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

Column Generation

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

June 7, 2010; revised: July 1, 2010

Abstract

Column generation is an indispensable tool in computational optimization to solve a mathematical program by iteratively adding the variables of the model. Even though the method is simple in theory there are many algorithmic choices and we discuss the most common ones. Particular emphasis in put on the dual interpretation, relating column generation to Langrangian relaxation and cutting plane algorithms, which revealed several critical issues like the need for dual variable stabilization techniques. We conclude with some advise for computer implementations.

Key words: Linear programming; column generation; Dantzig-Wolfe decomposition; Lagrangian relaxation; dual variable stabilization; branch-and-price.

Column generation is a classical technique to solve a mathematical program by iteratively adding the variables of the model [1]. Typically, only a tiny fraction of the variables is needed to prove optimality which makes the technique interesting for problems with a huge number of variables. The method is often said in one sentence with Dantzig-Wolfe decomposition [2] (see also 1.1.2.3, Dantzig-Wolfe decomposition), as it is particularly effective when the matrix has a special structure like bordered block-diagonal or staircase forms.

We make one point clear right from the beginning. Even though the method was termed *generalized linear programming* in the early days, it never became competitive for solving linear programs, except for special cases [3]. In addition, tailored implementations were designed to exploit matrix structures, but they did not perform better than the simplex method [4]. In contrast, column generation is a real winner in the context of integer programming (see also 1.4.2.4, branch, price, and cut algorithms). This made the powerful method a *must-have* in the computational mixed integer programming "bag of tricks."

We assume the reader to be familiar with basic linear programming duality and the simplex method (see also 1.1.1, fundamental techniques).

1 Column Generation

We would like to solve a linear program, called the *master problem* (MP)

$$\begin{aligned} v(\text{MP}) \;:=\; \min \quad & \sum_{j\in J} c_j\lambda_j \\ \text{subject to} \quad & \sum_{j\in J} \mathbf{a}_j\lambda_j \;\geq\; \mathbf{b} \\ & \lambda_j \;\geq\; 0,\;\; j\in J \end{aligned} \tag{1}$$

with $|J| = n$ variables and m constraints. In many applications n is exponential in m and working with (1) explicitly is not an option because of its shear size. Instead, consider the *restricted master problem* (RMP) which contains only a subset $J' \subseteq J$ of variables. An optimal solution $\boldsymbol{\lambda}^*$ to the RMP needs not be optimal for the master problem, of course. Denote by $\boldsymbol{\pi}^*$ an optimal dual solution to the RMP. In the *pricing step* of the simplex method (see also 1.1.1.3) we look for a non-basic variable of negative reduced cost to enter the basis. To accomplish this in column generation, one solves the *pricing problem* (or *subproblem*) PP

$$v(\text{PP}) \;:=\; \min\{c_j - \boldsymbol{\pi}^*\mathbf{a}_j \mid j\in J\} \;\;. \tag{2}$$

When $v(\text{PP}) < 0$, the variable λ_j and its coefficient column $(c_j, \mathbf{a}_j)$ corresponding to a minimizer j are added to the RMP; this is solved to optimality to obtain optimal dual variable values, and the process iterates until no further improving variable is found. In this case, $\boldsymbol{\lambda}^*$ optimally solves the master problem (1) as well. In particular, column generation inherits finiteness and correctness from the simplex method, when cycling is taken care of.

It seems not clear why (2) should be of any help when $|J|$ is large. However, in almost every application, indices in J enumerate entities which can be well described as the feasible domain X of an optimization problem

$$\min_{\mathbf{x}\in X}\{c(\mathbf{x}) - \boldsymbol{\pi}^* a(\mathbf{x})\} \;\;, \tag{3}$$

where $c_j = c(\mathbf{x}_j)$ and $\mathbf{a}_j = a(\mathbf{x}_j)$, and $\mathbf{x}_j \in X$ corresponds one-to-one to $j \in J$. That is, instead of *explicitly* pricing all candidate variables, we solve a typically well-structured optimization problem, making the search for a variable of negative reduced cost *implicit*. Technically, our notation suggests that X be finite, but this needs not be the case.

Consider the one-dimensional cutting stock problem, the classical example in column generation introduced in [5]. We are given paper rolls of width W, and m demands b_i, $i = 1, \ldots, m$, for orders of width w_i. The goal is to minimize the number of rolls to be cut into orders, such that the demand is satisfied. A standard formulation is

$$\min\{\mathbf{1}\boldsymbol{\lambda} \mid A\boldsymbol{\lambda} \geq \mathbf{b},\ \boldsymbol{\lambda} \in \mathbb{Z}_+^{|J|}\}\ , \tag{4}$$

where A encodes the set of $|J|$ feasible cutting patterns, i.e., $a_{ij} \in \mathbb{Z}_+$ denotes how often order i is obtained when cutting a roll according to $j \in J$. From the definition of feasible patterns, the condition $\sum_{i=1}^{m} a_{ij} w_i \leq W$ must hold for every $j \in J$, and λ_j determines how often the cutting pattern $j \in J$ is used. The linear relaxation of (4) is then solved via column generation, where the pricing problem is a *knapsack problem*.

Dual Bounds

During column generation we have access to a dual bound on $v(\text{MP})$ so that we can terminate the algorithm when a desired solution quality is reached. Let $v(\text{RMP})$ denote the optimum of the current RMP. When we know that $\sum_{j \in J} \lambda_j \leq \kappa$ for an optimal solution of the MP, one cannot improve $v(\text{RMP})$ by more than κ times the smallest reduced cost $v(\text{PP})$, hence

$$v(\text{RMP}) + \kappa \cdot v(\text{PP}) \ \leq\ v(\text{MP})\ . \tag{5}$$

An important special case is $\kappa = 1$ when a *convexity constraint* is present, see Sect. 1.1. This bound is tight as $v(\text{PP}) = 0$ when column generation terminates. Note that $v(\text{PP})$ is not available when the pricing problem is solved heuristically. When the objective function is a sum of all variables, i.e., $\mathbf{c} \equiv \mathbf{1}$, we use $\kappa = v(\text{MP})$ and obtain $v(\text{RMP})/(1 - v(\text{PP})) \leq v(\text{MP})$. There are other proposals, e.g. [6], and also tailored bounds for special problems, e.g. [7]. In general, the dual bound is not monotone over the iterations (*yo-yo effect*).

