On The Choice of Explicit Stabilizing Terms in Column Generation

Hatem M.T. Ben Amor[*] Jacques Desrosiers[†]
Antonio Frangioni[‡]

Abstract

Column generation algorithms are instrumental in many areas of applied optimization, where linear programs with an enormous number of columns need to be solved. Although succesfully employed in many applications, these approaches suffer from well-known *instability* issues that somewhat limit their efficiency. Building on the theory developed for nondifferentiable optimization algorithms, a large class of stabilized column generation algorithms can be defined which avoid the instability issues 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 for a great degree of flexibility in the choice and in the management of the stabilizing term, one can use piecewise-linear or quadratic functions that can be efficiently dealt with off-the-shelf solvers. The effectiveness in practice of this approach is demonstrated by extensive computational experiments on large-scale Vehicle and Crew Scheduling problems. Also, the results of a detailed computational study on the impact of the different choices in the stabilization term (shape of the function, parameters), and their relationships with the quality of the initial dual estimates, on the overall effectiveness of the approach are reported, providing practical guidelines for selecting the most appropriate variant in different situations.

[*] Ad-Opt Division, Kronos Canadian Systems, 3535 Queen Mary Rd, Suite 650, Montreal (Canada) H3V1H8, email: hbenamor@kronos.com, and Groupe d'Études et de Recherche en Analyse des Décisions (GERAD), 3000, chemin de la Côte-Sainte-Catherine, Montréal (Canada) H3T 2A7, e-mail: hatem.ben.amor@gerad.ca

[†] HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal (Canada) H3T 2A7, e-mail: jacques.desrosiers@gerad.ca

[‡] Dipartimento di Informatica, Università di Pisa, Polo Universitario della Spezia, Via dei Colli 90, 19121 La Spezia (Italy), e-mail: frangio@di.unipi.it

Keywords: *Column Generation, Proximal Point methods, Bundle methods, Vehicle and Crew Scheduling problems*

1 Introduction

Column Generation (CG) has proven to be very successful in solving very large scale optimization problems, such as those obtained as the result of decomposition/reformulation approaches applied to some original integer programming formulations. It has been introduced independently by Gilmore and Gomory [15] and Dantzig and Wolfe [8] in the early sixties. 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 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 (LP) 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 LP. A restricted problem using a subset of extreme points and rays is solved, obtaining optimal dual multipliers that are used to generate positive reduced cost columns, if any. In both cases, optimality is reached when no such column exists. Hence, CG 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 [21]; it solves convex problems by generating supporting hyperplanes of the objective function. At each iteration, the dual of the restricted problem in D-W is solved and cuts are added until dual feasibility, and therefore optimality, are reached. Thus, the *column generation*, or *pricing*, problem in the primal is a *separation problem* in the dual, seeking for cuts which separate the current estimate of the dual optimal solution from the true value [13].

Although CG/CP algorithms have been used with success in many applications, difficulties appear when solving very large scale degenerate problems. It is well-known that primal degeneracy may cause a “tail-off” effect in column generation. Moreover, instability in the behavior of dual variables are more frequent and harmful when problems get larger (cf. e.g. [6, §4(ii)]): it is possible to move from a good dual point to a much worse one, which affects the quality of columns to be generated in the following iteration, and therefore the overall convergence speed of the algorithm. This effect can be countered by employing *stabilization approaches*.

A first form of stabilization has been proposed in the early seventies within the nondifferentiable optimization community (e.g. [22]): a "good" dual point among those visited so far is taken to define the *stability center*, and an *explicit Stabilizing Term* (ST) that penalizes moves far from the center is added to the dual objective function. The stability center is changed if a "sufficiently better" dual point is found. A variety of stabilized algorithms of this kind has been proposed [26, 32, 20, 23, 19], and a deeper theoretical understanding of the underlying principles [18, 33, 12] has been achieved over time; we especially refer the interested reader to [24].

A different form of stabilization involves avoiding extremal solutions in the restricted problem and insisting that an interior solution has to be used [31]. This can be done for instance by defining an appropriate notion of *center* of the *localization set* (the portion of dual space where the dual optimal solution is known to be), and calling the oracle on that point in order to shrink the size of the localization set as rapidly as possible. Although this approach is ideally alternative to the introduction of an explicit ST, the latest developments indicate that explicit stabilization also improves the performances of centers-based stabilized algorithms [2, 27].

In this paper, we study the practical effect of different variants of explicit STs on the performances of *Stabilized CG* (SCG) approaches. The aim of the paper is threefold:

- to briefly overview the issue of instability in CG and remind that a variety of stabilizing methods [12] can be implemented with relatively few modifications to existing CG algorithms using standard software tools;

- to prove by computational experiments that different forms of ST can have different and significant positive impacts in real-world, large-scale, challenging applications;

- to assess, by means of a computational study, the impact of the different choices in the ST (shape of the function, parameters), and their relationships with the quality of the initial dual estimates, on the overall effectiveness of the SCG approach.

We limit ourselves to the effect of stabilization on the standard CG approach, i.e., without any other form of centers-based stabilization. The rationale of this choice is that inserting an explicit ST is required anyway for optimal performances [2, 27], so developing guidelines about the best form of the ST is already a relevant issue. Besides, mixing two types of stabilization would make the contribution of each technique more difficult to ascertain, thus requiring a separate study.

The paper is organized as follows: in Section 2 the problem is stated, the standard CG approach is reviewed, its relationships with CP algorithms are underlined and the issues of the approach are discussed. In Section 3 we present a class of SCG approaches that avoid the instability problems by using an explicit ST in the dual and we discuss its primal counterparts. Then, in Section 4 we describe several STs that fit under the general SCG framework, discussing the relevant implementation details. Then, in Section 5 we present a set of computational experiments on, respectively, large-scale Multi-Depot Vehicle Scheduling (MDVS) problems (§5.1) and simultaneous Vehicle and Crew Scheduling (VCS) problems (§5.2), aimed at proving the effectiveness of the proposed approach in practice. Finally, in Section 6 we conduct an extensive computational comparison aimed at assessing the impact of the different choices in the ST, and their relationships with the quality of the initial dual estimates, on the overall effectiveness of the SCG approach; Section 7 summarizes our observations and draws some 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. Given a set X, $I_X(x) = 0$ if $x \in X$ (and $+\infty$ otherwise) is its *indicator function*. Given a problem (F) $\inf[\sup]\{f(x) : x \in X\}$, $v(F)$ denotes its optimal value; as usual, $X = \emptyset \Rightarrow v(F) = +\infty[-\infty]$.

2 Column Generation and Cutting Planes

2.1 The CG/CP algorithm

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

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

where $\mathcal{A}$ is the set of columns, each $a \in \mathcal{A}$ being a vector of $\mathbb{R}^m$, and $b \in \mathbb{R}^m$. In many applications, the number of columns is so large that they are impossible or impractical to handle at once; alternatively, the columns just cannot be determined a priori in practice. However, some structure exists in the set $\mathcal{A}$ so that optimization over its elements is possible; in particular, the *separation problem*

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

can be solved in relatively short time for all values of $\pi \in \mathbb{R}^m$.

In this case, (P) and (D) can be 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 and dual master*—or *restricted*—problems

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

The optimal solution $\hat{x}$ to $(P_\mathcal{B})$, completed with zeroes as needed, is feasible to (P), whereas the optimal solution $\hat{\pi}$ to $(D_\mathcal{B})$ may be unfeasible for (D); however, checking whether or not some dual constraint $\pi a \geq c_a$ for $a \in \mathcal{A}\backslash\mathcal{B}$ is violated can be accomplished by solving (P_π) with $\pi = \hat{\pi}$. If $v(P_{\hat{\pi}}) \leq 0$, then $\hat{\pi}$ is actually feasible for (D), and therefore $(\hat{x}, \hat{\pi})$ is a pair of primal and dual optimal solutions to (P) and (D), respectively. Otherwise, the optimal solution $\bar{a}$ of $(P_{\hat{\pi}})$ identifies the dual constraint $\pi\bar{a} \geq c_{\bar{a}}$ violated by $\hat{\pi}$ (equivalently, one column $\bar{a}$ with positive *reduced cost* $c_{\bar{a}} - \hat{\pi}\bar{a}$) that can be added to $\mathcal{B}$. This iterative process has to finitely terminate, at least if no column is ever removed from $\mathcal{B}$, because $\hat{\pi}$ must change at every iteration; the dual constraint corresponding to $\bar{a}$ *separates* $\hat{\pi}$ from the dual feasible region. Hence, solving (P) by CG is equivalent to solving (D) by Kelley's CP algorithm [21].

2.2 Special structures in (P)

In many relevant cases, the primal constraint matrix contains, possibly after a rescaling, a set of *convexity constraints*; that is, $\mathcal{A}$ can be partitioned into k disjoint subsets $\mathcal{A}_1, \ldots, \mathcal{A}_k$ such that k of the m rows of (P) correspond to the constraints $\sum_{a\in\mathcal{A}_h} x_a = 1$ for $h = 1, \ldots, k$. In particular, this is the case if (P) is the explicit representation of the *convexified relaxation* of a combinatorial optimization problem [24, 13]. When this happens, it is convenient to single out the dual variables η_h corresponding to the convexity constraints, i.e., to consider (D) written as

$$\begin{array}{ll} \min & \sum_{h=1}^{k} \eta_h + \pi b \\ & \eta_h \geq c_a - \pi a \quad a \in \mathcal{A}_h \quad h = 1, \ldots, k \end{array}$$

This corresponds to the fact that the separation problem decomposes into k separate optimization problems

$$(P_\pi^h) = \max\{ c_a - \pi a : a \in \mathcal{A}_h \} \ \ ,$$

one for each set $\mathcal{A}_h$. Another set $\mathcal{A}_0$ may need to be defined if some columns do not belong to any convexity constraint; these often correspond to rays of the feasible region of separation problems that are unbounded for $\pi = \hat{\pi}$, but we will avoid this complication for the sake of notational simplicity. Accordingly, in $(P_\mathcal{B})/(D_\mathcal{B})$ the set $\mathcal{B}$ of currently available columns is partitioned into the subsets $\mathcal{B}_1, \ldots, \mathcal{B}_k$. The usefulness of this form lies in the fact that, defining

$$\phi(\pi) = \pi b + \sum_{h=1}^{k} v(P_\pi^h) \quad ,$$

one has

$$v(P_\mathcal{B}) \leq v(P) \leq v(D) \leq \phi(\hat{\pi}) \ \ .$$

Hence, $\phi(\hat{\pi}) - v(P_\mathcal{B}) \leq \varepsilon$ ensures that $\hat{x}$ is a $\varepsilon-$optimal solution to (P), thereby allowing to early terminate the optimization process if ε is deemed small enough. More in general, improvements (decreases) of the ϕ-value can be taken as an indication that $\hat{\pi}$ is nearer to an optimal solution $\tilde{\pi}$ to (D), which may be very useful as discussed below.

