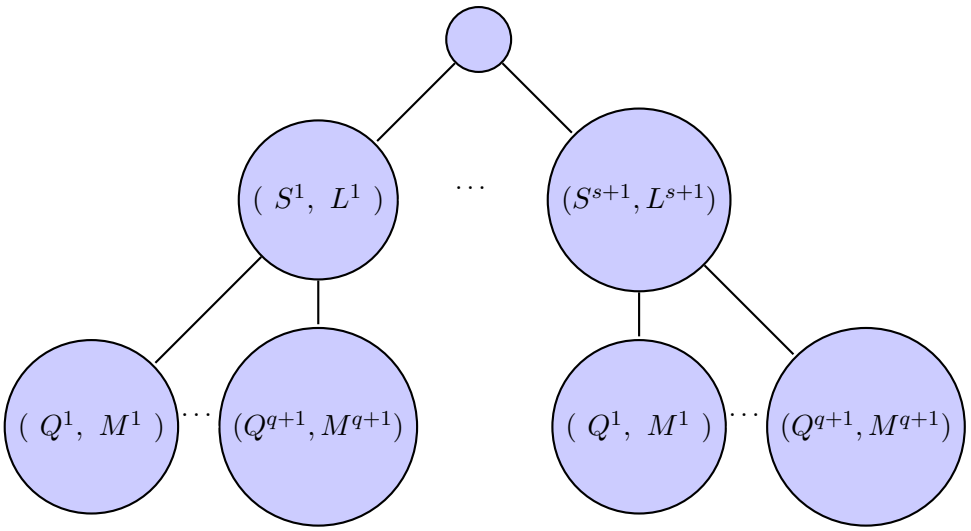


Figure 6: Example for serial blocking with two block types. (S^p, L^p) means the pth branching constraint for block 1, resp. (Q^r, M^r) for block 2. But notice: the second level of the tree may not need to be full computed or it can get different if there are linking variables.

The example shows that serial blocking is more adaptable than the parallel approach. Moreover in the second level of the tree the component bound Q could get another one if there are linking variables in the two blocks. The depth is higher but the width becomes shorter.

Prune by dominance can now be applied like it is discribed in [1] and the depth and width correspond to the one in the analysis of the tree in section 3.3.

5.1.3 The Priority

The priority of each component chosen in the method *Separate* resp. *Explore* is not specified in [1] at all.

Setting the priority all on the **same priority**, e.g. $priority_i = 1 \ \forall i$ will end up in a possible unbalanced tree as always the first component in the current index set will be chosen. So this is not the best choice at all.

A better choice for priority is **max-min priority**, the number of different entries of the generators in the current column subclass. So components with many different values in the generator are preferred to others. This fact can speed up the overcovering of a component what was a usefull tool in the analysis of the scheme to show that a component becomes integer.

Another approach to solve this problem is a **random priority**. For example will each i in the current index set I get the same probability, i.e. $P(i^* = i) = \frac{1}{|I|}\ \forall i \in I$.

A different idea came out while taking a look at the in the first section mentioned pseudocost branching. A **priority by pseudocosts**.
If there is a pseudocost for each component, which gives a value how efficient it is to branch on it, then this should be a good priority. But while the ideas for strong branching and reliability can keep up, the scheme does not branch on a component like it is described for branching on original variables. So the pseudocosts have to be computed in a different way.

If *Separate* or *Explore* is called we noticed that the last component (i, γ, b) in the computed component bound sequence S has to satisfy the fractionality width condition $\lambda(S) \notin \mathbb{N}$. So compute the costs for this component is a possible good idea.

Similar to the upwards and downwards pseudocosts, one can compute $|S| + 1$ pseudocosts for x_i by similar taking the difference of the objective value from the current node and the corresponding successor node and for the difference of the value of the original variable use the mapping discribed in section 3.1.

But it could happen that the original variable gets the same value than before, because unlike the branching on the variables directly only the width of the strip is changed in a branching step and another master solution can yield the same value for this component. So when choosing this sort of pseudocosts this case has to be carried out. And for large S there will be many costs of this type.

