Les Cahiers du GERAD ISSN: 0711–2440

Stabilization in Column Generation

H. Ben Amor, J. Desrosiers
A. Frangioni

G–2004–62

August 2004

Stabilization in Column Generation

Hatem Ben Amor
GERAD and Département de mathématiques et de génie industriel
École Polytechnique de Montréal
C.P. 6079, Succ. Centre-ville
Montréal (Québec) Canada, H3C 3A7
hatem.ben.amor@gerad.ca

Jacques Desrosiers
GERAD and Méthodes quantitatives de gestion
HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada, H3T 2A7
jacques.desrosiers@gerad.ca

Antonio Frangioni
Dipartimento di Informatica
Università di Pisa
largo Bruno Pontecorvo 1
56127 Pisa, Italy
frangio@di.unipi.it

August, 2004

Les Cahiers du GERAD
G–2004–62

Abstract

Column Generation (CG) algorithms are instrumental in many areas of applied optimization, where Linear Programs with an enormous number of columns need to be solved. Although successfully used in many applications, the standard CG algorithm suffers from well-known "instability" issues that somewhat limit its efficiency and usability. Building on the theory developed for NonDifferentiable Optimization algorithm, we propose a large class of Stabilized Column Generation (SCG) algorithms which avoid the instability problems of the standard approach by using an explicit stabilizing term in the dual; this amounts at considering a (generalized) Augmented Lagrangian of the primal Master Problem. Since the theory allows a great degree of flexibility in the choice and in the management of the stabilizing term, we can use piecewise-linear functions that can be efficiently dealt with off-the-shelf LP technology, as well as being related in interesting ways with some previous attempt at stabilizing the CG algorithm. The effectiveness in practice of this approach is demonstrated by extensive computational experiments on large-scale Multi-Depot Vehicle Scheduling problems and simultaneous Vehicle and Crew Scheduling problems.

Key Words: Column Generation, NonDifferentiable Optimization, Proximal Point methods, Bundle methods, Multi-Depot Vehicle Scheduling problem, Vehicle and Crew Scheduling problem

Résumé

Les algorithmes de génération de colonnes sont de plus en plus utilisés pour la résolution de grands programmes d'optimisation linéaire comportant un nombre élevé de variables. Malgré son succès, la méthode standard de génération de colonnes est pourtant souvent instable, ce qui limite son efficacité et son usage. En se basant sur la théorie développée pour l'optimisation non différentiable, nous proposons une classe d'algorithmes de stabilisation de la méthode de génération de colonnes en ajoutant explicitement un terme de stabilisation à la formulation duale. Le problème primal correspondant fait alors appel à une généralisation du Lagrangien augmenté. Puisque la théorie permet beaucoup de flexibilité au niveau du choix et de la gestion de ce terme de stabilisation, nous proposons l'utilisation d'une fonction linéaire par morceaux qui tire avantage des avancées récentes des systèmes commerciaux d'optimisation de programmes linéaires. L'efficacité pratique de cette approche est illustrée par une expérimentation numérique sur des instances de grandes tailles de problèmes de tournées de véhicules multi-dépôts ainsi que de problèmes de confection simultanée d'itinéraires d'autobus et d'horaires de chauffeurs.

1 Introduction

Column Generation (GC) has proven to be very successful in solving very large scale optimization problems. It has been introduced independently by Gilmore and Gomory in 1960 [16] and Dantzig and Wolfe in 1960 [8]. The formers proposed to solve the linear relaxation of the *Cutting Stock Problem* by considering only a subset of columns representing feasible cutting patterns. Other columns (feasible cutting patterns) are generated, if needed, by solving a knapsack problem whose costs are the dual optimal multipliers of the restricted problem. The latters introduced the Dantzig-Wolfe (D-W) decomposition principle, that consists in reformulating a structured linear problem using the extreme points and rays of the polyhedron defined by a subset of constraints. These extreme points and rays form the columns of the constraint matrix of a very large scale problem. A restricted problem using a subset of extreme points and rays is solved, obtaining optimal dual multipliers that are used to generate negative reduced cost columns, if any. In both cases, optimality is reached when no such column exists. Hence, column generation consists in solving a restricted version of the primal problem defined with a small subset of columns and adding columns, if needed, until optimality is reached.

From a dual viewpoint, adding columns to the master problem is equivalent to adding rows (cuts) to the dual. The classical Cutting Plane (CP) algorithm is due to Kelley 1960 [25]. It solves convex problems by generating supporting hyperplanes of the objective function (*optimality cuts*) or the feasible set (*feasibility cuts*). At each iteration the dual of the restricted problem in D-W, is solved and cuts are added until dual feasibility (and optimality) are reached. Optimality cuts give the value of the objective function at the obtained (feasible) dual point; the corresponding column, which is an extreme point in the D-W context, has contribution 1 to a convexity constraint present in the primal problem. Feasibility cuts correspond to extreme rays; they are generated when the dual point obtained by the restricted problem is infeasible, i.e., the corresponding value of the objective function is $+\infty$.

Difficulties of the column generation/cutting planes algorithm appear when solving very large scale degenerate problems. It is well accepted that primal degeneracy is responsible of the long tail effect of column generation. Moreover, dual degeneracy and instability in the behaviour of dual variables are more frequent and harmful when problems get larger. "Oscillations" are observed, that is, it is possible to move from a good dual point to a much worse one; indeed, while the upper bound given by the primal solution in nonincreasing, the lower bound computed for each dual point is not nondecreasing. This affects the quality of columns to be generated in the following iteration. Other instability aspects are discussed in [21].

Stabilization approaches use two main ideas. A "good" dual point among those visited so far is taken to define the *stability center*; then, a stabilizing term that penalizes moves far from the center is added to the dual objective function. This aims at reducing the number

and the magnitude of oscillations of dual variables during CG/CP solution process. The stability center is changed if a "sufficiently better" dual point is found. These ideas have been around since the early seventies in the NonDifferentiable Optimization community (e.g. [26]), and have lead to the development of a variety of stabilized algorithms [24, 27, 29, 33], as well as to a deeper theoretical understanding of the underlying principles [10, 12, 21, 22, 23, 34]; we especially refer the interested reader to [28]. Also, related but different approaches for addressing the instability issues of the cutting plane algorithm have been proposed [17]. However, not much of this development seems to have been adopted by the practitioners in need of solving very large linear problems. In this article, building on [12] we propose a unifying framework for column generation stabilization approaches which include as special cases approaches based on the proximal point algorithm and bundle methods schemes, demonstrating its practical interest by means of computational experiments on some large problems.

The paper is organized as follows: in Section 2 the problem is stated, the standard Column Generation approach is reviewed, its relationships with the Cutting Plane algorithms are underlined and the "instability" problems of the approach are described and exemplified. In Section 3 we present a class of Stabilized Column Generation (SCG) approaches that avoid the instability problems of the standard approach by using an explicit stabilizing term in the dual, we discuss its primal counterparts and review its theoretical convergence properties for a class of stabilizing terms of particular interest in this setting. Then, in Section 4 we describe one particular stabilization term that meets the requirement, its relationships with previous attempts in the same direction from the literature and the implementation details of using this particular term within an actual CG code. Finally, in Sections 5 and 6 we present a set of computational experiments on, respectively, large-scale Multi-Depot Vehicle Scheduling (MDVS) problems and simultaneous Vehicle and Crew Scheduling (VCS) problems, aimed at proving the effectiveness of the proposed approach in practice, and in Section 7 we draw some conclusions and directions for future work.

Throughout the paper the following notation is used. The scalar product between two vectors v and w is denoted by vw. $\|v\|_p$ stands for the L_p norm of the vector v, and the ball around 0 of radius δ in the L_p norm will be denoted by $B_p(\delta)$. Given a set X, $I_X(x) = 0$ if $x \in X$ (and $+\infty$ otherwise) is its *indicator function*. Given a function f, $S_\delta(f) = \{x : f(x) \leq \delta\}$ is its *level set* corresponding to the f-value δ. Given a problem

$$(P)\, \inf_x [\sup] \{f(x) : x \in X\} ,$$

$v(P)$ denotes the optimal value of f over X; as usual, $X = \emptyset \Rightarrow v(P) = +\infty[-\infty]$.

2 Column Generation and Cutting Planes

We consider a linear program (P) and its dual (D)

$$(P) \quad \begin{array}{rll} \max & \sum_{a\in\mathcal{A}} c_a x_a & \\ & \sum_{a\in\mathcal{A}} a x_a & = b \\ & x_a \geq 0 & a \in \mathcal{A} \end{array} \qquad (D) \quad \begin{array}{rll} \min & \pi b & \\ & \pi a \geq c_a & a \in \mathcal{A} \end{array}$$

where $\mathcal{A}$ is the set of columns, each $a \in \mathcal{A}$ being a vector of $\mathbb{R}^m$, and $b \in \mathbb{R}^m$.