2.3 Issues in the CG approach

The CG/CP approach in the above form is simple to describe and, given the availability of efficient and well-engineered LP solvers, straightforward to implement. However, several nontrivial issues have to be addressed.

Empty master problem In order to be well-defined, the CG method needs a starting set of columns such that $(P_\mathcal{B})$ has a finite optimal solution, that is, $(D_\mathcal{B})$ is bounded below. This is typically done as follows: assuming without loss of generality that $b \geq 0$, artificial columns of very high negative cost (*trippers*), each one covering exactly one of the constraints, are added to (P), yielding the modified pair or problems

$$(\bar{P}) \quad \begin{array}{ll} \max & \sum_{a \in \mathcal{A}} c_a x_a - Ms \\ & \sum_{a \in \mathcal{A}} a x_a + s = b \\ & x_a \geq 0 \ \ a \in \mathcal{A} \ , \ s \geq 0 \end{array} \qquad (\bar{D}) \quad \begin{array}{ll} \min & \pi b \\ & \pi a \geq c_a \quad a \in \mathcal{A} \\ & \pi \geq -M \end{array} \qquad (1)$$

The set of artificial variables s provides a convenient initial $\mathcal{B}$; they can be discarded as soon as they are found to be zero in the optimal solution of $(P_{\mathcal{B}})$.

Albeit simple to implement, such an initialization phase has issues. Roughly speaking, the quality of the columns generated by (P_{π}) can be expected to be related to the quality of π as an approximation of the optimal solution $\tilde{\pi}$ to (D); this ultimately boils down to obtaining reasonable estimates of the large price M, which however is difficult in practice. This usually results in $\hat{\pi}$ far off $\tilde{\pi}$ in the initial stages of the CG algorithm, which causes the generation of bad columns, ultimately slowing down the approach.

Instability The above discussion may have mislead the reader in believing that generating a good approximation of the dual optimal solution $\tilde{\pi}$ is enough to solve the problem; unfortunately, this is far from being true. The issue is that there is no control over the oracle; even if it is called at the very optimal point $\tilde{\pi}$, there is no guarantee that it returns the *whole* set of columns that are necessary to *prove* its optimality. Indeed, for several separation problems it may be difficult to generate but one solution (i.e., column) for each call. Thus, in order to be efficient, a CG algorithm, provided with knowledge about $\tilde{\pi}$, should sample the dual space near $\tilde{\pi}$, in order to force the subproblem to generate columns that have zero reduced cost in $\tilde{\pi}$.

However, this is not the case for the standard CG algorithm: even if a good approximation of $\tilde{\pi}$ is obtained at some iteration, the dual solution at the subsequent iteration may be arbitrarily far from optimal. In other words, the CG approach is almost completely unable of exploiting the fact that it has already reached a good dual solution in order to speed up the subsequent calculations; this is known as the *instability* of the approach, which is the main cause of its slow convergence rate on many practical problems.

One possibility is to introduce some mean to stabilize the sequence of dual iterates. If $\tilde{\pi}$ were actually known, one may simply restrict the dual iterates in a small region surrounding it, forcing the subproblem to generate columns that are almost optimal in $\tilde{\pi}$ and, consequently, efficiently accumulate the optimal set of columns. The practical effect of this idea is shown in Table 1. The first column reports the width of the hyperbox, centered on $\tilde{\pi}$, to which all dual iterates are restricted: the first row corresponds to the non-stabilized CG approach. Then, column “cpu” reports the total cpu time (in seconds), column “itr” reports the number of CG iterations, column “cols” reports the total number of columns generated by the subproblem and column "MP iters" reports the total number of simplex iterations performed to solve the master problem; the percentage of the corresponding measure w.r.t. that of the non-stabilized approach is shown in brackets.

width	cpu	itr	cols	MP iters
$+\infty$	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 with perfect dual information

Even with a large box width (200.0) there is a significant improvement in solution efficiency; the tighter the box, the more efficient the algorithm is. This suggests that properly limiting the changes in the dual variables may lead to substantial improvements in the performances; of course, the issue is that $\tilde{\pi}$ is in general *not* known, so one must account for the case where the current estimate of the dual optimal solution is not exact.

3 A Stabilized Column Generation approach

To stabilize the CG approach, we exploit some ideas originally developed in the field of nondifferentiable optimization; in particular, here we will rely upon the theory of [12] to introduce a general framework for Stabilized Column Generation (SCG) algorithms.

3.1 The stabilized master problems

In order to avoid large fluctuations of the dual multipliers, a *stability center* $\bar{\pi}$ is chosen as an estimate of $\tilde{\pi}$, and a proper convex explicit stabilizing term $\mathcal{D}_\tau : \mathbb{R}^m \to \mathbb{R} \cup \{+\infty\}$, dependent on some vector of parameters τ, is added to the objective function of $(D_\mathcal{B})$, thus yielding the *stabilized dual master problem*

$$(D_{\mathcal{B},\bar{\pi},\tau}) \qquad \begin{array}{ll} \min & \sum_{h=1}^k \eta_h + \pi b \ +\mathcal{D}_\tau(\pi - \bar{\pi}) \\ & \eta_h \geq c_a - \pi a \quad a \in \mathcal{A}_h \qquad\qquad h = 1, \ldots, k \end{array} \ . \qquad (2)$$

The optimal solution $\tilde{\pi}$ of (2) is then used in the separation problem. The ST $\mathcal{D}_\tau$ is meant to penalize points "too far" from $\bar{\pi}$; at a first reading, a norm-like function can be imagined there. As already mentioned in the introduction, other, more or less closely related, ways for stabilizing CP algorithms have been proposed [23, 2]; a thorough discussion of the relationships among them can be found in [18, 24].

Solving (2) is equivalent to solving a *generalized augmented Lagrangian* of $(P_{\mathcal{B}})$, using as augmenting function the *Fenchel's conjugate* of $\mathcal{D}_\tau$; in fact, the *Fenchel's dual* of (2) is

$$(P_{\mathcal{B},\bar{\pi},\tau}) \qquad \begin{array}{rl} \max & \sum_{a\in\mathcal{B}} c_a x_a - \bar{\pi}s - \mathcal{D}^*_\tau(s) \\ & \sum_{a\in\mathcal{B}} a x_a - s = b \\ & \sum_{a\in\mathcal{B}_1} x_a = 1 \qquad x_a \geq 0 \quad , \quad a \in \mathcal{B} \end{array} \quad . \tag{3}$$

For any convex function $f(x)$, its Fenchel's conjugate $f^*(z) = \sup_x \{ \; zx - f(x) \; \}$ characterizes the set of all vectors z that are support hyperplanes to the epigraph of f at some point. f^* is a closed convex function and enjoys several properties, for which the reader is referred e.g. to [18, 12]; here we just remind that from the definition one has $f^*(0) = -\inf_x \{ \; f(x) \; \}$. Using $\mathcal{D}_{[t]} = \frac{1}{2t}\| \cdot \|_2^2$, which gives $\mathcal{D}^*_{[t]} = \frac{1}{2}t\| \cdot \|_2^2$, one immediately recognizes in (3) the *augmented Lagrangian* of $(P_{\mathcal{B}})$, with both a first-order Lagrangian term, corresponding to the stability center $\bar{\pi}$, and a second-order Augmented Lagrangian term, corresponding to the stabilizing function $\mathcal{D}_\tau$, added to the objective function to penalize violation of the constraints, expressed by the slack variable s. In general, (3) is a *nonquadratic* augmented Lagrangian [33] of $(P_{\mathcal{B}})$. Note that $\mathcal{D}_\tau = 0$ corresponds to $\mathcal{D}^*_\tau = I_{\{0\}}$; that is, with no stabilization at all (3) collapses back to $(P_{\mathcal{B}})$. An appropriate choice of $\mathcal{D}^*_\tau$ will easily make (3) feasible even for "small" $\mathcal{B}$; indeed, comparing (1) with (3) shows that the trippers in the (1) are nothing but a (very coarse) stabilization device, only aimed at avoiding the extreme instability corresponding to an unbounded $(D_{\mathcal{B}})$.

We will denote by $(P_{\bar{\pi},\tau})$ and $(D_{\bar{\pi},\tau})$, respectively, the *stabilized primal and dual problems*, that is, (3) and (2) with $\mathcal{B} = \mathcal{A}$. Extending the above derivation to multiple subproblems' case is straightforward. Also, it is easy to extend the treatment to the case of inequality constraints in (P), which produce dual constraints $\pi \geq 0$; they simply correspond to a sign constraint $s \geq 0$ on the slack variables.

3.2 A Stabilized Column Generation framework

The stabilized master problems provide means for defining a general Stabilized Column Generation framework, such as that of Figure 1.

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

Figure 1: The general SCG algorithm

The algorithm generates at each iteration a *tentative point* $\hat{\pi}$ for the dual and a (possibly unfeasible) primal solution $\hat{x}$ by solving $(D_{\mathcal{B},\bar{\pi},\tau})/(P_{\mathcal{B},\bar{\pi},\tau})$. If $\hat{x}$ 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). More in general, one can stop whenever $\phi(\bar{\pi}) - \sum_{a\in\mathcal{B}}(c_a - \bar{\pi}a)\hat{x}_a - \bar{\pi}b \ (\geq 0)$ and $\| \sum_{a\in\mathcal{B}} a\hat{x}_a - b\|$ are both "small" numbers: this means that $\hat{x}$ is both almost optimal for the stabilized problem $(P_{\bar{\pi},\tau})$ (with *all* columns) and almost feasible for (P), and therefore a good solution for (P) if the slight unfeasibilty can be neglected. Otherwise, the new columns generated using $\hat{\pi}$ are added to $\mathcal{B}$. If $\phi(\hat{\pi})$ is "substantially lower" than $\phi(\bar{\pi})$, then it is worth to update the stability center: this is called a *Serious Step (SS)*. Otherwise $\bar{\pi}$ is not changed, and we rely on the columns added to $\mathcal{B}$ for producing, at the next iteration, a better tentative point $\hat{\pi}$: 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, then it may be worth to lessen the penalty for moving far from $\bar{\pi}$. Conversely, a *NS* might be due to an insufficient stabilization, thereby suggesting to increase the penalty. The algorithm can be shown to finitely converge to a pair $(\tilde{\pi}, \tilde{x})$ of optimal solutions to (D) and (P), respectively, under a number of different hypotheses; the interested reader is referred to [12].