A last idea is a more heuristic one which respects the structure of the current strip. Assume $\gamma = \geq$ (the case $\gamma = <$ can applied inverse) and let $\mu \notin \mathbb{N}$ denote the width of the strip, i.e. $\mu = \lambda(S)$. The $|S| + 1$ successor nodes can be ordered into two types.
The first one implies $\lambda(S) \geq \lceil \mu \rceil$, in other words the width of the strip is increased by at least $\lceil \mu \rceil - \mu$. Using the fact that all components in the strip satisfies $\geq b$ the difference on the original variable in this strip is $\geq b(\lceil \mu \rceil - \mu)$. Notice that the width of the complementary strip is decreased by 1, so this value is not a bound on the whole component. For a heuristic cost value one can take $b(\lceil \mu \rceil - \mu)$ in the objective gain.
The other type of successor nodes implies $\lambda(S) \leq \lfloor \mu \rfloor$. If the component is overcovered then all components in the strip satisfies $\leq b$ too. The difference to the original variable in this strip is $\leq b(\mu - \lfloor \mu \rfloor)$. Notice that in the complementary strip the values in the component can satisfy $\geq b$ too and the component does not really have to be overcovered, i.e. fixed, at this time. So this value is not a bound on the whole component. For a heuristic cost value one can take $b(\mu - \lfloor \mu \rfloor)$ in the objective gain.
The resulting values computed with this objective gains, yield $|S| + 1$ likely pseudocosts which can be seen as similar values like one upwards and $|S|$ downwards pseudocosts like by branching in the original variable.

5.1.4 ILO or not ILO

As already mentioned is the lexicographic order not necessary for the branching scheme. Only the induced order is important. But given a BLP it is only needed in theory as integrality of the mapped original solution is euqivalent to the integrality on the master variables. (see [1] lemma 1). In particular this means that single strips have an integral width and can therefore be switched in the order of the column classes and still yield up in an integral original solution under the mapping of section 2.1. Moreover does the separate method *Explore* respect the nested partition of the column classes done at previous nodes in the B&P tree. So an ILO on F is not necessary at all. As this also holds for a MIP the ILO still becomes important in the following way:

Example 19. Let $K' = 1,\ n_1 = 2,\ U^1 = 2$ and the current master solution λ given in lexicographic order is

λ_g	0.5	0.5	0.5	0.5
x_1	2	2	0	0
x_2	1	0	1	0

Applying the mapping from section 3.1 yields the fractional original solution

$$x^1 = \begin{pmatrix} 2 \\ 0.5 \end{pmatrix},\ x^2 = \begin{pmatrix} 0 \\ 0.5 \end{pmatrix}$$

Now let be $C = \{\langle (2, >, 0) \rangle\}$. Then the current master solution λ given in ILO is

λ_g	0.5	0.5	0.5	0.5
x_1	2	0	2	0
x_2	1	1	0	0

Applying the mapping from section 3.1 on this order now yields

$$x^1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix},\ x^2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

and this is an integral solution.

The example shows, while not using an induced order on the column classes it is possible to get a fractional original solution but in e.g. ILO it would be integral. If this happens, the separate method *Explore* will not find a component bound sequence which separates the fractional solution. Hence this can be fixed while applying an ILO on the column strips with positive width and then use the mapping again to compute the integer solution x. Hence the number of strips is bounded by the number of constraints

m. Each compare of two strips takes at most $(O-L)$ operations. By using e.g. Quicksort or Mergesort which is in $O(m\ log(m))$ (see [13]) the whole ILO sort can be done in $O(m(O-L)\ log(m))$.

5.2 Computational Results

In this section we want to present some computational results. For the results we choose a small part of the binpacking instances called bison1 set by [24]. At each table are in the first column the name of the corresponding testset and then in the second, third and fourth column the number of computed nodes while using the generic branching scheme from section 3, the Ryan&Foster scheme (both on the discretization approach with aggregation) and the branching on the original variables while just applying the convexification approach. The first number is the number of the solving nodes and the second number in brackets gives the total number of computed nodes in the B&P tree. The next three columns contain the total time needed for solving and the last three columns contain the number of calls for each branching rule, i.e. the total number of branching steps.