If the number of columns is so large that they are impossible or impractical to handle at once, or if the columns just cannot be determined a priori in practice, then (P) is typically solved by Column Generation (CG). At any iteration of the CG algorithm, only a subset $\mathcal{B} \subseteq \mathcal{A}$ of the columns is handled; this defines the primal master – or restricted – problem $(P_\mathcal{B})$ and its dual $(D_\mathcal{B})$, as follows:

$$(P_\mathcal{B}) \quad \begin{array}{rll} \max & \sum_{a\in\mathcal{B}} c_a x_a & \\ & \sum_{a\in\mathcal{B}} a x_a & = b \\ & x_a \geq 0 & a \in \mathcal{B} \end{array} \qquad (D_\mathcal{B}) \quad \begin{array}{rll} \min & \pi b & \\ & \pi a \geq c_a & a \in \mathcal{B} \end{array}$$

By solving the master problem we obtain a primal feasible solution x^* along with an infeasible vector of dual multipliers π^*. With these multipliers, we compute the maximum reduced cost over all columns in order to generate positive reduced cost columns that will be added to $(P_\mathcal{B})$. This actually amounts at solving the following optimization problem

$$(P_{\pi^*}) \qquad \max\{c_a - \pi^* a : a \in \mathcal{A}\}$$

that is usually called subproblem or separation problem. If no column with positive reduced cost exists, i.e., $v(P_{\pi^*}) \leq 0$, then π^* is actually feasible for (D), and therefore (x^*, π^*) is a pair of primal and dual optimal solutions to (P) and (D). Since the number of columns is very large, solving (P_{π^*}), i.e., determining whether or not there are dual constraints violated by π^*, must be performed without evaluating the reduced cost of each column explicitly. Thus, the CG approach is feasible when "efficient" algorithms for optimizing over $\mathcal{A}$ are available, that is, when a *mechanized pricing* procedure can be implemented to effectively determine the next (set of) column(s) to be brought in the "active set" $\mathcal{B}$. This is true for many known problems. For the Cutting Stock Problem (CSP), for instance, the feasible cutting patterns can be defined as the feasible solutions of either an integer Knapsack problem (KP), a Constrained Shortest Path problem (SPC) or a longest path problem over a disaggregated network. In Vehicle Routing problems or Multicommodity flow problems the separation problems are usually (constrained) flow or shortest path problems.

The solution of the subproblem provides either a certificate of optimality of the current pair (x^*, π^*) or a new column a that will be added to the master problem. It is worth

pointing out that solving the subproblem to optimality is only needed to prove optimality of the current primal and dual solutions; one can stop solving the subproblem whenever a positive reduced cost column is found, as inserting in $\mathcal{B}$ even this single column ensures that the new dual solution π^* will be different, and therefore the termination of the algorithm. However, in many applications the practical success of the method is tied to the ability of providing, at each iteration, a fair number of columns with positive reduced cost, as this typically decreases the number of iterations of the CG algorithm, i.e., the number of subproblems (which can be costly) to be solved. This is why often the subproblem is solved to optimality: many exact solution techniques (e.g., dynamic programming) naturally generate many columns at once. In the case where the subproblem solution produces only one column, it is often useful to generate any positive reduced cost column, especially when this is significantly less costly.

Adding a column to $(P_{\mathcal{B}})$ results in adding a "cut" to $(D_{\mathcal{B}})$, i.e., a supporting hyperplane of the dual feasible domain that is violated by the current dual point π^*. Hence, solving (P) by column generation is equivalent to solving (D) by Kelley's cutting plane algorithm [25].

In many relevant cases, the primal constraint matrix contains a convexity constraint $ex = 1$, where e is the vector of all ones of proper dimension. When this happens, it is convenient to single out the dual variable η corresponding to the convexity constraint, i.e., to consider (D) written as

$$\begin{array}{lll} (D) & \min & \eta + \pi b \\ & & \eta \geq c_a - \pi a \quad a \in \mathcal{A} \end{array}$$

This allows us to rewrite the (D) as the following NonDifferentiable Optimization problem

$$\min \pi b + \Theta(\pi)$$

where

$$\Theta(\pi) = v(P_\pi) = \max\{c_a - \pi a : a \in \mathcal{A}\}$$

is a convex piecewise affine function with a finite number of pieces. Thus, at each cutting plane iteration the value of $\Theta(\pi^*)$ is computed, and the generated columns are in fact supporting hyperplanes of the epigraph of Θ; they are called "optimality cuts". A more general case, very common in applications, is when the column set $\mathcal{A}$ is partitioned into k disjoint subsets $\mathcal{A}_1, \ldots, \mathcal{A}_k$, each of which has a separate convexity constraint, and therefore a separate corresponding dual variable η_h. Thus, in this case (D) can be written as the problem of minimizing the "decomposable" function

$$\min \pi b + \Theta(\pi) = \pi b + \sum_{h=1}^{k} \Theta^h(\pi) = \sum_{h=1}^{k} \max\{c_a - \pi a : a \in \mathcal{A}_h\} \ ,$$

where the computation of each Θ^h requires solving a different separation problem for each of the subsets $\mathcal{A}_h$. Each column a_h computed by the h-th separation problem is a supporting

hyperplane of the epigraph of Θ^h, and their sum is a supporting hyperplane of the epigraph of the aggregate function Θ.

If the convexity constraint concerns only a subset of the columns, the value of $\Theta(\pi^*)$ is obtained whenever π^* is dual feasible, which means that generated columns contribute to the convexity constraints (they are optimality cuts). Otherwise, π^* is infeasible for the dual ($\Theta(\pi^*) = +\infty$), and feasibility cuts are obtained. The usefulness of rewriting (D) as a convex optimization problem lies in the possibility of computing at each iteration (where optimality cuts are obtained) the function value $\pi^* b + \Theta(\pi^*)$, that has to be minimized. Thus, improvements (decreases) of this value can be taken as an indication that the obtained dual point π^* is nearer to an optimal solution. Furthermore, in this case each iteration (where optimality cuts are obtained) provides an upper bound on the optimal value of (P), that can be used in various ways to early terminate the optimization process.

When no convexity constraint is present in (P), it is not possible to define a function to be minimized, and therefore no upper bound on the optimal value of (P) is available; formally speaking, $\Theta(\pi^*)$ is $+\infty$ at each iteration except the last one. The quality of π^* cannot be stated precisely, something that is needed in the general algorithm we propose. This suggests the introduction of "artificial" convexity constraints, as discussed in details later on.

The CG/CP approach in the above form is simple to describe and, given the availability of efficient and well-engineered linear optimization software, straightforward to implement. However, constructing an efficient implementation for large-scale problems is far from trivial, as many complex issues have to be addressed (cf. e.g. [15]). Among the thorniest of these issues one must surely mention the *instability* of the approach, i.e., the fact that the sequence of dual solutions $\{\pi^*\}$ has no locality properties. Instability is the main cause of its slow convergence rate on many practical problems (the well-known "tailing-off" effect); in turn, this may cause the master problem to become exceedingly large and costly.

One of the most striking proofs of the lack of locality of dual iterates, and of their adverse effects on performances, can be obtained by using as starting point for the algorithm – the dual solution π^* where the first subproblem is solved – a good approximation to the optimal solution of (D), or even the optimal solution itself. It is well-known that this has little to no effect on the convergence of the algorithm, i.e., that the CG approach is almost completely incapable of exploiting the fact that it has already reached a good solution in order to speed up the subsequent calculations. By contrast, modifying the CG approach in such a way that it is forced to take into account this information can accelerate the column generation procedure in an impressive way. This is shown in Table 1, taken from [3], for a large scale instance of Multi-Depot Vehicle Scheduling problem (see Section 5 for more details). The first row corresponds to the standard CG approach, while the following ones correspond to the modified approach, described in the next section, where a (5-piecewise linear) penalty term is added to the dual objective function. No penalty is incurred in a full-dimensional

hyperbox that contains the known dual optimal solution, while dual solutions outside the box are penalized; each of the other rows correspond to a different width of the box. For each method, the total cpu time (cpu), the number of column generation iterations (CG iter.), the total number of columns generated by the subproblem (SP cols.), and the total number of simplex iterations performed to solve the master problem (MP itrs.) are reported; for the stabilized ones, the percentage w.r.t. those of the standard column generation is also shown. Even with a large box width (200.0) there is a great improvement in solution efficiency; the tighter the box, the more efficient is the stabilized algorithm. For the smallest box width (0.2), the total cpu time, the number of CG iterations, the number of columns generated, and the number of simplex pivots are reduced by more than 96%.

	cpu(s)		CG iter.		SP cols.		MP itrs.	
Method		%		%		%		%
CG	4178.4		509		37579		926161	
200.0	835.5	20.0	119	23.4	9368	24.9	279155	30.1
20.0	117.9	2.8	35	6.9	2789	7.4	40599	4.4
2.0	52.0	1.2	20	3.9	1430	3.8	8744	0.9
0.2	47.5	1.1	19	3.7	1333	3.5	8630	0.9

Table 1: Solving a large scale MDVS instance when a dual optimal solution is known

The above example clearly shows why the standard CG/CP algorithm tends to be inefficient: unlike, say, Newton's method, its convergence speed does not improve moving towards the optimal solution. However, properly "controlling" the dual variables may lead to substantial improvements in the performances. In the next paragraph we will discuss how a general Stabilized Column Generation approach can be constructed.

3 A Stabilized Column Generation Approach

As shown in the previous section, (D) can be recast as the minimization of a polyhedral function over a polyhedral domain, i.e.,

$$(D) \quad \min\{\phi(\pi) : \pi \in \Pi\} \ .$$

If one or more convexity constraints are available that "cover" all the variables, then $\Pi = \mathbb{R}^m$, i.e., ϕ is finite everywhere (possibly decomposable). Otherwise, Π is the polyhedron

$$\Pi = \{\pi : \pi a \geq c_a, a \in \mathcal{A}_0\}$$

where $\mathcal{A}_0$ is the set of columns that are not covered by any convexity constraint; if $\mathcal{A}_0 = \mathcal{A}$, $\phi(\pi)$ is just the linear function πb.

The cutting plane algorithm applied to the minimization of ϕ, as described in the previous section, generates either feasibility cuts for Π or optimality cuts for ϕ; in other words, the set $\mathcal{B}$ of currently available cuts is partitioned into the (possibly empty) subsets $\mathcal{B}_0$ and $\mathcal{B}_1$ which describes two "outer approximations" of ϕ and Π, respectively,

$$\phi_{\mathcal{B}}(\pi) = \pi b + \max\{c_a - \pi a : a \in \mathcal{B}_1\} \leq \phi(\pi) \ \ \forall \pi \ ,$$

$$\Pi_{\mathcal{B}} = \{\pi : \pi a \geq c_a, a \in \mathcal{B}_0\} \supseteq \Pi \ .$$

In case of multiple convexity constraints, actually, ϕ is decomposable and separate sets of columns $\mathcal{B}_h = \mathcal{B} \cap \mathcal{A}_h$ are defined for each component $h = 1, \ldots, k$; in order not to clutter the notation, however, we will mainly describe the simple case with $k = 1$.