Note that when no convexity constraints are present in (P), the ϕ-value is not available and therefore $\bar{\pi}$ can only be updated when the stabilized primal and dual problems are solved to optimality. In this case the SCG algorithm reduces to a (nonquadratic) version of the *Proximal Point* (PP) approach [30, 33] applied to the solution of (D). Indeed, the *Bundle-type* SCG algorithm can be seen [12] as a PP approach where the stabilized dual problem $(D_{\bar{\pi},\tau})$ is in turn iteratively solved by CP, with an *early termination rule* that allows to interrupt the inner solution process, and therefore update $\bar{\pi}$, (much) before having actually solved $(D_{\bar{\pi},\tau})$ to optimality. This suggests that adding a redundant convexity constraint to (P), in order to have the corresponding dual variable η and therefore the ϕ-value defined, may be beneficial to the overall efficiency of the CG approach; this is confirmed by the results in §5.1 and §5.2.

4 Stabilizing functions

The SCG approach is largely independent on the choice of the stabilizing term $\mathcal{D}_\tau$: stabilizing $(D_\mathcal{B})$ corresponds to allowing the constraints of $(P_\mathcal{B})$ to be violated, but at a cost. Thus, the actual form of the problem to be solved only depends on $\mathcal{D}^*_\tau(s)$, allowing for several different STs to be tested at relatively low cost in the same environment.

A number of alternatives have been proposed in the literature for $\mathcal{D}_\tau$ or, equivalently, for the (primal) *penalty term* $\mathcal{D}^*_\tau$. In all cases, $\mathcal{D}_\tau$ is separable and therefore so is $\mathcal{D}^*_\tau$, that is

$$\mathcal{D}_\tau(d) = \textstyle\sum_{i=1}^m \Psi_{\tau[i]}(d_i) \qquad\qquad \mathcal{D}^*_\tau(s) = \textstyle\sum_{i=1}^m \Psi^*_{\tau[i]}(s_i)$$

where both $d = \pi - \bar{\pi}$ and the slack variables s take values in $\mathbb{R}^m$, and $\Psi_t : \mathbb{R} \to \mathbb{R} \cup \{+\infty\}$ is a family of functions depending on a subvector t of the parameters vector τ.

The boxstep method The boxstep method [26] uses $\Psi_t = I_{[-t,t]}$, that is, it establishes a trust region of radius τ around the stability center. In the primal viewpoint this corresponds to $\Psi^*_t = t|\cdot|$, i.e., to a linear penalty. Note that the absolute value forces one to split the vector of slack variables into $s = s^+ - s^-$ with $s^+ \geq 0$ and $s^- \geq 0$. Thus, the boxstep method is a simple modification of (1); however, in this case the cost of the artificial columns need not be very high, as the iterative process that changes $\bar{\pi}$ will eventually drive the dual sequence to a point where any chosen cost is large enough. On the other hand, since the sign of $\tilde{\pi} - \bar{\pi}$ is unknown, both sides must be penalized. Yet, the boxstep method have shown lackluster performances in practice due to a difficult choice of the parameters $\tau[i] = t_i$, that is, the cost of the trippers. The basic observation is that if t_i is "small" then one of the corresponding trippers $s_i^\pm$ will be in the primal optimal solution, and therefore $\bar{\pi} = \pm t_i$; in other words, the estimate of the (corresponding entry of the) dual optimal solution is only dependent on the guess t_i and owes nothing to the rest of the data of the problem. Conversely, if t_i is "large" then $s_i^+ = s_i^- = 0$ and no stabilization at all is achieved. Thus, typically either t_i is too large and little stabilization is achieved, or t_i is too small and very short steps are performed in the dual space, unduly slowing down convergence.

The dual boxstep method The method of [20] uses $\Psi_t^* = I_{[-1/t,1/t]}$, and therefore $\Psi = |\cdot|/t$. Because of nonsmoothness of $\mathcal{D}_\tau$ at 0, the algorithm requires a large enough penalty to converge [12]; since the primal penalty is a trust region, its radius has to be shrank in order to ensure that eventually s will converge to zero. Also, boundedness of the dual master problem is not granted. This algorithm has never been shown to be efficient in practice, and there is hardly reason to prefer it to the boxstep method.

The proximal bundle method The proximal bundle method [18, 32] uses $\tau = [t]$ (although scaled variants have sometimes been proposed [3]) and $\Psi_t = \frac{1}{2t}(\cdot)^2 \Rightarrow \Psi_t^* = \frac{1}{2}t(\cdot)^2$. Therefore, both the primal and dual master problems are convex quadratic problems with separable quadratic objective function. Since both $\mathcal{D}_\tau$ and $\mathcal{D}_\tau^*$ are smooth at 0, the algorithm will converge even for vanishing t and using "extreme" aggregation [12]; also, the dual master problem is always bounded. Bundle methods have proven efficient in several applications, even directly related to CG approaches, not least due to the availability of specialized algorithms for solving the master problems [11]; see e.g. [13, 24, 6] for some review.

The linear-quadratic penalty function In [28], the linear-quadratic ST

$$\Psi_{t,\varepsilon}^*(s) = t \begin{cases} s^2/\varepsilon & \text{if } s \in [-\varepsilon, \varepsilon] \\ |s| & \text{otherwise} \end{cases} \qquad \Psi_{t,\varepsilon}(d) = \begin{cases} \frac{\varepsilon}{4t}d^2 & \text{if } d \in [-t, t] \\ +\infty & \text{otherwise} \end{cases}$$

is proposed as a smooth approximation of the nonsmooth exact penalty function $t|\cdot|$ for $(P_{\mathcal{B},\bar{\pi},\tau})$. This can be seen as a modification of the boxstep method where nonsmoothness at zero of $\mathcal{D}_\tau^*$ is avoided, keeping all other positive aspects: convergence for vanishing τ, easy aggregation, boundedness of the dual master problem. However, this smoothing comes at the cost of a quadratic master problem similar to that of the proximal bundle approach, while, since ε is assumed to be small, the stabilizing effect should not be too different, qualitatively speaking, from that of the boxstep approach. It should also be remarked that the approach of [28] is a pure penalty method, i.e., the concept of stability center is ignored ($\bar{\pi} = 0$ all along) and convergence is obtained by properly managing t and ε.

k-piecewise linear penalty function The advantage of the quadratic ST over the linear ones can be thought to be that it has "infinitely many different pieces"; this somewhat avoids the need for a very accurate tuning of the tripper costs in order to attain both stabilization and a dual solution $\bar{\pi}$ that actually takes into account the problem's data. Clearly, a similar effect can be obtained by a piecewise-linear function with more than one piece. Reasonable requirements to any stabilizing function are that the steepest slope must be such as to guarantee boundedness of $(D_{\mathcal{B},\bar{\pi},\tau})$ (cf. M in (1)), and that $\mathcal{D}_\tau$ should be smooth at 0 (that is, $\mathcal{D}_\tau^*$ should be strictly convex at zero) so that convergence can be attained even for fixed or vanishing τ [12], and a primal optimal solution can be efficiently recovered [5]. A first attempt in this direction has been made in [10], where a 3-piecewise function is proposed that somewhat merges [26] with [20]: a linear stabilization is used, but only outside of a small region where violation of the constraints is not penalized. However, this may suffer from the same shortcomings of the Boxstep method, in that the penalties must be high to ensure boundedness (and, more in general, to avoid the same unstable behavior as CG), so only small moves in the non-penalized region may ultimately be performed, slowing down convergence. All this suggests to use a 5-piecewise stabilizing function with two sets of penalties: "large" ones to ensure stability, and "small" ones to allow for significant changes in the dual variables, i.e.,

$$\Psi_t(d) = \begin{cases} -(\zeta^- + \varepsilon^-)(d + \Gamma^-) - \zeta^-\Delta^- & \text{if} \quad d \leq -(\Gamma^- + \Delta^-) \\ -\varepsilon^-(d - \Delta^-) & \text{if} \quad -(\Gamma^- + \Delta^-) \leq d \leq -\Delta^- \\ 0 & \text{if} \quad -\Delta^- \leq d \leq \Delta^+ \\ +\varepsilon^+(d - \Delta^+) & \text{if} \quad \Delta^+ \leq d \leq (\Delta^+ + \Gamma^+) \\ +(\varepsilon^+ + \zeta^+)(d - \Gamma^+) + \zeta^+\Delta^+ & \text{if} \quad (\Delta^+ + \Gamma^+) \leq d \end{cases} \tag{4}$$

whose corresponding 6-piecewise primal penalty is

$$\Psi_t^*(s) = \begin{cases} +\infty & \text{if} \quad s < -(\zeta^- + \varepsilon^-) \\ -(\Gamma^- + \Delta^-)s - \Gamma^- \varepsilon^- & \text{if} \quad -(\zeta^- + \varepsilon^-) \leq s \leq -\varepsilon^- \\ -\Delta^- s & \text{if} \quad -\varepsilon^- \leq s \leq 0 \\ +\Delta^+ s & \text{if} \quad 0 \leq s \leq \varepsilon^+ \\ +(\Gamma^+ + \Delta^+)s + \Gamma^+ \varepsilon^+ & \text{if} \quad \varepsilon^+ \leq s \leq (\zeta^+ + \varepsilon^+) \\ +\infty & \text{if} \quad s > (\zeta^+ + \varepsilon^+) \end{cases} \tag{5}$$

where $t = [\zeta^\pm, \varepsilon^\pm, \Gamma^\pm, \Delta^\pm]$. This corresponds to defining $s = s_2^- + s_1^- - s_1^+ - s_2^+$, with

$$\zeta^+ \geq s_2^+ \geq 0 \qquad \varepsilon^+ \geq s_1^+ \geq 0 \qquad \varepsilon^- \geq s_1^- \geq 0 \qquad \zeta^- \geq s_2^- \geq 0$$

in the primal master problem, with objective function

$$(\bar{\pi} - \Delta^- - \Gamma^-)s_2^- + (\bar{\pi} - \Delta^-)s_1^- - (\bar{\pi} + \Delta^+)s_1^+ - (\bar{\pi} + \Delta^+ + \Gamma^+)s_2^+ \quad .$$

Hence, the primal master problem is still a linear program with the same number of constraints and a linear number of new variables. Clearly, this generalizes both (1) and all previous piecewise-linear STs; with a proper choice of the constants, $(P_{\mathcal{B},\bar{\pi},\tau})$ can be assumed to always be feasible. Piecewise-linear STs with more pieces can be used, at the cost of introducing more slack variables and therefore increasing the size of the master problem. We have found 5-pieces to often offer the best compromise between increased stabilization effect and increased size of the master problems, as the following paragraphs will show.

