Contributed to the *Wiley Encyclopedia of Operations Research and Management Science (EORMS).*

Branch-Price-and-Cut Algorithms

Jacques Desrosiers
HEC Montréal and GERAD
3000, chemin de la Côte-Sainte-Catherine
Montréal, Québec, Canada H3T 2A7
jacques.desrosiers@hec.ca

Marco E. Lübbecke
Technische Universität Darmstadt
Fachbereich Mathematik, AG Optimierung
Dolivostr. 15, 64293 Darmstadt, Germany
luebbecke@mathematik.tu-darmstadt.de

April 8, 2010; revised July 7, 2010

Abstract

In many mixed integer programs there is some embedded problem structure which can be exploited, often by a decomposition. When the relaxation in each node of a branch-and-bound tree is solved by column generation, one speaks of branch-and-price. Optionally, cutting planes can be added in order to strengthen the relaxation, and this is called *branch-price-and-cut*. We introduce the common concepts of convexification and discretization to arrive at a Dantzig-Wolfe type reformulation of a mixed integer program. The relation between the original and the extended formulations helps us understand how cutting planes should be formulated and how branching decisions can be taken while keeping the column generation subproblems manageable.

Key words: Integer programming; Dantzig-Wolfe decomposition; column generation; cutting planes; branch-and-bound.

Decompositions and reformulations of mixed integer programs are classical approaches to obtaining stronger relaxations and reduce symmetry. These often entail the dynamic addition of variables (columns) and/or constraints (cutting planes) to the model. When the linear relaxation in each node of a branch-and-bound tree is solved by column generation, one speaks of branch-and-price. Optionally, as in standard branch-and-bound, cutting planes can be added in order to strengthen the relaxation, and this is called *branch-price-and-cut.* Now, having understood and familiarized oneself with the three concepts, branch-and-bound (see also 1.4.2.1), column generation (see also 1.1.2.7), and cutting planes (see also 1.4.4) one may think that the above explanation is the end of the story. Actually, this understanding is precisely where the story begins. Strengthening the relaxation by means of cutting planes and performing branching on fractional variables both interfere with column generation and may entirely ruin the mentioned advantages of a decomposition when done naively.

In early years, one attempted to circumvent these complications in an *ad hoc* fashion [1] but over time a generic theoretical understanding developed. The state-of-the-art relies on the relation between the original problem and its extended reformulation, as first used in [2].

There are very successful applications of branch-and-price in industry (see [3], and also e.g., 4.4.4, vehicle routing and scheduling) and also to generic combinatorial optimization problems like bin packing and the cutting stock problem [4], graph coloring [5], machine scheduling [6], the p-median problem [7], the generalized assignment problem [8], and many others. The method today is an indispensable part of the integer programming toolbox.

1 Column Generation

Consider the following *integer master problem*

$$\begin{array}{rlcl} \min & \sum_{j\in J} c_j\lambda_j & & \\ \text{subject to} & \sum_{j\in J} \mathbf{a}_j\lambda_j & \leq & \mathbf{b} \\ & \boldsymbol{\lambda} & \in & \mathbb{Z}_+^{|J|} \; . \end{array} \tag{1}$$

In many applications, $|J|$ is huge (but always finite) and we solve the linear relaxation of (1), called the *master problem*, by column generation as follows (see also 1.1.2.7). The *restricted master problem* (RMP) contains only a subset $J' \subseteq J$ of variables, initially possibly none. In each iteration, (a) we obtain $\boldsymbol{\lambda}^*$ and $\boldsymbol{\pi}^*$, the primal and dual optimal solutions to the RMP, respectively. Then, (b) the following pricing problem (subproblem) is to be solved

$$v \;:=\; \min_{\mathbf{x}\in X}\{c(\mathbf{x}) - \boldsymbol{\pi} a(\mathbf{x})\} \;, \tag{2}$$

where $c_j = c(\mathbf{x}_j)$ and $\mathbf{a}_j = a(\mathbf{x}_j)$ reflect that each column $j \in J$ is associated with an element $\mathbf{x}_j \in X$ from a domain X over which we can optimize, often a set of combinatorial objects like paths or other subgraphs (so, usually, the $\mathbf{x}_j$ bears much more information than just the column $a(\mathbf{x}_j)$). When $v < 0$ the variable λ_j and its coefficient column $(c_j, \mathbf{a}_j)$ corresponding to a minimizer $\mathbf{x}_j$ are added to the RMP, and the process is iterated. Otherwise, $v \geq 0$ proves that there is no such improving variable, and the current $\boldsymbol{\lambda}^*$ is an optimal solution to the master problem.

2 Decompositions and Reformulations

In general, when solving mathematical programs involving integer variables, a *good* model is of utmost importance, see e.g., [9] for some general considerations (see also 1.4.1, models). After all, the whole integer programming machinery is about better describing the convex hull of feasible integer solutions: tighter relaxations from a different modeling choice and cutting planes fulfill this purpose. Also branch-and-price is motivated by the perspective for better dual bounds and reduced problem symmetry. In the following we describe the groundwork.

2.1 Dantzig-Wolfe Decompositions for Integer Programs

The Dantzig-Wolfe decomposition principle in linear programming [10] was devised to exploit sparsity and special structure in linear programs (see also 1.1.2.3, Dantzig-Wolfe decomposition). However, it reveals its true strength only when adapted to integer programs. We assume that optimizing over the mixed integer set $X = \{\mathbf{x} \in \mathbb{Z}^n_+ \times \mathbb{Q}^q_+ \mid D\mathbf{x} \leq \mathbf{d}\}$ is "relatively easy," which however, is overshadowed by additional *complicated* constraints, i.e.,

$$\begin{array}{rrcl} \min & \mathbf{cx} & & \\ \text{subject to} & A\mathbf{x} & \leq & \mathbf{b} \\ & \mathbf{x} & \in & X \end{array} \tag{3}$$

is rather hard to solve when the "hidden" structure X is not exploited. The mixed integer program (3) is called the *original* problem in this context and we call $\mathbf{x}$ the *original* variables. As we will see, a decomposition will lead to a master and a pricing problem as above. It was noted already by Geoffrion [11] in the context of Lagrangean relaxation (see also 1.1.2.4) that decomposing (3) by solving over X as a mixed integer program (and penalizing the violation of $A\mathbf{x} \leq \mathbf{b}$ in the objective function) may yield a stronger dual bound than the standard linear relaxation when the convex hull $\operatorname{conv}(X)$ is not an integral polyhedron. In this article, X is assumed to be a pure integer set, i.e., $q = 0$; we only point to the mixed integer generalization when needed.