The first group are the $N1*$ instances with $K' = 1,\ n = 2550,\ m = 100$ and $U = 50$.

	nodes			time			#calls		
instance	gen	R&F	orig	gen	R&F	orig	gen	R&F	orig
N1C1W1_C	15 (29)	8 (14)	1 (1)	0.69	0.50	10.53	10	7	0
N1C1W1_H	1 (1)	1 (1)	7 (12)	0.33	0.33	6.76	0	0	6
N1C1W1_O	15 (30)	5 (8)	1 (1)	0.54	0.42	7.37	10	4	0
N1C1W2_H	6 (11)	4 (6)	1(1)	0.49	0.45	7.53	3	3	0
N1C2W1_E	49 (80)	9 (16)	1 (1)	1.93	0.44	9.49	26	8	0
N1C2W1_J	3 (6)	5 (8)	1 (1)	0.41	0.41	8.06	2	4	0
N1C2W1_R	10 (16)	5 (8)	1 (1)	0.52	0.45	7.89	6	4	0
N1C2W1_T	8 (19)	5 (8)	5 (8)	0.68	0.52	11.36	6	4	4
N1C2W4_N	50 (68)	2 (4)	15 (28)	6.31	0.48	9.38	18	2	14
N1C3W1_A	18 (20)	13 (24)	23 (44)	1.19	0.52	13.55	7	12	22
N1C3W1_G	13 (15)	15 (28)	1 (1)	0.80	0.55	12.44	5	14	0
N1C3W1_H	237 (478)	7 (12)	1 (1)	13.01	0.49	13.85	151	6	0
N1C3W1_J	355 (456)	11 (20)	1 (1)	74.77	0.46	14.01	138	10	0
N1C3W1_M	174 (224)	9 (16)	1 (1)	26.63	0.51	13.74	70	8	0
N1C3W1_N	22 (33)	8 (14)	19 (36)	1.45	0.52	13.04	10	7	18
N1C3W1_O	150 (200)	12 (24)	18 (36)	15.22	0.48	13.61	61	12	18
N1C3W2_K	73 (155)	5 (10)	1 (1)	3.84	0.53	13.37	48	5	0
N1C3W2_P	105 (178)	17 (32)	25 (48)	6.60	0.58	15.79	61	16	24
N1C3W2_Q	119 (155)	15 (30)	1 (1)	9.46	0.53	12.63	49	15	0
N1C3W2_R	120 (227)	7 (14)	22 (44)	9.83	0.51	13.24	75	7	22
N1C3W4_F	51 (96)	5 (10)	1 (1)	3.83	0.53	14.46	32	5	0
N1C3W4_G	30 (73)	3 (4)	1 (1)	1.92	0.54	11.99	21	2	0
N1C3W4_L	35 (72)	10 (18)	29 (56)	1.75	0.57	13.52	25	9	28
N1C3W4_M	41 (95)	4 (6)	1 (1)	1.56	0.45	10.17	31	3	0
N1C3W4_R	119 (226)	8 (14)	27 (52)	7.71	0.50	11.29	70	7	26

The second group are the $N2*$ instances with $K' = 1,\ n = 10100,\ m = 200$ and $U = 100$.

