Automation and combination of linear-programming based stabilization techniques in column generation

A Pessoa, R Sadykov, E Uchoa, F Vanderbeck

Automation and combination of linear-programming based stabilization techniques in column generation

A. Pessoa (1), R. Sadykov (2,3), E. Uchoa (1), and F. Vanderbeck (3,2)

(1) Universidade Federal Fluminense, LOGIS lab
(2) INRIA Bordeaux-Sud-Ouest, team RealOpt
(3) University of Bordeaux, Mathematics Institute

Abstract. The convergence of a column generation algorithm can be improved in practice by using stabilization techniques. Smoothing and proximal methods based on penalizing the deviation from the incumbent dual solution have become standards of the domain. Interpreting column generation as cutting plane strategies in the dual problem, we analyze the mechanisms on which stabilization relies. In particular, the link is established between smoothing and in-out separation strategies to derive generic convergence properties. For penalty function methods as well as for smoothing, we describe proposals for parameter self-adjusting schemes. Such schemes make initial parameter tuning less of an issue as corrections are made dynamically. Such adjustments also allow to adapt the parameters to the phase of the algorithm. We provide extensive test reports that validate our self-adjusting parameter scheme and highlight their performances. Our results also show that using smoothing in combination with penalty function yields a cumulative effect on convergence speed-ups.

Keywords: Column Generation, Stabilization, Cutting Plane Separation

Introduction

In a decomposition approach for solving a Mixed Integer Program (MIP), a "tractable" subproblem is identified; it is defined by a subsystem of constraints that typically exhibits a block-diagonal structure; while the remaining constraints are qualified as "complicating" constraints. The problem is then reformulated in terms of variables that specify if a subproblem solution is to be used to form the solution to the full MIP. Column generation is used standardly to solve the linear program that results for such Dantzig-Wolfe reformulation. This so-called master program has indeed as many variables as subproblem solutions. Hence, to cope with such size, one needs to generate them dynamically in the course of the optimization procedure. Column generation is intimately linked to Lagrangian approaches, since the master is the linear programming equivalent of the Lagrangian dual resulting from relaxing the complicating constraints.

As reviewed in Section 1, column generation is a procedure where one iteratively solves a version of the master program restricted to a subset of its variables, collects the dual solution, and checks the optimality of the current solution by pricing over the whole set of columns. However, that pricing is not done by enumeration, but by optimizing the reduced-cost value of solutions of the subproblem mixed integer polyhedra. This pricing subproblem is precisely that resulting from the Lagrangian relaxation of the complicating constraints. Hence, its solution allows to define a valid Lagrangian dual bound. Thus, the procedure yields a sequence of price vectors, i.e., dual solutions to the master program, $\{\pi^t\}_t$, where t denotes the iteration counter, converging towards $\pi^* \in \Pi^*$, where Π^*

denotes the set of optimal dual solutions to the master LP. The associated Lagrangian bound values, $\{L(\pi^t)\}_t$, are converging towards the Lagrangian dual value L^*. Stabilization techniques are devised to accelerate the convergence of the dual sequence $\{\pi^t\}_t$ towards $\pi^* \in \Pi^*$.

As reviewed in Section 2, the convergence of the basic column generation procedure indeed suffers from several drawbacks : *dual oscillations*, *tailing-off effect*, *primal degeneracy*. Some stabilization techniques have been designed to address these drawbacks. The penalty function approaches [9, 15, 17] aim at controlling oscillations by explicitly penalizing dual solutions far from the current stabilization center. In the so-called smoothing techniques [19, 28], the dual price vector used for column generation is defined as a combination of the current dual solution to the master and a previous iterate solution. Some related algorithmic strategies for stabilization have been developed independently in the literature on cutting plane algorithms [4, 11]. Indeed, separation strategies from the cut generation literature and algorithmic strategies for stabilization in column generation algorithms are dual counterparts: the pricing procedure in column generation is understood as a separation routine for the dual of the master program. Therefore, efficient strategies to define the separation point or to select cuts translate into stabilization techniques and column generation strategies, as emphasized in several papers including [4, 5, 26, 27]. Note however that the primal-dual analogy between pricing and separation is not complete, as some of the results derived herein do not have their equivalent in terms of separation strategies.

The first contributions of the present paper concern dual price smoothing, a technique that addresses at once the dual oscillations, tailing-off, and degeneracy drawbacks of the basic column generation procedure. It acts through both exploiting history and through centralization, and it is particularly simple to implement. We complete here the results that were presented in our pre-publication taking the form of a conference proceeding [21] (which also included preliminary computational tests). We specifically formalize the link between in-out separation [4, 11] and dual price smoothing. We show that dual price smoothing schemes (such as that of [19, 28]) can be understood as an extension of in-out separation, introducing an in-point updating strategy that relies on a valid dual bound computation. We establish the detailed properties of a generic smoothing scheme and a global bound convergence property for the Wentges smoothing rule [28] that has no equivalent in a general in-out procedure. Moreover, we extend the smoothing paradigm by combining it with an ascent method; the combination is experimentally shown to lead to significant improvements.

The second set of contributions concern the parameterization of both penalty function and smoothing stabilizations. In fact, a major drawback of each of these approaches is to require a delicate and application-specific parameter tuning that is critical for the efficiency. Our work aims at limiting the number of parameters and at developing automatic-adjustment of the remaining parameters in the course of the algorithm. For smoothing, our parameter self-adjusting scheme uses gradient information. For piecewise linear stability function, we mimic the adjustment of the curvature of a quadratic function. Such automation aims at making the method parameter-tuning-free, as the initial parameter, even if wrongly set, will be corrected by the self-adjusting scheme. But it also results into a dynamic parameter setting as the "ideal parameter values" are typically not the same during the heading-in phase, in the core of the algorithm, and during the tailing-off phase. The third advantage of a dynamic adjustment of parameters is to alleviate the need for more complex stabilization schemes: for instance, our experiments show that, with automated parameter adjustments, a 3-piecewise linear function can perform just as well as, if not better than, a 5-piecewise linear function. Hence, we see our proposals for parameter self-adjusting schemes as an

important step for transforming stabilization techniques into general purpose techniques well suited for a generic branch-and-price solver.

Finally, we also contribute by proposing a combination of smoothing and penalty functions in a powerful semi-automatic stabilization scheme that is shown experimentally to perform better that either technique alone.

This paper addressed stabilization methods as tools to enhance the "poorly" performing basic column generation procedure, that is known in the literature as Kelley's cutting plane algorithm for the dual master program. Kelley's procedure uses the Simplex algorithm (as a black-box LP solver) to solve the restricted master programs. However, alternative techniques to solve the master problem exists that have inherent features that reduce the convergence drawbacks of a basic Simplex-based approach: The Improved-Primal-Simplex method [10] targets degeneracy; the Bundle approach and its variants [5, 16] use a quadratic penalty for deviation from the incumbent dual solution to implement proximality; the ACCPM [12] and other interior point based approaches, such as the recent primal-dual approach of [13, 18], use a central point as a dual candidate, intending to produce deeper cuts in the master dual and to avoid primal degeneracy.

Although the stabilization schemes that we consider are general paradigms that could also be applied in non-Simplex-based methods (as the above mentioned), our extensive experiments are done in the context of a Simplex-based column generation. This focus of scope in the use of penalty function and smoothing techniques is widespread in the literature. Our extensions consists in performing semi-automated parameter tuning and proposing a combined scheme that largely enhance the performance of a standard Kelley's algorithm. Thereby, we hope to have an impact on standard approaches. Moreover, our results offer new benchmarks for comparison with non-Simplex-based approaches. However, analyzing how the proposed methods extend and perform in each possible non-Simplex alternative is out of our scope. Developing the stabilization paradigms, while taking advantage the already existing inherent stabilization features of non-Simplex methods, requires a specific study that is left for future work. Nevertheless, a limited experiment in this direction was performed to evaluate the potential of such research direction. We experimented with using our stabilization scheme in combination with an interior point algorithm such as the Barrier method that is available in most LP-Solvers. As observed in our conclusion, it seems that, as the interior point method already has an inherent stabilization feature, the added stabilization procedures have a less drastic impact.

The paper is organized as follows. In Section 1, we review the primal and dual polyhedral representation of a Lagrangian dual problem, while introducing our notations. In Section 2, we discuss the stabilization issue and we review standard methods to speed-up convergence. In Section 3, the link is made between in-out separation and smoothing, from which we derive convergence properties for smoothing schemes. In Sections 4 and 5, we present parameter adjustment schemes and their hybridization with ascent methods. In Section 6, we discuss penalty function, limiting their parametrization to a single but important parameter. In Section 7, we outline the applications and instances that are used in our numerical experiments. Sections 8 and 9 include respectively our results with smoothing and the combination with penalty function approaches. In the conclusion, we also point to further work on applying stabilization beyond Simplex based approaches and we report on preliminary comparative results with a black-box Barrier method.

1. Column Generation

The main concepts underlying column generation approaches are reviewed here, emphasizing the properties on which rely stabilization schemes. Consider the integer program:

$$[\text{F}] \equiv \min\{cx : x \in X\} \tag{1}$$

where

$$X := \{x \in I\!R^n_+ : \ Ax \geq a, x \in Z\} \text{ and } Z := \{x \in I\!N^n : \ Bx \geq b, \ l \leq x \leq u\}.$$

In the decomposition of system X, it is assumed that Z defines a subproblem (assumed to be non-empty and bounded to simplify the presentation) that is "tractable" to solve, but $Ax \geq a$ are "complicating constraints". In other words, we assume that subproblem

$$[\text{SP}] \equiv \min\{cx : x \in Z\} \tag{2}$$

is "relatively easy" to solve compared to problem [F]. Then, a natural approach to solve [F], or to estimate its optimal value, is to exploit our ability to optimize over Z. We review this technique below.

Let Q be the enumerated set of subproblem solutions (it is a finite set given the boundedness of Z), i.e. $Q = \{z^1, \ldots, z^{|Q|}\}$ where $z^q \in Z$ is a subproblem solution vector. Abusing notations, $q \in Q$ is used hereafter as a short-cut for $z^q \in Q$. Thus, we can reformulate Z and *conv*(Z), the convex-hull of the integer solutions to Z, as:

$$Z = \{x \in I\!N^n : x = \sum_{q \in Q} z^q \lambda_q, \ \sum_{q \in Q} \lambda_q = 1; \ \lambda_q \geq 0 \ \forall q \in Q\} \ , \tag{3}$$

$$\textit{conv}(Z) = \{x \in I\!R^n_+ : x = \sum_{q \in Q} z^q \lambda_q, \ \sum_{q \in Q} \lambda_q = 1, \ \lambda_q \geq 0 \ \forall q \in Q\} \ . \tag{4}$$

As *conv*(Z) defines an ideal formulation for Z, [SP] can be rewritten as:

$$[\text{SP}] \equiv \min\{cx : x \in Z\} \equiv \min\{cz^q : q \in Q\} \equiv \min\{cx : x \in \textit{conv}(Z)\}. \tag{5}$$

In a Lagrangian relaxation approach, one exploits the subproblem tractability to derive dual bounds for [F]. Dualizing constraints $Ax \geq a$ with penalty vector $\pi \in I\!R^m_+$, the *Lagrangian function*,

$$L(\pi, x) := \pi \ a + (c - \pi A)x \ , \tag{6}$$

is optimized over Z, i.e., one solves the *Lagrangian subproblem*:

$$[\text{LSP}(\pi)] \equiv L(\pi) := \min_{x \in Z} L(\pi, x) \tag{7}$$

that defines a valid dual bound on [F]. The *Lagrangian dual function* is defined by $L : \pi \in I\!R^m_+ \rightarrow L(\pi)$. Maximizing function L leads to the best dual bound that can be derived from the Lagrangian relaxation. The *Lagrangian dual problem* is defined as:

$$[\text{LD}] \equiv \max_{\pi \in I\!R^m_+} L(\pi) \ . \tag{8}$$

The Lagrangian dual problem can be reformulated as a max-min problem, or as a linear program:

$$
\begin{aligned}
[\text{LD}] &\equiv \max_{\pi \in I\!R^m_+} \min_{x \in Z} \{\pi\, a + (c - \pi A)x\}; && (9)\\
&\equiv \max\{\eta, && (10)\\
&\quad \eta \leq cz^q + \pi(a - Az^q) \quad \forall q \in Q, && (11)\\
&\quad \pi \in I\!R^m_+, \eta \in I\!R^1\}; && (12)\\
&\equiv \min\{\sum_{q \in Q} (c\, z^q)\lambda_q, && (13)\\
&\quad \sum_{q \in Q} (Az^q)\lambda_q \geq a, && (14)\\
&\quad \sum_{q \in Q} \lambda_q = 1, \quad \lambda_q \geq 0 \quad \forall q \in Q\}; && (15)\\
&\equiv \min\{cx : Ax \geq a,\ x \in \mathit{conv}(Z)\ \}. && (16)
\end{aligned}
$$

The *Dantzig-Wolfe reformulation* is a valid reformulation of [F] expressed in terms of variables λ_q that were introduced for the reformulation of Z given in (3). Its linear programming (LP) relaxation, which we denote by [M], is precisely the form (13-15) of [LD].