2.1.1 Convexification

The classical decomposition approach builds on the representation theorems by Minkowski and Weyl [12] and *convexifies* X, hence the name. Each $\mathbf{x} \in X$ can be expressed as a convex combination of finitely many extreme points $\{\mathbf{x}_p\}_{p\in P}$ plus a non-negative combination of finitely many extreme rays $\{\mathbf{x}_r\}_{r\in R}$ of $\operatorname{conv}(X)$, i.e.,

$$\mathbf{x} = \sum_{p\in P} \mathbf{x}_p\lambda_p + \sum_{r\in R} \mathbf{x}_r\lambda_r, \quad \sum_{p\in P} \lambda_p = 1, \quad \boldsymbol{\lambda} \in \mathbb{Q}^{|P|+|R|}_+ \quad . \tag{4}$$

Substituting for $\mathbf{x}$ in (3) and applying the linear transformations $c_j = \mathbf{cx}_j$ and $\mathbf{a}_j = A\mathbf{x}_j$, $j \in P \cup R$ one obtains an *extended formulation* equivalent to (3), which is an *integer master* *problem* as (1):

$$\begin{array}{rrcl}
\min & \sum_{p\in P} c_p\lambda_p + \sum_{r\in R} c_r\lambda_r & & \\
\text{subject to} & \sum_{p\in P} \mathbf{a}_p\lambda_p + \sum_{r\in R} \mathbf{a}_r\lambda_r & \leq & \mathbf{b} \\
& \sum_{p\in P} \lambda_p & = & 1 \\
& \boldsymbol{\lambda} & \geq & \mathbf{0} \\
& \mathbf{x} & = & \sum_{p\in P} \mathbf{x}_p\lambda_p + \sum_{r\in R} \mathbf{x}_r\lambda_r \\
& \mathbf{x} & \in & \mathbb{Z}^n_+ \ .
\end{array} \tag{5}$$

The constraints involving only $\boldsymbol{\lambda}$ variables are called *coupling constraints* and *convexity constraint*, respectively. It is important to note that integrality is still imposed on the original $\mathbf{x}$ variables. Due to the large cardinality of $P \cup R$ the LP relaxation of (5) is solved by column generation (see also 1.1.2.7 and Sect. 1), where the constraints linking $\mathbf{x}$ and $\boldsymbol{\lambda}$ variables can be dropped. Thus, only the dual variables $\boldsymbol{\pi}$ and π_0 remain relevant where π_0 corresponds to the convexity constraint. The pricing problem is the integer program

$$\min\{\mathbf{cx} - \boldsymbol{\pi}A\mathbf{x} - \pi_0 \mid \mathbf{x} \in X\} \ , \tag{6}$$

for which, ideally, a tailored combinatorial algorithm is available; and if not, we need to solve it with some standard IP solver which may be very time consuming.

2.1.2 Discretization

In contrast to convexification where $\operatorname{conv}(X)$ is reformulated, *discretization* is a reformulation of X itself. It enables us to require integrality on the master variables which is not valid in (5). Vanderbeck [13] introduced the concept since "it allows for the development of a unifying and complete theoretical framework to deal with all relevant issues that arise in the implementation of a branch-and-price algorithm." As we will see he was in particular thinking of cutting planes and branching. We need the fact [9] that every integer $\mathbf{x} \in X$ can be written as an *integral* combination

$$\mathbf{x} = \sum_{p\in P} \mathbf{x}_p\lambda_p + \sum_{r\in R} \mathbf{x}_r\lambda_r, \ \sum_{p\in P} \lambda_p = 1, \ \boldsymbol{\lambda} \in \mathbb{Z}^{|P|+|R|}_+ \tag{7}$$

with finite sets of *integer* points $\{\mathbf{x}_p\}_{p\in P} \subseteq X$ and *integer* rays $\{\mathbf{x}_r\}_{r\in R}$ of X. Note that we slightly abused notation here since the set P of *generators* is usually not identical to the corresponding extreme points of the convexification approach, and rays in R are scaled to be integer. Substitution for $\mathbf{x}$ in (3) yields an *integer* master problem

$$\begin{array}{rrcl}
\min & \sum_{p\in P} c_p\lambda_p + \sum_{r\in R} c_r\lambda_r & & \\
\text{subject to} & \sum_{p\in P} \mathbf{a}_p\lambda_p + \sum_{r\in R} \mathbf{a}_r\lambda_r & \leq & \mathbf{b} \\
& \sum_{p\in P} \lambda_p & = & 1 \\
& \boldsymbol{\lambda} & \in & \mathbb{Z}^{|P|+|R|}_+ \ ,
\end{array} \tag{8}$$

where integrality is now imposed on the master variables $\boldsymbol{\lambda}$, and again $c_j = \mathbf{c}\mathbf{x}_j$ and $a_j = A\mathbf{x}_j$, $j \in P \cup R$. Note that in fact $\lambda_p \in \{0, 1\}$ for $p \in P$. It is known [13] that the LP relaxation gives the same dual bound as (5). When solving (8) by column generation the pricing problem is the same as above, but it needs to be able to generate integer solutions in the interior of X. When X is bounded, (8) is a linear integer program even when the original problem (3) has non-linear cost function $c(\mathbf{x})$: Because of the convexity constraint, variables $\boldsymbol{\lambda}$ are binary, thus $c(\mathbf{x}) = c(\sum_{p\in P} \mathbf{x}_p\lambda_p) = \sum_{p\in P} c_p\lambda_p$ turns into a linear objective function.

The discretization approach generalizes to a mixed integer set X in that integer variables are discretized and continuous variables are reformulated using the convexification approach [14]. For general integer variables this is not straight forward using the convexification approach, but in the important special case $X \subseteq [0, 1]^n$ of combinatorial optimization, convexification and discretization coincide. Both approaches yield the same dual bound which equals that of Lagrangean relaxation (see also 1.1.2.5 and [15]).