With the above notation, the Dual Master Problem can be written as

$$(D_{\mathcal{B}}) \quad \min\{\phi_{\mathcal{B}}(\pi) : \pi \in \Pi_{\mathcal{B}}\} \ . \tag{1}$$

Since only a finite number of cuts is required for obtaining an "exact" representation of ϕ and Π, it is clear that the algorithm finitely terminates if all the generated information is kept in $\mathcal{B}$. However, this means that $(P_{\mathcal{B}})$ can become prohibitively large, especially if many columns are generated at each iteration. Furthermore, the CP algorithm suffers from the instability issue, as described in the previous section.

Since solving (P) by column generation amounts at solving one NonDifferentiable Optimization problem via a cutting plane approach, we can use in the CG context some ideas for the "stabilization" of the CP approach originally developed in the field of NonDifferentiable Optimization. In particular, here we will use the theory developed in [12] to introduce a large class of Stabilized Column Generation (SCG) algorithms. The development of [12] is by no means the only attempt at constructing a large class of Stabilized Cutting Plane (SCP) approaches (e.g. [4, 10, 22, 23, 34]), although it does have some technical advantages that allow to accommodate, among the others, the particular stabilizing function employed in Section 4.

The main idea of SCP approaches, as the name suggests, is to introduce some "stabilization device" in the dual problem to avoid the large fluctuations of the dual multipliers. In this particular incarnation this is done by choosing a *current center* $\bar{\pi}$ and a proper convex *stabilizing function* $\mathcal{D} : \mathbb{R}^m \to \mathbb{R} \cup \{+\infty\}$, and by solving the *stabilized master problem*

$$(D_{\mathcal{B},\bar{\pi},\mathcal{D}}) \quad \min\{\phi_{\mathcal{B}}(\pi) + \mathcal{D}(\pi - \bar{\pi}) : \pi \in \Pi_{\mathcal{B}}\} \tag{2}$$

instead of (1) at each iteration. The optimal solution π^* of (2) is used in the separation subproblem, i.e., to add either optimality cuts to $\phi_{\mathcal{B}}$ or feasibility cuts to $\Pi_{\mathcal{B}}$. The stabilizing function $\mathcal{D}$ is meant to avoid large fluctuations of π^* by penalizing points "too far" from the current point $\bar{\pi}$; at a first reading a norm-like function can be imagined there, with more details due soon. It should be noted that other, more or less closely related,

ways for stabilizing CP algorithms have been proposed [27, 17]; a through discussion of the relationships among them can be found in [21, 28].

Solving (2) is equivalent to solving its *Fenchel's dual*

$$(P_{\mathcal{B},\bar{\pi},\mathcal{D}}) \qquad \begin{array}{ll} \max & \displaystyle\sum_{a\in\mathcal{B}} c_a x_a + \bar{\pi}\left(b - \sum_{a\in\mathcal{B}} a x_a\right) - \mathcal{D}^*\left(\sum_{a\in\mathcal{B}} a x_a - b\right) \\ & \displaystyle\sum_{a\in\mathcal{B}_1} x_a = 1 \\ & x_a \geq 0 \quad , \quad a \in \mathcal{B} \end{array} \tag{3}$$

(cf. [21]); indeed, we can assume that an optimal solution of (3), x^*, is computed together with π^*. $\mathcal{D}^*$ is the *Fenchel's conjugate* of $\mathcal{D}$:

$$\mathcal{D}^*(w) = \sup_{\pi}\{\pi w - \mathcal{D}(\pi)\}$$

(cf. [21]), which characterizes the set of all vectors w that are support hyperplanes to the epigraph of $\mathcal{D}$ at some point. $\mathcal{D}^*$ is a closed convex function and enjoys several properties, for which the reader is referred e.g. to [21, 12]. It is easy to check from the definition the following relevant (for our application) examples of conjugate functions:

- the conjugate of $\mathcal{D} = \frac{1}{2t}\|\cdot\|_2^2$ is $\mathcal{D}^* = \frac{1}{2}t\|\cdot\|_2^2$;
- the conjugate of $\mathcal{D} = I_{B_\infty(t)}$ is $\mathcal{D}_t^* = t\|\cdot\|_1$;
- the conjugate of $\mathcal{D} = \frac{1}{t}\|\cdot\|_1$ is $\mathcal{D}^* = I_{B_\infty(1/t)}$.

So, stabilizing the dual master problem is equivalent to solving a *generalized augmented Lagrangian* of the primal master problem ($P_{\mathcal{B}}$), with a "first-order" Lagrangian term corresponding to the current point $\bar{\pi}$ and a "second-order" term corresponding to the stabilizing function $\mathcal{D}$. Thus, stabilized cutting plane algorithms applied to the solution of (D) can be seen as generalized augmented Lagrangian approaches applied to the solution of (P), where the augmented Lagrangian problem is approximately solved by inner linearization, as shown in more details in the following. If the current point $\bar{\pi}$ and the stabilizing function $\mathcal{D}$ are properly updated, the process can be shown to finitely solve (D) and (P).

More specifically, let us consider the general Stabilized Column Generation algorithm of Figure 1.

The algorithm generates at each iteration a *tentative point* π^* for the dual and a (possibly unfeasible) primal solution x^* by solving $(D_{\mathcal{B},\bar{\pi},\mathcal{D}})/(P_{\mathcal{B},\bar{\pi},\mathcal{D}})$. If x^* is feasible and has a cost equal to the lower bound $\phi(\bar{\pi})$, then it is clearly an optimal solution for (P), and $\bar{\pi}$ is an optimal solution for (D). Otherwise, new columns are generated using the tentative point π^*, i.e., $\phi(\pi^*)$ is computed. If $\phi(\pi^*)$ is "substantially lower" than $\phi(\bar{\pi})$, then it is worth to update the current point: this is called a *Serious Step (SS)* (note that a *SS* can be performed only if optimality cuts have been obtained). Otherwise the current point is

⟨ Initialize $\bar{\pi}$ and $\mathcal{D}$ ⟩
⟨ solve $P_{\bar{\pi}}$, initialize $\mathcal{B}$ with the resulting columns ⟩
repeat
  ⟨ solve $(D_{\mathcal{B},\bar{\pi},\mathcal{D}})/(P_{\mathcal{B},\bar{\pi},\mathcal{D}})$ for π^* and x^* ⟩
  if($\sum_{a\in\mathcal{B}} c_a x_a^* = \phi(\bar{\pi})$ **and** $\sum_{a\in\mathcal{B}} a x_a^* = b$)
  then stop
  else
    ⟨ solve P_{π^*}, i.e., compute $\phi(\pi^*)$ ⟩
    ⟨ possibly add some of the resulting columns to $\mathcal{B}$ ⟩
    ⟨ possibly remove columns from $\mathcal{B}$ ⟩
    if($\phi(\pi^*)$ is "substantially lower" than $\phi(\bar{\pi})$)
    then $\bar{\pi} = \pi^*$ /**Serious Step**/
    ⟨ possibly update $\mathcal{D}$ ⟩
while(not stop)

Figure 1: The general SCG/SCP algorithm

not changed, and we rely on the columns added to $\mathcal{B}$ for producing, at the next iteration, a better tentative point π^*: this is called a *Null Step (NS)*. In either case the stabilizing term can be changed, usually in different ways according to the outcome of the iteration. If a *SS* is performed, i.e., the current approximation $\phi_{\mathcal{B}}$ of ϕ was able to identify a point π^* with better function value than $\bar{\pi}$, then it may be worth to "trust" the model more and lessen the penalty for moving far from $\bar{\pi}$, e.g., by employing a stabilizing term $\mathcal{D}' < \mathcal{D}$; this corresponds to using a "steeper" penalty term $\mathcal{D}'^* > \mathcal{D}^*$ in the primal. If a *NS* is done instead, this "bad" outcome might be due to an excessive trust in the model, i.e., an insufficient stabilization, thereby suggesting to adopt a stabilizing term $\mathcal{D}' > \mathcal{D}$; this corresponds to using a "flatter" penalty term $\mathcal{D}'^* < \mathcal{D}^*$ in the primal.

When no convexity constraint is present in the primal formulation, which is the case of general column generation context, only "feasibility cuts" are generated and the dual objective value is $-\infty$ unless $\bar{\pi}$ is optimal to $(D_{\mathcal{B},\bar{\pi},\mathcal{D}})$. Hence, the center can only be updated when the *stabilized primal and dual problems*

$$(P_{\bar{\pi},\mathcal{D}}) \quad \begin{array}{ll} \max & \displaystyle\sum_{a\in\mathcal{A}} c_a x_a + \bar{\pi}\left(b - \sum_{a\in\mathcal{A}} a x_a\right) - \mathcal{D}^*\left(\sum_{a\in\mathcal{A}} a x_a - b\right) \\ & \displaystyle\sum_{a\in\mathcal{A}_1} x_a = 1 \\ & x_a \geq 0 \quad , \quad a \in \mathcal{A} \end{array} \tag{4}$$

$$(D_{\bar{\pi},\mathcal{D}}) \quad \min\{\phi(\pi) + \mathcal{D}(\pi - \bar{\pi}) : \pi \in \Pi\} \tag{5}$$

are solved to optimality. Thus, in this case a stabilized algorithm is a essentially a *proximal point* approach [32], even if a generalized one [10, 22, 34]. These approaches suffer from the inherent difficulty that, in order to solve $(P)/(D)$, a sequence of their stabilized counterparts $(P_{\bar{\pi},\mathcal{D}})$ / $(D_{\bar{\pi},\mathcal{D}})$ has to be solved, where each stabilized problem in the sequence is in principle as difficult to solve as the original one. Hence, most often these approaches are computationally viable only if coupled with rules that allow the early termination of the solution process to $(P_{\bar{\pi},\mathcal{D}})/(D_{\bar{\pi},\mathcal{D}})$; if the latter is solved by a cutting plane approach, as is clearly the case here, the result is a (generalized) *bundle* method [21, 12, 23, 26, 33].