A *column generation* procedure to solve [M] proceeds as follows. At iteration t, the restriction of [M] to columns defined in $Q^t = \{z^1, \ldots, z^t\}$ is denoted by [M^t]. This *restricted master LP* is:

$$
[\text{M}^t] \equiv \min\{\sum_{\tau=1}^{t} c\, z^\tau \lambda_\tau : \sum_{\tau=1}^{t} Az^\tau \lambda_\tau \geq a; \sum_{\tau=1}^{t} \lambda_\tau = 1; \lambda_\tau \geq 0,\ \tau = 1, \ldots, t\ .\} \tag{17}
$$

Linear program [M^t] is solved to optimality. Let λ^t denote an optimal solution to [M^t]. Its projection in X is:

$$
x^t := \sum_{\tau=1}^{t} z^\tau\ \lambda^t_\tau\ . \tag{18}
$$

Let $c\, x^t$ denote its objective value. The linear programming dual of [M^t] is:

$$
[\text{DM}^t] \equiv \max\{\eta : \pi(Az^\tau - a) + \eta \leq cz^\tau,\ \tau = 1, \ldots, t; \pi \in I\!R^m_+; \eta \in I\!R^1\};. \tag{19}
$$

Let (π^t, η^t) denote an optimal solution to [DM^t]. Using this dual solution, one searches for the most negative reduced cost column, by solving the subproblem:

$$
z^{t+1} \leftarrow z_{\pi^t} := \mathit{argmin}_{x \in Z}\{(c - \pi^t A)x\}\ . \tag{20}
$$

If $(c - \pi^t A)\ z_{\pi^t} + \pi^t\ a - \eta^t < 0$, then z_{π^t} defines a negative reduced cost column that is added to the restricted master. Otherwise, the current LP solution is optimal for the unrestricted master program [M].

The above column generation procedure should be seen essentially as a method for the dual problem, known as Kelley's cutting plane algorithm. It outputs a sequence of values for the Lagrangian price vector: $\{\pi^t\}_t$, that converges towards an optimal dual price vector, π^*. In the process, one can also derive a sequence of candidate primal solutions, $\{x^t\}_t$, converging towards an optimal solution x^* of problem (16), but primal solutions should be understood as a by-product used to prove optimality of the dual solution. One can observe the following properties:

Observation 1
(i) The vector x^t defined in (18) *is a solution to [M^t]* $\equiv \min\{cx : Ax \geq a,\ x \in \text{conv}(\{z^1, \ldots, z^t\})\}$.
(ii) The solution of the dual restricted master, [DM^t], is such that $\pi^t = \text{argmax}_{\pi \in \mathbb{R}^m_+} L^t(\pi)$ *where* $L^t()$ *defines an approximation of the Lagrangian dual function* $L()$*, considering only the subset of subproblem solutions* $\{z^1, \ldots, z^t\}$*: i.e.,*

$$L^t() : \pi \rightarrow \min_{z \in \{z^1, \ldots, z^t\}} \{\pi a + (c - \pi A)z\} = \min\{L(z^1, \pi), \ldots, L(z^t, \pi)\}\ . \tag{21}$$

Function $L^t()$ is an upper approximation of function $L()$: $L^t(\pi) \geq L(\pi)\ \forall \pi \in \mathbb{R}^m_+$. The hypo-graph of function $L^t()$ defines a polyhedral outer approximation (illustrated in the left picture of Figure 1) of the master LP dual program (10-12). By duality, $L^t(\pi^t) = \eta^t = c\ x^t$.
(iii) Solving [LSP(π^t)] exactly serves four purposes simultaneously:
$(iii.a)$ it yields the most negative reduced cost column: $z^{t+1} = z_{\pi^t} \in Q$ for [M];
$(iii.b)$ it yields the most violated constraint defined by a subproblem solution $z^q \in Q$ for [DM];
$(iii.c)$ the constraint violation of the oracle solution z_{π^t} defines a sub-gradient of $L(.)$ at point π^t:

$$g^t := (a - A\ z_{\pi^t})\ ; \tag{22}$$

$(iii.d)$ the correct value of the Lagrangian function $L()$ is now known at point π^t: $L(\pi^t) = \pi^t a + (c - \pi^t A) z_{\pi^t}$, and therefore this value remains unchanged in any further approximation of $L()$, i.e., $L^\tau(\pi^t) = L(\pi^t)\ \forall \tau > t$.
(iv) At iteration t, $\text{conv}(\{(\pi^\tau, L^{\tau+1}(\pi^\tau))\}_{\tau=1,\ldots,t})$ *defines an inner approximation (illustrated in the middle picture of Figure 1) of the master LP dual program (10-12). Outer and inner approximation are equal at these points, $\{(\pi^\tau, L^{\tau+1}(\pi^\tau))\}_{\tau=1,\ldots,t}$, as $L^{\tau+1}(\pi^\tau) = L(\pi^\tau)$ for $\tau = 1, \ldots, t$. One of these points defines the incumbent dual solution $\hat{\pi} = \text{argmax}_{\tau=1,\ldots,t} L^{\tau+1}(\pi^\tau)$ with value $\hat{L} = L(\hat{\pi}) = \hat{\eta}$.*
(v) If $L^t(\pi^t) = \hat{L}$, or practically $(cx^t - \hat{L}) \leq \epsilon$ for a given precision level $\epsilon > 0$, then the optimal solution is reached, i.e., $\eta^* = \hat{L}$.

In the sequel, (π^*, η^*) denotes an optimal solution to the Lagrangian dual, while $\hat{L} = L(\hat{\pi})$ denotes the current best dual (lower) bound on η^*.

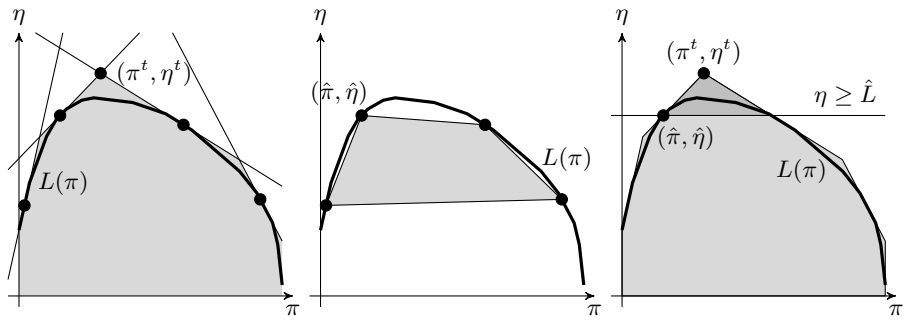


Fig. 1: Outer and Inner polyhedral approximation for the dual master polyhedron [DM] (10-12) and the trust polyhedra defined by adding a cut on the objective to the outer approximation.

2. Stabilization Techniques in Column Generation

Stabilization techniques are devised to accelerate the convergence of the column generation algorithm, by targeting its drawbacks as reviewed in [26]:

- *Dual oscillations:* Solutions π^t jump erratically. One extreme solution of the restricted dual master (10-12) at iteration t, [DM^t], is followed by a different extreme point of [DM^{t+1}], leading to a behaviour often referred to as *"bang-bang"*. Because of these oscillations, it might be that $||\pi^{t+1} - \pi^*|| > ||\pi^t - \pi^*||$. Moreover, the dual bounds $L(\pi^t)$ are converging non monotonically, with ups and downs in the value curve (the *yo-yo* phenomenon).
- *The tailing-off effect:* Towards the end of the algorithm, added inequalities in [DM^t] tend to cut only a marginal volume of the dual solution space, making progress very slow.
- *Primal degeneracy and alternative dual optimal solutions:* An extreme point λ of polyhedron [M^t] has typically fewer non zero values than the number of master constraints. The complementary dual solution solves a system with fewer constraints than variables that admits many alternative solutions. As a consequence, the method iterates between alternative dual solutions without making any progress on the objective value.

Techniques to stabilize column generation belongs to one of the three standard families listed in [26]:

Proximality of a stability center: To avoid oscillation, one attempts to remain in the proximality of the current incumbent dual solution, denoted by $\hat{\pi}$, used as a *stability center*. This can be implemented by adding a penalty function to the dual objective to drive the optimization towards the stability center. Using the modified objective function, the dual problem (19) is replaced by

$$\pi^t := argmax_{\pi \in I\!R^m_+} \{L^t(\pi) - \hat{S}(\pi)\}, \tag{23}$$

where the *penalty function*,

$$\hat{S} : \pi \in I\!R^m_+ \rightarrow I\!R_+ \ ,$$

is typically convex, takes value zero at $\hat{\pi}$, and increases as $||\pi - \hat{\pi}||$ increases. The Bundle method [5] is a special case where $\hat{S}(\pi) = \frac{1}{2\theta}||\pi - \hat{\pi}||^2$. One can also make use of a *piecewise linear penalty function* S (see [9] for instance) in order to ensure that the master problem is still a linear program (with additional artificial variables whose costs and bounds are chosen to model a piecewise linear stabilizing function). Penalty function methods require delicate tuning of several parameters. Alternatives are to set proximality as a constraint: $||\pi - \hat{\pi}||^2 \leq \kappa$ giving rise to a trust-region approach [16]; or to minimize $||\pi - \hat{\pi}||$ subject to a parametric target constraint on the Lagrangian function value: $L^t(\pi) \geq \bar{L}$ leading to a level set approach [7, 16].

Smoothing techniques: The dual solution π^t used for pricing is "corrected" based on previous dual solutions. In particular, Neame [19] proposes to define smoothed price as:

$$\tilde{\pi}^t = \alpha\tilde{\pi}^{t-1} + (1-\alpha)\pi^t \ , \tag{24}$$

i.e., $\tilde{\pi}^t$ is a weighted sum of previous iterates: $\tilde{\pi}^t = \sum_{\tau=0}^{t}(1-\alpha)\alpha^{t-\tau}\pi^\tau$. Wentges [28] proposes another smoothing rule where:

$$\tilde{\pi}^t = \alpha\hat{\pi} + (1-\alpha)\pi^t \ . \tag{25}$$

i.e., $\tilde{\pi}^t = \hat{\pi} + (1-\alpha)(\pi^t - \hat{\pi})$, which amounts to taking a step of size $(1-\alpha)$ from $\hat{\pi}$ in the direction of π^t. In both rules, $\alpha \in [0, 1)$ parameterizes the level of smoothing. The pricing problem is then solved using the

smoothed prices, $\tilde{\pi}^t$, instead of π^t:

$$z_{\tilde{\pi}^t} := \textit{argmin}_{x \in Z}\{(c - \tilde{\pi}^t A)x\} \; . \tag{26}$$

Solving this modified pricing problem might not yield a negative reduced cost column, even when one exists for π^t. This situation is the result of a *mis-pricing*. In such case, applying (24) or (25) with the same π^t solution leads to a new dual price vector that is closer to π^t, as we shall see. Note moreover, that the incumbent $\hat{\pi}$ is updated each time the current Lagrangian bound improves over $\hat{L}$.

Centralized prizes: One makes faster progress in improving the polyhedral outer approximation of the master LP dual program (10-12) when separating a point (π, η_π) in the interior of (10-12) rather than an extreme point. The analytic center cutting-plane method (ACCPM) [12] defines iterate π^t as the analytic center of the linear program (10-12) augmented with an optimality cut $\eta \geq \hat{L}$ that defines a trust region, as illustrated in the right picture of Figure 1; the latter is found using a barrier method. A more recent example of centralization approach is the Primal Dual Column Generation Method (PDCGM) proposed in [13] that combines column generation with a central path following algorithm. Alternatives exist to implement centralization within a Simplex-based approach (see [15] for instance and the references in [26]).

Note that using a stability function approach, a smoothed price vector, or an interior point for pricing can have a drawback: for some oracle, the pricing problem can be harder to solve when the price vector is dense (few non zero components) and more evenly distributed between items (with less clear dominance that can be exploited in dynamic programming recursions for instance).