1.1 Dantzig-Wolfe Decomposition

The classical scenery of column generation is set in the context of Dantzig-Wolfe decomposition [2] in which a special structure of the typically very sparse coefficient matrix is exploited (see also 1.1.2.3, Dantzig-Wolfe decomposition). Consider a linear program, called the *original formulation* in this context

$$\begin{array}{rlcl} \min & \mathbf{c}\mathbf{x} & & \\ \text{subject to} & A\mathbf{x} & \geq & \mathbf{b} \\ & D\mathbf{x} & \geq & \mathbf{d} \\ & \mathbf{x} & \geq & \mathbf{0}\ . \end{array} \tag{6}$$

Let $X = \{\mathbf{x} \in \mathbb{Q}_+^n \mid D\mathbf{x} \geq \mathbf{d}\}$. By the representation theorems for convex polyhedra by Minkowski and Weyl [8] we can write each $\mathbf{x} \in X$ as finite convex combination of extreme

points $\{\mathbf{x}_p\}_{p\in P}$ plus finite non-negative combination of extreme rays $\{\mathbf{x}_r\}_{r\in R}$ of X, i.e.,

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

Substituting for $\mathbf{x}$ in (6), thereby eliminating constraints $D\mathbf{x} \geq \mathbf{d}$, and letting $c_j = \mathbf{c}\mathbf{x}_j$ and $\mathbf{a}_j = A\mathbf{x}_j$, $j \in P \cup R$, we obtain an equivalent *extended formulation*

$$\begin{array}{rrcl} \min & \displaystyle\sum_{p\in P} c_p\lambda_p + \sum_{r\in R} c_r\lambda_r & & \\ \text{subject to} & \displaystyle\sum_{p\in P} \mathbf{a}_p\lambda_p + \sum_{r\in R} \mathbf{a}_r\lambda_r & \geq & \mathbf{b} \\ & \displaystyle\sum_{p\in P} \lambda_p & = & 1 \\ & \boldsymbol{\lambda} & \geq & \mathbf{0} \ , \end{array} \tag{8}$$

which is solved by column generation. Let $\boldsymbol{\pi}^*, \pi_0^*$ denote a dual optimal solution to the RMP obtained from (8), where variable π_0 corresponds to the *convexity constraint* $\sum_{p\in P} \lambda_p = 1$. The subproblem (3) is to check whether $\min_{j\in P\cup R}\{c_j - \boldsymbol{\pi}^*\mathbf{a}_j - \pi_0^*\} < 0$. By our previous linear transformation this results in solving the linear program

$$\min\{(\mathbf{c} - \boldsymbol{\pi}^*A)\mathbf{x} - \pi_0^* \mid D\mathbf{x} \geq \mathbf{d}, \mathbf{x} \geq \mathbf{0}\} \ . \tag{9}$$

When the minimum is negative and finite, an optimal solution to (9) is an extreme point $\mathbf{x}_p$ of X, and we add a variable with coefficient column $[\mathbf{c}\mathbf{x}_p, (A\mathbf{x}_p), 1]$ to the RMP. When the minimum is minus infinity, we obtain an extreme ray $\mathbf{x}_r$ of X as a homogeneous solution to (9), and we add the column $[\mathbf{c}\mathbf{x}_r, (A\mathbf{x}_r), 0]$ to the RMP. It is particularly interesting that the master problem stays a linear program even when the subproblem is non-linear.

The usefulness of Dantzig-Wolfe decomposition becomes more apparent in the practically relevant case that D has block diagonal structure, i.e.,

$$D = \begin{pmatrix} D^1 & & & \\ & D^2 & & \\ & & \ddots & \\ & & & D^K \end{pmatrix} \qquad \mathbf{d} = \begin{pmatrix} \mathbf{d}^1 \\ \mathbf{d}^2 \\ \vdots \\ \mathbf{d}^K. \end{pmatrix} \tag{10}$$

Each $X^k = \{D^k\mathbf{x}^k \geq \mathbf{d}^k, \ \mathbf{x}^k \geq \mathbf{0}\}$, $k = 1, \ldots, K$, gives rise to a representation as in (7). The decomposition yields K subproblems, each with its own convexity constraint and associated dual variable π_0^k:

$$\min\{(\mathbf{c}^k - \boldsymbol{\pi}A^k)\mathbf{x}^k - \pi_0^k \mid \mathbf{x}^k \in X^k\}, \quad k = 1, \ldots, K, \tag{11}$$

where $\mathbf{c}^k$ and A^k correspond to variables $\mathbf{x}^k$. An optimal solution to the RMP is found when no minimum in (11) is negative. The dual bound (5) can be adapted.

There are other special matrix structures which can be exploited, e.g., the so-called staircase form of matrices which arises in multi-period or multi-stage planning problems, in particular in stochastic programming. In the easiest case, the matrix of the pricing problem has bordered block-diagonal structure again, and the Dantzig-Wolfe decomposition can be iteratively applied (also known as *nested* column generation).

1.2 Lagrangian Relaxation

In particular for a bordered block-diagonal matrix, Dantzig-Wolfe decomposition can be interpreted as keeping complicating constraints in the master problem while exploiting a particular structure in the subproblems. *Lagrangian relaxation* [9] proceeds the other way round: The complicating constraints $A\mathbf{x} \geq \mathbf{b}$ are relaxed and their violation is penalized in the objective function via multipliers $\boldsymbol{\pi} \geq \mathbf{0}$ (see also 1.1.2.4, Lagrangian optimization for LP). This results in the *Lagrangian subproblem*

$$L(\boldsymbol{\pi}) := \min_{\mathbf{x} \in X} \ \mathbf{c}\mathbf{x} - \boldsymbol{\pi}(A\mathbf{x} - \mathbf{b}) \ , \tag{12}$$

which gives a lower bound on the optimum in (6) for any $\boldsymbol{\pi} \geq \mathbf{0}$. We obtain the best such bound by solving the *Lagrangian dual problem*

$$\max_{\boldsymbol{\pi} \geq \mathbf{0}} \ L(\boldsymbol{\pi}) \ . \tag{13}$$

The Lagrangian function $L(\boldsymbol{\pi})$ is piecewise linear, concave, and sub-differentiable (but not differentiable). The most popular, since very easy to implement, choice to obtain optimal or near optimal multipliers are subgradient algorithms (see also 1.2.3.4, subgradient optimization). By duality, in the optimum $v(\text{RMP}) = \boldsymbol{\pi}\mathbf{b}$, and (12) can be written as

$$L(\boldsymbol{\pi}) = \boldsymbol{\pi}\mathbf{b} + \min_{\mathbf{x} \in X}(\mathbf{c} - \boldsymbol{\pi}A)\mathbf{x} \ \ = \ \ v(\text{RMP}) + v(\text{PP}) \ ,$$

that is, the dual bound in Dantzig-Wolfe decomposition and the Lagrangian bound coincide (see also 1.1.2.5, relationship among Benders, Dantzig-Wolfe, and Lagrangian optimization).