5 Practical impact of stabilization

We first report some experiments on large-scale practical problems, aimed at proving that different forms of stabilization can indeed have a significant positive impact in real-world, challenging applications. These results have been obtained using a customized version of the state-of-the-art, commercial `GenCol` code [9].

5.1 The Multiple-Depot Vehicle Scheduling problem

The Multiple-Depot Vehicle Scheduling problem (MDVS) can be described as follows. A set of p tasks have to be covered by vehicles, each with a maximum capacity, available at d different depots. Vehicles can be seen as following a path (cycle) in a *compatibility network*, starting and ending at the same depot. Using a binary variable for each feasible path, of which there are exponentially many, the problem can be formulated as a very-large-scale Set Covering (SC) problem with $p + d$ constraints. Due to its large size, MDVS is usually solved by branch-and-price where linear relaxations are solved by CG [29, 17]; given the set of multipliers produced by the master problem, columns are generated by solving d shortest path problems, one for each depot, on the compatibility network. We are interested in stabilizing the CG process at the root node; the same process, possibly adapted, may then be used for any other branch-and-price node.

The test problems Test problem sets are generated following the scheme of [7]. 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. The instances, described in Table 2, are the same used in [4]; for each instance, the number p of tasks, the number d of depots, and the number a (in units of one million) of arcs of the compatibility network are reported.

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
p	400	400	400	400	800	800	1000	1000	1200	1200
d	4	4	4	4	4	4	5	5	5	4
a	0.21	0.21	0.21	0.20	0.76	0.82	1.30	0.97	1.50	1.10

Table 2: MDVS: instances' characteristics

Initialization All stabilization approaches that are tested use the same initialization procedure; by performing a *depot aggregation procedure* (see [5] for more details), an instance of the Single Depot Vehicle Scheduling problem (SDVS) can be constructed which approximates the MDVS instance at hand. SDVS is a minimum cost flow problem over the compatibility network, and therefore can be solved in polynomial time. Its primal optimal solution may be used to compute an initial integer feasible solution for MDVS as well as an upper bound on the integer optimal value, while 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 $\bar{\pi}$ in the algorithm.

Pure Proximal approach Experiments with a Pure Proximal (PP) approach on these instances have already been performed in [4]; however, no direct comparison with the use of a 3-piecewise ST, nor with a Bundle-type approach, was attempted there. Since there are many possibilities for the parameters' setting strategy, we used an improved version of the PP strategy found to be the best in [4]. The ST 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 intial values equal to 1 (the right-hand side of stabilized constraints), which ensures boundedness of the master problem à-la (1). Since the problem contains no explicit convexity constraint, Serious Steps are performed only when no positive reduced column is generated, i.e., optimality of $(P_{\bar{\pi},\tau})$ 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. Both a 3-pieces and a 5-pieces ST are tested; the 3-pieces function is obtained from the 5-pieces one by simply removing the small penalties.

Bundle-type approach When fixed costs are sufficiently large, the number of vehicles $\bar{b}$ obtained by solving the SDVS problem in the initialization phase is the minimum possible number of vehicles; the instances considered here use a large enough fixed cost to ensure this property. Thus, a redundant constraint ensuring that at least $\bar{b}$ vehicles are used can be safely added to the problem; this is not meant to serve as a cutting plane in the sense of Branch&Cut methods—indeed, in itself it typically does not impact the master problem solution—but rather to allow defining a proper objective function ϕ, and therefore to use a Bundle-type approach, where the stability center is updated (much) before optimality of CG applied to the stabilized problem is reached. For the rest, the same parameters strategy used in the PP case is adopted here. While different strategies may help in improving the performances of the Bundle-type approach, we found this simple one to be already quite effective; furthermore, this ensures a fair comparison where the different efficiency of the different approaches cannot be due to different strategies for updating the τ parameters.

Results Results are given in Table 3 for standard column generation (CG), the pure proximal approach with 3-pieces and 5-pieces ST (PP-3 and PP-5, respectively), and the Bundle-type approach (BP). In this Table, rows labeled "cpu", "mp", and "itr" report respectively the total and master problem computing times (in seconds) and the number of CG iterations needed to reach optimality.

Pb		p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
cpu	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
	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-3	82	92	104	75	181	129	134	145	144	189
	PP-5	47	47	49	45	93	64	98	83	86	150
	BP	37	43	44	36	57	53	59	49	51	101
mp	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
	BP	9.9	13.7	14.9	10.1	100.2	70.0	329.3	206.3	334.2	982.5

Table 3: Computational results for MDVS problems

Analyzing the results leads to the following conclusions:

- all stabilized approaches are substantially better that the standard CG, in terms of computation time, on all problems; this is mainly due to the reduction of the number of iterations, a clear sign that stabilization do actually improve the convergence of the dual iterates;
- both PP algorithms improve standard CG substantially; however, PP-5 clearly outperforms PP-3 on all aspects, especially total computing time and iterations number, while in turn being outperformed by BP;
- the improvement is more uniform among PP-5 and BP for small size problems, but as the size grows BP becomes better and better; this is probably due to the fact that *for larger problems the initial dual solution is worse*, and the good performances of PP are more dependent on the availability of a very good initial dual estimate to diminish the total number of (costly) updates of $\bar{\pi}$, while the cost for updating $\bar{\pi}$ is substantially less for BP;
- BP has a slightly higher *average master problem computation time per iteration* than PP, especially for larger instances; this may be explained by higher master problem reoptimization costs due to a larger number of Serious Steps.

Thus, the larger size of the master problem associated to a 5-pieces ST does not increase too much the master problem cost, at least not enough to vanish the effect of the better stabilization achieved w.r.t. 3-pieces only. Yet, a 5-pieces ST is clearly more costly than a 3-pieces one. A possible remedy, when m is too large, is to penalize only a subset of the rows, i.e., to only partially stabilize the dual vector π. Identifying the "most important" dual variables, such as those with largest multiplier, or those whose multiplier varies more wildly, can help in choosing an adequate subset of rows to be penalized. Alternatively, one may choose the number of pieces dynamically, and independently, for each dual 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 ST are not needed, and the corresponding 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-pieces function that allows small number of stabilization variables and ensures primal feasibility. We have experimented with this *5-then-3* strategy, and although we don't report full results for space reasons, these seem to be able to further improve the performances of the SCG approach by about 10%–20%, although the improvement is larger for smaller instances, and tends to diminish as the size of the instance grows.

5.2 The Vehicle and Crew Scheduling problem

The simultaneous Vehicle and Crew Scheduling problem (VCS), described in [16], 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, ...). This problem can be formulated, similarly to MDVS, as a very-large-scale SC problem where each column is associated to a proper path in a suitably defined network.

However, the need of expressing the time at which events take place, in order to synchronize vehicles and crews, makes the separation subproblem much more difficult to solve than in the MDVS case; when formulated as a Constrained Shortest Path (CSP) problem using up to 7 resources, its solution can be 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 a unique network in which CSPs with many resources need to be solved, 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) CSP 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 *when optimality has to be proved*, and therefore all the subnetworks have to be solved. It must be remarked at this point that, because not all the subproblems are solved at every CG iteration, the actual value of ϕ is not known, and therefore the standard descent rule of Bundle methods cannot be directly used. In our implementation we simply moved the stability center whenever the decrease *for the evaluated components alone* was significant; a theoretical study of conditions guaranteeing convergence of CG approaches with partial solution of the separation problem can be found in [25].

The test problems We use a set of 7 instances taken from a real-world urban bus scheduling problem. They are named pm, where m is the total number of covering constraints in the master problem. Their characteristics are presented in Table 4, where p, k, $|N|$ and $|A|$ are respectively the total number of constraints in the master problem, the number of subnetworks, and the size (number of nodes and arcs) of each subnetwork.

	p199	p204	p206	p262	p315	p344	p463
p	1096	1123	1134	1442	1734	1893	2547
k	822	919	835	973	1039	1090	1238
$\|N\|$	1528	1577	1569	1908	2180	2335	2887
$\|A\|$	3653	3839	3861	4980	6492	7210	9965

Table 4: VCS: instances' characteristics

The algorithms We tested different stabilized CG approaches for the VCS problem. Somewhat surprisingly, a PP stabilized CG approach turned out to be *worse* than the non-stabilized CG. This is due to the fact that a PP stabilized algorithm needs to *optimally solve the subproblem many times, each time that optimality of the stabilized problem has to be proved.* 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 PP approach competitive. Instead, a Bundle-type approach, that does not need to optimally solve the stabilized problem except at the very end, was found to be competitive.

For implementing the Bundle-type 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 the stabilizing term is decreased using proper simple rules, while after a Null Step the stabilizing term is kept unchanged. Note that since each dual variable must be in $[-1, 1]$, this property is preserved while updating the stability center.

Results Results of the experiments on VCS are given in Table 5. The meaning of the rows in this Table is the same as in Table 3, except that running times are in minutes.

		p199	p204	p206	p262	p315	p344	p463
cpu	CG	26	26	30	68	142	238	662
	BP	12	13	14	40	73	163	511
itr	CG	167	129	245	263	239	303	382
	BP	116	119	173	160	213	201	333
mp	CG	13	9	14	35	43	90	273
	BP	3	3	4	7	19	20	93

Table 5: Computational results for VCS

The results show that, as expected, stabilization reduces the number of CG iterations. Also, the use of a Bundle-type approach, as opposed to a PP one, allows this reduction in iteration time to directly translate into a reduction of the total computing time. This happens even if the subproblem computing time is increased, as it is the case for the largest problem p463, for which CG requires $662-273 = 389$ minutes, while BP requires $511-93 = 418$ minutes. Thus, the Bundle-type approach once again proves to be the best performing stabilization procedure among those tested in this paper.