2.2 Bordered Block-Diagonal Matrices

For many problems, $X = X_1 \times \cdots \times X_K$ (possibly permuting variables), that is, X decomposes into $X_k = \{\mathbf{x}_k \in \mathbb{Z}_+^{n_k} \times \mathbb{Q}_+^{q_k} \mid D_k\mathbf{x}_k \leq \mathbf{d}_k\}$, $k = 1, \ldots, K$, with all matrices and vectors of compatible dimensions and $\sum_k n_k = n$, $\sum_k q_k = q$. This is another way of saying that D can be brought into a block-diagonal form, so that the original problem (3) reads

$$\begin{array}{rrcl} \min & \sum_k \mathbf{c}_k\mathbf{x}_k & & \\ \text{subject to} & \sum_k A_k\mathbf{x}_k & \leq & \mathbf{b} \\ & \mathbf{x}_k & \in & X_k \ , \quad k = 1, \ldots, K \ . \end{array} \tag{9}$$

This is the classical situation for applying a Dantzig-Wolfe decomposition. Each $\mathbf{x}_k$ is expressed using (4) or (7), with the introduction of λ_{kj} variables, $j \in P_k \cup R_k$, where $P = \bigcup_{k=1}^K P_k$ and $R = \bigcup_{k=1}^K R_k$. It is an important special case that some or all $(D_k, \mathbf{d}_k)$ are identical, e.g., for bin packing, vertex coloring, or vehicle routing problems. This implies a symmetry since “the same” solution can be expressed in many different ways by permuting the k indices. Symmetry is beautiful in many areas of mathematics, however, for an integer program it may be a major source of inefficiency in a branch-and-bound algorithm and should be avoided by all means. Typically, one aggregates (sums up) the λ_{kp} variables, substituting $\nu_p := \sum_k \lambda_{kp}$, and adding up the K convexity constraints. Extreme rays need no aggregation. Choosing a representative P_1, we obtain the *aggregated* extended formulation

$$\begin{array}{rrcl} \min & \sum_{p\in P_1} c_p\nu_p + \sum_{r\in R} c_r\lambda_r & & \\ \text{subject to} & \sum_{p\in P_1} \mathbf{a}_p\nu_p + \sum_{r\in R} \mathbf{a}_r\lambda_r & \leq & \mathbf{b} \\ & \sum_{p\in P_1} \nu_p & = & K \\ & \boldsymbol{\nu} & \in & \mathbb{Z}_+^{|P_1|} \\ & \boldsymbol{\lambda} & \in & \mathbb{Z}_+^{|R|} \ , \end{array} \tag{10}$$

here in the discretization version. This also condenses the original $\mathbf{x}_k$ variables into aggregated original variables $\mathbf{z} = \sum_k \mathbf{x}_k$.

2.3 Extended Reformulations

Lifting a mixed integer problem to a higher-dimensional space, obtaining a stronger formulation there, and projecting this back to the original variables' space is a well-known concept in integer programming (see also 1.4.5.1, disjunctive inequalities and higher-dimensional representations). Not only for branch-and-price it is helpful to know some basic ideas. The very recommendable survey [16] presents decomposition approaches in this context.

A polyhedron $Q = \{(\mathbf{x}, \boldsymbol{\lambda}) \in \mathbb{R}^n \times \mathbb{R}^\ell \mid A\mathbf{x} + L\boldsymbol{\lambda} \leq \mathbf{b}\}$ is called an *extended formulation* of a polyhedron $O \subseteq \mathbb{R}^n$ if Q can be projected to O, i.e., if $O = \text{proj}_\mathbf{x}(Q)$, where $\text{proj}_\mathbf{x}(Q)$ denotes the projection of Q on the $\mathbf{x}$ variables, i.e., $\text{proj}_\mathbf{x}(Q) = \{\mathbf{x} \in \mathbb{R}^n \mid \exists \boldsymbol{\lambda} \in \mathbb{R}^\ell : (\mathbf{x}, \boldsymbol{\lambda}) \in Q\}$. Polyhedron Q is an extended formulation of the integer set X if $X = \text{proj}_\mathbf{x}(Q) \cap \mathbb{Z}^n_+$. More generally, Q may itself be a mixed integer set, and we call Q an extended formulation of the mixed integer set X if $X = \text{proj}_\mathbf{x}(Q)$. The discretization approach provides an example. We also speak of a *problem* being an extended formulation, e.g., we call the linear relaxation of the master problem (5) an extended formulation of the original problem (3).

As stated above, extended formulations are typically stronger than their original counterparts, and the special interest in Dantzig-Wolfe type extended reformulations Q of mixed integer sets X lies in the fact that they are *tight*, i.e., $\text{proj}_\mathbf{x}(Q) = \text{conv}(X)$. That is, they are best in a well-defined sense at the expense that they may contain an exponential number of variables.

It is of main interest in our context that some experience and creativity with projections and extended formulations may help us reformulating a problem *before* a Dantzig-Wolfe decomposition is applied, be it implicit or explicit. This becomes important when formulating cutting planes (Sect. 3) and branching rules (Sect. 4). In order to provide some intuition consider a flow-based formulation for the minimum spanning tree problem in a graph $G = (V, E)$ where one sends one unit of flow from a designated root node to all other nodes, i.e., $|V| - 1$ units in total. An integer variable x_{ij} represents the amount of flow on edge $(i, j) \in E$, and binary variables y_{ij} indicate whether $(i, j) \in E$ is in the spanning tree. Edge $(i, j) \in E$ may carry flow only when it is part of the tree, i.e., $x_{ij} \leq (|V| - 1) \cdot y_{ij}$. This "big-$M$" formulation can be much improved by extending the formulation by introducing a separate flow commodity $k = 1, \ldots, |V|$ for each node, that is, reformulate $x_{ij} = \sum_k x^k_{ij}$. The above mentioned constraint becomes $x^k_{ij} \leq y_{ij}$ which obviously better reflects the integrality of the y_{ij} variables, and thus gives stronger branching and cutting opportunities.

Reducing problem symmetry can be another reason to consider an extended formulation [17]. For problems like bin packing, graph coloring, and many others, binary variables x_{ij} assign items i to identical entities j. The symmetry in index j can be broken e.g., by binary variables z_{ij} which reflect the assignment of items i and j to the same entity but not items $i < k < j$. Cutting planes are reported to be effective at the cost of a more complicated pricing problem.