3. Smoothing and In-Out Separation

The above smoothing techniques are related to the in-out separation scheme of [4, 11]. The solution over the outer approximation of dual polyhedron (10-12) at iteration t (see Figure 1, left part) defines an *out-point*, i.e., a point outside polyhedron (10-12):

$$(\pi^{\text{out}}, \eta^{\text{out}}) := (\pi^t, L^t(\pi^t)) \; . \tag{27}$$

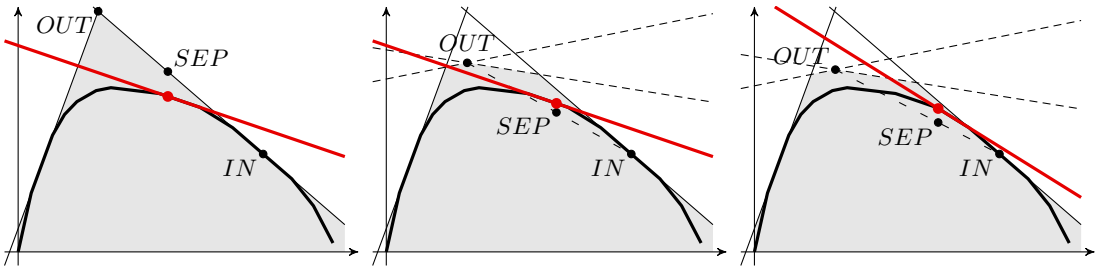
Symmetrically, consider a point inside the inner approximation (see Figure 1, middle part) of polyhedron (10-12). Possible definitions of such *in-point* are provided by the smoothing rules described above:

$$(\pi^{\text{in}}, \eta^{\text{in}}) := \begin{cases} (\tilde{\pi}^{t-1}, L(\tilde{\pi}^{t-1})) & \text{under rule (24),} \\ (\hat{\pi}, \hat{L}) & \text{under rule (25).} \end{cases} \tag{28}$$

These are in-points, because $L(\tilde{\pi}^{t-1})$ and $L(\hat{\pi})$ have been computed exactly when pricing as noted in Observation 1-$(iii.d)$. On the segment between the in-point and the out-point, one defines a *sep-point* at distance α from the out-point:

$$(\pi^{\text{sep}}, \eta^{\text{sep}}) := \alpha \, (\pi^{\text{in}}, \eta^{\text{in}}) + (1 - \alpha) \, (\pi^{\text{out}}, \eta^{\text{out}}) \; . \tag{29}$$

The in-out separation strategy consists in attempting to cut such sep-point. If an exact separation/pricing oracle fails to yield a separation hyperplan that cuts this sep-point, the sep-point is proved to be a valid in-point. Else, the out-point is updated (as cutting the sep-point implies cutting the out-point). For standard in-out separation, where either the in-point or the out-point is replaced by the sep-point at each iteration, [4] proves that the distance



Case A Case B Case C

Fig. 2: The three cases that can arise when attempting to cut point (π^{sep}, η^{sep}). Case A: (π^{sep}, η^{sep}) is cut-off by z^t, and hence so is (π^{out}, η^{out}). Case B: (π^{sep}, η^{sep}) is not cut but (π^{out}, η^{out}) is. Case C: neither (π^{sep}, η^{sep}), nor (π^{out}, η^{out}) are cut, which results in a *mis-pricing*.

between them tends to zero during a mis-pricing sequence.

The following proposition formalizes the properties common to Neame's and Wentges' smoothing schemes for column generation. Observe that the smoothing schemes described by rule (24) and (25) differ from the above standard in-out separation by the way in which the component η of the current dual solution is updated. Indeed, solving the separation/pricing problem yields a supporting hyperplane and a valid Lagrangian bound which is exploited in point (ii) below.

Proposition 1 *Common properties of Smoothing Schemes, in particular that of Neame or Wentges.*
(i) *If the separation point* (π^{sep}, η^{sep}) *is cut by the inequality defined by* $z_{\pi^{sep}}$*, i.e., if* $L(\pi^{sep}) = \pi^{sep} a + (c - \pi^{sep} A) z_{\pi^{sep}} < \eta^{sep}$*, then* (π^{out}, η^{out}) *is cut off and* $z_{\pi^{sep}}$ *defines a negative reduced cost column for* $[M^t]$*, as illustrated in Case A of Figure 2.*
(ii) *In the case* (π^{sep}, η^{sep}) *is not cut, which we refer to as a* "mis-separation"*, i.e., when* $L(\pi^{sep}) \geq \eta^{sep}$*, then* $(\pi^{sep}, L(\pi^{sep}))$ *defines a new in-point that may be used for the next iteration. Moreover, as* $\eta^{sep} = \alpha\eta^{in} + (1-\alpha)\eta^{out}$*, the new dual bound,* $L(\pi^{sep})$*, obtained when solving the pricing problem, improves the optimality gap at the smoothed price by a factor* α*, as illustrated in Figure 3, i.e.,*

$$(c\, x^t - L(\tilde{\pi}^t)) \leq \alpha\, (c\, x^t - L(\tilde{\pi}^{t-1}))\,, \tag{30}$$

provided that the in-point has been defined in such a way that

$$\eta^{in} \geq L(\tilde{\pi}^{t-1})\,, \tag{31}$$

which is the case under both rules (24) and (25).
(iii) *The cut defined by* $z_{\pi^{sep}}$ *can cut-off* (π^{out}, η^{out}) *even if it did not cut* (π^{sep}, η^{sep})*, as illustrated in Case B of Figure 2. If it does, both the in-point and the out-point can be updated. Otherwise, failing to cut the out-point, as illustrated in Case C of Figure 2, leads to a so-called* "mis-pricing"*. Then,* $(\pi^{out}, \eta^{out}) = (\pi^t, \eta^t)$ *remains solution for* $[DM^{t+1}]$ *defined from* $[DM^t]$ *by adding generator* $z_{\pi^{sep}}$*; but, the smoothed prices of the next iterate get closer to kelley's price vector,* π^{out}*:*

$$||\tilde{\pi}^{t+1} - \pi^{t+1}|| = \alpha\, ||\tilde{\pi}^t - \pi^t|| < ||\tilde{\pi}^t - \pi^t||\,. \tag{32}$$

Proof: Proposition 1-(i) results from the convexity of polyhedron (10-12). Proposition 1-(ii) states that if the separation point cannot be cut, it is proven feasible for polyhedron (10-12). One can then check that,

$$(c\,x^t - L(\tilde{\pi}^t)) = (\eta^{\text{out}} - L(\pi^{\text{sep}})) \leq (\eta^{\text{out}} - \eta^{\text{sep}}) = \alpha\,(\eta^{\text{out}} - \eta^{\text{in}}) \leq \alpha\,(c\,x^t - L(\tilde{\pi}^{t-1}))\ : \tag{33}$$

the first inequality in (33) is a simple re-writing of the assumed condition $L(\pi^{\text{sep}}) \geq \eta^{\text{sep}}$, while the second inequality derives from the assumption of Condition (31). It is satisfied at equality for Neame's scheme using rule (24) and derives from $\eta^{\text{in}} = \hat{L} \geq L(\tilde{\pi}^{t-1})$ for Wentges' scheme using rule (25). Proposition 1-(iii) observes that in the case (ii), there can be a mis-pricing or not. If there is a mis-pricing at iteration t, $\pi^{t+1} = \pi^t$ and the next sep-point can be shown to be closer to the out-point. Indeed, as $\pi^{\text{sep}} = \alpha\,\pi^{\text{in}} + (1-\alpha)\,\pi^{\text{out}}$, $||\pi^{\text{out}} - \pi^{\text{sep}}|| = \alpha\,||\pi^{\text{out}} - \pi^{\text{in}}||$. And, at iteration $t+1$, $\tilde{\pi}^{t+1} = \pi^{\text{sep}}$, $\pi^{\text{in}} = \tilde{\pi}^t$, while $\pi^{\text{out}} = \pi^{t+1} = \pi^t$. ∎

Smoothing schemes other than Neame and Wentges that differ by the rules for updating the in-point are possible. Note that Property (30) remains valid provided Condition (31) is met.

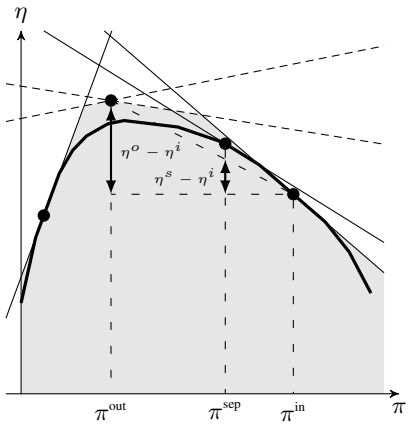


Fig. 3: In the case where $(\pi^{\text{sep}}, \eta^{\text{sep}})$ cannot be cut, then $(\eta^{\text{out}} - L(\pi^{\text{sep}})) \leq (\eta^{\text{out}} - \eta^{\text{sep}}) = \alpha\,(\eta^{\text{out}} - \eta^{\text{in}})$, as outlined in Proposition 1-(ii).

Hence, the smoothing rules of Neame and Wentges can be understood as a projection in the π-space of the in-out separation procedure of [4], with an extra feature however: in smoothing techniques, the in-point is typically updated even when the sep-point is cut. The update of the η value to a valid dual bound guarantees the feasibility of the updated in-point. In Wentges' smoothing scheme [28], the in-point is redefined as the dual incumbent at each iterate. Note however that when the separation point cannot be cut, $L(\pi^{\text{sep}}) > \hat{L}$ according to Proposition 1-(ii), and $\hat{\pi}$ is updated to π^{sep}. Thus, Wentges' smoothing conforms to the standard in-out paradigm. However, Neame's smoothing scheme [19] differs from the standard in-out procedure by the fact that π^{in} is updated to π^{sep} whether or not the sep-point was cut. It can be seen as a valid variant of the in-out procedure as, even if $(\pi^{\text{sep}}, \eta^{\text{sep}})$ is not an in-point, $(\pi^{\text{sep}}, L(\pi^{\text{sep}}))$ defines an in-point that can used in the next iteration, as done implicitly in rule (24). In any case, Proposition 1 holds true for Neame's smoothing scheme, as well as for Wentges. We emphasize that Proposition 1-(ii) is valid when a mis-separation arises, even if there is no mis-pricing. It has no equivalent for the general in-out procedure of [4] where no special component is associated with the objective value. To the best of our knowledge, such results had not been proven for Neame's smoothing scheme [19]. For

Wentges smoothing, Property (iii) was already mentioned in [28], while Property (ii), which then takes the form $(c\,x^t - L(\hat{\pi}^t)) \leq \alpha\,(c\,x^t - L(\hat{\pi}^{t-1}))$, was proven in [22], but in a more intricate manner relying on the concavity of function $L^t()$ defined in (21) and under a mis-pricing assumption.

Proposition 1-(ii)-(iii) are presented as convergence measures that are local to a mis-pricing sequence. However, they induce a global convergence measure for Wentges smoothing rule:

Corollary 1 *Global convergence of Wentges Smoothing Schemes.*
The optimality gap $(c\,x^t - \hat{L})$ decreases strictly with every mis-separation. Hence, the number of mis-separation is bounded.

Proof: The results follows immediately from Proposition 1-(ii), as with Wentges, $L(\tilde{\pi}^t) = L(\hat{\pi}^t) = \hat{L}$, while $c\,x^t$ decreases monotonically in the course of the column generation, and the gap $(c\,x^t - \hat{L})$ decreases by a factor α every time the sep point is not cut. ∎

For Neame smoothing scheme, in any sequence of separation failures, the value of $L(\tilde{\pi}^t)$ increases strictly according to Proposition 1-(ii). As $\hat{L} = \max_t\{L(\tilde{\pi}^t)\}$, either $\hat{L}$ eventually increases during the mis-separation sequence, or the sequence ends with an effective separation of $(\pi^{\text{sep}}, \eta^{\text{sep}})$. But, in the latter case, the value of $L(\tilde{\pi}^t)$ at the next mis-separation sequence is not necessarily larger than the last Lagrangian value of the previous sequence. So there is no guarantee of a monotonic decrease in the optimality gap over mis-separation sequences. An hybrid scheme based on smoothing rule (24) but with a reset of the in-point using rule (25) on the first iterate of a mis-separation sequence would have the global convergence property of Corollary 1 over mis-separation sequences.

The results of this section trivially extend to the case of multiple subproblems.