6 Assessing the impact of stabilizing term choices

We now present a computational study aimed at more precisely assessing the impact of the different choices in the ST (shape of the function, parameters), and their relationships with the quality of the initial dual estimates, on the overall effectiveness of the SCG approach. The SCG algorithm uses a Bundle-type approach where the ST is symmetrical, as in previous sections. To avoid any artifact due to the dynamic updating of the ST parameters, the ST is kept unchanged both for Null and Serious steps. Updating $\bar{\pi}$ is done whenever no columns are generated or 10^{-4} relative improvement of lower bound value occurs.

Instances For our study we have selected one "easy" and two "difficult" classes of instances. The easy ones are the MDVS instances described in §5.1; for these, optimization is stopped whenever a relative gap $\leq 10^{-7}$ is reached, or the maximum number of 700 CG iterations is reached. The first group of difficult instances is the Long-Horizon Multiple-Depot Scheduling (LH-MDVS) benchmark used in [1]. These are randomly generated MDVS instances where the horizon is extended from one day up to a whole week; as a consequence the routes are longer, and the columns in an optimal solution have many ones, which may make the CG process very inefficient [1]. 14 instances are considered, 2 for each horizon length from 1 to 7 days; for the results they are arranged into three groups, "lh1" (4 instances) with horizons 1 and 2 days, "lh2" (6 instances) with horizons 3, 4, and 5 days, and "lh3" (4 instances) with horizons 6 and 7 days. For these, optimization is stopped whenever a relative gap $\leq 10^{-4}$ or the maximum number of 1500 CG iterations is reached. Finally, we examine Urban Bus Scheduling (UBS) instances [7]. These are randomly generated in the same way as MDVS instances with one additional resource constraint that need to be satisfied by routes, which makes them more difficult to solve than ordinary MDVS instances, albeit less than LH-MDVS ones. We consider two instances for each number of tasks in $\{500, 700, 1000, 1200, 1500, 2000\}$; they are denoted "u$nsi$", where n is the number of tasks (divided by 100) and i is the seed number used to initialize the random number generator. The same stopping criteria as for LH-MDVS are used.

Stabilizing terms For our experiments, we compared quadratic STs (the Proximal Bundle method) and piecewise-linear STs with, respectively, one piece (Boxstep), three pieces [10] and five pieces [4]. A particular effort has been made to *compare different functions with analogous setting of the parameters*, in order to be able to separate the role of the "shape" of the function from that of the parameters defining its "steepness". Thus, the ST have been constructed as follows:

- The quadratic ST (Q) only depends on one single parameter t. We defined five possible values for t, of the form $t = 10^j$ for $j \in T = \{7, 5, 3, 2, 1\}$.
- Similarly, the Boxstep ST (1P) only depends on the single parameter Δ. We defined the five possible values $\{1000, 500, 100, 10, 1\}$ for Δ. Note that t and Δ have qualitatively the same behavior: the larger they are, the "less stabilized" the dual iterates are.

- The 3-pieces linear ST (3P) is built using the values of Δ as interval widths, and computing the slope parameter ε so that the ST is tangent to the corresponding quadratic ST; the values of t, Δ, and ε therefore satisfy $t\varepsilon = 2\Delta$.
- Finally, 5-pieces linear ST (5P) are built from the 3-pieces ones as follows. For each value of (Δ, ε), the interval (right or left) is split into two sub-intervals with equal width $\Delta/2$. The slope parameters are computed in a unique way: if $\varepsilon > 1.0$ for the 3-pieces ST, then the outer slope parameter takes value 1.0 (actually the absolute value of the right-hand side b_i) and the inner slope parameter takes value $(\varepsilon - 1.0)$, otherwise both slopes take the value $\varepsilon/2$.

Thus, there are 5 Q algorithms, 5 1P algorithms, and as much as 25 3P and 5P algorithms. However, not all pairs of parameters actually make sense, as several combinations lead to values of ε that are either "too small" or "too large". This is described in Table 6, where:

- cases where $\varepsilon < 10^{-5}$ are marked with "$(*)$", and are dropped due to possible numerical problems;
- cases where $\varepsilon < 2 \cdot 10^{-3}$ are marked with "$(\circ)$", and are dropped, too, since the tests showed that for those values the behaviour of the corresponding SCG algorithm is very close to the behaviour of the standard CG algoritm;
- for every Δ with several $\varepsilon \geq 1.0$ (marked with "$(\square)$"), we consider only one with $\varepsilon = 1.1$, since all right-hand sides of constraints to be stabilized are equal to 1.

$\Delta \backslash t$	10^7	10^5	10^3	10^2	10
1000	$2 \cdot 10^{-4}$ $(\circ)$	$2 \cdot 10^{-2}$	2 $(\square)$	20 $(\square)$	$2 \cdot 10^2$ $(\square)$
500	10^{-4} $(\circ)$	10^{-2}	1 $(\square)$	10 $(\square)$	10^2 $(\square)$
100	$2 \cdot 10^{-5}$ $(\circ)$	$2 \cdot 10^{-3}$	$2 \cdot 10^{-1}$	2 $(\square)$	20 $(\square)$
10	$2 \cdot 10^{-6}$ $(*)$	$2 \cdot 10^{-4}$ $(\circ)$	$2 \cdot 10^{-2}$	$2 \cdot 10^{-1}$	2 $(\square)$
1	$2 \cdot 10^{-7}$ $(*)$	$2 \cdot 10^{-5}$ $(\circ)$	$2 \cdot 10^{-3}$	$2 \cdot 10^{-2}$	$2 \cdot 10^{-1}$

Table 6: Quadratic-Linear correspondance

Initial dual points In order to test the effect of the availability of good dual information on the performances of the SCG algorithm, we also generated, starting from the known dual optimal solution $\tilde{\pi}$, perturbed dual information, to be used as the starting point, as follows:

- **α-points:** the initial points have the form $\alpha\tilde{\pi}$ for $\alpha \in \{0.9, 0.75, 0.5, 0.25, 0.0\}$, i.e., are a convex combination between the optimal dual solution $\tilde{\pi}$ and the all-0 dual solution (which is feasible) that is typically used when no dual information is available.

- **Random points:** the initial points are chosen uniformly at random in a hyper-cube centered at $\tilde{\pi}$, and so that their distance in the $\|.\|_\infty$ norm from $\tilde{\pi}$ is comprised into a given interval $[\delta^1, \delta^2]$ for the three possible choices of (δ^1, δ^2) in $\{(0, 0.5), (0, 1), (0.5, 1)\}$.

Note that α-points are likely to be better dual information than random points, since they are collinear to the true optimal dual solution $\tilde{\pi}$.

6.1 MDVS: using initial dual α-points

First we consider the results obtained using initial dual α-points for different values of α. Table 7 compares k-pieces linear STs among them. Each tested variant corresponds to a column, whose headers indicate the shape of the ST (1P, 3P, or 5P) and the values of Δ and t (where applicable). This Table is divided in two parts:

- the topmost part reports results for each of the MDVS instances *averaged w.r.t. the five possible values of* α; for each algorithm, both the mean and the standard deviation of the total number of CG iterations needed to reach optimality is reported;

- the bottom part of the table reports results for each of the five possible values of α *averaged w.r.t. the 10 possible MDVS instances*; for each algorithm, the mean of the total number of CG iterations needed to reach optimality is reported.

This table is arranged for decreasing values of Δ, i.e., for increasing strength of the stabilizing term; for each value, all k-pieces STs are compared, with different values of t where applicable, with again t ordered in decreasing sense. Thus, roughly speaking, the penalties become stronger going from left to right: the leftmost part of this Table corresponds to "weak" penalties and the rightmost part corresponds to "strong" penalties.

Table 7 contains a wealth of information, that can be summarized as follows:

- Already weak penalties produce significantly better results than standard CG; this probably means that the large interval value ($\Delta = 1000$)