1.3 Row and Column Generation

Linear programs may not only have a large number of variables but also (too) many rows, e.g., when constraints are formulated on all subsets of a given ground set (like subtour elimination constraints for the TSP). In such cases one iteratively adds only those constraints which are violated by the current solution. The identification of a violated constraint (or the detection that none exists) is called *separation*. Embedded in a branch-and-bound algorithm, *cutting plane methods* became instrumental (and thus the standard) in solving mixed *integer* programs. Now, row and column generation obviously cannot be viewed independently. Even though some general ideas exist on how the pricing problem needs to be modified in order to cope with the dual variables from the additional rows, such approaches are still mainly problem specific (see also 1.4.2.4, branch, price, and cut algorithms).

1.4 Mixed Integer Programs

When solving a mixed integer program by branch-and-bound, the (linear) relaxation serves the purpose of providing a dual bound on the optimal objective function value. When the relaxation is solved by column generation in each node one speaks of branch-and-price (see also 1.4.2.4, branch, price, and cut algorithms). We cannot overstress the fact that the primary use of column generation is in this context, and it is becoming increasingly popular as column generation re-formulations often give much stronger bounds than the original LP relaxation. Many people actually refer to branch-and-price when they speak of column generation.

The Dantzig-Wolfe decomposition principle can be generalized to mixed integer programs in several ways. However, the basic column generation procedure to solve the linear relaxation remains the same. One drawback, the slow convergence, see Sect. 3, may even become smaller. When a dual bound LB is available, and the objective function coefficients are all integers, i.e., $c_j \in \mathbb{Z}$, $j \in J$, column generation can be stopped as soon as $\lceil LB \rceil = \lceil \bar{z} \rceil$. When one is aiming for quick integer solutions one may even terminate prematurely, and take a branching decision as soon as column generation starts tailing off. In this case the node's dual bound is not valid, so it is set to that of the father node, and this *early termination* even is exact in principle.

2 Algorithmic Issues

The dual of the RMP is the dual of the master problem with rows omitted, hence a relaxation. Therefore, the pricing problem is a *separation problem* for the dual; column generation is a cutting plane method to solve the Lagrangian dual (13). This explains why many researchers relate it to the Kelley [10] and Cheney-Goldstein [11] cutting plane methods known from maximizing a concave continuous function. The dual point of view, see [12] for a more detailed discussion, revealed central algorithmic issues in column generation. In particular, one should re-read this section after having read Sect. 3 on dual variable stabilization.

Note that there is a theoretical consequence from the equivalence of separation and optimization [13]. Even exponential size RMPs (linear programs) are solvable in polynomial time (in theory by the Ellipsoid method) when the pricing problem is.

2.1 Master Problem: Computing Primal and Dual Solutions

The purpose of the RMP is to provide dual variable values: To communicate with the pricing problem which primal variables are needed to come closer to dual feasibility, and thus primal optimality. Note that we never need a primal feasible solution before optimality is reached, not even to calculate dual bounds. That is, the RMP serves the same purpose as e.g., subgradient methods in Lagrangian relaxation, and this connection can be exploited.

2.1.1 Initialization, Infeasibility, and Farkas Pricing

Even when the master problem has a feasible solution, there are two important situations when the RMP is not feasible: In the beginning when no variables have been generated yet, and after branching when solving an integer program. In the traditional "phase I" approach [14] artificial variables with a "big M" penalty cost are introduced. A smaller M gives a tighter upper bound on the respective dual variables, and may reduce the *heading-in effect* [15] of initially producing irrelevant columns. Heuristic estimates of the optimal dual variable values can be used for this purpose [16]. Furthermore, one may warm-start from a previous similar run [17] or use a primal heuristic to produce an initial solution.

Column generation provides another way of turning an infeasible RMP feasible via the well-known fact that the dual of an infeasible linear program is unbounded (if not infeasible). This is formalized in Farkas' Lemma which states that either $A\mathbf{x} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$ is feasible or there is a vector $\boldsymbol{\pi}$ with $\boldsymbol{\pi}A \leq \mathbf{0}$ and $\boldsymbol{\pi}\mathbf{b} > 0$. Such a vector $\boldsymbol{\pi}$, which is interpreted as a ray in the

dual, *proves* the infeasibility of the first system as $\boldsymbol{\pi} A\mathbf{x} = \boldsymbol{\pi}\mathbf{b}$ cannot be fulfilled. The idea is now to add a variable to A with coefficient column $\mathbf{a}$ with $\boldsymbol{\pi}\mathbf{a} > 0$ which thus *destroys* this proof of infeasibility. Such a variable can be found (or concluded that none exists) by solving

$$\max_{x \in X}\{\boldsymbol{\pi}^* a(\mathbf{x})\} \;=\; \min_{x \in X}\{-\boldsymbol{\pi}^* a(\mathbf{x})\} \tag{14}$$

which is nothing else but the standard pricing problem (3) with cost coefficients $c(\mathbf{x}) = 0$. The dual ray $\boldsymbol{\pi}^*$ is typically provided by the LP solver in the case of an infeasible linear program. While this method appears to belong to the *folklore*, the name *Farkas pricing* has been introduced only recently in [18] within the `SCIP` framework (see Sect. 4).

2.1.2 Algorithms: Pivots, Subgradients, Bundles, and Volumes

As for any linear program, it is not *a priori* clear which method to solve the RMP will perform "best." This may depend on the problem and the available solvers. Traditionally, primal or dual simplex methods are used (see [19] for general comments on their suitability), but there are many alternatives. The *sifting method* [20], which is some sort of static column generation, can be a reasonable complement for large scale RMPs [17, 21]. Interior point methods like the *barrier method* can prove effective, although there is no warm start (yet). Also the *analytic center cutting plane method* [22] is advantageous as it produces interior point dual solutions. In addition to these general purpose methods, one may stronger exploit duality.

As stressed before, the RMP should furnish dual multipliers. After some initial iterations, a simplex method may produce relevant dual solutions which lead to progress, but then switching to a subgradient or more elaborate method to improve the dual solution may produce better dual bounds, and thus faster termination [23, 24, 25]. This can be cheaper and more stable, see Sect. 3, and may considerably reduce computation times. As the literature an Lagrangian relaxation is rich, there are many proposals for multiplier adjustment in subgradient methods which can be adapted to the column generation context. The RMP may itself be solved by subgradient algorithms by relaxing all its constraints in the objective function. This can be used as a primal heuristic as well, as proposed for set covering applications [26, 27, 28].