4. Smoothing with a self adjusting parameter

When using a smoothing scheme, instead of using the same α for all iterations, one can define iteration-dependent values α_t. We name α-schedule, the procedure used to select values of α_t dynamically. Intuitively, a large α can yield deeper cut if no mis-pricing occurs, while a small α can yield large dual bound improvement if a mis-pricing occurs. But a large α resulting in a mis-pricing or a small α with no mis-pricing may result in an iterate with little progress being made. The primary concern should be the overall convergence of the method, which can be guaranteed by Proposition 1. If no smoothing is used, i.e., $\alpha_t = 0\ \forall t$, the procedure is a standard Simplex based column generation for which finite convergence is proven, provided a cycle breaking rule that guarantees that each basis is visited at most once. When smoothing is used on the other hand, the same basis can remain optimal for several iterations in a sequence of mis-pricings. However, the convergence measures of Proposition 1-(ii)-and-(iii) provide bounds on the number of mis-pricings for a given LP solution π^{out}. Thus, in the line of the asymptotic convergence proof for in-out separation of [4], one can show that:

Proposition 2 *Finite convergence of the column generation procedure using smoothing.*
Applying a Simplex based column generation procedure to (13-15) while pricing on smoothed prices as set in (26), using a smoothing scheme satisfying Condition (31) *(in particular Neame* (24) *or Wentges* (25)*'s rule), converges to an optimal solution after a finite number of iterations, for any α-schedule with $\alpha_\tau \in [0, 1)$ for all iterations τ; i.e., for some $t \in \mathbb{N}$, $(\pi^t, \eta^t) = (\pi^*, \eta^*)$, where (π^*, η^*) is an optimal solution to (10-12).*

Proof: The number of columns that can be generated is finite. A column, once added, cannot price out negatively: the associated dual constraint is satisfied by both the in-point and the out-point (hence by the sep-point). However, in the case of a mis-pricing, there can be several iterations where no new column arises as a solution to the pricing problem. So the proof relies on bounding the number of such iterations. Consider a sequence of mis-pricing iterations, starting with $\pi^{\text{in}} = \tilde{\pi}^0$. Then, at iteration $t+1$,

$$||\tilde{\pi}^{t+1} - \pi^{\text{out}}|| = \alpha_t \, ||\tilde{\pi}^t - \pi^{\text{out}}|| = \ldots = \Pi_{\tau=0}^{t} \alpha_\tau \, ||\tilde{\pi}^0 - \pi^{\text{out}}||$$

and, under Condition (31),

$$||c\, x^{t+1} - L(\tilde{\pi}^{t+1})|| \leq \alpha_t \, ||c\, x^t - L(\tilde{\pi}^t)|| \leq \ldots \leq \Pi_{\tau=0}^{t} \alpha_\tau \, ||c\, x^0 - L(\tilde{\pi}^0)|| \,.$$

Hence, since $\alpha_\tau \in [0,1)$, $\Pi_{\tau=0}^{t} \alpha_\tau \to 0$, $\tilde{\pi}^t \to \pi^{\text{out}}$, and $L(\tilde{\pi}^t) \to cx^t = \eta^t = \eta^{\text{out}}$, as $t \to \infty$. Thus, combining the argument of finiteness of the Simplex algorithm and the asymptotic convergence over a mis-pricing sequence, one can guarantee that $\exists t$ such that such that, $\pi^t = \pi^{\text{out}} = \pi^* \in \Pi^*$ and $\eta^t = \eta^{\text{out}} = \eta^*$, and therefore, for any given precision level $\epsilon > 0$, $\exists t$ such that $||\pi^* - \tilde{\pi}^t|| \leq \epsilon$ and $||\eta^* - L(\tilde{\pi}^t)|| \leq \epsilon$ and the master is considered as optimized. ∎

However, asymptotically convergent algorithms might not be suitable for practical purposes. For instance, consider setting $\alpha = 0.8$ for all t, as illustrated in the left of Figure 4. Then, the distance reduction in a mis-pricing sequence becomes small very quickly. In practice, it would be better to choose an α-schedule such that $\tilde{\pi}^t = \pi^t$ after a small number of mis-pricing iterations t, as illustrated in the right of Figure 4. Given a static baseline α, we propose as outlined on the left side in Table 1, to adapt α_t during a mis-pricing sequence in such a way that $(1 - \Pi_{\tau=0}^{k} \alpha_\tau) = k * (1-\alpha)$. Hence, $\alpha_t = 0$ after $k = \left\lceil \frac{1}{(1-\alpha)} \right\rceil$ mis-pricing iterations, at which point smoothing stops, as $\tilde{\pi}^t = \pi^t$, which forces the end of a mis-pricing sequence.

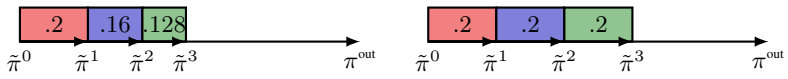


Fig. 4: At the t-th mis-pricing iteration, $||\tilde{\pi}^0 - \pi^{\text{out}}||$ is cut by a factor $(1 - \Pi_{\tau=0}^{t} \alpha_\tau)$.

So far we assumed a static baseline α provided as an input. Let us now consider how the user could be free from having to tune α for his application. In deriving an auto-adaptive α-schedule, one could consider using high α while the out-point is believed to be a bad approximation of π^*, and reducing α as the method converges, which is measured by smaller gaps $(\eta^t - \hat{L})$, and the purpose becomes to prove optimality. Alternatively, one could rely on local information, as we do. We propose to decrease α when the sub-gradient at the sep-point indicates that a larger step from the in-point would further increase the dual bound (i.e., when the angle of the ascent direction, g^{sep}, as defined in (22), and the direction $(\pi^{\text{out}} - \pi^{\text{in}})$ is less than $90°$), and vice versa. We outline this procedure on the right side in Table 1, while Figure 5 offers an illustration. The functions that we use for increasing and decreasing α are illustrated in Figure 6. We add to α ten percent of the amount that is missing for it to reach the value one when increasing. Non-symmetrically, we remove 0.1 from α when decreasing it. I.e., $f_{\text{incr}}(\alpha_t) = \alpha_t + (1 - \alpha_t) \cdot 0.1$, while $f_{\text{decr}}(\alpha_t) = \max\{0, \alpha_t - 0.1\}$. Such a choice for these functions comes from the following observation we had during preliminary computational experiments: α_t should asymptotically approach value one when we need to

Table 1: α-schedule in a mis-pricing sequence for a given initial α (on the left) and dynamic α-schedule based on sub-gradient information for a given initial α (on the right).

Step 0: $k \leftarrow 1$, $\pi^0 \leftarrow \pi^{\text{in}}$	Step 0: Let $\alpha_0 \leftarrow \alpha$, $t \leftarrow 0$.
Step 1: $\tilde{\alpha} \leftarrow [1 - k * (1 - \alpha)]^+$	Step 1: Call pricing on $\pi^{\text{sep}} = \alpha_t \pi^{\text{in}} + (1 - \alpha_t)\pi^{\text{out}}$.
Step 2: $\pi^{\text{sep}} = \tilde{\alpha}\, \pi^0 + (1 - \tilde{\alpha})\, \pi^{\text{out}}$	Step 2: If a mis-pricing occurs, start the mis-pricing schedule.
Step 3: $k \leftarrow k + 1$	Step 3: Let g^{sep} be the sub-gradient in sol $z_{\pi^{\text{sep}}}$.
Step 4: call the pricing oracle on π^{sep}	Step 4: If $g^{\text{sep}}(\pi^{\text{out}} - \pi^{\text{in}}) > 0$, $\alpha_{t+1} \leftarrow f_{\text{incr}}(\alpha_t)$; otherwise, $\alpha_{t+1} \leftarrow f_{\text{decr}}(\alpha_t)$.
Step 5: if a mis-pricing occurs while $\tilde{\alpha} > 0$, goto Step 1; else, return $z_{\pi^{\text{sep}}}$.	Step 5: Let $t \leftarrow t + 1$, solve the master and goto Step 1.

increase stabilization, but we should be able to decrease α_t sufficiently fast when strong stabilization is not needed anymore.

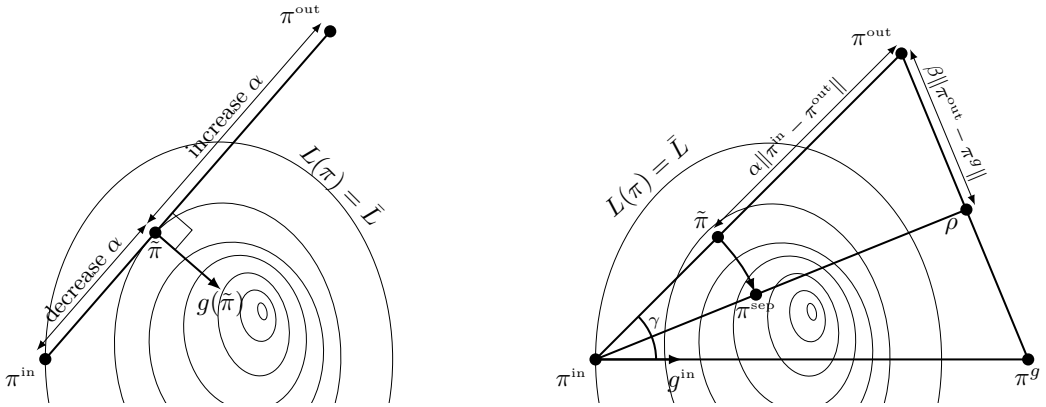


Fig. 5: Left: Auto-adaptive α-schedule: condition for increasing or decreasing α depending on whether the π^{sep} is below or beyond the threshold value $\tilde{\pi}$. Right: Directional Smoothing: combining sub-gradient and Wentges smoothing

5. Hybridizing smoothing with an ascent method

With a pure smoothing technique, the price vector is defined by taking a step $(1 - \alpha)$ from the in-point in the direction of the out-point: $\pi^{\text{sep}} = \pi^{\text{in}} + (1 - \alpha)(\pi^{\text{out}} - \pi^{\text{in}})$. Here, we consider modifying the direction $(\pi^{\text{out}} - \pi^{\text{in}})$ by twisting it towards the direction of ascent observed in π^{in}. The resulting method can be viewed as a hybridization of column generation with a sub-gradient method. When Wentges's rule (25) is used, the resulting hybrid method is related to the Volume algorithm [2] where π^t is obtained by taking a step from $\hat{\pi}$ in a direction that combines previous iterate information with the current sub-gradient.

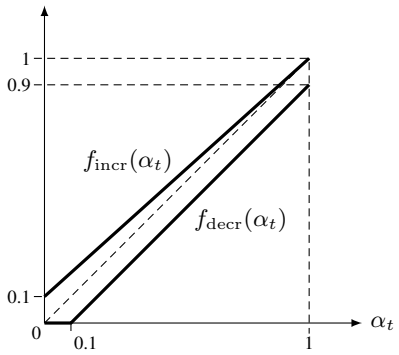


Fig. 6: Functions for increasing or decreasing α

Table 2: Directional smoothing with parameter β. Below, π^g denotes the dual price obtained by taking a step in the sub-gradient direction g^{in} computed in π^{in} in the standard smoothing scheme; while ρ denotes the dual price obtained by taking a unit step in the direction obtained as a weighted combination of g^{in} and $(\pi^{\text{out}} - \pi^{\text{in}})$.

Step 1: $\tilde{\pi} = \pi^{\text{in}} + (1-\alpha)(\pi^{\text{out}} - \pi^{\text{in}})$
Step 2: $\pi^g = \pi^{\text{in}} + \frac{g^{\text{in}}}{\|g^{\text{in}}\|} \|\pi^{\text{out}} - \pi^{\text{in}}\|$
Step 3: $\rho = \beta\pi^g + (1-\beta)\pi^{\text{out}}$
Step 4: $\pi^{\text{sep}} = \left(\pi^{\text{in}} + \frac{\|\tilde{\pi} - \pi^{\text{in}}\|}{\|\rho - \pi^{\text{in}}\|} (\rho - \pi^{\text{in}})\right)^+$

The hybrid procedure, that we call *directional smoothing*, is outlined in Table 2 and illustrated in the right part of Figure 5. Let g^{in} denote the sub-gradient associated to oracle solution $z_{\pi^{\text{in}}}$. In Step 1, $\tilde{\pi}$ is computed by applying smoothing. In Step 2, π^g is computed as the point located on the steepest ascent direction at a distance from π^{in} equal to the distance to π^{out}. In Step 3, a rotation is performed, defining target ρ as a convex combination between π^g and π^{out}. Then, in Step 4, the sep-point is selected in direction $(\rho - \pi^{\text{in}})$ at the distance from π^{in} equal to $\|\tilde{\pi} - \pi^{\text{in}}\|$ and it is projected on the positive orthant. As is the case with non-directional smoothing, using modified dual prices can result in mis-pricing. When this arises, we switch off directional smoothing by setting $\beta = 0$ in the next iterates of the mis-pricing sequence.

