COLUMN GENERATION

MARCO E. LÜBBECKE
Fachbereich Mathematik, Technische Universität Darmstadt, Darmstadt, Germany

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 ***Dantzig–Wolfe Decomposition***), as it is particularly effective when the matrix has a special structure like bordered block-diagonal or staircase forms.

We wish to clarify one point right at the start. 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 ***Branch and Price***). This made the powerful method a *must-have* in the computational mixed integer programming "bag of tricks."

We assume that the reader is familiar with basic linear programming duality and the simplex method (see also the section titled "Fundamental Techniques" in this encyclopedia).

COLUMN GENERATION

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

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

with $|J| = n$ variables and m constraints. In many applications, n is exponential in m and working with Equation (1) explicitly is not an option because of its sheer 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 need not be optimal for the MP, of course. Denote an optimal dual solution to the RMP by $\boldsymbol{\pi}^*$. In the *pricing step* of the simplex method (see also ***Simplex Method and Complexity***), we look for a nonbasic variable of negative reduced cost to enter the basis. To accomplish this in column generation, one solves the *pricing problem* (or *subproblem*) PP: Q1

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

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 MP (1) as well. In particular, column generation inherits finiteness and correctness from the simplex method, when cycling is taken care of.

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

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

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

Wiley Encyclopedia of Operations Research and Management Science, edited by James J. Cochran
Copyright © 2010 John Wiley & Sons, Inc.

optimization problem, making the search for a variable of negative reduced cost *implicit*. Technically, our notation suggests that X be finite, but this need not be the case.

Consider the one-dimensional cutting-stock problem, the classical example in column generation introduced in Gilmore and Gomory [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

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

where A encodes the set of $|J|$ feasible cutting patterns, that is, $a_{ij} \in \mathbb{Z}_+$ 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 Equation (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, we 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 the section titled “Dantzig–Wolfe Decomposition.” 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, that is, $\mathbf{c} \equiv \mathbf{1}$, we use $\kappa = v(\text{MP})$ and obtain $v(\text{RMP})/(1 - v(\text{PP})) \leq v(\text{MP})$. There are other proposals, for example, such as that of Farley [6], and also tailored bounds for special problems, for example, such as that of Valério de Carvalho [7]. In general, the dual bound is not monotone over the iterations (*yo-yo effect*).

Dantzig–Wolfe Decomposition

The classical scenario 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 ***Dantzig–Wolfe Decomposition***). Consider a linear program, called the *original formulation* in this context:

$$\begin{aligned} \min\quad & \mathbf{cx} \\ \text{subject to}\quad & A\mathbf{x} \geq \mathbf{b} \\ & D\mathbf{x} \geq \mathbf{d} \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned} \tag{6}$$

Let $X = \left\{ \mathbf{x} \in \mathbb{Q}_+^n \mid D\mathbf{x} \geq \mathbf{d} \right\}$. Using the representation theorems for convex polyhedra by Minkowski and Weyl [8], we can write each $\mathbf{x} \in X$ as a finite convex combination of extreme points $\{\mathbf{x}_p\}_{p \in P}$ plus finite nonnegative combination of extreme rays $\{\mathbf{x}_r\}_{r \in R}$ of X, that is,

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

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

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

which is solved by column generation. Let $\boldsymbol{\pi}^*, \pi_0^*$ denote a dual-optimal solution to the RMP obtained from Equation (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

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

When the minimum is negative and finite, an optimal solution to Equation (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 Equation (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 MP stays a linear program even when the subproblem is nonlinear.

The usefulness of Dantzig–Wolfe decomposition becomes more apparent in the practically relevant case where D has a block-diagonal structure, that is,

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

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 Equation (7). The decomposition yields K subproblems, each with its own convexity constraint and associated dual variable π_0^k:

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

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 Equation (11) is negative. The dual bound (5) can be adapted.

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

Lagrangian Relaxation

For a bordered block-diagonal matrix, in particular, Dantzig–Wolfe decomposition can be interpreted as keeping complicating constraints in the MP 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 ***Lagrangian optimization for LP***). This results in the *Lagrangian subproblem*

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

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

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

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

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

that is, the dual bound in Dantzig–Wolfe decomposition and the Lagrangian bound coincide (see also ***Relationship Among Benders, Dantzig–Wolfe, and Lagrangian Optimization***).