Subgradient algorithms suffer from a very restricted information; only the current subgradient is available. *Bundle methods* [29, 30] therefore work with a set of subgradients, the *bundle*, from which the methods borrow their name. It is true that a simplex method maintains a kind of bundle as well (the variables in the basis) but bundle methods may be more flexible. Bundle methods apply the proximal point idea of (quadratically) penalizing a deviation of the next iterate from the currently best one in terms of the dual bound. This makes them attractive in the context of Sect. 3 and explains their use in column generation [31]. It usually only takes a few iterations to produce an approximately optimal primal-dual pair.

The *volume algorithm* [32] is another extension of subgradient algorithms which also rapidly produces good approximations. It is named after a new way of looking at linear programming duality, using volumes below the active faces to compute the dual variable values and the direction of movement. The pricing subproblem is called with a dual solution "in a neighborhood" of an optimal dual solution. One can compute the probability that a particular column (which induces a face of the dual polyhedron) is generated. A modified subgradient method furnishes estimates of these probabilities, i.e., approximate primal solutions. Primal feasibility may be mildly violated.

When used in alternation with the simplex method, the volume algorithm produces dual solutions with a large number of non-zero variables [17] which may accelerate column generation. Promising computational experience is given [23, 33] for various combinatorial optimization problems. Advantages of the volume algorithm are a straight forward implementation with small memory requirements, numerical stability, and fast convergence.

2.1.3 Row Aggregation for Set Partitioning Problems

Primal degeneracy is an efficiency issue also in column generation, e.g., for large-scale set partitioning problems. Because of the degenerate pivots, dual variables yield less reliable information for the pricing problem. A possible remedy is to group *similar* constraints and aggregate them into one [34], thus working with an RMP with much less rows. The intuition is that in applications like vehicle routing and crew scheduling, some activity sequences are more likely to occur than others: In airline crew scheduling a pilot usually stays on the same aircraft for several flight legs. Since aircraft itineraries are known prior to solving the crew pairing problem, it is natural to "guess" some aggregation of the flights to cover.

The method is not particular to column generation but can be used in this context. Most importantly, an aggregated RMP gives aggregated dual variables which need to be disaggregated. This should (and can) be done carefully so that the disaggregated dual solution fulfills many of the dual constraints already. To ensure proper convergence and optimality, the aggregation is dynamically updated throughout the solution process. Tests conducted on the linear relaxation of the simultaneous vehicle and bus driver scheduling problem in urban mass transit show that this solution approach significantly reduces the size of the master problem, the degeneracy, and the solution times, especially for larger problems: For an instance with 1600 set partitioning constraints, the RMP solution time is reduced by a factor of eight. A partial pricing strategy, dubbed *multi-phase dynamic constraint aggregation* [35], gives further significant speedup.

2.2 The Pricing Problem

The pricing problem provides a column that prices out profitably or proves that none exists. *Any* variable with negative reduced cost will do, be it obtained by an exact, approximate or heuristic algorithm (the latter are first choice in terms of speed). One may even add positive reduced cost variables (possibly to a pool first). Sometimes relaxations of the pricing problem are solved, at the expense of a weaker dual bound, like for vehicle routing problems [36]. Highly complex pricing problems (like in staff and duty scheduling) may be better solved by constraint programming as this offers a strong expressiveness of the model [37].

2.2.1 Pricing Schemes and Pricing Rules

For the simplex method many proposals have been made as to which columns to consider and according to which rule to choose when selecting a variable to enter the basis. Schemes like *full*, *partial*, or *cyclic* pricing find their analogs in column generation pricing. When there are *many* subproblems it may be sensible to use partial/cyclic pricing in order to avoid the generation of many similar columns [38], but the number of iterations may increase. Dantzig's classical most-negative reduced cost pricing rule is not the only choice. The `Devex` rule [39]

(a practical variant of *steepest-edge* [40, 41]) is reported to perform particularly well for set partitioning RMPs [42]. The dual analog, the *deepest-cut* rule [43] tries to cut away as much of the dual space as possible. It can be implemented heuristically and is reported to offer some speedup [44].

While steepest-edge is inherently based on the simplex method, deepest-cut is more independent from a particular solution method. This leads to the *lambda pricing* rule [20]. Assume that $c_j \geq 0$, $j \in J$. Clearly, the reduced cost $c_j - \boldsymbol{\pi}^* \mathbf{a}_j$ are non-negative for all $j \in J$ iff

$$\min_{j \in J} \left\{ \frac{c_j}{\boldsymbol{\pi}^* \mathbf{a}_j} \mid \boldsymbol{\pi}^* \mathbf{a}_j > 0 \right\} \quad \geq 1 \ . \tag{15}$$

At first glance, this is just a reformulation. However, (15) takes advantage of structural properties of (particular) set partitioning problems: Picking columns with a small ratio accounts for smaller cost coefficients as well as for more non-zero entries in $\mathbf{a}_j$.

It is common in cutting plane algorithms to fill a *cut pool* first and select a *good* subset of cuts from it according to criteria like efficiency, orthogonality, sparsity, etc. [18]. Defining and applying such criteria to selecting good columns remains to be seen. Attempts to characterize dual facets [42] do not appear to have any practical impact so far. It would be interesting to see other pricing rules *particular to* column generation, e.g., with the aim of stabilization.

2.2.2 Pricing Problems when solving Integer Programs

When the subproblem's domain X in (3) is a mixed integer set, e.g., when a Dantzig-Wolfe type decomposition is applied to a mixed integer original problem (6), pricing problems become mixed integer programs themselves (see also 1.4.2.4, branch, price, and cut algorithms). It is well-known [9] that the dual bound from the RMP can be stronger than the LP relaxation only when the subproblem does not possess the *integrality property*. That is, the linear relaxation of the pricing problem should not give an integer solution. The tradeoff in choosing a decomposition is between a strong dual bound (by adding also complicating constraints to the subproblem) and the manageability of the subproblem (by avoiding this). Sometimes a combinatorial algorithm is available for the pricing problem and a faster alternative to an integer program; often this is a dynamic program (like for resource constrained shortest path problems in routing applications) which has the advantage of providing more than one solution to the pricing problem. The latter can be achieved with integer programs as well by using the *solution pool* that state-of-the-art solvers offer.

In particular with the help of pricing heuristics, one often generates columns which resemble a good *integer* solution rather than an optimal *fractional* one (which may be much harder to characterize). One should keep in mind that what helps the integer program need not help the linear program. Still, e.g., for set partitioning RMPs a reasonable strategy is to generate columns of a rich diversity [45] (*complementary columns*).