	nodes			time			#calls		
instance	gen	R&F	orig	gen	R&F	orig	gen	R&F	orig
N2C1W1_A	30 (62)	10 (18)	1 (1)	5.61	1.72	61.97	21	9	0
N2C1W1_B	232 (424)	15 (30)	36 (70)	32.16	1.66	52.06	144	15	35
N2C1W1_J	20 (23)	4 (6)	1 (1)	2.71	1.49	41.70	8	3	0
N2C1W1_L	31 (62)	3 (6)	1 (1)	3.32	1.52	44.30	21	3	0
N2C1W1_M	231 (467)	10 (18)	1 (1)	139.18	1.83	75.44	152	9	0
N2C1W1_P	114 (173)	10 (20)	9 (18)	10.12	1.55	47.97	59	10	9
N2C1W1_Q	84 (169)	14 (26)	41 (80)	43.45	1.84	74.26	54	13	40
N2C1W1_R	5 (5)	2 (2)	19 (38)	1.66	1.59	50.46	2	1	19
N2C1W2_T	3 (11)	2 (4)	13 (26)	1.65	1.46	41.94	3	2	13
N2C2W1_A	103 (191)	11 (22)	1 (1)	15.23	1.55	50.01	65	11	0
N2C2W1_F	48 (89)	7 (12)	1 (1)	6.16	1.80	49.29	30	6	0
N2C2W1_G	159 (309)	17 (34)	27 (52)	18.80	1.80	56.41	99	17	26
N2C2W1_L	20 (38)	4 (8)	20 (38)	2.67	1.48	50.69	12	4	19
N2C2W1_N	40 (70)	12 (22)	36 (70)	7.69	1.77	71.65	24	11	35
N2C2W1_R	621 (1063)	16 (32)	58 (114)	411.38	1.90	84.37	359	16	57
N2C2W2_F	106 (248)	4 (6)	1 (1)	50.92	1.77	62.94	74	3	0
N2C2W2_S	3 (10)	2 (2)	1 (1)	1.67	1.51	54.45	3	1	0
N2C3W2_A	441 (898)	32 (64)	55 (108)	204.01	2.19	79.85	304	32	54
N2C3W2_B	348 (709)	25 (48)	48 (96)	236.22	2.01	79.65	236	24	48
N2C3W2_J	139 (300)	11 (20)	38 (74)	47.44	1.89	80.26	94	10	37
N2C3W2_K	101 (152)	26 (52)	64 (126)	89.87	2.19	80.82	45	26	63
N2C3W2_L	108 (108)	13 (24)	46 (90)	54.89	2.05	80.79	28	12	45
N2C3W4_B	294 (676)	8 (14)	38 (74)	152.06	2.08	62.03	197	7	37
N2C3W4_D	102 (234)	15 (28)	48 (96)	33.21	1.81	60.83	75	14	48
N2C3W4_E	115 (192)	7 (12)	1 (1)	92.40	1.70	67.33	58	6	0
N2C3W4_I	60 (96)	29 (58)	61 (122)	44.91	1.90	75.14	31	29	61
N2C3W4_L	16 (25)	15 (28)	44 (86)	4.50	1.82	60.63	8	14	43
N2C3W4_N	124 (253)	5 (8)	66 (130)	31.83	1.78	74.81	87	4	65
N2C3W4_O	47	16 (67)	1 (1)	34.87	1.78	70.29	19	15	0
N2C3W4_Q	304 (649)	9 (16)	17 (32)	340.25	1.87	65.53	191	8	16
N2C3W4_T	87 (214)	20 (38)	50 (100)	30.34	1.74	50.81	62	19	50

The third group are the $N3*$ instances with $K' = 1,\ n = 40200,\ m = 400$ and $U = 200$.

	nodes			time			#calls		
instance	gen	R&F	orig	gen	R&F	orig	gen	R&F	orig
N3C1W1_B	76 (146)	8 (16)	1 (1)	103.61	7.78	352.27	51	8	0
N3C1W1_P	58 (110)	6 (10)	1 (1)	58.58	7.95	363.73	37	5	0
N3C1W1_R	202 (329)	11 (20)	65 (130)	475.19	8.74	450.55	104	10	65
N3C1W2_Q	12 (40)	9 (16)	34 (68)	18.01	7.08	275.96	11	8	34
N3C1W2_R	32 (58)	7 (12)	54 (106)	120.40	7.41	329.22	17	6	53
N3C2W1_F	205 (380)	18 (36)	79 (158)	652.86	9.61	481.32	121	18	79
N3C2W2_P	24 (74)	2 (4)	98 (196)	41.84	7.65	379.38	20	2	98

One can see the size of the tree using the generic scheme is in practice often much larger than the tree for the Ryan&Foster branching without any fixed factor. This could depend on the formulation of the binpacking instances as a *set cover*, since for a set cover proposition 6 does not hold. Also the fact that sometimes up to four nodes are computed in one branching step. But the number of total nodes is always smaller than noted in section 3.3.