In the absence of mis-pricing, we fix the value of parameter β using a simple procedure that is based on our computational experiments that showed that the larger the angle γ between vectors $(\pi^{\text{out}} - \pi^{\text{in}})$ and $(\pi^g - \pi^{\text{in}})$, the smaller the value for β should be. This is especially true at the end of column generation when angle γ is close to $90°$, and large value β results in many mis-pricings. Based on this observation, our proposal is to use an adaptive β-schedule by setting $\beta = \cos\gamma$. As γ is always less than $90°$, since vector $(\pi^{\text{out}} - \pi^{\text{in}})$ is an ascent direction, $\beta \in (0, 1]$.

6. Piecewise penalty function approaches

As formulated in (23), stabilization can be obtained by adding a penalty function to the dual objective that drives the optimization towards dual solutions that are close to a *stability center*, $\hat{\pi}$, which we define as the incumbent dual solution that gave the best Lagrangian bound so far. The stability function implement a penalty

proportional to the norm of the deviation from the stability center. To remain with a modified master program that takes the form of a linear program, we restrict our attention to the $\mathcal{L}_1$ norm: $||\pi-\hat{\pi}||_1 = \sum_i |\pi_i-\hat{\pi}_i|$, possibly with weights indexed by i or the $\mathcal{L}_\infty$ norm: $||\pi-\hat{\pi}||_\infty = \max_i |\pi_i-\hat{\pi}_i|$, or a combination of these two. Although there are many possible ways to define such stability functions, our numerical experience indicates that the efficiency of the scheme can depend more on the parameterization of such function than on the function definition it-self.

We have restricted our attention to 3-piece and 5-piece piecewise linear functions that are further assumed to be symmetric, where the middle piece is defined by a tolerance (or confidence) interval with penalty zero up to a threshold on the $\mathcal{L}_\infty$ norm; beyond the confidence interval, the $\mathcal{L}_1$ norm is used as a measure of deviation. Our experiments have shown that using the $\mathcal{L}_\infty$ norm penalty function instead of the $\mathcal{L}_1$ norm leads to poor performance in terms of stabilization because some dual component can take values that are far away for the stability center, while others remain very close. However, using both $\mathcal{L}_1$ norm and $\mathcal{L}_\infty$ norm in combination performs sometimes better than the $\mathcal{L}_1$ norm alone, but it requires tuning two parameters instead of one of the two. Hence, we focus on using the $\mathcal{L}_1$ norm alone to measure deviation beyond the confidence interval.

Our 3-piece function, minimizes the $\mathcal{L}_1$ norm of the deviation with weight γ, once outside a confidence interval of size Δ. I.e., it penalizes each dual variable deviation $|\pi_i - \hat{\pi}_i|$ individually by a factor γ when it lies outside of the confidence interval $[\hat{\pi}_i - \Delta, \hat{\pi}_i + \Delta]$. The confidence interval could alternatively be defined in relative value around $\hat{\pi}_i$ as $[\hat{\pi}_i(1-\delta), \hat{\pi}_i(1+\delta)]$, but such approach leads to a very small confidence interval when $\hat{\pi}_i$ has small absolute value. Hence, a confidence interval defined by an absolute value parameter is easier to control. To account for individual variations between the complicating constraints, we use a normalization factor, $|a'_i|$, over γ, with $a'_i = a_i$, i.e., the right-hand-side of the corresponding constraint, if this right-hand-side is non-zero, and $a'_i = 1$ otherwise. Overall, the penalization factor of a dual variable π_i is only differentiated by the normalization constant $|a'_i|$. Our experience with "individual" penalty functions (component wise dependent factors Δ_i and γ_i) resulted in poor performance, besides it was difficult to parameterize these values.

Hence, at iteration t, the modified dual restricted master and the associated bi-dual take the form:

$$
\begin{aligned}
\max \eta \; - \; & \textstyle\sum_i (|a'_i|\,\gamma\,\pi_i^- + |a_i|\,\gamma\,\pi_i^+) \\
& \textstyle\sum_i \pi_i (A_i z^\tau - a_i) + \eta \le c z^\tau \quad \tau = 1,\dots,t \\
& \pi_i - \pi_i^+ \le \hat{\pi}_i^{t-1} + \Delta \quad \forall i \\
& \pi_i + \pi_i^- \ge \hat{\pi}_i^{t-1} - \Delta \quad \forall i \\
& \pi_i, \pi_i^-, \pi_i^+ \in \mathbb{R}_+ \quad \forall i
\end{aligned}
\qquad\qquad
\begin{aligned}
\min & \textstyle\sum_{\tau=1}^t (c z^\tau)\,\lambda_\tau + \sum_i (\hat{\pi}_i^{t-1} + \Delta)\,\rho_i^+ \\
& \qquad - \textstyle\sum_i (\hat{\pi}_i^{t-1} - \Delta)\,\rho_i^- \\
& \textstyle\sum_{\tau=1}^t (A_i z^\tau)\lambda_\tau + \rho_i^+ - \rho_i^- \ge a_i \quad \forall i \\
& \textstyle\sum_{\tau=1}^t \lambda_\tau = 1 \\
& 0 \le \rho_i^+, \rho_i^- \le |a'_i|\gamma \quad \forall i \\
& \lambda_\tau \ge 0 \quad \tau = 1,\dots,t.
\end{aligned}
$$

where $\hat{\pi}^t$ denote the best dual solution up to iteration t of the column generation procedure. Note that with the $\mathcal{L}_\infty$ norm, the bounds: $\rho_i^+, \rho_i^- \le |a'_i|\gamma$, on the artificial variables would be replaced by a normalization condition : $\sum_i (\rho_i^+ + \rho_i^-) \le \gamma$.

Let $[\mathrm{M}_S^t]$ denote the modified master as presented above. If $[\mathrm{M}_S^t]$ admits no more negative reduced cost columns, there remains to check whether its solution is optimal for the unmodified master, [M].

Observation 2

(i) *The optimal value of* $[M_S^t]$ *defines a dual bound on the optimal value of* $[M^t]$.

(ii) At any iteration t, if $(\lambda^t, {\rho^+}^t = 0, {\rho^-}^t = 0)$ is an optimal solution for $[M_S^t]$, i.e., if the artificial variables ρ^+ and ρ^- take value zero in the solution to $[M_S^t]$, then λ^t is an optimal solution to the associated unmodified master $[M^t]$.
(iii) In particular, when there are no more negative reduced cost columns for $[M_S^t]$, if the artificial variables ρ^+ and ρ^- take value zero in the current solution, the current λ solution defines an optimal solution for $[M]$.

Indeed, to observe (i) note that any feasible solution to $[\mathrm{M}^t]$ is also feasible for $[\mathrm{M}_S^t]$ where it takes the same cost value. For (ii), note that λ^t is feasible for $[\mathrm{M}^t]$ and therefore defines a primal bound on $[\mathrm{M}^t]$; but the value of solution λ^t also defines a dual bound on $[\mathrm{M}^t]$ as seen in (i). Furthermore, for (iii) note that if $\rho^+ = \rho^- = 0$, the bounds on the artificial variables are not binding, and therefore their dual solution are zero, so the pricing problem are the same for both $[\mathrm{M}_S^t]$ and $[\mathrm{M}^t]$. When the modified master $[\mathrm{M}_S]$ has converged to a solution with positive values ρ^+ or ρ^-, it indicates that the penalty function has driven the search away from the optimal solution of the unmodified master $[\mathrm{M}]$. Then, one needs to update the penalty function parameters so that the impact of the penalties tend to vanish progressively.

The only two parameters of our 3-piecewise linear penalty function are γ and Δ. Below, we present two schemes to update these parameters. Both schemes rely on a single input parameter, which we denote κ. Thus, our proposal amounts to define self-adjusting γ and Δ-schedules. However, unlike for the parameterization of smoothing, we need here to select the input κ, that can be critical for the efficiency of the stabilization. We say that a parameter is *critical* if its value affects the performance of the stabilization scheme by a "large margin" (say a factor ten in the number of iterations for instance) and the "best parameter value" differs for different problems. For all non-critical parameters , we used the same value for all problems tested in our computational experiments.

We developed a scheme for automated parameter adjustment that is "curvature-based": the idea is to approximate the quadratic function $q_i^c(\pi) = |a_i'| \cdot (\pi_i - \hat{\pi}_i)^2/c$, where c is the parameter which represents the function's curvature. Parameters γ and Δ are chosen in such a way that the resulting 3-piecewise linear penalty function is an outer approximation of $q_i^c(\pi)$ and tangent to it in 3 points. The penalty function on iteration t is depicted in Figure 7. Let π_{diff}^t be the average component-wise difference between π^t and $\hat{\pi}^{t-1}$,

$$\pi_{diff}^t = \frac{\sum_{i \in I} |\pi_i^t - \hat{\pi}_i^{t-1}|}{\mid I \mid},$$

where I is the set of indices of complicating constraints in the master and π^t is the solution to $[\mathrm{DM}_S^t]$ at iteration t, for $t \geq 2$, while π_{diff}^1 is the average component-wise difference between the dual solution on the first iteration and zero vector of the appropriate dimension. We set γ and Δ values at iteration t as follows:

$$\Delta^t = \frac{\pi_{diff}^t}{2} \text{ and } \gamma^t = \frac{\pi_{diff}^t}{c}.$$

While the curvature, c, is set to π_{diff}^1/κ. So, κ is the only parameter of the stabilization scheme. If some artificial variables ρ_i^+ or ρ_i^- remain in the optimal solution to $[\mathrm{M}_S]$, the value of curvature c is increased, so as to reduce the weight of the penalty function in the master objective. It is multiplied by 10 in our procedure, but this factor is not critical.

The above "curvature-based" parameterization of the penalty function decreases the number of critical parameters from two to one. Its drawback however is to lead to modifying the restricted master objective coefficients and

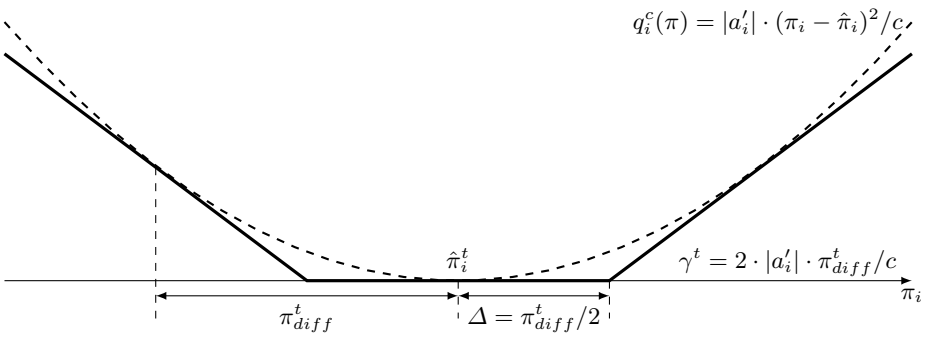


Fig. 7: Curvature based penalty function on iteration t

artificial variable bounds at each iteration, making re-optimization more computationally demanding. Moreover, parameter κ remains very critical. A "small" value can make stabilization very weak, while a "large" value can make column generation converge to a solution which is "far away" from the true optimum of the unmodified master. Our attempts to come up with an auto-adjusting scheme for the curvature c (or equivalently for parameter κ) were not successful. Such an auto-adjusting scheme remains an interesting research goal for the future.

The alternative parameter adjustment scheme that we considered is an "explicit" setting of parameters γ and Δ. Our preliminary experiments on a range of different problems showed that a good setting of parameter γ is 0.9. Then, Δ is set to π_{diff}^1/κ, where π_{diff}^1 is defined as above. Again, κ is the only critical parameter to set. During the course of the column generation procedure, parameter Δ is divided by 2 each time all the artificial variables ρ_i^+ and ρ_i^- are absent from the solution of the restricted master. At the end of the column generation procedure, if there are artificial variables ρ_i^+ or ρ_i^- in the solution, the value of γ is decreased (reducing the impact of the penalty function in the objective allows the procedure to move to better solutions for the unmodified master): we divide it by 10. These two settings (division of Δ by 2 and division of γ by 10) are not critical.

Although, this "explicit" penalty function does not "stabilize as much" as the "curvature-based" penalty function, as shown by the comparison of the number of iterations in our experiment, it has the following advantage. Bounds on the artificial variables remain unchanged over the course of the algorithm (except if artificial variables remain in the optimal solution), thus the re-optimization of the restricted master is easier for the LP solver. This is beneficial for the overall running time.