Δ			1000					500					100					10							1				
alg		CG	1P	3P		5P		1P	3P		5P		1P	3P		5P		1P	3P			5P			1P	3P		5P	
t				10^5	10^3	10^5	10^3		10^5	10^3	10^5	10^3		10^3	10^2	10^3	10^2		10^3	10^2	10	10^3	10^2	10		10^2	10	10^2	10
p1	avg	134	85	110	79	110	72	80	118	72	120	65	122	74	58	75	58	408	115	73	110	113	73	111	255	123	96	126	97
	dev		0.08	0.07	0.06	0.02	0.03	0.13	0.09	0.02	0.1	0.07	0.45	0.04	0.06	0.04	0.13	0.45	0.05	0.06	0.13	0.07	0.04	0.05	0.36	0.04	0.03	0.02	0.04
p2	avg	151	92	117	92	114	84	84	115	83	111	77	119	84	81	80	83	387	118	88	130	112	104	145	306	133	110	126	107
	dev		0.08	0.05	0.03	0.04	0.04	0.11	0.03	0.04	0.04	0.17	0.49	0.05	0.19	0.05	0.12	0.39	0.04	0.17	0.14	0.09	0.04	0.12	0.36	0.02	0.08	0.04	0.07
p3	avg	183	117	162	114	161	99	109	163	96	158	89	129	103	80	94	81	473	156	97	122	150	106	150	442	168	122	166	125
	dev		0.08	0.04	0.08	0.03	0.05	0.11	0.05	0.04	0.03	0.04	0.47	0.04	0.05	0.03	0.23	0.44	0.04	0.05	0.17	0.02	0.06	0.09	0.41	0.02	0.03	0.03	0.05
p4	avg	137	83	124	83	115	80	79	123	76	120	76	115	81	75	78	74	396	122	79	99	128	81	111	271	131	101	127	100
	dev		0.05	0.07	0.07	0.04	0.04	0.17	0.05	0.08	0.04	0.16	0.45	0.06	0.13	0.05	0.13	0.38	0.03	0.03	0.13	0.05	0.05	0.06	0.36	0.05	0.07	0.05	0.04
p5	avg	592	343	481	256	468	222	260	498	223	493	194	200	274	241	278	279	544	474	346	388	475	332	406	620	481	336	489	335
	dev		0.07	0.04	0.05	0.05	0.05	0.03	0.03	0.04	0.05	0.08	0.35	0.13	0.32	0.09	0.36	0.36	0.04	0.17	0.2	0.06	0.16	0.17	0.29	0.05	0.16	0.01	0.14
p6	avg	505	195	394	177	384	150	158	423	149	422	126	151	177	125	181	121	556	368	192	205	369	194	239	599	386	203	377	205
	dev		0.04	0.02	0.06	0.02	0.05	0.08	0.03	0.07	0.05	0.07	0.33	0.08	0.19	0.11	0.2	0.37	0.02	0.03	0.29	0.06	0.03	0.17	0.38	0.03	0.05	0.04	0.04
p7	avg	287	152	271	125	275	110	125	284	110	281	107	181	164	141	160	159	554	289	185	244	275	183	286	601	286	204	282	193
	dev		0.09	0.04	0.06	0.04	0.05	0.11	0.04	0.04	0.03	0.17	0.49	0.17	0.35	0.15	0.4	0.38	0.08	0.04	0.09	0.08	0.06	0.11	0.36	0.11	0.08	0.11	0.03
p8	avg	192	126	172	108	178	96	107	190	97	187	94	184	139	115	142	118	466	183	150	174	181	152	179	508	189	156	188	154
	dev		0.09	0.05	0.06	0.04	0.08	0.12	0.08	0.06	0.06	0.14	0.54	0.21	0.28	0.16	0.27	0.44	0.08	0.09	0.21	0.1	0.1	0.2	0.25	0.04	0.07	0.06	0.12
p9	avg	258	161	224	127	215	109	140	225	110	223	101	179	130	140	130	170	505	222	178	235	219	179	233	531	242	178	232	183
	dev		0.09	0.07	0.04	0.05	0.04	0.14	0.08	0.03	0.04	0.12	0.57	0.1	0.32	0.11	0.37	0.46	0.11	0.13	0.25	0.06	0.13	0.28	0.03	0.07	0.14	0.04	0.12
p10	avg	298	214	244	144	238	120	177	242	128	257	120	254	146	163	147	207	566	232	160	302	231	151	329	655	244	176	246	176
	dev		0.21	0.1	0.04	0.02	0.06	0.19	0.05	0.02	0.06	0.14	0.61	0.07	0.36	0.05	0.31	0.35	0.13	0.11	0.14	0.08	0.11	0.16	0.04	0.13	0.06	0.05	0.03
α																													
0.9	avg	274	139	227	133	226	113	112	236	113	238	91	68	122	84	119	83	214	218	139	155	215	139	179	541	231	155	235	156
0.75	avg	274	157	228	130	224	109	130	236	111	236	99	100	128	100	132	112	380	221	148	187	222	147	205	305	237	160	231	161
0.5	avg	274	167	231	126	225	115	132	240	115	232	105	174	137	125	141	149	561	225	154	217	230	163	223	516	239	167	232	168
0.25	avg	274	161	227	131	225	118	142	237	116	240	109	229	144	147	146	161	650	230	160	215	228	166	240	518	237	175	238	171
0	avg	274	159	236	133	230	117	145	242	118	240	120	247	155	155	146	170	623	245	173	232	233	163	248	514	248	185	245	181

Table 7: Comparing linear STs using $\alpha-$initial dual points

is not actually that large, considering that the penalty becomes $+\infty$ outside the box.

- Initially, the performances improve when Δ decreases, and is best for medium stabilizations; however, when Δ further decreases the performances degrade, ultimately becoming (much worse) than these of standard CG, meaning that too strong a stabilization forces too many steps to be performed.

- Something similar happens for t: for good values of Δ, a larger t (for 3P and 5P) is typically worse than a smaller one. For $\Delta = 10$, where three different values of t are available, the middle value is the best, indicating again that a good compromise value has to be found.

- Boxstep (1P) profits more from good initial dual points, achieving the overall best performance for $\alpha = 0.9$ and $\Delta = 100$; however its performance is strongly dependent on α, and quickly degrades as the initial point get worse. Indeed, 3P and especially 5P are much more robust: the standard deviation is usually much smaller. This is not always true, in particular for strong penalties where 1P behaves very badly, which means that it consistently behaves so; indeed, 3P and 5P are much less affected by the parameters values being extremal, i.e. too weak or too strong.

- With only one exception (p5 for $\Delta = 100$), for each value of Δ there is one value of t such that either 3P or 5P outperforms 1P. Most of the time 5P gives the best performance, and indeed it is the overall fastest algorithm for all values of α except the extreme ones. The improvement of 5P over 3P is somewhat smaller than that seen in §5; this is likely to be due to our "artificial" choice of the constants, intended to mimic the quadratic penalty rather than to be suited to the instances at hand, indicating that the extra flexibility of 5P requires some effort to be completely exploited.

Table 8 compares in a similar fashion the k-pieces ST and the quadratic one; we focus on the values of Δ which provide the best results, hence some of the worst performing cases of the linear STs are eliminated to allow for better readability. This Table is organized similarly to Table 7, except that algorithms are grouped for t first and for Δ second, both ordered in decreasing sense; this allows to better compare Q with the piecewise-linear functions with similar shape, while keeping the same qualitative ordering of penalties.

t			10^7		10^5				10^3								10^2						10					
alg		CG	1P	Q	1P	3P	5P	Q	1P	3P			5P			Q	1P	3P		5P		Q	1P	3P		5P		Q
Δ			1000		500	1000	1000		100	1000	500	100	1000	500	100		10	100	10	100	10		1	10	1	10	1	
p1	avg	134	85	100	80	110	110	88	122	79	72	74	72	65	75	63	408	58	73	58	73	188	255	110	96	111	97	423
	dev		0.08	0.06	0.13	0.07	0.02	0.04	0.45	0.06	0.02	0.04	0.03	0.07	0.04	0.22	0.45	0.06	0.06	0.13	0.04	0.18	0.36	0.13	0.03	0.05	0.04	0.46
p2	avg	151	92	112	84	117	114	103	119	92	83	84	84	77	80	86	387	81	88	83	104	243	306	130	110	145	107	385
	dev		0.08	0.03	0.11	0.05	0.04	0.02	0.49	0.03	0.04	0.05	0.04	0.17	0.05	0.07	0.39	0.19	0.17	0.12	0.04	0.36	0.36	0.14	0.08	0.12	0.07	0.34
p3	avg	183	117	146	109	162	161	125	129	114	96	103	99	89	94	103	473	80	97	81	106	261	442	122	122	150	125	494
	dev		0.08	0.08	0.11	0.04	0.03	0.03	0.47	0.08	0.04	0.04	0.05	0.04	0.03	0.38	0.44	0.05	0.05	0.23	0.06	0.39	0.41	0.17	0.03	0.09	0.05	0.37
p4	avg	137	83	115	79	124	115	98	115	83	76	81	80	76	78	73	396	75	79	74	81	235	271	99	101	111	100	499
	dev		0.05	0.06	0.17	0.07	0.04	0.03	0.45	0.07	0.08	0.06	0.04	0.16	0.05	0.37	0.38	0.13	0.03	0.13	0.05	0.43	0.36	0.13	0.07	0.06	0.04	0.37
p5	avg	592	343	422	260	481	468	288	200	256	223	274	222	194	278	213	544	241	346	279	332	336	620	388	336	406	335	583
	dev		0.07	0.04	0.03	0.04	0.05	0.03	0.35	0.05	0.04	0.13	0.05	0.08	0.09	0.3	0.36	0.32	0.17	0.36	0.16	0.37	0.29	0.2	0.16	0.17	0.14	0.29
p6	avg	505	195	318	158	394	384	199	151	177	149	177	150	126	181	109	556	125	192	121	194	300	599	205	203	239	205	610
	dev		0.04	0.05	0.08	0.02	0.02	0.01	0.33	0.06	0.07	0.08	0.05	0.07	0.11	0.06	0.37	0.19	0.03	0.2	0.03	0.41	0.38	0.29	0.05	0.17	0.04	0.23
p7	avg	287	152	223	125	271	275	155	181	125	110	164	110	107	160	118	554	141	185	159	183	316	601	244	204	286	193	568
	dev		0.09	0.04	0.11	0.04	0.04	0.03	0.49	0.06	0.04	0.17	0.05	0.17	0.15	0.12	0.38	0.35	0.04	0.4	0.06	0.44	0.36	0.09	0.08	0.11	0.03	0.32
p8	avg	192	126	162	107	172	178	126	184	108	97	139	96	94	142	112	466	115	150	118	152	212	508	174	156	179	154	513
	dev		0.09	0.03	0.12	0.05	0.04	0.02	0.54	0.06	0.06	0.21	0.08	0.14	0.16	0.27	0.44	0.28	0.09	0.27	0.1	0.43	0.25	0.21	0.07	0.2	0.12	0.4
p9	avg	258	161	213	140	224	215	152	179	127	110	130	109	101	130	132	505	140	178	170	179	263	531	235	178	233	183	539
	dev		0.09	0.07	0.14	0.07	0.05	0.03	0.57	0.04	0.03	0.1	0.04	0.12	0.11	0.19	0.46	0.32	0.13	0.37	0.13	0.45	0.03	0.25	0.14	0.28	0.12	0.38
p10	avg	298	214	204	177	244	238	148	254	144	128	146	120	120	147	153	566	163	160	207	151	339	655	302	176	329	176	594
	dev		0.21	0.05	0.19	0.1	0.02	0.02	0.61	0.04	0.02	0.07	0.06	0.14	0.05	0.47	0.35	0.36	0.11	0.31	0.11	0.39	0.04	0.14	0.06	0.16	0.03	0.28
α																												
0.9	avg	274	139	200	112	227	226	149	68	133	113	122	113	91	119	84	214	84	139	83	139	134	541	155	155	179	156	269
0.75	avg	274	157	200	130	228	224	148	100	130	111	128	109	99	132	94	380	100	148	112	147	199	305	187	160	205	161	431
0.5	avg	274	167	203	132	231	225	147	174	126	115	137	115	105	141	114	561	125	154	149	163	286	516	217	167	223	168	585
0.25	avg	274	161	202	142	227	225	149	229	131	116	144	118	109	146	139	650	147	160	161	166	338	518	215	175	240	171	662
0.0	avg	274	159	203	145	236	230	149	247	133	118	155	117	120	146	151	623	155	173	170	163	390	514	232	185	248	181	658

Table 8: Comparing quadratic vs linear ST using $\alpha-$initial dual points

The results in Table 8 can be commented as follows:

- For weak penalties ($t = 10^7$, $t = 10^5$), 1P performs better than Q that is in turn better than 3P and 5P; weak Q is probably too weak, and the infinite slope of 1P is the most important factor for its relatively good performances. 3P and 5P are weaker than Q since they underestimate it.
- As t decreases, Q becomes better than 1P, and initially it outperforms 3P and 5P; however, while Q becomes more and more competitive w.r.t. 1P as α decreases (the quality of the initial dual point worsens), so do 3P and 5P w.r.t. Q, and for low-quality initial points they become better than Q.
- As t further decreases in the strong range, 3P and 5P become better than Q (for selected values of Δ); again, a worse quality of the initial point has much less of an impact on 3P and 5P than on Q, as testified by the standard deviation values.