2.4 Reversing Dantzig-Wolfe Decomposition

Column generation can be applied without a prior decomposition. Examples are set covering or set partitioning problems like in the classical cutting stock problem or in vehicle and crew scheduling (see also 4.4.4). One directly formulates an integer master problem (1) together with a pricing problem (2). Without a decomposition, there is no *original* problem. However, as we describe later, such an original problem can be very helpful in designing branching rules and cutting planes. Consequently, we would like to construct a corresponding original problem from (1) and (2). That is, in a sense, one aims at *reversing* the Dantzig-Wolfe decomposition. Under a mild assumption which is typically true, this can be done [18]. The idea exploits the fact that the pricing problem is formulated in original variables, e.g., when the pricing problem constructs a path in a network, the decisions taken are typically whether an edge is included in the path or not. Conforming with our previous notation, we assume that we know an integer upper bound K on $\sum_{j\in J}\lambda_j$ in (1) (the maximum number of vehicles, paper rolls, bins, etc.). When the zero column $\mathbf{a}_0 = \mathbf{0}$ has cost $c_0 = 0$ we can assume equality constraints in an original formulation. We duplicate the pricing problem's domain K times, i.e., we set $X_k = X$, $k = 1, \ldots, K$, and obtain a bordered block-diagonal form:

$$\begin{array}{rrcll} \min & \displaystyle\sum_{k=1}^{K} c(\mathbf{x}_k) & & & \\ \text{subject to} & \displaystyle\sum_{k=1}^{K} a(\mathbf{x}_k) & = & \mathbf{b} & \\ & \mathbf{x}_k & \in & X_k & k = 1, \ldots, K\ . \end{array} \tag{11}$$

Performing a Dantzig-Wolfe decomposition on (11) one obtains a formulation equivalent to (1) where each of the K pricing problems contributes, due to the convexity constraints, at most one unit to the master solution. Note that this introduces the symmetry of identical subproblems by design of the construction and one needs to aggregate the $\mathbf{x}_k$ variables. We have produced a projection to the variables implicitly given in the pricing problem. This is just one option; symmetry avoiding projections are preferable, e.g., by explictly producing different pricing problems.

3 Cutting Planes

Just as in standard branch-and-cut adding valid inequalities can strengthen the LP relaxation, yielding what is known as branch-price-and-cut. In the earlier days, this combination has been considered problematic since the pricing problem must be aware of the coefficients in cutting planes as these need to be lifted when new variables are generated. With the introduction of the two viewpoints presented in this section the notion of *compatibility* became obsolete.

In the convexification approach (5) integrality is required on the $\mathbf{x}$ variables just as in the original problem (3). Thus, it is only natural to formulate valid inequalities on these variables. On the other hand, remembering the motivation for extended formulations (cf. Section 2.3), this ignores the potential of the higher-dimensional space, or in other words: We would like to formulate valid inequalities on the integer master variables of the discretization reformulation (8) as well. Our presentation follows [19], and several examples can be found therein.

3.1 Cutting Planes on the Original Variables

Assume that we know a set $F\mathbf{x} \leq \mathbf{f}$ of inequalities valid for the original problem (3), i.e.,

$$\begin{array}{rrcl} \min & \mathbf{cx} & & \\ \text{subject to} & A\mathbf{x} & \leq & \mathbf{b} \\ & F\mathbf{x} & \leq & \mathbf{f} \\ & \mathbf{x} & \in & X \end{array} \tag{12}$$

has the same integer feasible solutions. Via a Dantzig-Wolfe reformulation these inequalities directly transfer to the master problem, both in convexification and discretization:

$$\sum_{p\in P} \mathbf{f}_p \lambda_p + \sum_{r\in R} \mathbf{f}_r \lambda_r \leq \mathbf{f} \tag{13}$$

with the linear transformations $\mathbf{f}_j = F\mathbf{x}_j$ for $j \in P \cup R$. The dual variables $\boldsymbol{\alpha}$ of the additional inequalities (13) can be easily taken care of in the pricing problem, cf. (6):

$$\min\{\mathbf{cx} - \boldsymbol{\pi}A\mathbf{x} - \boldsymbol{\alpha}F\mathbf{x} - \pi_0 \mid \mathbf{x} \in X\} \tag{14}$$

or in other words, with the dynamic addition of valid inequalities formulated in the original $\mathbf{x}$ variables, only the pricing problem's objective function needs to be updated. We may alternatively enforce the cutting planes in the pricing problem by reducing its domain to $X_F = \{\mathbf{x} \in X \mid F\mathbf{x} \leq \mathbf{f}\}$. The reason for doing so is that inequalities added to X are convexified when we solve the pricing problem as an integer program, and thus we may hope for a stronger dual bound. The pricing problem (6) becomes

$$\min\{\mathbf{cx} - \boldsymbol{\pi}A\mathbf{x} - \pi_0 \mid \mathbf{x} \in X_F\} \ , \tag{15}$$

which, interestingly, may sometimes be of the same structure as before the modification. Sets P and R need to be updated in the master problems (5) or (8); moreover, variables λ_j with $F\mathbf{x}_j > \mathbf{f}$, $j \in P \cup R$, have to be eliminated.

Generic cutting planes (see also 1.4.4, polyhedral combinatorics) formulated on the original $\mathbf{x}$ variables sometimes seem to have little or no effect on improving the usually already strong dual bound (see also ?.?.?.?, decomposition techniques). This may also be due to the fact that usually no basic solution to the original problem is available on which several generic cutting planes rely (a crossover might help here). Currently, problem specific valid inequalities are the alternative to go.

It is known that separation of particularly structured (e.g., integer) points may be considerably easier than separating arbitrary fractional solutions. This motivates to use the decomposition to aid separation: A fractional master solution is a convex combination of integer solutions to the pricing problems which can be recovered and separated separately. In [20] this is called *structured separation* or *decompose and cut*.

3.2 Cutting Planes on the Master Variables

In many applications, and in particular in the discretization approach, the master variables are integer variables. It is clear that not every inequality in the master $\boldsymbol{\lambda}$ variables can be derived from the original $\mathbf{x}$ variables via a Dantzig-Wolfe decomposition, and this complicates matters as we will see. Assume that we already separated a set $G\boldsymbol{\lambda} \leq \mathbf{g}$ of valid inequalities in the master problem, i.e.,

$$\begin{array}{rrcl} \min & \sum_{p\in P} c_p\lambda_p + \sum_{r\in R} c_r\lambda_r & & \\ \text{subject to} & \sum_{p\in P} \mathbf{a}_p\lambda_p + \sum_{r\in R} \mathbf{a}_r\lambda_r & \leq & \mathbf{b} \\ & \sum_{p\in P} \mathbf{g}_p\lambda_p + \sum_{r\in R} \mathbf{g}_r\lambda_r & \leq & \mathbf{g} \\ & \sum_{p\in P} \lambda_p & = & 1 \\ & \boldsymbol{\lambda} & \in & \mathbb{Z}_+^{|P|+|R|} \ . \end{array} \tag{16}$$