The modified restricted dual and primal master formulations and the above observations can easily be extended to a 5-piecewise linear penalty function. For the "curvature-based" parameterization, one can define the 5-piece variant in which each component of the penalty function is tangent to $q_i^c(\pi)$ in 5 points: $\hat{\pi}_i^t$, $\hat{\pi} \pm \pi_{diff}^t$, $\hat{\pi} \pm \pi_{diff}^{t-1}$. However, our computational experiments revealed that the 5-piece variant does not bring any significant improvement in the computational time. For the "explicit" parameterization, we consider two values of γ and Δ: the inner and the outer. We fix the inner γ to 0.1, while the inner Δ is set to $\Delta/10$. Only inner Δ is divided by 2 each time all the artificial variables are absent from the solution of the restricted master. All these settings are not critical. With the "explicit" parameterization, we observed that the performance obtained with the 5-piece variant

is better than with the 3-piece variant; however when the penalty function is used in combination with smoothing, the 3-piece variant performs just as well as the 5-piece.

7. Application test bed

We experiment the above stabilization techniques on a significantly large scope of applications and data sets. We briefly describe these problems and instances. To increase representativity, for each problem, we chose instances with both good and bad convergence properties: i.e. we measure the number of column generation iterations relative to the number of master constraints; when this ratio is low, we consider that convergence is good; when it is high, convergence is bad; we select instances at both end of this spectum.

All experiments were run on a Dell PowerEdge 1950 (32Go, Intel Xeon X5460, 3.16GHZ). To solve the restricted master LP, Cplex 12.4 was used as an LP solver: using Cplex default option that automatically selects between primal and dual Simplex. No time limit was set.

7.1 Generalized Assignment

In the generalized assignment problem, we are given n tasks to be assigned to m machines (or agents), where each machine i has capacity u_i, and each task j, when assigned to machine i, uses d_{ij} units of resource and costs c_{ij}, for $j = 1, \ldots, n$, and $i = 1, \ldots, m$. The problem consists in assigning each task to exactly one machine, such that the total resource usage on each machine does not exceed its capacity, and the total assignment cost is minimized. This application involves **multiple distinct subproblems**. A solution of the pricing subproblem consists of a set of tasks to be assigned to one of the machines, while respecting its capacity. To solve this binary knapsack pricing subproblem, we used the *minknap* solver of Pisinger [23]. This oracle can by downloaded from `http://www.diku.dk/~pisinger/minknap.c`.

The instances of the generalized assignment problem were taken from the OR-Library [3] and the paper by Yagiura et al. [29]. Each instance name is in the C-m-n format, where C is instances class, m is the number of machines (or agents), and n is the number of jobs (or tasks). We used the following instances: D-20-100, E-20-100, D-10-100, E-10-100, D-20-200, E-20-200, D-40-400, E-40-400, D-5-100, E-5-100, D-10-200, E-10-200, D-20-400, E-20-400, D-5-200, E-5-200, D-10-400, E-10-400.

7.2 Multi-Item Multi-Echelon Lot-Sizing

The multi-item lot-sizing problem consists in planning production so as to satisfy demands d_t^k (either from stock or from production) for item $k = 1, \ldots, K$, over a discrete time horizon with periods $t = 1, \ldots, T$. The production of an item involves production stages (echelons), $e = 1, \ldots, E$, each of which takes place on a different machine that can only process one product in each period (under the so-called small bucket assumption). Usage of a machine at a time period entails a setup cost. The objective is to minimize the sum of all production, setup, and storage costs. This application involves **multiple distinct subproblems**. The pricing subproblem is a uncapacitated single-item lot-sizing problem. It is solved using a standard dynamic programming algorithm of complexity $O(T^2)$ in the case there is a single-echelon. For instances with multiple echelons, such dynamic program can be

extended in an algorithm of complexity $O(ET^4)$ (see Zangwill [30]), which is what we use.

Our instances have been generated randomly using uniform distributions. The size of an instance is determined by triple (E, K, T), where E is the number of echelons, K is the number of items, and T is the number of periods. Setup costs are generated in interval $[20, 100]$, production costs are zero, and storage costs h_e^k are generated as $h_{e-1}^k + \gamma$, where γ is in the interval $[1, 5]$, and $h_0^k = 0$. For each period, there is a positive demand for 3 items on average. Demands are in the interval $[10, 20]$. We generated one instance for each of the following size triples: $(1, 20, 100)$, $(1, 40, 200)$, $(1, 10, 100)$, $(1, 20, 200)$, $(1, 40, 400)$, $(2, 20, 50)$, $(2, 10, 50)$, $(2, 20, 100)$, $(2, 10, 100)$, $(3, 20, 50)$, $(3, 10, 50)$, $(3, 20, 100)$, $(3, 10, 100)$, $(5, 20, 50)$, $(5, 10, 50)$, $(5, 20, 100)$, $(5, 10, 100)$.

7.3 Multi-Commodity Flow

In the multi-commodity flow problem, we are given a directed graph and a set of commodities to be routed in it. Each commodity has its own flow balance constraint to satisfy; while the arc capacities are shared by all commodities. The objective is to minimize the total cost of the flows in the arcs. This application involves **multiple distinct subproblems**. The pricing subproblem associated to a pair "source-sink" of a commodity is the shortest path problem. To solve all shortest path problems for a given commodity, we used the iterative shortest path tree algorithm from [25] with at most 5 iterations.

Our instances are six freight railcar routing instances taken from [25]. They involve 11 commodities corresponding to difference railcar types, and a planning horizon going from 40 to 140 days. Graphs contain from approximately 110 thousand nodes and 1.7 million arcs to 340 thousand nodes and 7.5 million arcs. Convergence is slower for instances with larger time horizon.

7.4 Parallel Machine Scheduling

In the parallel machine scheduling problem, we are given n jobs to be processed by m identical machines, where job j has a processing time p_j, a due date d_j, and a weight w_j, for $j = 1, \ldots, n$. The problem consists of assigning a sequence of jobs to each machine, each job j being assigned to exactly one machine, while minimizing $\sum_{j=1}^{n} w_j T_j$, where $T_j = \max\{0, C_j - d_j\}$ is the tardiness of job j, and C_j is the completion time of job j, for $j = 1, \ldots, n$. This application, denoted as $P \,||\, \sum w_j T_j$, involves **multiple identical subproblems**. A solution of the pricing subproblem consists of a sequence of jobs to be assigned to one of the machines, possibly processing a job more than once. To solve it, we apply a standard dynamic programming algorithm of complexity $O(nT)$, where T is the time horizon length: $T = \lceil \sum_{j=1}^{n} p_j/m \rceil + p_{\max}$. Here $p_{\max}$ is the maximum processing time of a job.

Our instances were generated similarly to single machine weighted tardiness scheduling instances of the OR-Library [3]. Instances are generated using the procedure of Potts and van Wassenhove [24]: integer processing times p_j are uniformly distributed in interval $[1, p_{\max}]$ and integer weights w_j in $[1, 10]$ for jobs j, $j = 1, \ldots, n$, while integer due dates have been generated from a uniform distribution in $[P(1-TF-RDD/2)/m, P(1-TF+RDD/2)/m]$, where $P = \sum_j p_j$, TF is a tardiness factor, and RDD is a relative range of due dates. Instance sizes are determined by a triple $(m, n, p_{\max})$. We generated instances of sizes determined by triples $(1, 50, 100)$, $(1, 100, 100)$, $(2, 50, 100)$, $(2, 100, 100)$, $(4, 100, 100)$, $(4, 200, 100)$. For each size, one instance was generated for

each of the following pairs (TF, RDD): $(0.2, 0.2)$, $(0.4, 0.4)$, $(0.6, 0.6)$, $(0.8, 0.8)$, $(1.0, 1.0)$. Therefore, in total, 30 instances were generated.

7.5 Capacitated Vehicle Routing

The capacitated vehicle routing problem is defined in a complete graph with $n + 1$ nodes: a depot and n customers, each of which requiring a delivery quantity d_i, $i = 1, \ldots, n$. The problem consists in finding a set of K delivery routes, starting and ending at the depot, that visit every customer exactly once, where the sum of the demands of the customers visited by each route does not exceed the vehicle capacity Q. The objective is to minimize travel costs defined in an $(n + 1) \times (n + 1)$ matrix $[c_{ij}]$. This application involves **multiple identical subproblems**. The pricing subproblem consists in building a single route that respects the vehicle capacity but may visit a customer more than once. To limit cycling and get stronger Lagrangian bound, we consider so-called ng-route as subproblem solution: a set of neighbors is associated to each customer; a customer i that has already been visited can only appear again in the route after visiting a customer j that does not include i as its neighbor. For solving the pricing subproblem, we used the algorithm of Baldacci et al. [1], considering the three closest customers to define neighborhood.

The instances of the capacitated vehicle routing problem that we use are classic symmetric instances. They can be downloaded, for example, at `http://branchandcut.org/VRP/data`. Names of the instances are in the format C-nN-kK, where C is the instance class, $N = n + 1$, and K is the number of identical vehicles. We used instances A-n63-k10, A-n64-k9, A-n69-k9, A-n80-k10, B-n50-k8, B-n68-k9, E-n51-k5, E-n76-k7, E-n76-k8, E-n76-k10, E-n76-k14, E-n101-k8, M-n121-k7, M-n151-k12, M-n200-k16, P-n50-k8, P-n70-k10, P-n76-k5, P-n101-k4.

7.6 Multi-Activity Shift Scheduling

In the multi-activity shift scheduling problem, n activities must be scheduled in a planning horizon divided into m periods. The number b_{ij} of employees that is required at each period $i = 1, \ldots, m$ for each activity $j = 1, \ldots, n$ is set. For each employe $e \in E$, a set Ω of feasible shifts is implicitly defined by a set of rules (accounting for skills, preferences, availability, work regulation, and ergonomic considerations); it is identical for all employees in our instances. Each shift $s \in \Omega$ has a cost c_s. The difference between the number of employees assigned to activity j in period i and b_{ij} is penalized with coefficients o_{ij} for over-covering, and u_{ij} for undercovering. The problem consists in assigning a feasible shift for each employee such that the sum of over-covering, under-covering, and shift costs is minimized. This application involves **multiple identical subproblems**. The pricing subproblem for an employee consists in finding a feasible shift with the minimum cost. In [6], it is shown that the set of feasible shifts can be described by context-free grammars. This description allows one to solve a pricing subproblem by dynamic programming which runs linearly in the number of hyper-arcs of the graph modeling transitions, which what we implemented?

The instances that we used come from the paper by Demassey et al. [8]. In these instances, there are 96 periods and up to 20 identical employees. We used instances number 1, 4, 7, and 10, with 2, 4, 6, 8, and 10 activities. Therefore, there are 20 instances in total.

7.7 Bin Packing

In the bin packing problem, we are given n items to be assigned to bins of size S, each item i having a size s_i, for $i = 1, \ldots, n$. The problem consists in assigning each item to exactly one bin such that the total size of all items assigned to a bin does not exceed S, and the number of bins used is minimized. This application involves **multiple identical subproblems**. A solution of the pricing subproblem consists of a set of items to be assigned to one bin respecting its size. For solving such binary knapsack pricing subproblem, we used the *minknap* solver of Pisinger [23] as for Generalized Assignment.

Our instances of the bin packing problem have been generated randomly using uniform distributions. Instance classes "a2", "a3", and "a4" (the number refers to the average number of items per bin) contain instances with bin capacity equal to 4000 where item sizes are in intervals $[1000, 3000]$, $[1000, 1500]$, and $[800, 1300]$, respectively. We generated four instances for each class: two instances with 400 items and two instances with 800 items.

7.8 Cutting Stock

The cutting stock problem is similar to the bin packing problem. The only difference is that each item i have d_i copies of the same size s_i. So the problem consists in distributing all copies of all items between bins. This application involves **multiple identical subproblems**. A solution of the pricing subproblem consists of the number of copies of each item assigned to one bin respecting its size. For solving such integer knapsack pricing subproblem, we used the same *minknap* solver of Pisinger [23]. For this, we transformed each instance of the bounded integer knapsack problem to an equivalent instance of the binary knapsack problem by binary expansion.

Our instances of the cutting stock problem have been generated randomly using uniform distributions. Instance classes "a2", "a3", "a4", and "a5" (the number refers to the average number of items per bin) contain instances with bin capacity equal to 10000 where item sizes are in intervals $[3000, 5000]$, $[2500, 4000]$, $[1500, 3500]$, and $[1000, 3000]$ respectively. Values d_i are uniformly distributed in interval $[1, 10]$. We generated six instances for each class (two instances with 250 items, two with 500 items, and two with 1000 items).