Row and Column Generation

Linear programs may have not only a large number of variables but also (too) many rows, for example, 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 that 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 ***Branch and Price***).

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 is referring to branch-and-price (see also ***Branch and Price***). 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 reformulations often give much stronger bounds than the original LP relaxation. Many people actually refer to branch-and-price when they speak of column generation. Q2

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 the section titled “Stabilization of Dual Variables”), may even become less severe. When a dual bound LB is available, and the objective function coefficients are all integers, that is, $c_j \in \mathbb{Z}$, $j \in J$, 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 the process 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 farther node, and this *early termination* even is exact in principle. Q3

ALGORITHMIC ISSUES

The dual of the RMP is the dual of the MP with rows omitted, and 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 that are known for maximizing a concave continuous function. The dual point of view (see Briant *et al*. [12] for a more detailed discussion) revealed central algorithmic issues in column generation. In particular, one should re-read this section after having read the section titled “Stabilization of Dual Variables” 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 solvable.

Master Problem: Computing Primal and Dual Solutions

The purpose of the RMP is to provide dual variable values: To communicate to 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, for example, subgradient methods in Lagrangian relaxation, and this connection can be exploited.

Initialization, Infeasibility, and Farkas Pricing. Even when the MP 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 for 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 the condition $\boldsymbol{\pi}A\mathbf{x} = \boldsymbol{\pi}\mathbf{b}$ 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

$$\max_{x \in X} \left\{ \boldsymbol{\pi}^* a(\mathbf{x}) \right\} = \min_{x \in X} \left\{ -\boldsymbol{\pi}^* a(\mathbf{x}) \right\}, \quad (14)$$

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 Achterberg [18] within the SCIP framework (see section titled "Acceleration Techniques and Implementation Issues").

Algorithms: Pivots, Subgradients, Bundles, and Volumes. As for any linear program, it is not *a priori* clear which method of solving the RMP will perform "best." This may depend on the problem and the available solvers. Traditionally, primal or dual simplex methods are used (see Lasdon [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.

Q4

As we have 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 [23–25]. This can be cheaper and more stable (see the section titled "Stabilization of Dual Variables"), and may considerably reduce computation times. As the literature an Lagrangian relaxation is rich, there are many proposals for multiplier adjustment in subgradient methods that 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 [26–28].

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 the name is derived. 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 current best one in terms of the dual bound. This makes them attractive in the context of the section titled "Stabilization of Dual Variables" and explains their use in column generation [31]. It usually takes only a few iterations to produce an approximately optimal primal–dual pair.

The *volume algorithm* [32] is another extension of subgradient algorithms that also rapidly produces good approximations. It is so named because of 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, that is, approximate primal solutions. Primal feasibility may be mildly violated.

The volume algorithm, when used in alternation with the simplex method, produces dual solutions with a large number of nonzero variables [17], which may accelerate column generation. The computational experience has been promising [23,33] for various combinatorial optimization problems. Advantages of the volume algorithm are straight forward implementation with small memory requirements, numerical stability, and fast convergence. Q5

Row Aggregation for Set Partitioning Problems. Primal degeneracy is an efficiency issue also in column generation, for example, 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 that need to be disaggregated. This should (and can) be done carefully so that the disaggregated dual solution fulfills many of the dual constraints. 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 MP, 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 8. A partial pricing strategy, called *multiphase dynamic constraint aggregation* [35], gives further significant speedup.

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, such as 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].

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 $c_j - \boldsymbol{\pi}^*\mathbf{a}_j$ is nonnegative for all $j \in J$ iff

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

At first glance, this is just a reformulation. However, Equation (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 nonzero 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, and others [18]. It remains to be seen how such criteria can be defined and applied for selecting good columns. Attempts to characterize dual facets [42] do not appear to have had any practical impact so far. It would be interesting to see other pricing rules *particular to* column generation, for example, with the aim of stabilization.

Pricing Problems when Solving Integer Programs. When the subproblem's domain X in Equation (3) is a mixed integer set, for example, when a Dantzig–Wolfe type decomposition is applied to a mixed integer original problem (6), pricing problems become mixed integer programs themselves (see also ***Branch and Price***). 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 trade-off 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 as by using the *solution pool* that state-of-the-art solvers offer.