Thus, the 3- and 5-pieces linear STs offer more robustness and good performances in most cases. Quadratic STs produce acceptable, sometimes very good, improvement if t is neither too large nor too small, and they seem somewhat more capable of exploiting the availability of a good initial dual point. For very good initial dual points, 1P with a carefully selected value of Δ provides the best performances; however this choice is the least robust, and Q is clearly a much less risky choice if one does not want to handle multiple stabilization parameters. One final observation is that the good behaviour of 1P with large α is likely to be due to the fact that the initial dual point has the same structure as an optimal one, since the all-zero dual solution is feasible in our case (the same situation postulated in [25]); results may be less favorable to 1P, and perhaps to Q, too, if this is not the case.

6.2 MDVS: using randomly generated initial dual points

We now turn to randomly generated initial dual solutions; since these are somewhat more difficult to solve, we only require a relative gap of 10^{-4} to be reached. Table 9 reports the results obtained using randomly generated initial dual points; each column reports averaged results (number of iterations required to reach optimality) for a group of instances, "md1" being those with 400 tasks, "md2" being those with 800 tasks, and "md3" being the remaining ones with 1000 or 1200 tasks. This Table is arranged similarly to Tables 7 and 8, except for being transposed; thus, penalties become stronger going from the top to the bottom of the Table.

The results in Table 9 confirm the importance of a properly structured initial point for 1P, as its performances are substantially worse than these obtained using α-points. Q now shows much better performances than 1P almost everywhere, except for very strong penalties; furthermore, it attains the best performances in some cases ($(\delta^1, \delta^2) = (0.0, 0.5)$). However, in all other cases the 3-pieces and especially the 5-pieces ST, with a proper choice of the parameters, are more efficient than Q; besides, the latter is sometimes considerably more affected by the choice of the initial points, whereas 3P and 5P are most often largely insensitive to this. Thus, k-pieces linear ST seem capable to offer both performances and robustness without requiring initial dual points with specific structure or high quality.

6.3 LH-MDVS: using randomly generated initial dual points

We now report on the same experiments of the previous section on the much more difficult LH-MDVS instances; indeed, the maximum of 1500 iterations allowed to SCG approaches is far less than the maximum number of iterations needed by standard CG. The results are presented in Table 10, that has the same structure as Table 9; only, since not all instances are solved to the prescribed accuracy within the allotted iteration limit, a further column "slv" is added which reports the total number of instances, across the three groups, for which the algorithms did actually stop for having reached a gap less than 10^{-4}.

The results mostly mirror those previously shown. With no choice of Δ the boxstep (1P) solves all instances of a group within 1500 iterations (cf. column "slv"); only occasionally it even solves more instances than CG. 3P encountered more difficulties with these more challenging instances, but still did much better than standard CG in all cases, and performed very well in more than half of the cases. For these instances 5P performed significantly better than 3P across the board, much more evidently so than in the easier MDVS cases. However, the best performing ST for LH-MDVS, basically always attaining the best results (for carefully chosen t) is Q, except for the case of too strong penalty function where 5P and 3P significantly outperformed it.

δ^1—δ^2			0.0—0.5			0.0—1.0			0.5—1.0		
t	alg	Δ	md1	md2	md3	md1	md2	md3	md1	md2	md3
	CG		151	549	259	151	549	259	151	549	259
10^7	1P	1000	632	700	700	611	700	700	529	600	514
	Q		115	367	207	114	376	198	114	361	200
10^5	1P	500	414	527	543	520	421	393	550	627	345
	3P	500	126	447	222	131	434	226	126	437	221
	5P	500	127	437	218	125	434	225	122	440	230
	Q		101	240	144	101	246	145	104	259	143
10^3	1P	1e2	212	255	176	331	274	285	293	337	261
	3P	1000	96	232	123	94	216	121	94	221	121
		500	83	189	103	85	182	107	80	185	107
		100	85	203	130	85	201	123	85	206	132
	5P	1000	84	181	107	87	182	109	84	184	110
		500	74	156	99	74	155	97	74	156	95
		100	82	193	130	86	199	137	84	203	139
	Q		54	118	71	87	157	111	111	141	113
10^2	1P	1e1	300	464	468	312	471	503	300	517	544
	3P	100	63	129	92	71	137	94	65	147	93
		10	80	211	142	82	207	138	82	221	146
	5P	100	58	122	97	72	123	94	60	136	97
		10	79	203	140	82	218	149	86	215	155
	Q		184	190	193	369	386	447	352	396	468
10	1P	1	320	651	509	363	683	487	294	700	508
	3P	10	96	226	198	118	213	174	97	213	192
		1	91	221	160	89	224	161	88	237	167
	5P	10	122	364	214	117	256	190	117	254	202
		1	92	235	156	84	225	163	92	248	161
	Q		456	673	618	531	700	675	491	700	688

Table 9: MDVS: using randomly generated initial dual points

δ^1—δ^2			0.0—0.5				0.0—1.0				0.5—1.0			
t	alg	Δ	lh1	lh2	lh3	slv	lh1	lh2	lh3	slv	lh1	lh2	lh3	slv
	CG		629	1866	3588	6	629	1866	3588	6	629	1866	3588	6
10^7	1P	1000	1500	1500	1500	0	1500	1500	1500	0	1500	1500	1500	0
	Q		448	1283	1500	9	446	1247	1500	8	458	1249	1500	9
10^5	1P	500	1208	1500	1500	1	1500	1500	1500	0	1224	1500	1500	1
	3P	500	494	1265	1500	8	494	1283	1500	8	510	1306	1500	8
	5P	500	483	1231	1484	9	483	1251	1486	9	489	1227	1498	9
	Q		331	880	1314	12	343	882	1316	12	330	878	1331	12
10^3	1P	100	624	1500	1500	4	476	1374	1500	7	452	1382	1500	8
	3P	1000	370	1443	1500	6	384	1394	1500	7	382	1442	1500	6
		500	298	1161	1500	9	314	1186	1487	9	315	1183	1500	8
		100	245	651	1155	13	249	696	1189	13	258	688	1209	13
	5P	1000	298	1203	1484	9	293	1172	1450	9	314	1166	1473	10
		500	240	882	1377	11	246	900	1362	11	253	885	1347	11
		100	233	528	867	14	244	559	948	14	239	566	931	14
	Q		136	283	396	14	155	323	457	14	152	317	460	14
10^2	1P	10	505	1284	1500	8	611	1484	1500	5	546	1435	1500	5
	3P	100	191	578	1085	13	199	755	915	10	200	626	1131	13
		10	216	418	573	14	222	469	717	14	226	466	713	14
	5P	100	164	436	793	14	170	519	822	14	175	481	797	14
		10	217	396	518	14	220	447	685	14	225	444	641	14
	Q		282	293	676	14	617	864	1249	10	608	777	1362	12
10	1P	1	969	1500	1500	4	1133	1500	1500	2	1106	1500	1500	3
	3P	10	205	308	448	14	248	571	610	14	232	575	540	14
		1	224	434	526	14	247	529	757	14	297	501	702	14
	5P	10	234	303	614	14	270	636	651	14	250	618	491	14
		1	249	442	532	14	257	608	880	14	263	485	682	14
	Q		838	1486	1500	5	1233	1500	1500	2	1163	1500	1500	3

Table 10: LH-MDVS: using randomly generated initial dual points

6.4 UBS: relative gap evolution

Finally, we report results for the UBS instances; these have also been obtained with randomly generated initial dual points, with $(\delta^1, \delta^2) = (0.0, 1.0)$. In Table 11 we first report detailed results for all 12 instances; this Table is organized exactly as Table 9.

t	alg	Δ	u5s0	u5s1	u7s0	u7s1	u10s0	u10s1	u12s0	u12s1	u15s0	u15s1	u20s0	u20s1
	CG		106	132	158	169	321	300	371	506	858	785	1004	989
10^7	1P	1000	1500	285	1500	1500	1500	1500	1500	1500	1500	1500	1500	1500
	Q		98	97	143	152	331	304	361	471	681	694	871	1105
10^5	1P	500	1373	209	1500	1020	1500	1500	1500	1500	1500	664	231	729
	3P	500	99	125	169	181	458	321	394	590	944	1012	1251	1375
	5P	500	111	113	164	193	404	362	441	602	920	816	1230	1490
	Q		81	92	111	110	188	166	206	220	339	271	298	357
10^3	1P	100	336	167	350	328	675	417	953	315	690	286	260	391
	3P	1000	75	85	107	98	169	149	181	188	277	243	233	335
		500	72	75	90	88	141	133	155	167	243	191	181	230
		100	77	80	101	102	171	156	178	203	342	270	279	399
	5P	1000	71	74	97	87	154	131	150	159	248	191	194	239
		500	62	68	82	78	122	107	130	129	199	139	158	186
		100	77	76	99	99	166	155	175	196	317	304	297	412
	Q		107	55	108	80	171	171	119	93	195	163	106	108
10^2	1P	10	259	458	349	461	645	534	698	725	811	787	763	950
	3P	100	55	60	77	69	107	95	114	116	199	124	150	245
		10	106	125	99	131	167	154	176	274	309	336	391	448
	5P	100	52	59	70	83	109	89	104	209	187	118	159	227
		10	107	80	102	140	171	154	178	268	388	380	372	499
	Q		364	261	384	327	427	451	452	337	529	388	296	404
10	1P	1	361	391	559	462	729	748	879	1098	1397	1472	1500	1500
	3P	10	134	140	183	172	198	297	213	320	278	270	362	370
		1	111	119	128	143	182	171	210	254	407	372	368	516
	5P	10	147	136	199	186	250	328	265	305	301	370	393	389
		1	104	113	103	131	192	189	194	291	371	361	355	501
	Q		506	486	634	877	1352	1273	1485	1125	1500	1276	995	1276