7.9 Vertex Coloring

In the vertex coloring problem, we are given a simple undirected graph. We need to label each vertex of the graph with a color such that no two adjacent vertices share the same color. The problem consists in coloring the graph using the minimum number of colors. This application involves **multiple identical subproblems**. A solution of the pricing problem consists in choosing a set of vertices which are non-adjacent to each other. So, the pricing problem here is the weighted stable set problem, or the weighted clique problem once seen in the complementary graph. To solve the pricing problem, we used either the *cliquer* solver by Östergård [20] (which can be downloaded from `http://users.aalto.fi/~pat/cliquer.html`), if the graph's density is large, or Cplex MIP solver, otherwise.

The instances that we used are from the DIMACS benchmark [14] and can be downloaded, for example, at `http://mat.gsia.cmu.edu/COLOR03/`. We used the following instances: `mulsol.i.2`, `mulsol.i.4`, `DSJC250.9`, `DSJC500.9`, `zeroin.i.1`, `zeroin.i.2`, `DSJC125.1`, `DSJC250.5`, `myciel6`, `myciel7`, `queen11_11`, `queen12_12`, `miles250`, `miles750`, `miles1500`, `mugg88_25`, `mugg100_1`. The *cliquer*

solver was used for solving the pricing problem for all instances except the last five. Convergence is slower for instances with larger average number of vertices in a stable set, i.e. with smaller density of the graph.

8. Numerically testing smoothing schemes

In the experiments we describe next, we assess numerically the stabilization effect of applying Wentges smoothing with static α-schedule versus our auto-adaptive schedule starting with $\alpha_0 = 0.5$. Additionally, we estimate the effect of using directional smoothing, with static and auto-adaptive value of parameter β, in combination with Wentges smoothing.

To tune the static α-value, we determine experimentally for each instance separately the best α (denoted $best$) among values in $\{0.1, 0.2, \ldots, 0.8, 0.9, 0.95\}$ as the value that yields the minimum total computing time of the column generation procedure used to solve the master LP. This is illustrated in Figure 8 on two different instances. One can note in this figure that the best α value depends highly on the application. Moreover, it differs a lot from one instance to the next even within the same application, as reflected by the range of best α reported in Table 3. Similarly, the best static β-value is determined by testing all β values in $\{0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$.

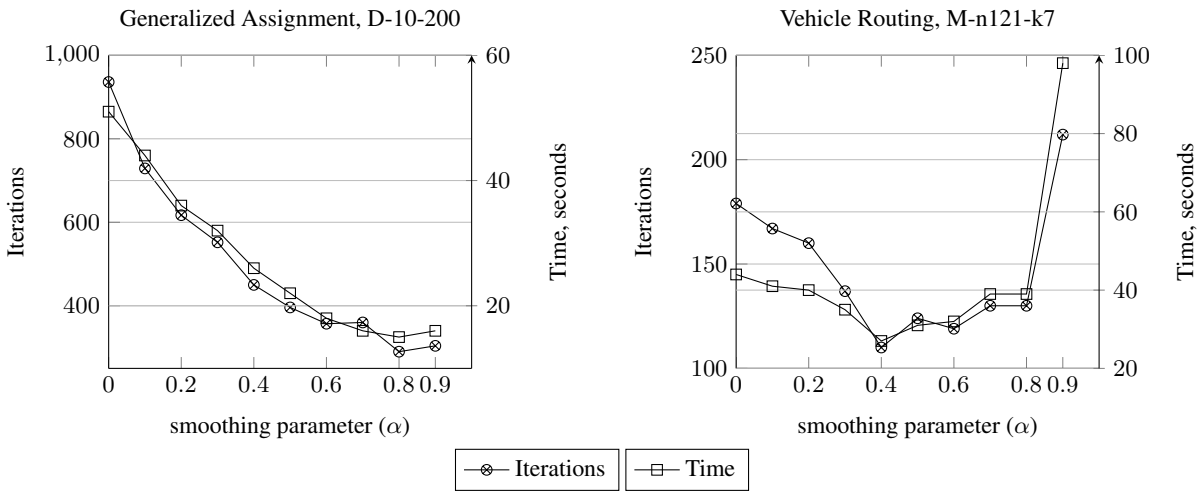


Fig. 8: Sensitivity of master iterations number and solution time to a static α

Table 3 gives an overview of column generation results with Wentges and directional smoothing. Five variants of column generation are compared: (i) that without any stabilization ($\alpha = 0$, $\beta = 0$), (ii) that with static Wentges stabilization ($\alpha = best$, $\beta = 0$), (iii) that with auto-adaptive Wentges stabilization ($\alpha = auto$, $\beta = 0$), (iv) that with combined automatic Wentges and static directional stabilization ($\alpha = auto$, $\beta = best$), and (v) that with combined automatic Wentges and directional stabilization ($\alpha = auto$, $\beta = auto$). The first column is the problem name; the second one reports the geometric mean of CPU time in seconds without stabilization; the third one gives the range of best α-values for each application; while the last four columns give the geometric means of the ratio

of the solution time of variant (i) against variant (ii), (iii), (iv), and (v), respectively. The first observation is that our automatic Wentges smoothing scheme manages to reproduce the results of the smoothing with instance-wise tuned parameter for the first six problems. For the last three problems, it is not the case. For the Bin Packing and Cutting Stock problems, we conjecture that this is due to the very high number of identical pricing subproblems (at least 250). Because of this, the pricing problem can have a large number of mutually "perpendicular" optimal solutions which are generated on subsequent column generation iterations. This results in a sub-gradient that is perpendicular to the "in-out" direction; then the sub-gradient provides no indication on how to update parameter α. The explanation for the Vertex Coloring problem case is different: as it will become clear from Table 4, automatically adjusted dual prices make the pricing problem harder to solve. The second observation is that directional smoothing can improve the convergence significantly for some applications. Self-adjusting scheme for parameter β does not always reproduce results for the tuned static parameter. However, the advantage of the former is that it never deteriorate results, whereas static setting of β may cause that. An interesting observation is that automatic directional smoothing "corrects" the poorer performance of automatic Wentges smoothing when combined with it for the Bin Packing and Cutting Stock problems.

			Ratio of solution time of variant $\alpha, \beta = 0$ to			
	Time of	Range of	$\beta = 0$ and		$\alpha = auto$ and	
Application	$\alpha, \beta = 0$	best α	$\alpha = best$	$\alpha = auto$	$\beta = best$	$\beta = auto$
Generalized Assignment	75	$[0.3, 0.9]$	3.91	3.83	9.58	7.93
Multi-Echelon Lot Sizing	87	$[0.6, 0.95]$	3.41	4.09	4.46	5.99
Multi-Commodity Flow	914	$[0.1, 0.9]$	2.67	2.73	3.79	3.11
Parallel Machine Scheduling	34	$[0.7, 0.95]$	2.96	2.91	2.42	3.61
Capacitated Vehicle Routing	5.5	$[0.2, 0.8]$	1.57	1.55	1.40	1.54
Multi-Activity Shift Scheduling	4.7	$[0.7, 0.95]$	1.73	1.70	1.68	1.71
Bin Packing	4.1	$[0.2, 0.9]$	2.18	1.69	1.94	1.81
Cutting Stock	3.7	$[0.1, 0.95]$	1.16	0.98	1.43	1.32
Vertex Coloring	2.2	$[0.3, 0.8]$	1.22	1.01	(*)	(*)

Table 3: Overview of results for column generation with Wentges and directional smoothing; a (*) denotes cases where the time limit was reached

In Figure 9, we plot the behaviour of α and β values in the course of the algorithm using the self-adjusting schemes, versus the best static α value, for representative instances. One can identify two stages in the solution process. The first stage ends with the first major pick, when the (possibly artificial) columns used to initialize the restricted master get out of the solution. A steep decreasing of the α value represents a mis-pricing sequence, while a slower decrease or increase represents an adjustment based on sub-gradient information. The β value gets small when the gradient direction in $\hat{\pi}$ makes a large angle with the direction to π^{out}. For the last Lot Sizing instance, the self-adjusting scheme is much more efficient than a static α. In this instance, a large α is desirable at the beginning while a small α value is best towards the end. This desirable setting cannot be approximated by a fixed α. This explains the fact that, for the Lot Sizing problem, automatic smoothing performs noticeably better than smoothing with tuned parameter, as shown in Table 3.

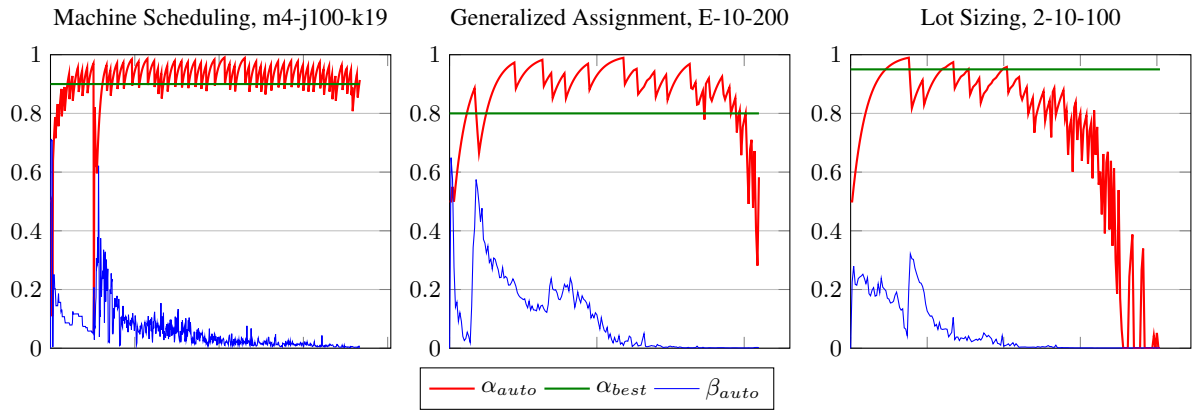


Fig. 9: Behaviour of values α and β in the course of the column generation procedure using self-adjusting schemes

In Table 4, we give more details on the column generation performance under Wentges and directional smoothing. Comparing different variants against the basic approach without any stabilization ($\alpha = 0$, $\beta = 0$), we report ratios of geometric means for the following statistics: `It` is the number of iterations in column generation; `Sp` is the number of calls to the pricing subproblems; `Cl` is the number of columns generated; `T` is the solution time. Note that the larger is the difference between `It` and `Sp`, the more often mis-pricing occurs. Statistics `Cl` is different from `Sp` because generated columns may already be present in the master, in this case they are not counted in `Cl`. The first observation here is that, for most problems, smoothing improves solution times by a larger factor than the number of iterations. This is due to the fact that the master requires more time to solve towards the end when there are many columns and hence fewer iterations means easier master on average. Note that greater time improvement is not observed for the Vehicle Routing, Cutting Stock and Vertex Coloring problems, for which pricing subproblems become harder when stabilization is applied. It occurs that for the Vertex Coloring problem, the automatic smoothing makes pricing subproblems even harder than static parameter smoothing. This explains worse results for the former, as the improvement in the number of iterations is similar. We do not give results for the directional smoothing applied to the Vertex Coloring problem because the pricing subproblems become so difficult for some instances that the column generation could not be terminated. From Table 4 we can also understand why automatic directional smoothing is more "consistent" than with the static parameter β. This is due to the fact that self-adjusting scheme for value β proposed in the paper can yield an significant decrease in the number of mis-pricings.