In particular, with the help of pricing heuristics, one often generates columns that 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, for example, for set partitioning RMPs a reasonable strategy is to generate columns of a rich diversity [45] (*complementary columns*).

STABILIZATION OF DUAL VARIABLES

Column generation is known to suffer from *tailing off* [46], that is, there is only incremental progress per iteration as we get closer to the optimum, in particular, for large and degenerate problems. There are several partial explanations (see Desrosiers and Lübbecke [47] for a summary), but the 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 Briant *et al*. [12] an example provided by 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 nonlinear 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 the method employed, one typically observes a reduction in the number of column generation iterations. The downside of it is that the pricing problems become harder to solve on an average. However, among more sophisticated implementation techniques, stabilization may provide the largest performance gains [15].

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, for example, 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 off the whole optimal face. In that sense, a simplex method may yield a "bad representative" of the optimal face. An immediate remedy to this may 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 [48].

A simplex-method-based approach to obtain a solution in the interior of the dual-optimal face is taken in Rousseau *et al*. [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 by 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.

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 RMP that is thus restricted is reoptimized. If the new dual optimum is attained on the boundary of the box, we have a direction toward 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*, that is, our current best guess of an optimal dual solution that is in some sense "more reliable" than the other dual solutions. This is well known, for example, in trust-region methods, and it is the underlying mechanism of in what follows.

Polyhedral Penalty Terms

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

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

and its dual:

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

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 Equation (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 Equation (16) we obtain an optimal solution to the unperturbed problem $\min\{\mathbf{c}\boldsymbol{\lambda} \mid A\boldsymbol{\lambda} = \mathbf{b}, \boldsymbol{\lambda} \ge \mathbf{0}\}$ when $\boldsymbol{\varepsilon}_- = \boldsymbol{\varepsilon}_+ = \mathbf{0}$ or $\boldsymbol{\delta}_- < \hat{\boldsymbol{\pi}} < \boldsymbol{\delta}_+$, where $\hat{\boldsymbol{\pi}}$ is an optimal solution to Equation (17). Therefore, the stopping criteria of a column generation algorithm becomes $v(PP) = 0$ 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 in the RMP requires addition of 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 Equation (16) is a relaxation of the unperturbed RMP, and it may be computed faster. Q6

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 Briant *et al*. [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, for example, in quadratic programming.

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 $\hat{\boldsymbol{\pi}}$, that is, the pricing problem is called with $\alpha\hat{\boldsymbol{\pi}} + (1-\alpha)\boldsymbol{\pi}^*$ for $0 \le \alpha \le 1$. 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 when 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 $(1-\alpha)^{-1}$, that is, the method not only converges but does so at 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.

Q7

Q8

The convex combination with a dual solution that produced the current best dual bound is a rediscovery 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.

Q9

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 $\boldsymbol{\pi}E \le \mathbf{e}$, 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 $\min\{\mathbf{c}\boldsymbol{\lambda} + \mathbf{e}\mathbf{y} \mid A\boldsymbol{\lambda} + E\mathbf{y} \ge \mathbf{b}, \boldsymbol{\lambda}, \mathbf{y} \ge \mathbf{0}\}$. *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* $\pi_1 \le \pi_2 \le \cdots \le \pi_m$. These $m-1$ dual constraints can be generalized to larger sets [58,59]. Let $S_i = \{s \mid w_s < w_i\}$. Then

$$\sum_{s\in S} w_s \le w_i \;\Rightarrow\; \sum_{s\in S} \pi_s \le \pi_i, \quad S \subset S_i, \qquad (18)$$

which significantly reduces the number of iterations in 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, for example, for the cutting stock triplet-problems, where each roll is cut into exactly 3 orders without any wastage, 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].

ACCELERATION TECHNIQUES AND IMPLEMENTATION ISSUES

Column generation is easily understood in theory but an implementation may suddenly reveal that there are many small pieces that need to fit together. We mention some of these in the sequel.

Libraries, Frameworks, Modeling Languages

Most people who implement a column generation code will at least rely on some package that provides an efficient simplex algorithm. There are plenty of packages available, both commercial and open-source, such as 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 implement only 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 that support its implementation (and thus in particular column generation) such as 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 that are unlikely to be tight at optimality [67]. The main benefit from a framework is that it manages the branch-and-price tree, and that standard branching schemes, and others are available.