Table 11: Detailed results for UBS instances

The results in Table 11 basically confirm those previously seen: 1P is not significantly more (and often much less) efficient than standard CG, 3P and 5P significantly outperform 1P, with 5P most often being preferable to 3P, Q is most often a good choice, even superior to 5P, except for high penalty values. There seem to be *some relationship between the difficulty of the instances and the trends seen in the results*: UBS are more difficult than MDVS but easier than LH-MDVS, and this seems to impact in a predictable way on the relative behavior of the different STs. In particular, 5P is discernibly better than 3P, less in LH-MDVS but more in MDVS. Similarly, Q appears to often be the better choice, less than in the more difficult LH-MDVS instances but more than in the easier MDVS instances. All this seems to indicate that, at least on this test bed, *smoother ST tend to be more and more efficient as the difficulty of the instance grows*; while 1P can be very efficient on the easy MDVS instances with a very good initial dual point, Q is definitely the best choice on the very difficult LH-MDVS instances with random initial point, and 3P and 5P seems to fall in the middle. This does not contradict the results in [6], while painting a richer and possibly more interesting picture. It is worth reminding again that all this holds for a very "rigid" setting, i.e., no on-line tuning of the steepness of the ST and a fixed choice of the intermediate parameters in 5P, which in our opinion is more likely to damage the latter than Q, which has infinitely many slopes. However, the results seem to indicate that more flexible ST, like 5P and Q, may definitely have a role in building efficient SCG approaches for very difficult instances.

We finish our analysis by presenting, in Table 12, results depicting the evolution of the relative gap w.r.t the number of iterations. Three groups of UBS instances were formed: "ubs1" contains instances with 500 and 700 tasks, "ubs2" contains instances with 1000 and 1200 tasks, and "ubs3" contains instances with 1500 and 2000 tasks. For each group, the four gap values 10^{-1}, 10^{-2}, 10^{-3}, and 10^{-4} were considered; in Table 12, for each group and ST the average numbers of CG iterations needed to *decrease the gap starting from the previous value* is reported, with a blank entry indicating that the gap could not be reached within the 1500 iterations limit.

The results in Table 12 confirm that 1P is very slow in obtaining even the very coarse precision of 10^{-1}, clearly indicating that a lot of effort is ill-spent due to an inefficient stabilization which prevents from obtaining good columns early on. The lower bound improvements basically mirror the general efficiency of the algorithms, with a fast initial convergence being strictly (positively) correlated with a good overall efficiency of the approach; 3P, 5P and Q show the same relative behavior seen in Table 11. This once again shows the importance, provided that the strength of the ST is properly chosen, of rapidly obtaining a good estimate of the optimal dual solution for the overall efficiency of a SCG approach.

			ubs1				ubs2				ubs3			
t	alg	Δ	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-1}	10^{-2}	10^{-3}	10^{-4}
10^7	1P	1000	1188	5	3		1500				1500			
	Q		89	23	8	3	319	31	13	5	788	34	13	4
10^5	1P	500	1005	12	7	2	1500				750	22	7	2
	3P	500	108	23	12	2	394	33	9	5	1095	36	11	4
	5P	500	109	25	9	4	406	30	12	4	1063	36	11	5
	Q		66	20	10	3	151	28	13	4	269	34	11	4
10^3	1P	100	236	11	38	10	501	2	53	34	317	0	56	34
	3P	1000	55	23	11	3	125	31	12	3	221	37	10	4
		500	46	24	9	2	99	34	12	4	135	60	12	4
		100	57	21	10	2	133	30	12	3	268	27	16	12
	5P	1000	48	23	9	3	102	30	13	4	168	35	11	4
		500	39	21	10	2	77	32	11	3	123	32	12	3
		100	55	20	10	4	126	33	11	4	258	30	20	25
	Q		33	17	12	25	85	22	13	19	83	25	15	21
10^2	1P	10	291	64	14	14	514	110	19	8	669	144	10	5
	3P	100	32	21	10	3	63	29	8	8	150	8	7	16
		10	56	17	12	31	136	23	13	22	275	25	37	35
	5P	100	32	19	8	7	62	26	12	28	130	7	12	24
		10	57	17	10	24	135	22	12	25	288	25	50	48
	Q		265	9	52	8	346	0	45	26	315	0	41	48
10	1P	1	305	81	23	34	691	119	22	33	1322	108	5	32
	3P	10	38	24	56	40	79	45	76	58	223	0	80	18
		1	57	17	20	32	138	24	13	30	294	25	55	43
	5P	10	39	23	66	39	90	31	119	47	201	0	149	14
		1	56	17	19	21	140	21	16	40	288	25	50	34
	Q		461	111	35	19	1037	244	8	20	1144	109	5	4

Table 12: Gap evolution for UBS instances

7 Conclusions

Using a general theoretical framework developed in the context of NonDifferentiable Optimization, a generic Stabilized Column Generation algorithm is defined where an explicit Stabilizing (resp. Penalty) Term is added to the dual (resp. primal) master problem in order to stabilize the dual iterates. The general framework leaves great freedom for a number of crucial details in the implementation, such as the "shape" of the ST, the specific choices of its parameters, influencing its "strength", and the strategies for the on-line updating of these parameters. A crucial aspect of this approach is the availability of convexity constraints which allow to define an objective function, whose value can be monitored in order to evaluate whether the tentative dual point, where CG is performed, is better than the stability center. This allows to move away from Proximal Point approaches, which already offer better performances than non-stabilized CG ones, and towards Bundle-type approaches, which typically outperform PP ones, being the only viable alternative in some cases (cf. §5.2).

We have computationally analyzed several different STs, as well as a large number of different choices for the parameters, in several practical applications using a state-of-the-art Column Generation code. The results show that a careful choice of the parameters may lead to very substantial performance improvements w.r.t. non-stabilized CG approaches. An extensive computational experience, aimed at determining guidelines for the selection of the shape of the ST, has shown evidence that "simple" STs, with one or three pieces, may be the best choice for easier instances, especially if a very good estimate of the dual optimal solution is available. However, as the instances become more difficult to solve, and the quality of the initial dual solution deteriorates, "more complex" STs become more efficient. In particular, a 5-pieces linear ST seems to offer a good compromise between flexibility and implementation cost, allowing to obtain very good results most of the time, only provided one avoids falling into extreme cases where the penalty is either too week or too strong.

In conclusion, 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. It will be interesting to verify if the present findings extend to approaches combining centers-based stabilization with explicit introduction of a stabilizing term, as predicated in [2, 27]. Also, it is surely worth testing whether the recently proposed modified form of Bundle approach of [25] improves performances significantly w.r.t. the ones tested in this paper, and/or changes in some way the guidelines for the selection of the shape of the ST obtained in this context.

References

[1] A. Oukil, H. Ben Amor, and J. Desrosiers *Stabilized Column Generation for Highly Degenerate Multiple-Depot Vehicle Scheduling Problems*, Computers & Operations Research, 33(4), 910–927, 2006.

[2] F. Babonneau, C. Beltran, A. Haurie, C. Tadonki, and J.-Ph. Vial *Proximal-ACCPM: a versatile oracle based optimization method*, in Advances in Computational Management Science Volume 9, Springer, 67–89, 2007.

[3] L. Bacaud, C. Lemaréchal, A. Renaud and C. Sagastizábal *Bundle Methods in Stochastic Optimal Power Management: A Disaggregated Approach Using Preconditioners*, Computational Optimization and Applications 20, 227–244 (2001).

[4] H. Ben Amor, J. Desrosiers *A Proximal Trust Region Algorithm for Column Generation Stabilization*, Computers & Operations Research, 33(4), 910–927 (2006).

[5] 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, QC, Canada (2002).

[6] O. Briant, C. Lemaréchal, Ph. Meurdesoif, S. Michel, N. Perrot, F. Vanderbeck *Comparison of Bundle and Classical Column Generation*, Mathematical Programming, 113(2), 299–344 (2008).

[7] 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).

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

[9] G. Desaulniers, J. Desrosiers, I. Ioachim, M.M. Solomon, F. Soumis, and D. Villeneuve *A Unifed Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems*, in "Fleet Management and Logistics", T. Crainic and G. Laporte (eds.), Kluwer, 57–93 (1998).

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

[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 *About Lagrangian Methods in Integer Optimization*, Annals of Operations Research 139, 163–193 (2005).

[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", N.H.M. Wilson (ed.), Lecture Notes in Economics and Mathematical Systems 471, Springer, 441–460 (1999).

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

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

[17] A. Hadjar, O. Marcotte, and F. Soumis *A Branch-and-Cut Algorithm for the Mutiple Depot Vehicle Scheduling Problem*, Operations Research 54(1), 130–149 (2006).

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

[19] K. Kiwiel *A Bundle Bregman Proximal Method for Convex Nondifferentiable Optimization*, Mathematical Programming 85, 241–258 (1999).

[20] 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).

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

[22] 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).

[23] C. Lemaréchal, A. Nemirovskii, and Y. Nesterov *New Cariants of Bundle Methods*, Mathematical Programming 69, 111–147 (1995).

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

[25] C. Lemaréchal, and K. Kiwiel *An Inexact Conic Bundle Variant Suited to Column Generation*, Mathematical Programming, to appear (2008).

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

[27] A. Ouorou *A Proximal Cutting Plane Method Using Chebychev Center for Nonsmooth Convex Optimization*, Mathematical Programming, to appear (2008).

[28] M.C. Pinar, and S.A. Zenios *On Smoothing Exact Penalty Functions for Convex Constrained Optimization*, SIAM Journal on Optimization 4, 486–511, (1994).

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

[30] R.T. Rockafellar *Monotone Operators and the Proximal Point Algorithm*, SIAM Journal on Control and Optimization 14(5), 877–898 (1976).

[31] L.M. Rousseau, M. Gendreau, and D. Feillet *Interior Point Stabilization in Column Generation* Operations Research Letters 35(5), 660–668 (2007).

[32] H. Schramm, and 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).

[33] M. Teboulle *Convergence of Proximal-like Algorithms*, SIAM Journal on Optimization 7, 1069–1083 (1997).