3 Stabilization of Dual Variables

Column generation is known to suffer from *tailing off* [46], i.e., there is only incremental progress per iteration the closer we get to the optimum, in particular for large and degenerate

problems. There are several partial explanations (see [47] for a summary), but a main reason lies in the *unstable* behavior of the dual variables. A dual solution may be far apart from the previous one (*bang-bang effect*, in [12] an example by A. Nemirovskii is cited which drastically shows this behavior). *Stabilization* of the dual variables tries to reduce this effect. The principles are well-established in the non-linear programming world; choosing *good separation points* in cutting plane algorithms is the analogous concept [48].

It should be noted that in the case that stabilization is successful, regardless of what method is employed, one typically observes a reduction in the number of column generation iterations. At the downside of it, the pricing problems become harder to solve on average. However, among more sophisticated implementation techniques, stabilization may promise the largest performance gains [15].

3.1 Interior Point Stabilization

Solving the RMP by a simplex method gives an extreme point of the optimal face of the dual polyhedron. When this face has a large dimension, e.g., when the primal is highly degenerate, there may be many extreme points, and the one obtained is essentially a "random choice" [20]. This extreme point is cut off in the next iteration, however, one would rather like to cut of the whole optimal face. In that sense, a simplex method may yield a "bad representative" of the optimal face. An immediate remedy to this can be to use an interior point method instead as one would cut off an interior point of the optimal face. Particular proposals have been using *analytic centers* [49], *volumetric centers*, and *central paths* [50], among others. Such concepts have been discussed for cutting plane algorithms as well, see [48].

A simplex method based approach to obtain a solution in the interior of the dual optimal face is taken in [51]. It works in two steps and exploits the extremity of basic solutions. First, the RMP is solved and the objective function value is fixed to the optimum via adding an additional constraint. Then, several random objective functions $\mathbf{c}$ are chosen (and also the opposite direction $-\mathbf{c}$), each of which produces an extreme point of the optimal face. The final dual solution is a convex combination of all extreme points obtained. This approach is computationally expensive but easy to implement.

3.2 Boxstep Method

Instead of producing rather arbitrary interior points, one may introduce a control of the dual solution's trajectory. By imposing lower and upper bounds, dual variables are constrained to lie "in a box around" the previous dual solution $\boldsymbol{\pi}^*$. The such restricted RMP is re-optimized. If the new dual optimum is attained on the boundary of the box, we have a direction towards which the box should be relocated. Otherwise, the optimum is attained in the box's interior, producing the sought global optimum. This is the principle of the `Boxtep` method [52, 53] and the basic idea of using a *stability center*, i.e., our current best guess of an optimal dual solution which is in some sense "more reliable" than other dual solutions. This is well-known, e.g., in trust-region methods, and it is the underlying mechanism of all what follows.

3.3 Polyhedral Penalty Terms

A hard-coded box is not very flexible. Instead, *stabilized column generation* [54] automates the re-centering of the box to the current dual solution. Consider the following linear program

$$\begin{array}{rrcl} \min & \mathbf{c}\boldsymbol{\lambda} - \boldsymbol{\delta}_-\mathbf{y}_- + \boldsymbol{\delta}_+\mathbf{y}_+ & & \\ \text{subject to} & A\boldsymbol{\lambda} - \mathbf{y}_- + \mathbf{y}_+ & = & \mathbf{b} \\ & \mathbf{y}_- & \leq & \boldsymbol{\varepsilon}_- \\ & \mathbf{y}_+ & \leq & \boldsymbol{\varepsilon}_+ \\ & \boldsymbol{\lambda}, \mathbf{y}_-, \mathbf{y}_+ & \geq & \mathbf{0} \end{array} \tag{16}$$

and its dual

$$\begin{array}{rrcl} \max & \boldsymbol{\pi}\mathbf{b} - \boldsymbol{\varepsilon}_-\mathbf{w}_- - \boldsymbol{\varepsilon}_+\mathbf{w}_+ & & \\ \text{subject to} & \boldsymbol{\pi}A & \leq & \mathbf{c} \\ & -\boldsymbol{\pi} - \mathbf{w}_- & \leq & -\boldsymbol{\delta}_- \\ & \boldsymbol{\pi} - \mathbf{w}_+ & \leq & \boldsymbol{\delta}_+ \\ & \mathbf{w}_-, \mathbf{w}_+ & \geq & \mathbf{0}\ . \end{array} \tag{17}$$

Surplus and slack variables $\mathbf{y}_-$ and $\mathbf{y}_+$, respectively, perturb $\mathbf{b}$ by $\boldsymbol{\varepsilon} \in [-\boldsymbol{\varepsilon}_-, \boldsymbol{\varepsilon}_+]$, which helps to reduce degeneracy. The interpretation of (17) is more interesting. The dual variables $\boldsymbol{\pi}$ are restricted to the interval $[\boldsymbol{\delta}_- - \mathbf{w}_-, \boldsymbol{\delta}_+ + \mathbf{w}_+]$, that is, deviation of $\boldsymbol{\pi}$ from the soft interval $[\boldsymbol{\delta}_-, \boldsymbol{\delta}_+]$ is allowed but penalized by an amount of $\boldsymbol{\varepsilon}_-, \boldsymbol{\varepsilon}_+$ per unit, respectively. From (16) we obtain an optimal solution to the unperturbed problem $\min\{\mathbf{c}\boldsymbol{\lambda} \mid A\boldsymbol{\lambda} = \mathbf{b}, \boldsymbol{\lambda} \geq \mathbf{0}\}$ when $\boldsymbol{\varepsilon}_- = \boldsymbol{\varepsilon}_+ = \mathbf{0}$ or $\boldsymbol{\delta}_- < \hat{\boldsymbol{\pi}} < \boldsymbol{\delta}_+$, where $\hat{\boldsymbol{\pi}}$ is an optimal solution to (17). Therefore the stopping criteria of a column generation algorithm become $v(PP) = 0$ and $\mathbf{y}_- = \mathbf{y}_+ = \mathbf{0}$.

This approach may need some parameter tuning, but it offers considerably speedup for some problems [54]. The change to the RMP requires adding upper bounded artificial variables only, which does not increase the size of the basis. It can be easily generalized to piecewise linear penalty functions with more pieces, where five pieces appear to give a good compromise [55], with a stronger penalty further away from the stability center. Note that (16) is a relaxation of the unperturbed RMP, and it may be faster computed.