It is a little less known fact that column generation can also be implemented 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 suitable for exactly solving very difficult problems, but it can be useful for practitioners working with the modeling language.

Suboptimal Solutions

Column generation and branch-and-price are exact methods, that is, 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}–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, that is, pregenerate a reasonable set of variables in several rounds, and then solve the resulting program with standard branch-and-bound.

Some Simple Ideas that 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 reoptimizing 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; see the section titled "Stabilization of Dual Variables." 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 are added to the RMP in each iteration; too few do not yield enough progress, and 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 pays to remove columns that are nonbasic 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 the section titled "Algorithmic Issues," all of which can be (and have been) modified and combined (see also the section titled "Nonsimplex Algorithms for LP" in this encyclopedia). 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.

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 lies 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 nondifferential 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 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 where branch-and-price codes solve many large-scale problems of "industrial difficulty," that no standard commercial solver could cope with.

FURTHER READING

Previous general reviews on column generation include Barnhart *et al*. [71], Desrosiers *et al*. [72], Soumis [73], and Wilhelm [74]. This article is based on Desrosiers and Lübbecke [47]. The literature on applications of branch-and-price and column generation grew so quickly in recent years (see the book by Desaulniers *et al*. [75]) that it is likely that someone already proposed at least a partial solution to the application you have in mind. For further reading on branch-and-price refer to Desrosiers and Lübbecke [76] and the article titled ***Branch and Price***.

REFERENCES

1. Ford LR, Fulkerson DR. A suggested computation for maximal multicommodity network flows. Manage Sci 1958;5:97–101.
2. Dantzig GB, Wolfe P. Decomposition principle for linear programs. Oper Res 1960;8: 101–111.
3. Ogtildeuz O. Generalized column generation for linear programming. Manage Sci 2002; 48(3):444–452.
4. Ho JK, Loute E. Computational experience with advanced implementation of decomposition algorithms for linear progamming. Math Program 1983;27:283–290.
5. Gilmore PC, Gomory RE. A linear programming approach to the cutting-stock problem. Oper Res 1961;9:849–859.