In order to allow changes in the current point before the actual solution of the stabilized problem, it may be convenient to introduce "artificial" convexity constraints when possible. In other words, if an estimate $\bar{b}$ can be given such that the constraint

$$\sum_{a\in\mathcal{A}} x_a \leq \bar{b}$$

can be guaranteed to hold at some (even if not all) optimal solutions of (P), then it is worth to add this constraint to (P) and use the corresponding dual variable – that will then be constrained to be nonnegative – to define Θ. That is, (D) is equivalent to

$$\min \left\{ \pi b + \bar{b} \max\{c_a - \pi a : a \in \mathcal{A}\} \right\} \tag{6}$$

in all iterations, since the value of the maximum is nonnegative. This gives an upper bound on $v(P)$ at each iteration, known as "the Lagrangian bound". Of course, the same can be done to define "disaggregated" functions Θ^h if artificial convexity constraints can be defined on subsets of the variables (columns). We will see examples of application of this technique in Sections 5 and 6.

Obviously, the smaller the value of the r.h.s. $\bar{b}$ of the artificial constraint, the better the obtained upper bound on $v(P)$. In general, it is possible that different optimal solutions of (P) have different values of $\sum_{a\in\mathcal{A}} x_a$; then, taking the minimum (or any specific) value for $\bar{b}$ may target some particular optimal solution of (P). It is also possible to dynamically adjust the value of $\bar{b}$, within the solution algorithms, whenever e.g. the feasibility of a smaller value is proved.

The algorithm in Figure 1 can be shown [12] to finitely converge to a pair $(\bar{\pi}, x^*)$ of optimal solutions to (D) and (P), respectively, under a number of different hypotheses. Within the CG environment, the following conditions can be imposed:

i) $\mathcal{D}$ is a convex nonnegative function such that $\mathcal{D}(0) = 0$, its level sets $S_\delta(\mathcal{D})$ are *compact and full-dimensional* for all $\delta > 0$; it is interesting to remark that these requirements are *symmetric* in the primal, i.e., they hold for $\mathcal{D}^*$ *if and only if* they hold for $\mathcal{D}$ [12].

ii) $\mathcal{D}$ is *differentiable* in 0, or, equivalently, $\mathcal{D}^*$ is *strictly convex* in 0.

iii) $\mathcal{D}$ is "steep enough" to guarantee that $(D_{\mathcal{B},\bar{\pi},\mathcal{D}})$ always attains a finite optimal solution (and therefore so does $(P_{\mathcal{B},\bar{\pi},\mathcal{D}})$).

iv) Chosen some $m \in (0,1]$, $\phi(\pi^*)$ is declared to be "substantially lower" than $\phi(\bar{\pi})$ if

$$\phi(\pi^*) - \phi(\bar{\pi}) \leq m(v(D_{\mathcal{B},\bar{\pi},\mathcal{D}}) - \phi(\bar{\pi})) \; ; \tag{7}$$

alternatively, the stronger condition

$$\phi(\pi^*) - \phi(\bar{\pi}) \leq m(\phi_{\mathcal{B}}(\pi^*) - \phi(\bar{\pi})) \tag{8}$$

can be used (as $\phi_{\mathcal{B}}(\pi^*) \leq v(D_{\mathcal{B},\bar{\pi},\mathcal{D}}) \leq \phi(\bar{\pi})$ since $\mathcal{D}$ is non-negative, $\bar{\pi}$ is a feasible solution for $D_{\mathcal{B},\bar{\pi},\mathcal{D}}$ and $\phi_{\mathcal{B}}(\bar{\pi}) = \phi(\bar{\pi})$). Note that one of the conditions above must be satisfied when a *SS* is performed, but the converse is not required, i.e., there is the possibility not to perform a *SS* if the conditions are satisfied, only provided that this does not happen infinitely many times. This allows for alternative actions, e.g., decreasing $\mathcal{D}$, to be taken in response to a "good" step, which considerably adds to the flexibility of the algorithm.

v) During a sequence of *consecutive NS*, $\mathcal{D}$ can change only *finitely many times*.

vi) There exists a fixed function $\bar{\mathcal{D}}$ that satisfies the conditions of point i) such that $\mathcal{D} \leq \bar{\mathcal{D}}$ at all iterations.

vii) If any column is removed from $\mathcal{B}$ at some iteration, no other removal is permitted until $v(D_{\mathcal{B},\bar{\pi},\mathcal{D}})$ increases by a fixed amount $\varepsilon > 0$.

These conditions are not the most general ones, but they are sufficient in the framework of CG. For instance, if $\mathcal{D}$ is always chosen in a family of functions $\mathcal{D}_t$ indiced over one continuous parameter $t > 0$ (think of the examples above), then conditions over t that are substantially weaker than those implied by v) and vi) can be imposed. Furthermore, if $\mathcal{D}$ is strictly convex, or, equivalently $\mathcal{D}^*$ is differentiable, condition vii) can be substantially relaxed allowing to drop any column a such that $x^*_a = 0$, and even to drop any number of "active" columns as long as the "aggregated column" $\bar{a} = \sum_{a \in \mathcal{B}} a x^*_a$ is added to $\mathcal{B}$ as their "representative"; this is interesting since it allows to keep $|\mathcal{B}|$ bounded to any fixed number (down to 2), although it only guarantees asymptotic convergence, as opposed to finite convergence, of the process.

The above scheme can also be made slightly more general by allowing to call the separation subproblem at a different point than $\bar{\pi}$, and/or not to add the resulting columns to $\mathcal{B}$, at some iterations; it is only required that, during a sequence of *consecutive NS*, this is done only a finite number of times. This is useful for instance to accommodate a further search on the π space "around" $\bar{\pi}$, such as a linear or curved search [33]. In the CG context, this is especially interesting because the rule of thumb is that generating "many" columns, provided they are reasonably good ones, is crucial for the performances of the approach; thus, generating columns in some points "near" $\bar{\pi}$ – without having to solve a

costly master problem for each one – offers a way to increase the number of generated columns. The effectiveness of this idea has been demonstrated in [3], where 5 different directions d^i, $i = 1, \ldots, 5$, are defined and used as extra displacement for updating the stability center: d^1 is the gradient of the dual problem objective b, d^2 is the ascent direction computed at the previous SS, d^3 is a subgradient of the Lagrangian function at the newly computed dual point, and d^4 and d^5 are the projections of d^2 and d^3 on the nonnegative orthant. Table 2, taken from [3], shows that a beneficial effect of using these extra displacements can observed for larger size instances for some Multiple Depot Vehicle Scheduling problem instances (see Section 5.2 for more details). In the table, SCG stands for using the "classical" solution d of the stabilized column generation while the other columns stand for using the corresponding modifier; lr, mp, and sp give the cpu times in seconds needed to solve linear relaxations, master problems, and subproblems, respectively, while itr give the number of column generation iterations needed to solve linear relaxations to optimality.

Pb	$p1$						$p3$					
d	SCG	d^1	d^2	d^3	d^4	d^5	SCG	d^1	d^2	d^3	d^4	d^5
lr(s)	42.2	36.0	35.7	37.1	37.4	40.0	44.9	45.4	45.5	46.0	48.5	48.2
mp(s)	13.7	11.9	12.6	12.5	12.1	14.2	20.0	17.6	19.4	18.0	17.7	18.1
sp(s)	28.5	24.1	23.1	24.6	25.3	25.8	24.9	27.8	26.1	28.0	30.8	30.1
itr	52	47	47	50	49	49	53	56	53	56	50	57
Pb	$p6$						$p5$					
d	SCG	d^1	d^2	d^3	d^4	d^5	SCG	d^1	d^2	d^3	d^4	d^5
lr(s)	595.5	465.3	458.7	452.5	480.2	477.5	306.4	268.4	260.4	253.2	251.4	252.9
mp(s)	216.2	148.9	157.2	167.4	153.9	157.1	141.9	111.2	114.7	111.0	114.9	114.5
sp(s)	379.3	316.4	301.5	285.1	326.3	320.4	164.5	157.2	145.7	142.2	136.5	138.4
itr	196	158	157	147	156	152	80	73	71	69	69	70
Pb	$p7$						$p8$					
d	SCG	d^1	d^2	d^3	d^4	d^5	SCG	d^1	d^2	d^3	d^4	d^5
lr(s)	1223.5	1110.3	1164.1	1058.9	1139.2	1111.5	837.1	726.2	740.5	730.7	679.3	740.5
mp(s)	610.9	619.5	634.0	616.7	598.4	633.0	479.6	254.8	282.6	272.0	331.6	270.8
sp(s)	612.6	490.8	530.1	442.2	540.8	478.5	357.5	471.4	457.9	458.7	347.7	469.7
itr	178	195	171	143	165	146	145	191	192	190	133	189

Table 2: Modifying the "classical" direction d of SCG.

This is why as much flexibility as possible has to be left to the actual handling of the penalty term when developing the underlying convergence theory of the approach, as done in [3, 2, 12]. By contrast, most theoretical developments of SCP methods require the penalty function to depend on only one or few parameters. For more details about the analysis and the general conditions, the interested reader is referred to [12].

It may be worthwhile to remark that, by choosing $m = 1$ and the stronger variant (8) for the descent test, the above general scheme becomes the (generalized) *proximal point* approach. Of course, choosing $m < 1$ can be thought to be a better choice in general because it allows to move to a better current point possibly far before than $(P_{\bar{\pi},\mathcal{D}})$ has been solved to optimality, thereby avoiding a number *NS*. However, this may also result in an overall increase in the number of *SS* required to reach an optimal solution; the balance between these two effects has to be carefully taken into account.