The dual variables $\boldsymbol{\beta}$ of these cuts need to be respected when calculating reduced costs in the pricing problem—if we don't we may re-generate "cut off" variables and end up in an infinite loop of separation and pricing. Certainly, we would lose the strength of a cut if we didn't lift it. If we think of the cuts' coefficients of a variable λ_j, $j \in P \cup R$, as the result of a function $\mathbf{g}_j = g(\mathbf{a}_j)$, the pricing problem reads

$$\min\{\mathbf{cx} - \boldsymbol{\pi}A\mathbf{x} - \boldsymbol{\beta}g(A\mathbf{x}) - \pi_0 \mid \mathbf{x} \in X\} \ . \tag{17}$$

Function g can be quite complicated; it may be non-linear as in the case of a Chvátal-Gomory rank-1 cut [9, 21] with rational multipliers $\mathbf{u} \in [0,1)^n$

$$\sum_{p\in P} \lfloor \mathbf{u}\mathbf{a}_p \rfloor \lambda_p + \sum_{r\in R} \lfloor \mathbf{u}\mathbf{a}_r \rfloor \lambda_r \ \leq \ \lfloor \mathbf{u1} \rfloor \ .$$

It can be helpful from a conceptual viewpoint to introduce new variables $\mathbf{y} = g(A\mathbf{x})$ to compute the coefficients from a solution to the pricing problem. An example is to introduce new resources, one for each cutting plane, in the resource-constrained shortest path pricing problem typically used in vehicle routing problems. As just mentioned we cannot hope that g is linear, but if it is, i.e., $\mathbf{y} = g(A\mathbf{x}) = F\mathbf{x}$ we immediately see that cutting planes on the master variables contain those which can be derived from the original variables.

Desaulniers et al. [19] note that introducing additional variables in the pricing problem hints at an extended original (possibly non-linear) formulation from which cutting planes in the master variables follow by Dantzig-Wolfe decomposition (in the spirit of Section 2.4):

$$\begin{array}{rrcl} \min & \mathbf{cx} & & \\ \text{subject to} & A\mathbf{x} & \leq & \mathbf{b} \\ & \mathbf{y} & \leq & \mathbf{g} \\ & \mathbf{x} & \in & X \\ & \mathbf{y} & = & g(A\mathbf{x}) \ , \end{array} \tag{18}$$

where $\mathbf{y} \leq \mathbf{g}$ remains in the master problem while $\mathbf{y} \leq g(A\mathbf{x})$ goes in the pricing problem.

Cutting planes on the master variables is a recent topic. So far a successful separation of clique inequalities [22], Chvátal-Gomory rank-1 cuts [21] (and subsets [23]) has been reported only for a very few problems. The modified subproblems become harder, in particular when the computation of cut coefficients requires severe modifications in the pricing problem.

3.3 Using an Extended Formulation

As noted in Sect. 2.3, extended formulations may give rise to tighter relaxations as they may "better reflect integrality requirements of the problem." Column generation based algorithms often offer natural candidates for such extended formulations when the pricing problem is solved via dynamic programming. Simple examples are the bin packing [24] or cutting stock [25] master problems where the subproblem is a knapsack problem. The dynamic program for the knapsack problem with capacity B can be formulated as a longest path problem in an acyclic network of pseudo-polynomial size, namely with B nodes representing the used capacity of the knapsack. Arcs between vertices i and j represent picking an item of size $j-i$ when already a capacity of i is used. Zero-cost arcs between consecutive vertices represent unused capacity. Using these arcs as variables for a reformulation of the pricing problem one obtains a network flow problem.

In general, variables represent state transitions of the dynamic program and this may allow to formulate complex cutting planes, expressed in a simple way and without significant changes to the pricing problem. In the capacitated vehicle routing problem [26] this approach leads to variables x_{ij}^d which state that some vehicle arrives in j, coming from i, with a remaining capacity of d. Besides such "capacity-indexed formulations" e.g., time-indexed formulations are used in scheduling problems. Flow conservation reformulated in these variables gives what they call a *base equality* from which many families of valid inequalities can be derived.

There is good experience with such kinds of cutting planes also for the capacitated minimum spanning tree problem [27], machine scheduling problems [28], and several more [16].

4 Branching

Even though branching decisions can be imposed by additional constraints (which we know how to do) we still have to *find* good branching rules. Disjunctive branching on the master variables is either not feasible (in convexification nobody asks for integer master variables) or not advisable: In discretization branching a master variable to zero has essentially no effect on the dual bound, while the up-branch significantly changes the solution, and thus potentially the dual bound. This produces an unbalanced search tree. Moreover, down-branching forbids certain solutions to the pricing problem to be re-generated. This problem brought up the notion of *compatibility* between pricing problem and branching rule which means that the pricing problem should not complicate after branching. Working simultaneously with original and master formulation, i.e., branching on original variables helped a lot, but introduces new problems, in particular when pricing problems are identical; we follow the classification of [16] which contains a thorough exposition. Let $\boldsymbol{\lambda}^*$ denote an optimal solution to the restricted master problem.

4.1 Convexification: Branching on Original Variables

When all pricing problems are distinct, in particular, when there is only one pricing problem, the convexification approach with its integrality requirement on original variables is the natural way to go. As is immediate by (5), branching candidates are all original x_i variables with $x_i^* = \sum_{j \in P \cup R} x_{ji} \lambda_j^* \notin \mathbb{Z}_+$, where x_{ji} denotes the i-th component of $\mathbf{x}_j$, $j \in P \cup R$. Dichotomic branching on x_i creates two new problems, on the down-branch by imposing $x_i \le \lfloor x_i^* \rfloor$, and on the up-branch by requiring $x_i \ge \lceil x_i^* \rceil$. There are two general options on how to enforce the branching decision, either in the master problem or in the pricing problem. We only discuss the down-branch, the up-branch is handled analogously.

Master Problem. The branching constraint $x_i \le \lfloor x_i^* \rfloor$ is reformulated via convexification, i.e., we add to the master problem (5) the constraint

$$\sum_{p \in P} x_{pi}\lambda_p + \sum_{r \in R} x_{ri}\lambda_r \quad \le \quad \lfloor x_i^* \rfloor \ . \tag{19}$$

The additional dual variable α_i is respected in the pricing problem as in (14), i.e., only its objective function needs modification. However, no integer points in the interior of $\operatorname{conv}(X)$ can be obtained from the pricing problem, and we may miss an optimal solution in the case of general integer variables $\mathbf{x}$.