6. Farley AA. A note on bounding a class of linear programming problems, including cutting stock problems. Oper Res 1990;38(5): 922–923.
7. Valério de Carvalho JM. A note on branch-and-price algorithms for the one-dimensional cutting stock problem. Comput Optim Appl 2002;21(3):339–340.
8. Schrijver A. Theory of linear and integer programming. Chichester: John Wiley & Sons; 1986.
9. Geoffrion AM. Lagrangean relaxation for integer programming. Math Program Stud 1974;2: 82–114.
10. Kelley JE Jr. The cutting-plane method for solving convex programs. J Soc Ind Appl Math 1961;8(4):703–712.
11. Cheney EW, Goldstein AA. Newton's method for convex programming and Tchebycheff approximation. Numer Math 1959;1(1): 253–268.
12. Briant O, Lemaréchal C, Meurdesoif Ph, *et al*. Comparison of bundle and classical column generation. Math Program Ser A 2008;113(2): 299–344.
13. Grötschel M, Lovász L, Schrijver A. Geometric algorithms and combinatorial optimization. Berlin: Springer; 1988.
14. Chvátal V. Linear programming. New York: W.H. Freeman and Company; 1983.
15. Vanderbeck F. Implementing mixed integer column generation. In: Desaulniers G, Desrosiers G, Solomon MM, editors. Column generation. Berlin: Springer; 2005. pp. 331–358.
16. Agarwal Y, Mathur K, Salkin HM. A set-partitioning-based exact algorithm for the vehicle routing problem. Networks 1989;19: 731–749.
17. Anbil R, Forrest JJ, Pulleyblank WR. Column generation and the airline crew pairing problem. Proceedings of the International Congress of Mathematicians; August 1998; Berlin. Documenta Mathematica Extra Volume ICM III 1998. pp. 677–686.
18. Achterberg T. SCIP: Solving constraint integer programs. Math Program Comput 2009;1 (1):1–41.
19. Lasdon LS. Optimization theory for large systems. London: Macmillan; 1970.
20. Bixby RE, Gregory JW, Lustig IJ, *et al*. Very large-scale linear programming: a case study in combining interior point and simplex methods. Oper Res 1992;40(5):885–897.
21. Chu HD, Gelman E, Johnson EL. Solving large scale crew scheduling problems. Eur J Oper Res 1997;97:260–268.
22. Goffin J-L, Haurie A, Vial J-Ph. Decomposition and nondifferentiable optimization with the projective algorithm. Manage Sci 1992;38(2):284–302.
23. Barahona F, Jensen D. Plant location with minimum inventory. Math Program 1998;83: 101–111.
24. Ceselli A, Righini G. A branch-and-price algorithm for the capacitated p-median problem. Networks 2005;45(3):125–142.
25. P Mahey. A subgradient algorithm for accelerating the Dantzig-Wolfe decomposition method. Proceedings of the X. Symposium on Operations Research, Part I: Sections 1–5, Volume 53 of Methods Opererations Research., Königstein/Ts: Athenäum/Hain/ Scriptor/Hanstein; 1986. pp. 697–707.
26. Caprara A, Fischetti M, Toth P. A heuristic method for the set covering problem. Oper Res 1999;47:730–743.
27. Caprara A, Fischetti M, Toth P. Algorithms for the set covering problem. Ann Oper Res 2000;98:353–371.
28. Wedelin D. An algorithm for large scale 0-1 integer programming with application to airline crew scheduling. Ann Oper Res 1995;57:283–301.
29. Lemaréchal C. An algorithm for minimizing convex functions. In: Rosenfeld JL, editor. Information Processing '74. Amsterdam: North Holland Publishing Co.; 1974. pp. 552–556.
30. Kiwiel KC. An aggregate subgradient method for nonsmooth convex minimization. Math Program 1983;27:320–341.
31. Kiwiel KC, Lemaréchal C. An inexact conic bundle variant suited to column generation. Math Program 2009;118(1):177–206.
32. Barahona F, Anbil R. The volume algorithm: producing primal solutions with a subgradient method. Math Program 2000;87(3):385–399.
33. Barahona F, Anbil R. On some difficult linear programs coming from set partitioning. Discrete Appl Math 2002;118(1–2):3–11.
34. Elhallaoui I, Villeneuve D, Soumis F, *et al*. Dynamic aggregation of set partitioning constraints in column generation. Oper Res 2005;53(4):632–645.
35. Elhallaoui I, Metrane A, Soumis F, *et al*. Multi-phase dynamic constraint aggregation for set partitioning type problems. Math Program 2010;123(2):345–370.

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

37. Junker U, Karisch SE, Kohl N, *et al*. A framework for constraint programming based column generation. In: Principles and practice of constraint programming. Volume 1713, Lecture Notes Computer Science. Springer; 1999. pp. 261–275.

Q10

38. Gamache M, Soumis F, Marquis G, *et al*. A column generation approach for large-scale aircrew rostering problems. Oper Res 1999;47(2):247–263.

39. Harris PMJ. Pivot selection methods of the Devex LP code. Math Program 1973;5:1–28.

40. Forrest JJ, Goldfarb D. Steepest-edge simplex algorithms for linear programming. Math Program 1992;57:341–374.

41. Goldfarb D, Reid JK. A practicable steepest-edge simplex algorithm. Math Program 1977; 12:361–371.

42. Sol M. Column generation techniques for pickup and delivery problems [PhD thesis]. Eindhoven University of Technology; 1994.

43. Vanderbeck F. Decomposition and column generation for integer programs [PhD thesis]. Université catholique de Louvain; 1994.

44. Papadakos N. Integrated airline scheduling. Comput Oper Res 2009;36:176–195.

45. Ghoniem A, Sherali HD. Complementary column generation and bounding approaches for set partitioning formulations. Optim Lett 2009;3(1):123–136.

46. Gilmore PC, Gomory RE. A linear programming approach to the cutting stock problem—Part II. Oper Res 1963;11:863–888.

47. Desrosiers J, Lübbecke ME. Selected topics in column generation. Oper Res 2005;53(6): 1007–1023.

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

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

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

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

52. Marsten RE. The use of the boxstep method in discrete optimization. Math Program Stud 1975;3:127–144.

53. Marsten RE, Hogan WW, Blankenship JW. The BOXSTEP method for large-scale optimization. Oper Res 1975;23:389–405.

54. du Merle O, Villeneuve D, Desrosiers J, *et al*. Stabilized column generation. Discrete Math 1999;194:229–237.