Finally, we also mention that a closely related approach has been often used in the literature, where the current point $\bar{\pi}$ is set to 0 at the beginning and *never* changed during all the course of the algorithm; looking at (4), this is a penalty approach to (P) where the penalized problem is solved, either exactly or approximately, by Column Generation. In order for this approach to converge to an optimal solution, it is in general required that $\mathcal{D} \to 0$ (pointwise) as the algorithm proceeds, unless $\mathcal{D}$ is an exact penalty function, i.e., there exists a $\mathcal{D} > 0$ such that $(P_{0,\mathcal{D}})$ is equivalent to (P). Of course the two approaches can be mixed, and in fact having $\mathcal{D} \to 0$ is necessary to guarantee convergence for some classes of stabilizing functions, most notably those that do not satisfy condition ii) above. An algorithmic framework that generalizes that of Figure 1 and allows for such an integration is proposed and analyzed in [12].

A number of stabilized cutting plane approaches have been proposed in the literature and used to solve structured linear programs. Penalty bundle methods, using $\mathcal{D} = \frac{1}{2t}\|\cdot\|_2^2$, have been reported to be successful [7, 13], even in the context of column generation [6]. Other closely related approaches have been proposed based over exponential [18] or linear-quadratic [30] stabilizing terms. In all these approaches, however, the Stabilized Master Problem is a nonlinear program. Although efficient specialized codes for some nonlinear SMP have been developed [11], it is reasonable to assume that the widely available and extremely efficient standard *LP* solvers will be a competitive long-term alternative to custom-made nonlinear solvers, at least in some cases; the CG setting is likely to be one of them, since the typical number of columns in the SMP is quite large. Attempts at developing SCP/SCG approaches that can exploit the available *LP* technology had already been done, either from the theoretical [4, 24] or from the practical [9] viewpoint. However, a number of important details were not taken into account, and the proposed methods were not flexible enough to accommodate a number of small but important variations of the basic algorithmic scheme, some of which will be illustrated in the next section. A substantial step towards practical application of these ideas to the CG framework was done in [3, 2]; however, only "pure" proximal point approaches had been used there. The main motivation for the development of [12] was exactly to build a solid theoretical foundation over which a number of different SCP/SCG approaches, comprised some *LP*-based ones, could be easily constructed. In the next sections we will show in details one of these possible approaches that attains promising results in the CG framework.

4 Piecewise Linear Penalty Functions

Selecting an appropriate penalty function $\mathcal{D}^*$ is a fundamental step in the development of an efficient SCG approach. The use of piecewise linear penalty functions leads to solving linear programs at each iteration. It also allows for exact solution of the stabilized primal and dual problems if desired, and thereby the implementation of "pure" (generalized) proximal point approaches.

In the context of the general algorithm presented in this paper, primal convergence is guaranteed whenever the penalty terms are differentiable at 0 (condition ii); for a piecewise linear penalty function $\mathcal{D}$, this requires that the associated *trust region* (the set where $\mathcal{D}$ is zero) must be full dimensional and contain 0 in its interior. This is the case of the boxstep method [29] and of [9, 3, 2]. For the linear norm penalty approach [24], because the corresponding penalty functions $\mathcal{D}$ are not differentiable at 0, convergence towards a primal optimal solution is not guaranteed. However, one can recover a primal optimal solution in an efficient way by using the method presented in Section 1 (see also [3]).

There are, of course, many possible ways for defining a piecewise linear penalty function. We found a 5-pieces function similar to the one used in [3, 2] to offer a good compromise between simplicity of implementation and flexibility. Given nonnegative vectors of interval widths Γ^-, Δ^-, Δ^+, and Γ^+, and penalty costs ζ^-, ε^-, ε^+, and ζ^+, $\mathcal{D}$ is defined by

$$\mathcal{D}(u) = \sum_{i=1}^{m} \mathcal{D}_i(u_i) \tag{9}$$

where

$$\mathcal{D}_i(u_i) = \begin{cases} -(\zeta_i^- + \varepsilon_i^-)\ (u_i + \Gamma_i^-) - \zeta_i^- \Delta_i^- & if\ -\infty \le u_i \le -\Gamma_i^- - \Delta_i^- \\ -\varepsilon_i^-(\ u_i - \Delta_i^-) & if\ -\Gamma_i^- - \Delta_i^- \le u_i \le -\Delta_i^- \\ 0 & if\ -\Delta_i^- \le u_i \le \Delta_i^+ \\ +\varepsilon_i^+(\ u_i - \Delta_i^+) & if\ \Delta_i^+ \le u_i \le \Delta_i^+ + \Gamma_i^+ \\ +(\varepsilon_i^+ + \zeta_i^+)\ (u_i - \Gamma_i^+) - \zeta_i^+ \Delta_i^+ & if\ \Delta_i^+ + \Gamma_i^+ \le u_i \le +\infty \end{cases} . \tag{10}$$

The corresponding Fenchel's conjugate $\mathcal{D}^*$ is

$$\mathcal{D}^*(y) = \sum_{i=1}^{m} \mathcal{D}_i^*(y_i) \tag{11}$$

where

$$\mathcal{D}_i^*(y_i) = \begin{cases} +\infty & if\ y_i < -(\zeta_i^- + \varepsilon_i^-) \\ -(\Gamma_i^- + \Delta_i^-)\ y_i - \Gamma_i^- \varepsilon_i^- & if\ -\zeta_i^- - \varepsilon_i^- \le y_i \le -\varepsilon_i^- \\ -\Delta_i^-\ y_i & if\ -\varepsilon_i^- \le y_i \le 0 \\ +\Delta_i^+\ y_i & if\ 0 \le y_i \le \varepsilon_i^- \\ +(\Gamma_i^+ + \Delta_i^+)\ y_i - \Gamma_i^+ \varepsilon_i^+ & if\ \varepsilon_i^+ \le y_i \le (\zeta_i^+ + \varepsilon_i^+) \\ +\infty & if\ y_i > (\zeta_i^+ + \varepsilon_i^+) \end{cases} . \tag{12}$$

This kind of function offers a better approximation of $\|.\|_2$ (or any $\|.\|_p$ with $2 < p < \infty$) than a similar function with only 3 pieces, especially far from the trust region. This is needed because the main penalty incurred should not play the role similar to that of any piece of the function to be minimized. For example, in the case where $\mathcal{B}_1 = \emptyset$, the dual objective function has only one linear piece. Using only one penalizing piece in each side of $\mathcal{D}$ may lead to either the penalty being less than the right-hand side, and therefore not being effective, or it being higher than the right-hand side, and therefore being effective but, probably, too high to allow interesting displacements outside the trust region. In the latter situation, dual points that are generated will be in most cases on the boundary of the trust region and little is done more than the boxstep method.

It is interesting to write down in details the primal and dual stabilized master problems with this choice of the stabilizing term. In order to simplify the formulation, we use the following notations:

$$\begin{aligned} \gamma_1 &= \bar{\pi} - \Delta^- - \Gamma^- \ , \\ \delta_1 &= \bar{\pi} - \Delta^- \ , \\ \delta_2 &= \bar{\pi} + \Delta^+ \ , \\ \gamma_2 &= \bar{\pi} + \Delta^+ + \Gamma^+ \ . \end{aligned}$$

The stabilized master dual problem is then

$$(D_{\mathcal{B},\bar{\pi},\mathcal{D}}) \qquad \begin{array}{ll} \min & \eta + \pi b + \zeta^- v^- + \varepsilon^- u^- + \varepsilon^+ u^+ + \zeta^+ v^+ \\ & -u^- + \delta_1 \leq \pi \leq \delta_2 + u^+ \\ & -v^- + \gamma_1 \leq \pi \leq \gamma_2 + v^+ \\ & \pi a \leq c_a \quad , \quad a \in \mathcal{B} \\ & \pi a + \eta \leq c_a \quad , \quad a \in \mathcal{B}_1 \\ & v^-, u^-, u^+, v^+ \geq 0 \end{array} \ . \tag{13}$$

Its dual, the stabilized primal master problem, is

$$(P_{\mathcal{B},\bar{\pi},\mathcal{D}}) \qquad \begin{array}{ll} \max & \displaystyle\sum_{a \in \mathcal{B}} c_a x_a + \gamma_1 z_1 + \delta_1 y_1 - \delta_2 y_2 - \gamma_2 z_2 \\ & \displaystyle\sum_{a \in \mathcal{B}} a x_a + z_1 + y_1 - y_2 - z_2 = b \\ & \displaystyle\sum_{a \in \mathcal{B}_1} x_a = 1 \\ & z_1 \leq \zeta^- \ , \ y_1 \leq \varepsilon_1 \ , \ y_2 \leq \varepsilon_2 \ , \ z_2 \leq \zeta^+ \\ & x_a \geq 0 \quad , \quad a \in \mathcal{B} \\ & z_1,\ y_1,\ y_2,\ z_2 \ \geq 0 \end{array} \ . \tag{14}$$

The number of constraints in the primal problem does not change whereas 4 variables are added for each stabilized constraint. Variables corresponding to the outmost penalty segments can act as artificial variables and hence no Phase I is needed to start the CG algorithm. Even if these variables are unit vectors and are easy to handle, the size of the

master problem may become too large to handle enough generated columns, especially when many columns are generated at each iteration. This might happen while using 7-piecewise penalty function, which would in principle offer even more flexibility but appears to leave little room for dynamically generated columns. 5-piecewise penalty function seems to give a good tradeoff between flexibility and practical efficiency of the approach.

The 5-pieces stabilizing term allows for a small penalty close to the trust region and much larger penalty far from it. In doing so, very large moves are allowed only when a high improvement in the stabilized dual objective is anticipated. More reasonable moves are accepted even if the objective improvement is not very high. This feature is useful, e.g., in the beginning of the stabilization process. 3-pieces function may suffer very small moves if the penalty is too high and may fall in the same instable behaviour as CG/CP if very small penalties are used. 5-pieces function can cope with this inconvenience by allowing small penalties in the outer box and imposing large enough penalties outside this box. This is shown in Table 3 for some Multiple Depot Vehicle Scheduling problem instances (see Section 5.2 for more details), where CG stands for standard column generation while PP-3 and PP-5 stand for stabilized approaches using, respectively, 3-piecewise stabilizing terms and 5-piecewise ones. In the table, cpu gives total computing time, lr, mp, and sp give the cpu times in seconds needed to solve linear relaxations, master problems, and subproblems, respectively, while itr give the number of column generation iterations needed to solve linear relaxations to optimality. Both stabilized algorithms improve standard CG substantially; however, PP-5 clearly outperforms PP-3 on all aspects, especially total computing time and CG iterations number.