3.4 Bundle Methods: Quadratic Penalty Term

The aim of the penalty terms is to encourage a dual solution to stay close to the stability center; so the penalty is larger the further away we go. Pictorially, a quadratic penalty function can achieve this goal better than a piecewise linear penalty, and *bundle methods* do precisely this: penalizing the Euclidean distance to the stability center. There is an extensive comparison between bundle methods and "classical" stabilization techniques in [12], and the current conclusion is that there is no clear winner. The situation may change in favor of bundle methods when future developments bring improvements e.g., in quadratic programming.

3.5 Convex Combinations with Previous Dual Solutions

A different approach to avoid (too) large steps in the dual space does not need any modification to the RMP at all, but convex combines the current dual solution $\boldsymbol{\pi}^*$ with a previous one $\hat{\boldsymbol{\pi}}$, i.e., the pricing problem is called with $\alpha\hat{\boldsymbol{\pi}} + (1-\alpha)\boldsymbol{\pi}^*$ for $0 \leq \alpha \leq 1$. When a column is

found it is added to the RMP only when it has negative reduced cost with respect to $\boldsymbol{\pi}^*$. The dual bound is updated whenever $L(\alpha\hat{\boldsymbol{\pi}} + (1-\alpha)\boldsymbol{\pi}^*) > L(\hat{\boldsymbol{\pi}})$. An interesting property is that even in the case that no column was added to the RMP (a *misprice*) it holds that the dual bound improves to at least $L(\hat{\boldsymbol{\pi}}) + \alpha(v(\text{RMP}) - L(\hat{\boldsymbol{\pi}}))$ [56]. As a consequence the duality gap $v(\text{RMP}) - L(\hat{\boldsymbol{\pi}})$ is reduced at least by a factor $(1-\alpha)^{-1}$, i.e., the method not only converges but also with a proven rate. Only a single parameter has to be calibrated, however, because of this static choice of α, the stability center moves with less flexibility than in the previous proposals.

The convex combination with a dual solution which produced the current best dual bound is a re-discovery of the *weighted Dantzig-Wolfe decomposition* method [57], in which α is updated in each iteration. The stability center $\hat{\boldsymbol{\pi}}$ becomes more reliable (larger α) the more often it leads to an improvement of the dual bound.

3.6 Valid Inequalities in the Dual Space

A complementary stabilization technique is to add valid inequalities to the dual. A simple proposal is the relaxation of RMP equalities to inequalities (when possible) which imposes sign constraints to the dual variables [5]. The concept of *dual-optimal inequalities* [58, 59] is more refined. One adds contraints $\boldsymbol{\pi}E \leq \mathbf{e}$ which are valid for the optimal face of the dual polyhedron. The consequence in the primal is that additional variables are introduced, and the RMP becomes $\min\{\mathbf{c}\boldsymbol{\lambda}+\mathbf{e}\mathbf{y} \mid A\boldsymbol{\lambda}+E\mathbf{y} \geq \mathbf{b},\, \boldsymbol{\lambda}, \mathbf{y} \geq \mathbf{0}\}$. *Deep* dual-optimal inequalities [59] may even cut away dual optimal solutions except at least one.

As an example consider the one-dimensional cutting stock problem (4). It can be easily shown that if the orders are ranked such that $w_1 < w_2 < \cdots < w_m$ then the dual variables satisfy the *ranking constraints* $\pi_1 \leq \pi_2 \leq \cdots \leq \pi_m$. These $m-1$ dual constraints can be generalized to larger sets [58, 59]. Let $S_i = \{s \mid w_s < w_i\}$. Then

$$\sum_{s\in S} w_s \leq w_i \;\; \Rightarrow \;\; \sum_{s\in S} \pi_s \leq \pi_i, \;\; S \subset S_i \;\; , \tag{18}$$

which significantly reduces the number of iterations on difficult instances [59].

As dual inequalities relax the primal RMP one has to ensure primal feasibility of the final $\boldsymbol{\lambda}^*$, which can be done by slightly perturbing the RMP [59]. The usefulness of adding valid dual inequalities has been demonstrated by constraining the dual variables to a small interval around their optimal values [55, 59] (or a heuristic good guess). Such *perfect dual information* is available e.g., for the cutting stock triplet-problems, where each roll is cut into exactly three orders without any waste, where $\pi_i = w_i/W$, $i = 1, \ldots, m$ is dual optimal. It is further known that restricting the dual space can reduce primal degeneracy [58].

4 Acceleration Techniques and Implementation Issues

Column generation is easily understood in theory but an implementation may suddenly reveal that there are many small pieces which need to fit together. In the sequel, we hint at some.

4.1 Libraries, Frameworks, Modeling Languages

Most people who implement a column generation code will at least rely on some package which provides an efficient simplex algorithm. There are plenty available, both commercial and open-source, like `CLP` [60], `GLPK` [61], and `SOPLEX` [62]. As noted above, there are alternatives (at least complements) to the simplex method, like the bundle method [63]. When we only do column generation, the main loop is quickly written. The major implementation effort then probably remains for the pricing problem. The situation is a little different when doing branch-and-price, but there are several frameworks which support its implementation (and thus in particular column generation) like `ABACUS` [64], `BCP` [65], `SCIP` [18], and `SYMPHONY` [66].

Frameworks have the advantage that they may automatically take care of features like using a *column pool* which contains variables from previous pricing rounds, or *lazy constraints* which are separated only when needed. This can be useful for constraints which are unlikely to be tight at optimality [67]. A main benefit from a framework is that it manages the branch-and-price tree, and that standard branching schemes etc. are available.

It is a little less known that column generation can be implemented also within several modeling languages like `GAMS` [68] or `OPL` [69]; but a true branch-and-price is usually not supported. Since the user does not have access to all the internals, this option is probably not quite suited for exactly solving very difficult problems, but it can be useful for practitioners working with the modeling language anyway.

4.2 Suboptimal Solutions

Column generation and branch-and-price are exact methods, i.e., in theory we obtain an optimal solution. The crux is that in practice, this may happen after a too long computation time, and one may wish to resort to a suboptimal solution. Fortunately, the dual bound gives a guarantee on its quality at any time. Heuristics should be used to construct or improve primal and dual solutions as often as it seems useful. This point cannot be overestimated.

Numerical computations on a computer are in limited precision and there are several tolerances to be thought of: What is negative reduced cost? When comparing against 0.0, one easily ends up in an infinite loop because of numerical inaccuracies. When does the primal bound match the dual bound closely enough? When an explicit perturbation of the right hand sides is used, of what magnitude will it be? Typically, for each of these tolerances one chooses some small value in the order of 10^{-3} to 10^{-6}. One can access the topic a bit more rigorously using the notion of ε-optimality [12]. An alternative is to resort to exact (rational) arithmetic; but due to performance reasons this is only advisable for mission critical linear and integer programs.