But nevertheless the time needed for branching is (except a few slow instances) between the time for the Ryan&Foster branching and the time needed for the branching on the original variables. Moreover by applying Ryan&Foster branching or the branching on the original variables the number of solving nodes is half of the number of total nodes, while by applying the generic scheme the numbers are nearly the same. This could stand for a balanced B&P tree. A last thing coming up while testing is the memory use of the scheme. As Ryan&Foster branching or the branching on original variables does not need information about data of the ancestor nodes (except the branching constraints), the generic scheme even does this. And keeping up the old information, say component bound sequences, the memory use grows quickly.

6 Conclusion

As seen, the generic branching scheme is a usefull scheme for practice and theory. As the time is in practice acceptable and even the number of nodes in the B&P tree, even more like it is in the theory, from the analysis of the scheme we know that each branching step is a real step towards integrality. Also on a *set partitioning* (omitting the possible stronger dual bounds) one would prefer the Ryan&Foster scheme, the generic scheme is not limited to this kind of instances. But as the numerical tests have shown, the scheme has it drawback in the use of memory. Even if the induced lexicographic sort is not needed for binary problems and only in a few leaf nodes for a MIP, the memory of C resp. component bound sequences of the ancestor nodes are still needed for pruning and separating. But omitting the large number of nodes and keep an eye on the CPU time and theory the generic scheme is an even good branching rule for practice like others.

References

[1] Francois Vanderbeck, Branching in branch-and-price: a generic scheme, Mathematical Programming Society, Ser. A (2011), http://dx.doi.org/10.1007/s10107-009-0334-1 130:249-294

[2] Gerad 25th anniversary series, Column Generation: edited by Guy Desaulniers, Jacques Desrosiers, Marius M. Solomon, Springer 2010

[3] Laurence A. Wolsey, Integer Programming, Wiley-Interscience series in discrete mathematics and optimization, 1998

[4] George L. Nemhauser and Laurence A. Wolsey. Integer and Combinatorial Optimization. John Wiley & Sons, 1988

[5] Christian Puchert. Primal Heuristics for Branch-and-Price Algorithms, Masterthesis, Technische Universität Darmstadt, Fachbereich Mathematik, Arbeitsgruppe Optimierung, Januar 2011

[6] Robert J. Wittrock. Dual nested decomposition of staircase linear programs, Mathematical Programming Studies, 1985, Volume 24, 65-86

[7] Gerald Gamrath. Generic Branch-Cut-and-Price. Masterthesis, Technische Universität Berlin, 2010

[8] Gerald Gamrath and Marco E. Lübbecke. Experiments with a Generic Dantzig-Wolfe Decomposition for Integer Programs. In Paola Festa, editor, Experimental Algorithms, volume 6049 of Lecture Notes in Computer Science, pages 239-252. Springer Berlin / Heidelberg, 2010

[9] Tobias Achterberg. SCIP: Solving constraint integer programs. Mathematical Programming Computation, 1(1), 2009. http://mpc.zib.de/index.php/MPC/ article/view/4

[10] Tobias Achterberg, Thorsten Koch, Alexander Martin. Branching rules revisited. Operations Research Letters 33 (2005) 42-54 , 2004. Elsevier

[11] Alexander Schrijver. Theory of linear and integer programming. Wiley 2000

[12] Marco E. Lübbecke. Lesson: Column Generation and Branch-and-Price. WS 11/12. RWTH Aachen

[13] Bernhard Korte, Jens Vygen. Kombinatorische Optimierung: Theorie und Algorithmen. Springer 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] A.M. Geoffrion, Lagrangian Relaxation for Integer Programming, Mathematical Programming Study 2 (1974) 82-114