55. Ben Amor HMT, Desrosiers J, Frangioni A. On the choice of explicit stabilizing terms in column generation. Discrete Appl Math 2009;157(6):1167–1184.

56. Pessoa A, Uchoa E, Poggi de Aragão M, *et al*. Algorithms over arc-time indexed formulations for single and parallel machine scheduling problems. Report RPEP Volume 8 no. 8. Universidade Federal Fluminense; 2008.

57. Wentges P. Weighted Dantzig-Wolfe decomposition of linear mixed-integer programming. Int Trans Oper Res 1997;4(2):151–162.

58. Valério de Carvalho JM. Using extra dual cuts to accelerate column generation. INFORMS J Comput 2005;17(2):175–182.

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

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

61. GNU linear programming kit. 2008. Available at http://www.gnu.org/software/glpk.

62. Sequential object-oriented simplex. 2010. Available at http://soplex.zib.de.

63. Helmberg C. ConicBundle library for convex optimization. 2009. Available at http://www-user.tu-chemnitz.de/helmberg/ConicBundle.

64. Jünger M, Thienel S. The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization. Softw Pract Exper 2000;30(11):1325–1352.

65. Ralphs TK, Ladányi L. COIN/BCP User's Manual. 2001. Available at http://www.coin-or.org/Presentations/bcp-man.pdf.

66. Ralphs TK. Symphony version 5.1 user's manual. Corl Laboratory Technical Report. 2006.

67. Cordeau J-F, Desaulniers G, Lingaya N, *et al*. Simultaneous locomotive and car assignment at VIA Rail Canada. Transp Res B 2001;35: 767–787.

68. General algebraic modeling system. 2010. Available at http://www.gams.com.

69. IBM ILOG CPLEX optimization studio. 2010. Available at http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio.
70. Desaulniers G, Desrosiers J, Solomon MM. Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. In: Ribeiro CC, Hansen P, editors. Essays and surveys in metaheuristics. Boston (MA): Kluwer Academic publishers; 2001. pp. 309–324.
71. Barnhart C, Johnson EL, Nemhauser GL, *et al*. Branch-and-price: column generation for solving huge integer programs. Oper Res 1998;46(3):316–329.
72. Desrosiers J, Dumas Y, Solomon MM, *et al*. Time constrained routing and scheduling. In: Ball MO, Magnanti TL, Monma CL, *et al*., editors. Network routing. Volume 8, Handbooks in operations research and management science. Amsterdam: North-Holland Publishing Co.; 1995. pp. 35–139.
73. Soumis F. Decomposition and column generation. In: Dell'Amico M, Maffioli F, Martello S, editors. Annotated bibliographies in combinatorial optimization. Chichester: John Wiley & Sons; 1997. pp. 115–126.
74. Wilhelm WE. A technical review of column generation in integer programming. Optim Eng 2001;2:159–200.
75. Desaulniers G, Desrosiers J, Solomon MM, editors. Column generation. Berlin: Springer; 2005.
76. Desrosiers J, Lübbecke ME. A primer in column generation. In: Desaulniers G, Desrosiers J, Solomon MM, editors. Column generation. Berlin: Springer; 2005. pp. 1–32.

Queries in Article eorms0158

Q1. There are two notations used (π^* and λ^*) for the optimal solution to the RMP. Please confirm if this is okay.

Q2. May we delete the sentence 'Many people actually refer to branch-and-price when they speak of column generation' as this seems a repetition of the second sentence in this paragraph.

Q3. We have changed 'terminate prematurely' to 'terminate the process prematurely'. Please confirm if this is fine.

Q4. May we rephrase this sentence as 'In addition to these general-purpose methods, one may exploit strong duality theorem'?

Q5. We have modified this sentence. Please confirm if the edit is fine.

Q6. We have modified this sentence. Please confirm if the edit is correct.

Q7. May we add 'combinations' after 'convex'?

Q8. We have modified this and the next sentences. Please confirm if the edit is fine.

Q9. May we rephrase this sentence as 'When the stability center $\hat{\pi}$ becomes more reliable (larger α), it often leads to an improvement in the dual bound'?

Q10. Please provide the editors names and place of publication for reference 37.

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 is placed 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 advice for computer implementations.

Keywords: linear programming; column generation; Dantzig–Wolfe decomposition; Lagrangian relaxation; dual variable stabilization; branch-and-price