Practitioners interested in primal solutions (found quickly) may choose some sort of price-and-branch, i.e., pre-generate a reasonable set of variables in several rounds, and then solve the resulting program with standard branch-and-bound.

4.3 Some Simple Ideas which often Work

Again: Think of heuristics everywhere. Preprocess your problem well, in particular when solving integer programs. For many problems on networks, the graph can be significantly

reduced. Use a profiler to identify which part of the algorithm is the bottleneck. Typically, this will be the pricing problem but sometimes re-optimizing the RMP can be extremely time-consuming as well. Try solving the RMP only approximately and improve the dual solution with some iterations with a method from the Lagrangian world. Try dual variable stabilization, cf. Sect. 3. Try to avoid solving the pricing problem to optimality too often. Again, use heuristics first, maybe even a cascade of heuristics of increasing complexity. Relaxations serve the same purpose. Experiment with different parameters, in particular how many columns you add to the RMP in each iteration; too few do not yield enough progress, too many slow down computations (a combinatorial algorithm, or the solution pool of your solver can return more than one column). For large-scale problems, it can pay to remove columns which were non-basic for too many iterations.

Many acceleration techniques are problem dependent, but can often be adapted. The survey [70] in the context of vehicle routing and crew scheduling is very helpful in this respect. Re-read about the algorithmic alternatives in Sect. 2, all of which can be (and have been) modified and combined (see also 1.1.3, non-simplex algorithms for LP). When everything fails, you need to research your problem (more thoroughly)! A proof that an optimal primal (or dual) solution you are looking for has a particular structure may restrict the search a lot.

5 Conclusions

Despite the obvious similarity to cutting plane techniques—both methods dynamically extend the model—column generation has significant differences. While cutting planes *can optionally be* added to the (already optimally solved) linear relaxation in order to strengthen it, one *has to* add negative reduced cost variables for otherwise one does not obtain a valid dual bound. This makes the competition a bit unfair but we believe that the future will lie in integrating the two methods into one anyway.

Even though column generation was incepted more than half a century ago, the last decade was the most active in research and implementation. The availability of powerful computers and electronic large-scale data of hard practical problems challenged the community. The influence of non-differential convex analysis, in particular the idea of dual variable stabilization, was beneficial for the field. Still, column generation and branch-and-price are available as generic implementations, and we are eager to see this change.

Until this happens, there are very elaborate suggestions for tailoring the method to particular problems, sometimes even particular problem instances. While this is questionable in terms of general purpose applicability, it is the driving force for pushing the border. Many interesting developments will certainly follow.

Column generation is clearly a success story in large-scale *integer* programming. The dual bound bound obtained from an extended reformulation is often stronger, the tailing off effect can be lessened, and the knowledge of the original formulation provides us with a guide for branching and cutting decisions in the search tree. Today we are in a position that branch-and-price codes solve many large-scale problems of "industrial difficulty," no standard commercial solver could cope with.

6 Further Reading

Previous general reviews on column generation include [71, 72, 73, 74]. This article is based on [47]. The literature on applications of branch-and-price and column generation grew so quickly in recent years (see the book [75]) that it is likely that someone already proposed at least a partial solution to the application you have in mind. Go on reading on branch-and-price in [76] and 1.4.2.4, branch, price, and cut algorithms.

References

[1] L.R. Ford and D.R. Fulkerson. A suggested computation for maximal multicommodity network flows. *Management Sci.*, 5:97–101, 1958.

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

[3] O. Ogtildeuz. Generalized column generation for linear programming. *Management Sci.*, 48(3):444–452, 2002.

[4] J.K. Ho and E. Loute. Computational experience with advanced implementation of decomposition algorithms for linear progamming. *Math. Programming*, 27:283–290, 1983.

[5] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting-stock problem. *Oper. Res.*, 9:849–859, 1961.

[6] A.A. Farley. A note on bounding a class of linear programming problems, including cutting stock problems. *Oper. Res.*, 38(5):922–923, 1990.

[7] J.M. Valério de Carvalho. A note on branch-and-price algorithms for the one-dimensional cutting stock problem. *Comput. Optim. Appl.*, 21(3):339–340, 2002.

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

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

[10] J.E. Kelley Jr. The cutting-plane method for solving convex programs. *J. Soc. Ind. Appl. Math.*, 8(4):703–712, 1961.

[11] E.W. Cheney and A.A. Goldstein. Newton's method for convex programming and Tchebycheff approximation. *Numer. Math.*, 1(1):253–268, 1959.

[12] O. Briant, C. Lemaréchal, Ph. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck. Comparison of bundle and classical column generation. *Math. Programming*, 113(2, Ser. A):299–344, 2008.

[13] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, Berlin, 1988.

[14] V. Chvátal. *Linear Programming*. W.H. Freeman and Company, New York, 1983.

[15] F. Vanderbeck. Implementing mixed integer column generation. In Desaulniers et al. [75], pages 331–358.

[16] Y. Agarwal, K. Mathur, and H.M. Salkin. A set-partitioning-based exact algorithm for the vehicle routing problem. *Networks*, 19:731–749, 1989.

[17] R. Anbil, J.J. Forrest, and W.R. Pulleyblank. Column generation and the airline crew pairing problem. In *Proceedings of the International Congress of Mathematicians Berlin*, volume Extra Volume ICM 1998 of *Doc. Math. J. DMV*, pages III 677–686, August 1998.

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

[19] L.S. Lasdon. *Optimization Theory for Large Systems*. Macmillan, London, 1970.

[20] R.E. Bixby, J.W. Gregory, I.J. Lustig, R.E. Marsten, and D.F. Shanno. Very large-scale linear programming: A case study in combining interior point and simplex methods. *Oper. Res.*, 40(5):885–897, 1992.

[21] H.D. Chu, E. Gelman, and E.L. Johnson. Solving large scale crew scheduling problems. *European J. Oper. Res.*, 97:260–268, 1997.

[22] J.-L. Goffin, A. Haurie, and J.-Ph. Vial. Decomposition and nondifferentiable optimization with the projective algorithm. *Management Sci.*, 38(2):284–302, 1992.

[23] F. Barahona and D. Jensen. Plant location with minimum inventory. *Math. Programming*, 83:101–111, 1998.

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

[25] P. Mahey. A subgradient algorithm for accelerating the Dantzig-Wolfe decomposition method. In *Proceedings of the X. Symposium on Operations Research, Part I: Sections 1–5*, volume 53 of *Methods Oper. Res.*, pages 697–707, Königstein/Ts., 1986. Athenäum/Hain/Scriptor/Hanstein.