Pb		p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
	method										
cpu(s)	CG	139.0	176.6	235.4	158.9	3138.1	3966.2	3704.3	1741.5	3685.2	3065..2
	PP-3	79.9	83.9	102.5	70.3	1172.5	818.7	1440.2	1143.3	1786.5	2282.8
	PP-5	31.3	36.4	37.8	27.8	481.9	334.6	945.7	572.3	1065.2	2037.4
itr	CG	117	149	200	165	408	524	296	186	246	247
	PP-3	82	92	104	75	181	129	134	145	144	189
	PP-5	47	47	49	45	93	64	98	83	86	150
mp(s)	CG	88.4	124.5	164.8	104.8	1679.4	2003.7	1954.6	924.8	1984.2	1742.6
	PP-3	44.0	46.6	59.6	42.0	571.5	399.4	740.4	542.5	858.3	1350.5
	PP-5	12.9	16.3	16.6	9.8	188.8	128.2	428.2	256.5	541.9	1326.0
sp(s)	CG	50.6	51.1	70.6	54.1	1458.7	1962.5	1749.7	816.7	1701.0	1322.6
	PP-3	33.9	37.3	42.9	28.3	601.0	419.3	700.8	590.8	928.2	932.3
	PP-5	18.4	20.1	21.2	18.0	293.1	206.4	517.5	315.8	523.3	711.4

Table 3: Comparing 5-piecewise and 3-piecewise penalty functions.

The proposed stabilizing term also provides more flexibility in updating strategies when compared to 3-piecewise linear functions or $l-p$ norms; this is represented by the many parameters $\Gamma^{\pm}$, $\Delta^{\pm}$, $\zeta^{\pm}$, and $\varepsilon^{\pm}$. These can be independently updated for each variable with several different strategies. This is important because all dual vector components do not behave the same way to converge towards optimality. A nearly stabilized component should not have the same stabilizing parameters as one that is far from being stabilized. The parameters allows for several different ways to lessen (or sharpen) the stabilizing term. One can either change the inner intervals, the outer intervals, the inner penalty parameters, the outer penalty parameters, or any combination of them. For example, lessening $\varepsilon^{\pm}$ and enlarging $\zeta^{\pm}$ leads to smaller penalty within the outer box and larger penalty outside this box, which cannot be done e.g. with $\|.\|_{\infty}$ and 3-piecewise linear functions.

However, a 5-piecewise stabilizing term is clearly more costly than a 3-piecewise one. A possible remedy, when the number of rows in the primal is too large, is to stabilize only a subset of the rows. Hence, the dual vector π is partially penalized; identifying the "most important" components can help in choosing an adequate subset of components to be penalized. Alternatively, one may choose the number of pieces dynamically, and independently, for each variable. In fact, at advanced stages of the process many dual components are near to their optimal value; in such a situation, the outer segments of the penalty term are not needed and the corresponding stabilization variables may be eliminated from the primal master problem. By doing so, in the last stages of the solution process, one should have a 3-piece function that allows small number of stabilization variables and ensures primal feasibility at the end of the algorithm. In [3], changing some components of the 5-piecewise penalty functions into 3-piecewise terms in the last stages of the process, whenever the corresponding dual variables values are found to be "stabilized enough", allowed a substantial improvement of the performances. Table 4 shows the results obtained on MDVS (again, see Section 5.2 for more details). In the table, CG stands for standard column generation, PP-5 stands for the Stabilized Column Generation approach using a fixed 5-pieces stabilizing term, and 5then3 designates the dynamic strategy; lr, mp, and sp give the cpu times in seconds needed to solve linear relaxations, master problems, and subproblems, respectively, while itr give the number of column generation iterations needed to solve linear relaxations to optimality. The table clearly shows that the dynamic strategy helps in improving performances, especially as the size of the instances increase.

Several different parameters updating strategies have been tested in different contexts [3, 2, 1]. For instance, in [3, 2] a strategy that enlarges the outer box if the corresponding component of the tentative point is out of the current box and lessens the inner penalty parameters at each serious step was shown to outperform a strategy that updates the penalty function parameters depending on the position of the corresponding component. In [1] a strategy where the shape of the function changes asymmetrically during the solution process was proposed that was able to outperform all other strategies for most of the problems. However, the approach leaves full scope for a large number of strategies, possibly problem-specific, to be tested.

Pb	$p1$			$p3$		
method	CG	PP-5	5then3	STD	SCG	5then3
lr(s)	203.5	42.5	37.0	285.2	44.9	48.8
mp(s)	125.9	13.7	12.1	180.5	20.0	18.1
sp(s)	77.6	28.5	24.9	104.7	24.9	30.7
itr	149	52	48	196	53	58
Pb	$p6$			$p5$		
method	CG	PP-5	5then3	STD	SCG	5then3
lr(s)	4178.4	595.5	501.2	3561.9	306.4	256.0
mp(s)	3149.2	216.2	166.2	2676.2	141.9	113.9
sp(s)	1029.2	379.3	335.0	885.7	164.5	142.1
itr	509	196	160	422	80	72
Pb	$p7$			$p8$		
method	CG	PP-5	5then3	STD	SCG	5then3
lr(s)	2883.4	1223.5	1068.3	1428.9	837.1	757.2
mp(s)	1641.3	610.9	593.4	779.4	479.6	272.0
sp(s)	1242.1	612.6	474.9	649.2	357.5	485.2
itr	380	178	145	259	145	134

Table 4: Results for 5-then-3-pieces approach.

Parameters updating strategies also depend on the specific form of SCG algorithm used. Pure proximal approaches, that solve the stabilized master problems exactly, may have different (simpler) parameters updating rules; in particular, typically the parameters are never changed in the sequence of iterations required to solve one stabilized problem. However, pure proximal approaches may suffer from tailing-off effects; if early stopping is possible, better performances are to be expected, as observed e.g. in [3]. Thus, bundle-type approaches, where the stability center is changed more frequently, may be more promising in general. However, it is customary in bundle approaches to modify the stabilizing term even during sequences of Null Steps, corresponding to the (approximate) solution process of the current stabilized primal and dual master problem. This may require different, more sophisticated parameter updating rules than in the pure proximal case.

5 Numerical Experiments: MDVS

5.1 The MDVS problem

Let $\{T_1, \ldots, T_m\}$ be a set of tasks to be covered by vehicles available at depots D_k ($k \in K$) with capacity n_k. Define Ω the set of routes that can be used to cover the given tasks. A route is a path between the source and the sink of a compatibility network. It has to start and end at the same depot. The nodes of the network correspond to tasks and depots.

Arcs are defined from depots to tasks, from tasks to depots and between tasks allowed to be visited one after another by a same vehicle. Define c_p $(p \in \Omega)$ as the cost of route p, binary constants a_{ip} $(p \in \Omega, i \in \{1, \ldots, m\})$ that take value 1 iff task T_i is covered by tour p and binary constants b_p^k $(p \in \Omega, k \in K)$ that take value 1 iff route p starts and ends at depot D_k. Using binary variables Θ_p $(p \in \Omega)$ which indicate if tour p is used by a vehicle to cover a set of tasks, the problem is formulated as:

$$\begin{aligned} Min \ & \sum_{p \in \Omega} c_p \Theta_p \\ & \sum_{p \in \Omega} a_{ip} \Theta_p = 1 \quad i = 1, \ldots, m \\ & \sum_{p \in \Omega} b_p^k \Theta_p \leq n_k \quad k \in K \\ & \Theta_p \in \{0, 1\} \quad p \in \Omega. \end{aligned} \tag{15}$$

Due to the very larger number of routes (columns), the problem is usually solved by branch-and-price where linear relaxations are solved by column generation [31, 20]. Given the set of multipliers produced by the master problem, columns are generated by solving a shortest path problem on the compatibility network to compute a negative reduced cost column, if any. We are interested in stabilizing the column generation process at the root node; the same process may then be used for any other branch-and-price node.

5.2 The test problems

Test problem sets are generated uniformly following the scheme of [5]. The cost of a route has two components: a fixed cost due to the use of a vehicle and a variable cost incurred on arcs. There are two types of problems A and B depending on the location of depots. Each problem is characterized by its number of tasks $m \in \{400, 800, 1000, 1200\}$, its type $T \in \{A, B\}$, and the number of depots $d \in \{4, 5\}$. The master problem has $m + d$ constraints and d subproblems are solved to generate columns. Table 5 gives the characteristics of the 10 instances.

5.3 The algorithms

Initialization All stabilization approaches that are tested use the same initialization procedure. The single depot vehicle scheduling problem, SDVS, is s special case of MDVS for which there is only one depot. But this is a minimum cost flow problem over the corresponding compatibility network and can be solved in polynomial time. Now, consider the following depot aggregation procedure in MDVS. All the depots are replaced with a single depot D_0. The cost of arc (D_0, T_i) is the minimum of the costs of arcs $(D_k, T_i), k \in K$. The cost of arc (T_i, D_0) is the minimum of the costs of arcs $(T_i, D_k), k \in K$. This lead to a SDVS with slightly fewer nodes and arcs. Its primal optimal solution may be used to

Problem	T	m	d	arcs number
$p1$	A	400	4	206696
$p2$	A	400	4	208144
$p3$	B	400	4	210328
$p4$	B	400	4	199560
$p5$	A	800	4	785880
$p6$	B	800	4	816780
$p7$	A	1000	5	1273435
$p8$	B	1000	5	973260
$p9$	A	1200	4	1472295
$p10$	B	1200	4	1147788

Table 5: MDVSP: instance characteristics.

compute an initial integer feasible solution for MDVS as well as an upper bound on the integer optimal value. The corresponding dual solution is feasible to (D) and provides a lower bound on the linear relaxation optimal value. This dual point is used as initial stability center in the algorithm (see [3] for more details).