Pricing Problem. The second option is to change the bound $x_i \le \lfloor x_i^* \rfloor$ directly in the pricing problem, which forbids the generation of extreme points and rays which violate the branching decision. Master variables already present but incompatible with the branching decision need to be eliminated. This can be done by adding

$$\sum_{\substack{j \in P \cup R: \\ x_{ji}=1}} \lambda_j \ = \ 0 \qquad \text{or equivalently,} \qquad \sum_{\substack{j \in P \cup R: \\ x_{ji}=0}} \lambda_j \ = \ 1 \tag{20}$$

to the master problem (5) which can be seen as modifying the convexity constraint (its dual variable is still denoted by π_0). The pricing problem then becomes

$$\min\{\mathbf{cx} - \boldsymbol{\pi} A\mathbf{x} - \pi_0 \mid \mathbf{x} \in X \cap \{\mathbf{x} \mid x_i \le \lfloor x_i^* \rfloor\}\} \ . \tag{21}$$

This may complicate the pricing problem, however, if it stays tractable, this option is to be preferred. The main reason is the potentially stronger dual bound from the master problem relaxation since the bound change is convexified:

$$\begin{aligned} & \min\{\mathbf{cx} \mid A\mathbf{x} \le \mathbf{b},\ \mathbf{x} \in \operatorname{conv}(X),\ x_i \le \lfloor x_i^* \rfloor\} \\ \le \quad & \min\{\mathbf{cx} \mid A\mathbf{x} \le \mathbf{b},\ \mathbf{x} \in \operatorname{conv}(X \cap \{\mathbf{x} \mid x_i \le \lfloor x_i^* \rfloor\})\} \ . \end{aligned} \tag{22}$$

Furthermore, adding disjunctive bounds to the pricing problem, i.e., partitioning its domain allows to generate points in the interior of $\operatorname{conv}(X)$ after branching.

Our presentation discussed the root node but both options directly extend to any node in the tree. One only needs to keep track of the modifications to the master and pricing problems which are local to subtrees. Both options generalize to the case of mixed integer programs.

4.2 Discretization: Avoiding the Symmetry

Branching on original variables works well when all pricing problems are distinct since

$$\mathbf{x}_k = \sum_{p \in P_k} \mathbf{x}_p \lambda_{kp} \tag{23}$$

defines a unique projection from the $\boldsymbol{\lambda}$ variables into the original $\mathbf{x}$ variable space. As pointed out in Section 2.2, the case of identical pricing problems bears a symmetry which should be avoided. One may aggregate original variables $\sum_k \mathbf{x}_k = \mathbf{z}$ and obtain a single pricing problem as above. Branching decisions disaggregate variables again and create distinct pricing problems [18]. However, as any permutation of the index set $\{1, \ldots, K\}$ gives an equivalent solution, this does not eliminate the symmetry in the $\mathbf{x}_k$ variables [29].

Consider the aggregated master problem (10) of the discretization approach. Disaggregation of the variables $\nu_p = \sum_k \lambda_{kp}$ and using (23) to obtain an original $\mathbf{x}$ solution is neither unique nor does integrality of $\boldsymbol{\nu}$ necessarily imply integrality of $\mathbf{x}$. The trick to avoid these shortcomings and symmetry at the same time is to present a projection from the master into the original variable space which does not use the one-to-one correspondence (23) between λ_{kp} and $\mathbf{x}_k$ variables. In other words, the grouping of $\boldsymbol{\lambda}$ variables is only implicit. Vanderbeck [29] (see also [16]) proposed to obtain values $\mathbf{x}_1^*, \ldots, \mathbf{x}_K^*$ (in that order) by summing variables λ_{kp} in lexicographic order of the corresponding $\mathbf{x}_p$, where $\mathbf{x}_q \prec \mathbf{x}_p$ means that $\mathbf{x}_q$ precedes $\mathbf{x}_p$ in that ordering. For all $k = 1, \ldots, K$ and $p \in P$ let

$$\lambda_{kp}^* \quad = \quad \min \left\{ 1,\ \nu_p - \sum_{\kappa=1}^{k-1} \lambda_{\kappa p}^*,\ \ \max \left\{ 0, k - \sum_{q : \mathbf{x}_q \prec \mathbf{x}_p} \nu_q^* \right\} \right\} \ . \tag{24}$$

We obtain $\mathbf{x}_k^* = \sum_{p \in P} \mathbf{x}_k \lambda_{kp}^*$. The lexicographic sorting guarantees that we always work with a unique representative solution $\mathbf{x}$ out of the many symmetric possibilities. This is a standard trick which has been used in symmetry breaking of integer programs recently [30].

Aggregate Original Variables. When there happens to be a fractional aggregate original variable value $y_i^* = \sum_{p \in P} x_{pi} \nu_p^* \notin \mathbb{Z}_+$, which needs not be the case in general, branching can be performed on such a variable by imposing

$$y_i = \sum_{p \in P} x_{pi} \nu_p^* \leq \lfloor y_i^* \rfloor \qquad \text{or} \qquad y_i = \sum_{p \in P} x_{pi} \nu_p^* \leq \lceil y_i^* \rceil \tag{25}$$

in the master. This only affects the pricing problem's objective function but this may considerably change it's character. This simple rule may give only little improvement on the dual bound [16].

Auxiliary Original Variables. When the previous rule fails, i.e., when an integer $\mathbf{y}$ does not yield an integer $\mathbf{x}$ ("the set of branching objects is not rich enough." [16]), one may try to work with an extended original formulation by introducing auxiliary variables to branch on, cf. Sect. 2.3. As an example consider the set partitioning problem

$$\min \left\{ \sum_{p \in P} \mathbf{c}\mathbf{x}_p \nu_p \mid \sum_{p \in P} \mathbf{x}_p \nu_p = \mathbf{1},\ \sum_{p \in P} \nu_p = K,\ \boldsymbol{\nu} \in \{0,1\}^{|P|} \right\} \ . \tag{26}$$

Many problems lead to such a Dantzig-Wolfe reformulation like bin packing or vertex coloring. For the latter problem, original variables $x_{ki} \in \{0, 1\}$ state whether vertex i receives color k and columns $\mathbf{x}_p$ correspond to independent sets in the underlying graph. Aggregate variables

$y_i^* = \sum_{p\in P} x_{pi} v_p^* = 1$ for any master solution, so the previous rule does not apply. However, it is well-known that in a fractional master solution there must exist rows i and j with

$$\sum_{\substack{p\in P\\ x_{pi}=x_{pj}=1}} \nu_p^* \;=:\; w_{ij}^* \notin \{0,1\} \ . \tag{27}$$

We (conceptually) introduce w_{ij} as auxiliary variables in the original problem for every pair (i,j) of vertices (and in the pricing problem as well) and branch on these variables. It may not be straight forward to impose the branching decision in the pricing problem directly; Ryan and Foster branching [31] is an example in which $w_{ij} \in \{0,1\}$ is enforced by letting $x_i = x_j$ in one branch and $x_i \neq x_j$ in the other. When the pricing problem is solved via a combinatorial algorithm, often a dynamic program, this naturally suggests an extended formulation of the pricing problem which translates to an extended original formulation [16], see also Sect. 3.3.