[16] G. L. Nemhauser and L. A. Wolsey. Integer and Combinatorial Optimization. Wiley, New York, 1988

[17] C. Joncour, S. Michel, Ruslan Sadykov, D. Sverdlov, François Vanderbeck. Column Generation based Primal Heuristics. International Conference on Combinatorial Optimization (ISCO), Electronic Notes in Discrete Mathematics, 36:695-702, Elsevier 2010

[18] F. Vanderbeck, M.W.P. Savelsbergh. A Generic View of Dantzig-Wolfe Decomposition in Mixed Integer Programming. Operations Research Letters (http://dx.doi.org/10.1016/j.orl.2005.05.009), Volume 34, Issue 3, Pages 296-306, May 2006

[19] Jacques Desrosiers and Marco E. Lübbecke. A Primer in Column Generation. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, Column Genera- tion, pages 1-32. Springer US, 2005

[20] T. Achterberg. Constraint Integer Programming. PhD thesis, Technische Universität Berlin, 2007

[21] F. Vanderbeck. On dantzig-wolfe decomposition in integer program- ming and ways to perform branching in a branch-and-price algorithm. Operations Research, 48(1):111-128, 2000

[22] Hans D. Mittelmann. Mixed Integer Linear Programming Benchmark (serial codes). http://plato.asu.edu/ftp/milpf.html.

[23] F. Vanderbeck. IMA (Institute for mathematics and its applications) University Minnesota. Hot Topics Workshop: Mixed-Integer Programming July 25-29, 2005, Slides: Branching in branch-and.price algorithms. http://www.ima.umn.edu/matter/W7.25-29.05/activities/Vanderbeck-Francois/gbrsSlides.pdf

[24] P. Schwerin and G. Wäscher. The binpacking problem: A problem generator and some numerical experiments with ffd packing and mtp. International Transactions in Operational Research, 4(5-6):377-389, 1997. instances could be find on a webpage from the university of Jena at: http://www.wiwi.uni-jena.de/Entscheidung/binpp/

7 German Summary

In der vorliegenden Masterarbeit wird die allgemeine Branchingregel von Vanderbeck wiedergegeben, implementiert und getestet. Dabei wird zunächst im 1. Kapitel die notwendige Theorie wiederholt und im 2. Kapitel die Hauptquelle [1] wiedergegeben. Allerdings wird hier direkt auf den Fall eines gemischtganzzahligen Programms mit unterschiedlichen aggregierten Blöcken eingegangen. Dies geschieht zunächst so, wie es in der Quelle im letzten Kapitel beschrieben wird. Die ganzzahligen (bzw gemischt ganzzahligen) Beispiele sind dort jedoch nicht zu finden, ebenso wie die Analyse der Baumgröße, die auf der in [1] gegebenen Analyse für binäre Programme mit einem Block Typ basiert und diese verallgemeinert. Im Abschnitt "Implementation & Testing" wird dann auf einige Probleme und neue Ideen eingegangen, die während der Implementierung aufgetreten sind. Zuletzt folgt dann noch der Vergleich mit anderen Branchingregeln anhand verschiedenster geeigneter Instanzen.
Der Hauptvorteil dieser Branchingregel ist, dass sie Symmetrie vermeidet durch frühes pruning, nutzen der Aggregation, das eigentliche Pricing Orakel nicht ändert und durch nicht disjunkte Branchingungleichungen einen balancierten Baum liefert. Im 3. Kapitel wird besonders der in [1] gegebene Vergleich zwischen dieser Branchingregel und dem Ryan-Foster-Branching wiedergegeben. Außerdem ist in der Implementierung das bisherige Pricing in GCG unverändert geblieben, d.h. die einzelne Subprobleme werden nicht wie in [1] noch weiter aufgesplittet, sondern blockweise gelöst.

Hiermit versichere ich, die vorliegende Masterarbeit selbstständig und ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Aachen, den 3. Dezember 2012

(Marcel Schmickerath)