[26] A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Oper. Res.*, 47:730–743, 1999.

[27] A. Caprara, M. Fischetti, and P. Toth. Algorithms for the set covering problem. *Ann. Oper. Res.*, 98:353–371, 2000.

[28] D. Wedelin. An algorithm for large scale 0-1 integer programming with application to airline crew scheduling. *Ann. Oper. Res.*, 57:283–301, 1995.

[29] C. Lemaréchal. An algorithm for minimizing convex functions. In J.L. Rosenfeld, editor, *Information Processing '74*, pages 552–556. North Holland, Amsterdam, 1974.

[30] K.C. Kiwiel. An aggregate subgradient method for nonsmooth convex minimization. *Math. Programming*, 27:320–341, 1983.

[31] K.C. Kiwiel and C. Lemaréchal. An inexact conic bundle variant suited to column generation. *Math. Programming*, 118(1):177–206, 2009.

[32] F. Barahona and R. Anbil. The volume algorithm: Producing primal solutions with a subgradient method. *Math. Programming*, 87(3):385–399, 2000.

[33] F. Barahona and R. Anbil. On some difficult linear programs coming from set partitioning. *Discrete Appl. Math.*, 118(1–2):3–11, 2002.

[34] I. Elhallaoui, D. Villeneuve, F. Soumis, and G. Desaulniers. Dynamic aggregation of set partitioning constraints in column generation. *Oper. Res.*, 53(4):632–645, 2005.

[35] I. Elhallaoui, A. Metrane, F. Soumis, and G. Desaulniers. Multi-phase dynamic constraint aggregation for set partitioning type problems. *Math. Programming*, 123(2):345–370, 2010.

[36] M. Desrochers, J. Desrosiers, and M.M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Oper. Res.*, 40(2):342–354, 1992.

[37] U. Junker, S.E. Karisch, N. Kohl, B. Vaaben, T. Fahle, and M. Sellmann. A framework for constraint programming based column generation. In *Principles and Practice of Constraint Programming*, volume 1713 of *Lect. Notes Comput. Sci.*, pages 261–275. Springer-Verlag, 1999.

[38] M. Gamache, F. Soumis, G. Marquis, and J. Desrosiers. A column generation apporach for large-scale aircrew rostering problems. *Oper. Res.*, 47(2):247–263, 1999.

[39] P.M.J. Harris. Pivot selection methods of the Devex LP code. *Math. Programming*, 5:1–28, 1973.

[40] J.J. Forrest and D. Goldfarb. Steepest-edge simplex algorithms for linear programming. *Math. Programming*, 57:341–374, 1992.

[41] D. Goldfarb and J.K. Reid. A practicable steepest-edge simplex algorithm. *Math. Programming*, 12:361–371, 1977.

[42] M. Sol. *Column Generation Techniques for Pickup and Delivery Problems*. PhD thesis, Eindhoven University of Technology, 1994.

[43] F. Vanderbeck. *Decomposition and Column Generation for Integer Programs*. PhD thesis, Université catholique de Louvain, 1994.

[44] N. Papadakos. Integrated airline scheduling. *Comput. Oper. Res.*, 36:176–195, 2009.

[45] A. Ghoniem and H.D. Sherali. Complementary column generation and bounding approaches for set partitioning formulations. *Optim. Letters*, 3(1):123–136, 2009.

[46] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem—Part II. *Oper. Res.*, 11:863–888, 1963.

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

[48] W. Ben-Ameur and J. Neto. Acceleration of cutting-plane and column generation algorithms: Applications to network design. *Networks*, 49(1):3–17, 2007.

[49] S. Elhedhli and J.-L. Goffin. The integration of an interior-point cutting-plane method within a branch-and-price algorithm. *Math. Programming*, 100(2):267–294, 2004.

[50] R. Kirkeby Martinson and J. Tind. An interior point method in Dantzig-Wolfe decomposition. *Comput. Oper. Res.*, 26(12):1195–1216, 1999.

[51] L.-M. Rousseau, M. Gendreau, and D. Feillet. Interior point stabilization for column generation. *Oper. Res. Lett.*, 35(5):660–668, 2007.

[52] R.E. Marsten. The use of the boxstep method in discrete optimization. *Math. Programming Stud.*, 3:127–144, 1975.

[53] R.E. Marsten, W.W. Hogan, and J.W. Blankenship. The BOXSTEP method for large-scale optimization. *Oper. Res.*, 23:389–405, 1975.

[54] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Math.*, 194:229–237, 1999.

[55] H.M.T. Ben Amor, J. Desrosiers, and A. Frangioni. On the choice of explicit stabilizing terms in column generation. *Discrete Appl. Math.*, 157(6):1167–1184, 2009.

[56] 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.

[57] P. Wentges. Weighted Dantzig-Wolfe decomposition of linear mixed-integer programming. *Int. Trans. Opl. Res.*, 4(2):151–162, 1997.

[58] J.M. Valério de Carvalho. Using extra dual cuts to accelerate column generation. *INFORMS J. Comput.*, 17(2):175–182, 2005.

[59] H. Ben Amor, J. Desrosiers, and J.M. Valério de Carvalho. Dual-optimal inequalities for stabilized column generation. *Oper. Res.*, 54(3):454–463, 2006.

[60] COIN-OR linear programming. `https://projects.coin-or.org/Clp`, 2010.

[61] GNU linear programming kit. `http://www.gnu.org/software/glpk`, 2008.

[62] Sequential object-oriented simplex. `http://soplex.zib.de`, 2010.

[63] C. Helmberg. ConicBundle library for convex optimization. `http://www-user.tu-chemnitz.de/~helmberg/ConicBundle`, 2009.

[64] 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.

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

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

[67] J.-F. Cordeau, G. Desaulniers, N. Lingaya, F. Soumis, and J. Desrosiers. Simultaneous locomotive and car assignment at VIA Rail Canada. *Transportation Res. B*, 35:767–787, 2001.

[68] General algebraic modeling system. `http://www.gams.com`, 2010.

[69] IBM ILOG CPLEX optimization studio. `http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio`, 2010.

[70] 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.

[71] 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.

[72] J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, pages 35–139. North-Holland, Amsterdam, 1995.

[73] F. Soumis. Decomposition and column generation. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 115–126. John Wiley & Sons, Chichester, 1997.

[74] W.E. Wilhelm. A technical review of column generation in integer programming. *Optimization and Engineering*, 2:159–200, 2001.

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

[76] J. Desrosiers and M.E. Lübbecke. A primer in column generation. In Desaulniers et al. [75], pages 1–32.