Pure proximal approach We use an improved version of the pure proximal strategy found to be the best in [3, 2]. The penalty functions are kept symmetric and the parameters $\Delta^{\pm}$ are kept fixed to a relatively small value (5). The outer penalty parameters $\zeta^{\pm}$ have their initial values equal to 1 (the right-hand side of stabilized constraints). The corresponding columns in the primal act as artificial variables which cost depends on the initial stability center, before the first *SS* is executed. Since the problem contains no explicit convexity constraint, Serious Steps are performed only when no positive reduced column is generated, i.e., optimality of stabilized problem is reached. In this case, the penalty parameters $\epsilon^{\pm}$ and $\zeta^{\pm}$ are reduced using different multiplying factors α_1, $\alpha_2 \in]0, 1[$. If the newly computed dual point is outside the outer hyperbox, the outer intervals are enlarged, i.e., $\Gamma_i^{\pm}$ is multiplied by a factor $\beta \geq 1$. Several triplets $(\alpha_1, \alpha_2, \beta)$ produced performant algorithms. Primal and dual convergence is ensured by using full dimensional trust regions that contain 0 in their interior and never shrink to a single point, i.e. $\Delta^{\pm} \geq \underline{\Delta} > 0$ at any CG iteration.

Bundle approaches When fixed costs are sufficiently large, the optimal solution of MDVS linear relaxation ensures the minimum number of vehicles. This minimum number is the same as the number of vehicles obtained by solving the SDVS problem obtained via the aggregation procedure described above. The instances considered here use a large enough fixed cost to ensure this property. The "objective function" used to verify for minimum improvement is then defined by

$$\min \pi b + \bar{b} \max\{c_a - \pi a : a \in \mathcal{A}\},$$

where $\bar{b}$ is the number of routes in the solution of the SDVSP problem solved in the initialization procedure.

Using this objective function, bundle approaches offer the opportunity of updating the stability center, and hence improving the quality of the "best" known dual point, without going to the optimality of column generation applied to the stabilized problem. As a consequence, the queuing effect is reduced, at least far from optimality.

The same parameters strategy used in the pure proximal case is adapted here: during Null Steps, the stabilizing term is kept unchanged. As previously noted, different strategies may help in improving the performances, but we found this simple one to be already quite effective.

5.4 Results

Results are given in Table 6 for standard column generation (CG), the pure proximal approach (PP), and the bundle approach (BP). The stabilized approaches are implemented using 5-piecewise linear penalty functions. In the table, cpu, mp, and sp are respectively the total, master problem, and subproblem computing times in seconds,while itr is the number of column generation iterations (null steps) needed to reach optimality.

Pb		p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
	method										
cpu(s)	CG	139.0	176.6	235.4	158.9	3138.1	3966.2	3704.3	1741.5	3685.2	3065..2
	PP	31.3	36.4	37.8	27.8	481.9	334.6	945.7	572.3	1065.2	2037.4
	BP	25.5	27.9	34.5	21.4	294.5	257.2	639.4	351.7	545.2	1504.5
itr	CG	117	149	200	165	408	524	296	186	246	247
	PP	47	47	49	45	93	64	98	83	86	150
	BP	37	43	44	36	57	53	59	49	51	101
mp(s)	CG	88.4	124.5	164.8	104.8	1679.4	2003.7	1954.6	924.8	1984.2	1742.6
	PP	12.9	16.3	16.6	9.8	188.8	128.2	428.2	256.5	541.9	1326.0
	BP	9.9	13.7	14.9	10.1	100.2	70.0	329.3	206.3	334.2	982.5
sp(s)	CG	50.6	51.1	70.6	54.1	1458.7	1962.5	1749.7	816.7	1701.0	1322.6
	PP	18.4	20.1	21.2	18.0	293.1	206.4	517.5	315.8	523.3	711.4
	BP	15.6	14.2	15.6	11.3	194.3	187.2	310.1	145.4	211.2	522.0

Table 6: Computational results for MDVS.

Both stabilization approaches are able to substantially improve the standard column generation approach according to computation time, on all problems. The stabilization effect is illustrated by the reduction of master problem computation times and column generation iterations numbers, for all instances.

The improvement is more uniform among the two approaches for small size problems ($m = 400$), while for medium size problems ($m = 800$) and especially large size ones ($m \in \{1000, 1200\}$) the bundle approach is substantially better. This is probably due to the fact that for larger problem the initial dual solution is of less good quality, and the good performances pure proximal approach are more dependent on the availability of a very good initial dual estimate to diminish the total number of (costly) updates of the stability center. This is clearly less critical for the bundle approach, that is capable of updating the stability canter at a substantial inferior cost, and therefore is less harmed by an initial estimate of lesser quality.

A particularity in the behaviour of the bundle approach is that master problem computation times are little bit higher in average. This may be explained by high master problem reoptimization costs due to larger number of serious steps.

6 Numerical Experiments: Vehicle and Crew Scheduling (VCS)

6.1 The VCS problem

We consider the simultaneous Vehicle and Crew Scheduling problem (VCS) described in [19]. The problem requires to simultaneously optimally design trips for vehicles (buses, airplanes, ...) which cover a given set of "work segments" and the duties of the personnel required to operate the vehicles (drivers, pilots, cabin crews, ...).

The problem can be formulated as follows. Let K be the set of types of duties (working days), Ω^k the set of duties of type k, S the set of segments to be covered by duties, V the set of trips, and H the set of possible departure times from parkings (h^* designates the latest possible departure time). Binary constants c_{ps}, i_{pv}, o_{pv}, d_{ph}, and a_{ph} ($p \in \Omega^k$, $s \in S$, $v \in V$, $h \in H$) are needed for the problem formulation. c_{ps} takes value 1 if and only if duty p covers segment s. i_{pv} and o_{pv} take value 1 if and only if duty p contains a deadhead or a walking that starts at the end of trip v or ends at the beginning of trip v, respectively. d_{ph}, respectively a_{ph}, takes value 1 if and only if duty p contains a deadhead starting, respectively ending, at the parking.

The formulation (16) uses binary variables θ_p ($p \in \Omega^k$, $k \in K$) that takes value 1 if and only if duty p is used in the solution. The objective function that is minimized counts the number of duties that are selected.

$$
\begin{aligned}
& Min \sum_{k\in K}\sum_{p\in\Omega^k} \Theta_p \\
& \sum_{k\in K}\sum_{p\in\Omega^k} c_{ps}\Theta_p = 1 && \forall\, s \in S \\
& \sum_{k\in K}\sum_{p\in\Omega^k} i_{pv}\Theta_p = 1 && \forall\, v \in V \\
& \sum_{k\in K}\sum_{p\in\Omega^k} o_{pv}\Theta_p = 1 && \forall\, v \in V \\
& \sum_{k\in K}\sum_{p\in\Omega^k} d_{ph}\Theta_p - \sum_{k\in K}\sum_{p\in\Omega^k} a_{ph}\Theta_p \le 0, && \forall h \in H - \{h^*\} \\
& \sum_{k\in K}\sum_{p\in\Omega^k} d_{ph^*}\Theta_p - \sum_{k\in K}\sum_{p\in\Omega^k} a_{ph^*}\Theta_p = 0 \\
& \Theta_p \in \{0,1\} && \forall p \in \Omega^k,\ k \in K.
\end{aligned}
\tag{16}
$$

Because the set of duties is extremely large, the linear relaxation of (16) can only be solved by column generation. In [19], the subproblem is formulated as a constrained shortest path problem using up to 7 resources. The subproblem solution is very expensive, especially for the last CG iterations, because some resources are negatively correlated. The solution time for the subproblem can be reduced by solving it heuristically, using an idea of [14]. Instead of building an unique network in which constrained shortest paths with many resources need to be solved, many (hundreds) different subnetworks, one for each possible departure time, are built. This allows to take into account several constraints that would ordinarily be modeled by resources while building the subnetworks. Of course, solving a (albeit simpler) constrained shortest path problem for each subnetwork would still be very expensive; therefore, only a small subset (between 10 and 20) of subnetworks are solved at each CG iteration. The subproblem cost thus becomes much cheaper, except for the very last iteration, and possibly one or two more, where optimality has to be proved, and therefore all the subnetworks have to be solved.

6.2 The test problems

We use a set of 7 test-problems taken from [19]. They are named pm, where m is the total number of covering constraints in the master problem. Their characteristics are presented in Table 7. In the table, Cov, Flow, Net, Nodes, Arcs give respectively the number of covering constraints, the number of flow conservation constraints in the master problem, the number of subnetworks, the number of nodes, and the number of arcs per subnetwork.

Problem	Cov	Flow	Net	Nodes	Arcs
$p199$	199	897	822	1528	3653
$p204$	204	919	829	1577	3839
$p206$	206	928	835	1569	3861
$p262$	262	1180	973	1908	4980
$p315$	315	1419	1039	2180	6492
$p344$	344	1549	1090	2335	7210
$p463$	463	2084	1238	2887	9965

Table 7: VCS: instance characteristics.

6.3 The algorithms

We tested different stabilized CG approaches for the VCS problem. Differently from the MDVS case, for VCS using a "pure proximal" stabilized CG approach for solving the linear relaxation turned out to *worsen* the performances of the CG approach. This is due to the fact that a pure proximal stabilized CG algorithm needs to optimally solve the subproblem many times, each time that optimality of the stabilized problem has to be proven. Thus, even if the CG iterations number is reduced by the stabilization, the subproblem computing time – and hence the total computing time – increases. Even providing very close estimates of dual optimal variables is not enough to make the pure proximal approach competitive.