Nested Partition of the Convexity Constraint. The most general rule is to split the master variables by modification of the convexity constraint [29]. If

$$\sum_{p\in P: x_{pi}\geq \ell_i} \nu_p = \delta \notin \mathbb{Z}_+ \tag{28}$$

for an index i (corresponding to original variable x_i) and an integer bound ℓ_i, one creates two branches with

$$\sum_{p\in P: x_{pi}\geq \ell_i} \nu_p \geq \lceil \delta \rceil \qquad \text{or} \qquad \sum_{p\in P: x_{pi}\leq \ell_i - 1} \nu_p \geq K - \lfloor \delta \rfloor \ . \tag{29}$$

The pricing problems must respect the variable bounds $x_i \geq \ell_i$ and $x_i \leq \ell_i - 1$, respectively. In order to guarantee a fractional δ in (28) one may need to impose bounds on a set S of original variables. In fact, such sets are found recursively, and this generalizes the partition in (29) in a nested way, producing more than two branches in general. The pricing problems must respect the bounds on variables in S and the respective complementary sets. There are several technical details to consider for which we refer to [29].

This last rule provides the strongest dual bound among the above proposals and implies the smallest impact in the pricing problem (only bound changes). It should be noted that points in the interior of $\text{conv}(X)$ can be generated with this generic rule and that the depth of the search tree is polynomially bounded.

5 Implementation Issues

Even when the whole lot of work of implementing a branch-price-and-cut algorithm will pay off, it will be a whole lot of work, even in 2010. We have seen that the freedom of choice can be enormous and some experience will certainly help. Nonetheless, it has never been easier than today with the great body of literature available.

At least when working with a convexification approach, one needs access to the values of the original variables. A trivial (but probably not efficient) way of doing this is to keep the constraints linking them to the master variables in the formulation (keeping the master constraints on the original variables is called the *explicit master* [17]). This facilitates e.g., branching on binary original variables since a simple bound change on the $\mathbf{x}$ variables implies eliminating incompatible master variables since their upper bounds are automatically changed to zero.

Even though an original (fractional) solution $\mathbf{x}^*$ is available, there is a drawback which has not been satisfactorily addressed so far: Typically, $\mathbf{x}^*$ is not a basic solution. This is important since several generic cutting planes and primal heuristics rely on that. Performing a cross-over has been suggested (Matthew Galati in personal communication referred to John Forrest) as a possible remedy but there are no experiences reported on this yet.

5.1 Frameworks

There are several frameworks which support the implementation of branch-and-price algorithms like `ABACUS` [32], `BCP` [33], and `MINTO` [34], `SCIP` [35], `SYMPHONY` [36], to name only a few. In addition there are codes which perform a Dantzig-Wolfe decomposition of a general (mixed) integer program, and handle the resulting column generation subproblems in a generic way. `BaPCod` [37] is a "prototype code that solves mixed integer programs (MIPs) by application of a Dantzig-Wolfe reformulation technique." The COIN-OR initiative (`www.coin-or.org`) hosts a generic decomposition code, called `DIP` [38] (formerly known as `DECOMP`), which is a "framework for implementing a variety of decomposition-based branch-and-bound algorithms for solving mixed integer linear programs" as described in [20]. The constraint programming `G12` project develops "user-controlled mappings from a high-level model to different solving methods," one of which is branch-and-price [39]. The attempt to turn the branch-price-and-cut framework `SCIP` into a branch-price-and-cut solver is called `GCG` [40].

5.2 When everything fails...

Many problems which are approached by branch-price-and-cut are so large and complex that optimal solutions are out of reach in a reasonable computation time. What to do then? The most honorable answer to that is: Research your problem! Is there any particular structure you can exploit, e.g., by using a combinatorial algorithm to solve your pricing problems (instead of solving them as MIPs); by formulating cutting planes; or by re-thinking the entire formulation? After all, this is what drives the innovations! The most practicable answer probably is to run a profiler to check where your code spends the CPU time, and search the bag of tricks for accelerating the weak spots. In particular, you may consider acceleration techniques [41] for solving the relaxations by column generation. The quickest (and sometimes the most promising, but certainly the dirtiest) answer is to go with a heuristic. In particular in practical applications you may not need to close the last percents of the optimality gap. Many practitioners will use *price-and-branch*, i.e., column generation is used only in the root node. In particular, if the problem is of set covering type (which it often is), one may branch on master variables and don't care about the theory. This is not elegant, but it often works. However, if the solver allows this, one should try to fix some variables (e.g., by branching) and generate further columns; this usually perceptively improves the solution quality.

6 A Remark and Recommendations for Further Reading

A final remark on the notion *branch-price-and-cut*. There is consent on using *branch-and-cut* and *branch-and-price*; so consequently the integration was named *branch-and-cut-and-price* in the first references. Adam Letchford (personal communication, 2005) remarked that a better style English is to omit the first *and*. Additionally exchanging *cut* and *price* reflects their order when solving the relaxation in each node, so we suggest to use *branch-price-and-cut*.

The classical—by now a bit outdated—survey on branch-and-price is [42]. The book [43] on column generation is, in fact, a book on branch-and-price and contains a lot of applications, in particular vehicle routing, the cutting stock problem, and machine scheduling. The book also contains an introductory text to the topic [44] with a focus on convexification. Implementation issues can be found in [45].

References

[1] G.L. Nemhauser and S. Park. A polyhedral approach to edge coloring. *Oper. Res. Lett.*, 10(6):315–322, 1991.

[2] J. Desrosiers, F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks*, 14:545–565, 1984.