Instead, a bundle-type approach, that do not need to optimally solve the stabilized problem except at the very end, was found to be competitive. For implementing the approach, an artificial convexity constraint was added to the formulation, using a straightforward upper bound on the optimal number of duties. As for the MDVS case, after a Serious Step – when the stability center is updated – the stabilizing term is decreased using proper simple rules, while after a Null Step the stabilizing term is kept unchanged. Note that since the dual variables must be in $[-1, 1]$, this property is preserved while updating the stability centers.

6.4 Results

Results of the experiments on VCS are given in Table 8. In the table, cpu, mp, and sp are respectively the total, master problem, and subproblem computing times in minutes, while itr is the number of column generation iterations (null steps) needed to reach optimality.

The results show that the number of CG iterations and the total computing time is reduced when using a bundle-type stabilized CG approach as opposed to an ordinary CG one. This happens even if the subproblem computing time is increased, as it is the case for the largest problem p463. Thus, the bundle-type approach allows to reduce the number of CG iterations, thanks to the stabilization, without the cost of optimally solving the subproblem too often; this directly translates in better overall performances of the algorithm.

Pb		p199	p204	p206	p262	p315	p344	p463
	method							
cpu(min)	CG	26	26	30	68	142	238	662
	BP	12	13	14	40	73	163	511
mp(min)	CG	13	9	14	35	43	90	273
	BP	3	3	4	7	19	20	93
sp(min)	CG	13	17	16	33	99	148	389
	BP	9	10	10	33	54	143	418
itr	CG	167	129	245	263	239	303	382
	BP	116	119	173	160	213	201	333

Table 8: Computational results for VCS.

7 Conclusions

Using a general theoretical framework developed in the context of NonDifferentiable Optimization, we have proposed a generic column generation stabilization algorithm. The definition of the algorithm leaves full scope for a number of crucial details in the implementation, such as parameters updating strategies for the stabilization terms. The framework has then been specialized to a specific 5-piecewise linear stabilization term that allows the stabilized primal and dual master problems to be solved by linear programming. The 5-pieces function is found to provide a good compromise between flexibility and master problem solution cost, especially when dynamic rules are used that discard "useless" pieces near termination of the algorithm. The meaning of the parameters in the 5-pieces function in terms of the original problem is discussed; the relationships allow for more easily defining reasonable initial values for the parameters and proper updating strategies.

The resulting algorithm has been tested in the context of two different practical applications: Multiple Depot Vehicle Scheduling and simultaneous Vehicle and Crew Scheduling. For both applications, the best stabilized column generation approaches are found to significantly improve the performances of classical (non-stabilized) column generation. In particular, "bundle" variants, where the stability center is updated whenever possible, are shown to be promising with respect to "pure proximal" variants, where the stability center is only updated when the stabilized problem is solved to optimality. This is particularly true for VCS, where the pure proximal approach requires to optimally solve the subproblem often, while the bundle approach allows to heuristically solve it at all but the very last iterations; since the subproblem cost is very significant in this case, this is crucial for the overall efficiency of the CG approach. For both applications, using bundle-type variants requires introducing an artificial convexity constraint to define an "objective function" whose decrease can be monitored in order to early perform the update of the stability center. In general, this requires computing some estimate of some quantity related to the optimal solution, which, especially if the estimate has to be tight, may in itself be a de-

manding task. However, at least for MDVS and VCS using easily found rough estimates to construct the right-hand-side of the artificial constraint seems to work quite well.

All in all, we believe that the present results show that stabilized column generation approaches have plenty of as yet untapped potential for substantially improving the performances of solution approaches to linear programs of extremely large size.

References

[1] H. Ben Amor, J. Desrosiers, and A. Oukil, A Numerical Study of a Pure Proximal Stabilized Column Generation Algorithm for Long-Horizon MDVSP, *Les Cahiers du GERAD* G-2004-57 (2004).

[2] H. Ben Amor, J. Desrosiers, A Proximal Trust Region Algorithm for Column Generation Stabilization, *Les Cahiers du GERAD* G-2003-43 (2003).

[3] H. Ben Amor, Stabilisation de l'algorithme de génération de colonnes, Ph.D. Thesis, Département de Mathématiques et de Génie Industriel, École Polytechnique de Montréal, Montréal, Canada (2002).

[4] C. Berger, Contribution à l'optimisation non-différentiable et à la décomposition en programmation mathématique, Ph.D. Thesis, Département de mathématiques et de génie industriel, École Polytechnique de Montréal, Montreal, Canada (1996).

[5] D. Carpaneto, M. Dall'Amico, M. Fischetti, and P. Toth, A Branch and Bound Algorithm for the Multiple Depot Vehicle Scheduling Problem, *Networks* 19, 531–548 (1989).

[6] P. Carraresi, L. Girardi, and M. Nonato, Network Models, Lagrangian Relaxations and Subgradient Bundle Approach in Crew Scheduling Problems, in *Computer Aided Scheduling of Public Transport, Lecture Notes in Economical and Mathematical Systems*, J. Paixao, ed., Springer-Verlag (1995).

[7] T.G. Crainic, A. Frangioni, and B. Gendron, Bundle-based relaxation methods for multicommodity capacitated fixed charge network design problems, *Discrete Applied Mathematics* 112, 73–99 (2001).

[8] G.B. Dantzig, P. Wolfe, The decomposition Principle for Linear Programs, *Operations Research* 8(1), 101–111 (1960).

[9] O. Du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen, Stabilized Column Generation, *Discrete Mathematics* 194, 229–237 (1999).

[10] J. Eckstein, Nonlinear proximal point algorithms using Bregman functions with applications to convex programming, *Mathematics of Operations Research* 18, 292–326 (1993).

[11] A. Frangioni, Solving semidefinite quadratic problems within nonsmooth optimization algorithms, *Computers & Operations Research* 23, 1099–1118 (1996).

[12] A. Frangioni, Generalized Bundle Methods, *SIAM Journal on Optimization* 13(1), 117–156 (2002).

[13] A. Frangioni, G. Gallo, A Bundle Type Dual-Ascent Approach to Linear Multicommodity Min Cost Flow Problems, *INFORMS Journal on Computing* 11(4), 370–393 (1999).

[14] R. Freiling, A.P.M. Wagelmans, and A. Paixao, An overview of models and techniques for integrating vehicle and crew scheduling, in *Computer-Aided Transit Scheduling. Lecture Notes in Economics and Mathematical Systems* 471, N.H.M. Wilson, ed., Springer, Berlin, Germany, 441–460 (1999).

[15] M. Gamache, F. Soumis, G. Marquis, and J. Desrosiers, A Column Generation Approach for Large Scale Aircrew Rostering Problems, *Operations Research* 47(2), 247–262 (1999).

[16] P.C. Gilmore, R.E. Gomory, A Linear Programming Approach to the Cutting Stock Problem, *Operations Research* 11, 849–859 (1961)

[17] J.-L. Goffin, J.-P. Vial, Convex nondifferentiable optimization: a survey focussed on the analytic center cutting plane method, *Optimization Methods and Software* 17(5), 805–867 (2002).

[18] M.D. Grigoriadis, L.G. Kahchiyan, An exponential-function reduction method for block-angular convex programs, *Networks* 26, 59–68 (1995).

[19] K. Haase, G. Desaulniers, and J. Desrosiers. Simultaneous Vehicle and Crew Scheduling in Urban Mass Transit Systems, *Transportation Science* 35(3), 286–303 (2001).

[20] A. Hadjar, O. Marcotte, and F. Soumis, A Branch-and-Cut Algorithm for the Mutiple Depot Vehicle Scheduling Problem, *Les Cahiers du GERAD* G-2001-25 (2001).

[21] J.-B. Hiriart-Urruty, C. Lemaréchal, *Convex Analysis and Minimization Algorithms*, Grundlehren Math. Wiss. 306, Springer-Verlag, New York (1993).

[22] A.N. Iusem, B.F. Svaiter, and M. Teboulle, Entropy-like proximal methods in convex programming, *Mathematics of Operations Research* 19, 790–814 (1994).

[23] K. Kiwiel, A bundle Bregman proximal method for convex nondifferentiable optimization, *Mathematical Programming* 85, 241–258 (1999).

[24] S. Kim, K.N. Chang, and J.Y. Lee, A descent method with linear programming subproblems for nondifferentiable convex optimization, *Mathematical Programming* 71, 17–28 (1995).

[25] J.E. Kelley, The Cutting-Plane Method for Solving Convex Programs, *Journal of the SIAM* 8, 703–712 (1960).

[26] C. Lemaréchal, Bundle Methods in Nonsmooth Optimization, in *Nonsmooth Optimization*, vol. 3 of IIASA Proceedings Series, C. Lemaréchal and R. Mifflin, eds., Pergamon Press (1978).

[27] C. Lemaréchal, A. Nemirovskii, and Y. Nesterov, New variants of bundle methods, *Mathematical Programming* 69, 111–147 (1995).

[28] C. Lemaréchal, Lagrangian Relaxation, in *Computational Combinatorial Optimization*, M. Jünger and D. Naddef, eds., Springer-Verlag, Heidelberg, 115–160 (2001).

[29] R.E. Marsten, W.W. Hogan, and J.W. Blankenship, The BOXSTEP Method for Large-scale Optimization, *Operations Research* 23(3), 389–405 (1975).

[30] M.C. Pinar, S.A. Zenios, On smoothing exact penalty functions for convex constrained optimization, *SIAM Journal on Optimization* 4, 486–511, (1994).

[31] C.C. Ribeiro, and F. Soumis, A Column Generation Approach to the Multi-Depot Vehicle Scheduling Problem, *Operations Research* 42(1), 41–52 (1994).

[32] R.T. Rockafellar, Monotone Operators and the Proximal Point Algorithm, *SIAM Journal on Control and Optimization* 14(5) (1976).

[33] H. Schramm, J. Zowe, A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results, *SIAM Journal on Optimization* 2, 121–152 (1992).

[34] M. Teboulle, Convergence of proximal-like algorithms, *SIAM Journal on Optimization* 7, 1069–1083 (1997).