[3] J. Desrosiers and M.E. Lübbecke. Selected topics in column generation. *Oper. Res.*, 53(6):1007–1023, 2005.

[4] F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Math. Programming*, 86(3):565–594, 1999.

[5] A. Mehrotra and M.A. Trick. A column generation approach for graph coloring. *INFORMS J. Comput.*, 8(4):344–354, 1996.

[6] J.M. van den Akker, J.A. Hoogeveen, and S.L. van de Velde. Parallel machine scheduling by column generation. *Oper. Res.*, 47(6):862–872, 1999.

[7] A. Ceselli and G. Righini. A branch-and-price algorithm for the capacitated p-median problem. *Networks*, 45(3):125–142, 2005.

[8] M.W.P. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Oper. Res.*, 45(6):831–841, 1997.

[9] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, Chichester, 1988.

[10] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Oper. Res.*, 8:101–111, 1960.

[11] A.M. Geoffrion. Lagrangean relaxation for integer programming. *Math. Programming Stud.*, 2:82–114, 1974.

[12] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Chichester, 1986.

[13] F. Vanderbeck. On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Oper. Res.*, 48(1):111–128, 2000.

[14] F. Vanderbeck and M.W.P. Savelsbergh. A generic view of Dantzig-Wolfe decomposition in mixed integer programming. *Oper. Res. Lett.*, 34(3):296–306, 2006.

[15] F. Vanderbeck. A generic view of Dantzig-Wolfe decomposition in mixed integer programming. *Oper. Res. Lett.*, 34(3):296–306, 2006.

[16] F. Vanderbeck and L. Wolsey. Reformulation and decomposition of integer programs. In M. Jünger, Th.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi, and L.A. Wolsey, editors, *50 Years of Integer Programming 1958–2008*. Springer, Berlin, 2010.

[17] Marcus Poggi de Aragão and Eduardo Uchoa. Integer program reformulation for robuts branch-and-cut-and-price. In *Annals of Mathematical Programming in Rio*, pages 56–61. Búzios, Brazil, 2003.

[18] D. Villeneuve, J. Desrosiers, M.E. Lübbecke, and F. Soumis. On compact formulations for integer programs solved by column generation. *Ann. Oper. Res.*, 139(1):375–388, 2005.

[19] G. Desaulniers, J. Desrosiers, and S. Spoorendonk. Cutting planes for branch-and-price algorithms. Les Cahiers du GERAD G-2009-52, HEC Montréal, 2009. Forthcoming in Networks.

[20] T.K. Ralphs and M.V. Galati. Decomposition and dynamic cut generation in integer linear programming. *Math. Programming*, 106(2):261–285, 2006.

[21] B. Petersen, D. Pisinger, and S. Spoorendonk. Chvátal-gomory rank-1 cuts used in a Dantzig-Wolfe decomposition of the vehicle routing problem with time windows. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 397–419. Springer, Berlin, 2008.

[22] S. Spoorendonk and G. Desaulniers. Clique inequalities for the vehicle routing problem with time windows. *INFOR*. Forthcoming.

[23] M. Jepsen, B. Petersen, S. Spoorendonk, and D. Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Oper. Res.*, 56(2):497–511, 2008.

[24] J.M. Valério de Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Ann. Oper. Res.*, 86:629–659, 1999.

[25] J.M. Valério de Carvalho. Exact solution of cutting stock problems using column generation and branch-and-bound. *Int. Trans. Opl. Res.*, 5(1):35–44, 1998.

[26] R. Fukasawa, H. Longo, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa, and R. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Math. Programming*, 106(3):491–511, 2006.

[27] E. Uchoa, R. Fukasawa, J. Lysgaard, A.A. Pessoa, M. Poggi de Aragão, and D. Andrade. Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation. *Math. Programming*, 112(2):443–472, 2008.

[28] A. Pessoa, E. Uchoa, M. Poggi de Aragão, and R. Rodrigues. Algorithms over arc-time indexed formulations for single and parallel machine scheduling problems. Report RPEP Vol. 8 no. 8, Universidade Federal Fluminense, 2008.

[29] F. Vanderbeck. Branching in branch-and-price: A generic scheme. *Math. Programming*, 2010. In press.

[30] F. Margot. Symmetry in integer linear programming. In M. Jünger, Th.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi, and L.A. Wolsey, editors, *50 Years of Integer Programming 1958–2008*. Springer, Berlin, 2010.

[31] D.M. Ryan and B.A.Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. North Holland, Amsterdam, 1981.

[32] Michael Jünger and Stefan Thienel. The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization. *Softw. Pract. Exper.*, 30(11):1325–1352, 2000.

[33] T.K. Ralphs and L. Ladányi. *COIN/BCP User's Manual*, 2001. `http://www.coin-or.org/Presentations/bcp-man.pdf`.

[34] G.L. Nemhauser, M.W.P. Savelsbergh, and G.S. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Oper. Res. Lett.*, 15:47–58, 1994.

[35] T. Achterberg. SCIP: Solving constraint integer programs. *Math. Programming Computation*, 1(1):1–41, 2009.

[36] T.K. Ralphs. Symphony version 5.1 users manual. Corl laboratory technical report, 2006.

[37] F. Vanderbeck. BaPCod – a generic branch-and-price code. `https://wiki.bordeaux.inria.fr/realopt/pmwiki.php/Project/BaPCod`, 2005.

[38] T.K. Ralphs and M.V. Galati. DIP – decomposition for integer programming. `https://projects.coin-or.org/Dip`, 2009.

[39] J. Puchinger, P.J. Stuckey, M.G. Wallace, and S. Brand. Dantzig-Wolfe decomposition and branch-and-price solving in G12. *Constraints*, 2010. To appear.

[40] G. Gamrath and M.E. Lübbecke. Experiments with a generic Dantzig-Wolfe decomposition for integer programs. In P. Festa, editor, *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA)*, volume 6049 of *Lect. Notes Comput. Sci.*, pages 239–252, Berlin, 2010. Springer-Verlag.

[41] G. Desaulniers, J. Desrosiers, and M.M. Solomon. Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 309–324, Boston, 2001. Kluwer.

[42] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.*, 46(3):316–329, 1998.

[43] G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors. *Column Generation.* Springer-Verlag, Berlin, 2005.

[44] J. Desrosiers and M.E. Lübbecke. A primer in column generation. In Desaulniers et al. [43], pages 1–32.

[45] F. Vanderbeck. Implementing mixed integer column generation. In Desaulniers et al. [43], pages 331–358.