9. Combining smoothing with a penalty function

In the experiment reported here, we compare numerically automatic smoothing stabilization scheme, to a piecewise linear penalty function stabilization scheme, and to the combination of the two stabilization schemes. Five variants of column generation are compared: (i) that with automatic smoothing (`Smooth`); (ii) that with "curvature-based" 3-piecewise linear penalty function with tuned parameter κ (`Curv`); (iii) that with combination of the two previous stabilization methods (`SmoothCurv`); (iv) that with "explicit" 3-piece and 5-piece linear

Application	Ratio of variant $\alpha, \beta = 0$ against															
	$\beta = 0$ and								$\alpha = auto$ and							
	$\alpha = best$				$\alpha = auto$				$\beta = best$				$\beta = auto$			
	It	Sp	Cl	T	It	Sp	Cl	T	It	Sp	Cl	T	It	Sp	Cl	T
Generalized Assignment	2.96	2.69	3.16	3.91	3.00	2.93	3.32	3.83	3.90	1.55	6.34	9.58	3.99	3.63	4.83	7.93
Multi-Echelon Lot Sizing	2.26	1.89	3.26	3.41	2.42	2.26	3.54	4.09	3.29	2.11	5.10	4.46	3.29	2.98	4.86	5.99
Multi-Commodity Flow	1.25	1.15	1.32	2.67	1.57	1.57	1.30	2.73	1.76	1.76	1.22	3.79	1.66	1.66	1.24	3.11
Parallel Machine Scheduling	2.22	2.12	2.19	2.96	2.16	2.12	2.14	2.91	2.33	1.63	2.05	2.42	2.47	2.40	2.43	3.61
Capacitated Vehicle Routing	1.65	1.59	2.01	1.57	1.67	1.61	2.37	1.55	1.60	1.47	2.51	1.40	1.68	1.60	2.56	1.54
Multi-Activity Shift Scheduling	1.74	1.57	1.74	1.73	1.66	1.63	1.66	1.70	1.73	1.47	1.73	1.68	1.67	1.63	1.67	1.71
Bin Packing	1.72	1.70	1.49	2.18	1.45	1.42	1.33	1.69	1.67	1.63	1.45	1.94	1.62	1.60	1.43	1.81
Cutting Stock	1.25	1.24	1.14	1.16	1.20	1.19	1.12	0.98	1.81	1.77	1.34	1.43	1.77	1.74	1.33	1.32
Vertex Coloring	1.45	1.40	1.38	1.22	1.39	1.31	1.36	1.01	(*)	(*)	(*)	(*)	(*)	(*)	(*)	(*)

Table 4: Detailed results for column generation with Wentges and directional smoothing; a (*) denotes cases where the time limit was reached

penalty function with tuned parameter κ (`Expl`); and (v) that with the combination of automatic smoothing and "explicit" penalty function stabilization (`SmoothExpl`). We do not present results for the Vertex Coloring problem since the application of linear penalty function stabilization sometimes makes the subproblem unsolvable in a reasonable time. Where automatic smoothing is used, we mean the combination of Wentges smoothing and directional smoothing, except for the Vehicle Routing and the Shift Scheduling problems where directional smoothing was not applied. For each application and each of the last four variants of column generation, we determined experimentally the best value for parameter κ for which the geometric mean (among all instances of the problem) of the master LP solution time was the lowest. This tuning is illustrated in Figure 10 for the Lot Sizing problem.

Table 5 gives an overview of column generation results for the 5 variants of column generation: `Smooth`, `Curv`, `SmoothCurv`, `Expl`, `SmoothExpl`. For the last two variants, we report results for 3-piece or 5-piece penalty functions, depending on which variant gave the better results. Columns `T` in Table 5 give the geometric mean of the ratio of the solution time of the corresponding variant against the basic approach without any stabilization. The main observation here is that the combination of two stabilization techniques: smoothing and penalty function, outperforms any of the two techniques applied separately; this is the case for all problems except Vehicle Routing. Note also that the "explicit" variant of the penalty function stabilization performs better than "curvature-based" variant for all applications except Lot-Sizing. Results for the Shift Scheduling, Bin Packing, and Cutting Stock problems are different from others. For these three applications, the overall effect of stabilization by penalty functions does not decrease running times noticeably or may increase it, even though the number of iterations decrease. This is due to the increase of the number of (artificial) variables in the master that is critical in making the master harder to re-optimize.

In Table 6, we give detailed results for two representative problems: Generalized Assignment and Lot Sizing. We can observe from the `It`, `Sp`, and `Cl` statistics that the "curvature-based" variant of the penalty function stabilization has larger stabilization effect than the "explicit" variant. However, as it was mentioned above, the restricted master becomes harder to solve for the former variant. Thus, the "curvature-based" outperforms the 'explicit" vari-

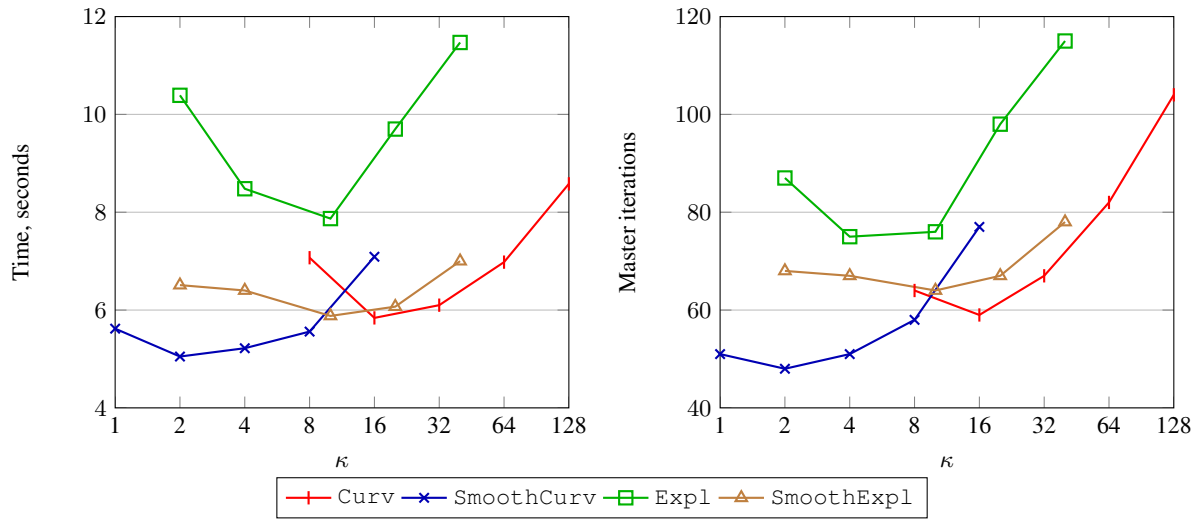


Fig. 10: Sensitivity of master iterations number and solution time to parameter κ for the Lot Sizing problem

Application	Smooth	Curv		SmoothCurv		Expl			SmoothExpl		
	T	κ	T	κ	T	pieces	κ	T	pieces	κ	T
Generalized Assignment	7.93	80	14.51	10	25.69	5	100	17.29	3	200	43.62
Multi-Echelon Lot Sizing	5.99	16	14.80	2	17.13	5	10	10.96	3	10	14.68
Multi-Commodity Flow	3.11	5	4.17	1	5.29	3	20	5.29	3	4	7.28
Parallel Machine Scheduling	3.61	4	1.77	2	4.19	5	10	2.20	3	10	5.74
Capacitated Vehicle Routing	1.54	1.6	1.89	0.1	1.83	5	1	2.34	3	2	2.02
Multi-Activity Shift Scheduling	1.71	25	0.73	25	1.33	3	0.2	1.06	3	0.1	1.73
Bin Packing	1.81	20	0.52	10	0.62	3	20	1.07	3	0.2	1.25
Cutting Stock	1.32	2	0.46	5	0.60	3	20	0.94	3	10	1.15

Table 5: Overview of results for column generation with smoothing and piecewise penalty function stabilization

ant in terms of the solution time only if most of the time is spent in solving the pricing subproblems, as in the case of the Lot Sizing problem. Another important observation is that using 5-piecewise linear penalty function significantly improves results for the variant `Expl`, but does not improve and sometimes even deteriorate results for the variant `SmoothExpl`.

In Figure 11, we provide performance profiles for five stabilization schemes: (i) no-stabilization; (ii) automatic Wentges smoothing; (iii) combined automatic Wentges and directional smoothing; (iv) piecewise penalty function stabilization alone (best variant); and (v) the combination of automatic smoothing and piecewise penalty function stabilization (best variant). A point (x, y) on the performance profile associated to a stabilization scheme indicates that, for a fraction y of instances, the algorithm was not more than x times slower than the best algorithm for those instances. Note that the horizontal axis is logarithmically scaled. The performance profiles support the conclusions made above based on the geometric mean ratios. We can observe also that application of smoothing stabilization techniques deteriorates the solution time of standard column generation only for very few instances of the last

Generalized Assignment						
Variant	pieces	κ	It	Sp	Cl	T
Smooth			3.99	3.63	4.83	7.93
Curv	3	80	5.39	5.41	5.36	14.51
SmoothCurv	3	40	7.78	7.08	9.66	25.69
Expl	3	200	2.95	2.94	2.75	6.50
Expl	5	100	4.67	4.68	4.83	17.29
SmoothExpl	3	200	6.46	6.03	8.36	43.62
SmoothExpl	5	40	7.74	7.15	9.71	33.62

Multi-Echelon Lot Sizing						
Variant	pieces	κ	It	Sp	Cl	T
Smooth			3.29	2.98	4.86	5.99
Curv	3	16	7.27	7.36	8.02	14.80
SmoothCurv	3	2	8.85	8.49	11.57	17.13
Expl	3	40	3.72	3.73	4.08	7.55
Expl	5	10	5.63	5.66	6.27	10.96
SmoothExpl	3	10	6.71	6.29	9.19	14.68
SmoothExpl	5	2	7.17	6.82	9.67	14.68

Table 6: Detailed results for column generation with smoothing and piecewise penalty function stabilization

three problems. An important observation is that, for all applications except two of them, there are cases for which the speed-up ratio of the best stabilization technique is one order of magnitude, or better. For the Generalized Assignment problem, the maximum speed-up ratio is about 6500 reducing the solution time from more than 6 hours to less than 4 seconds for the instance $D - 10 - 400$. This observation suggests that stabilization techniques should be considered systematically as a complement to basic Simplex-based column generation.

Conclusion and further discussion

We conclude this paper, by summarizing the work and the messages that we hope to convey, while pointing to further work that would complete the picture on the issue of stabilization in column generation approaches.

Our paper has specified the link between column generation stabilization by smoothing and in-out separation. We also extended the in-out convergence proof to smoothing schemes such as Neame's and Wentges', deriving an extra global convergence property on the optimality gap for Wentges' scheme. On the practical side, we proposed a simple scheme for dealing with a sequence of *mis-pricings*. Our main practical contribution however was to show that smoothing can be implemented in a way that does not require parameter tuning.

Our hard coded initialization and dynamic auto-adaptive scheme based on local sub-gradient information experimentally matches or improves the performance of the best user tuning as revealed in Table 3. The extra directional twisting feature (combining smoothing with an ascent method) is shown in Table 3 to bring further performance improvement for some problems. However, directional smoothing does not help in some applications with identical subproblems (as for Vehicle Routing or Shift Scheduling), when sub-gradient information is aggregated and hence probably less pertinent. The adaptive setting of parameter β outperforms the static value strategy leading to a generic parameter-tuning-free implementation of the combined smoothing scheme.

Our second practical output was to revisit penalty function approaches in an effort to reduce the number of parameters and to develop schedules for parameter self-adjustment. Although penalty function approaches can be more efficient than smoothing, there remains a single but critical parameter to tune penalty functions on a problem specific basis. Our main contribution here was to show that the combination of smoothing and penalty function approaches outperforms any of the two techniques applied separately. In any case, as shown by our performance

profile, stabilization should be understood as a key feature of LP based column generation approaches.

Our paper was deliberately restricted to experimentations where the master program is solved with the Simplex method. Although the stabilization features that we used could also be used in approaches relying on solving the master using a Quadratic Programming (QP) or an Interior Point (IP) master LP solver, they are expected to have a smaller impact in such cases, as QP and IP already incorporates stabilization features inherently. Besides, a genuine extension of stabilization techniques in these contexts would probably benefit from specific adaptations and tuning of the methods to the master solver. For instance, one could consider early termination of the IP solver to return an interior dual solution to the pricing oracle. Such extra study is beyond the scope of this paper and it should not alleviate our current message: if one is considering a Simplex based approach (in comparison to other approaches), one should at least enhance it with basic stabilization such as smoothing that can easily be put in place without any parameter tuning.

Indeed, with stabilization, Simplex-based column generation approaches can be competitive: our initial tests show that a Simplex approach is typically faster compared to using a back-box IP solver to tackle master LPs, even if Simplex uses more iterates, as reported in Table 7. The IP approach seems to be faster only on instances where convergence is really an issue; we suspect the same conclusion would hold for QP. Such statement is on the same line as text book advice on when using barrier versus Simplex in solving LPs. To illustrate this, we compared our Simplex results with those obtained using the Barrier algorithm (without the crossover) provided by Cplex, selecting three applications. Table 7 reports the speed-up ratios of geometric means of statistics `It`, `Sp`, `Cl`, `T`, using the Barrier algorithm instead of Simplex with default Cplex options.

	$\alpha = 0, \beta = 0$				$\alpha = auto, \beta = auto$				Smoothing + penalty function					
Application	`It`	`Sp`	`Cl`	`T`	`It`	`Sp`	`Cl`	`T`	Variant	κ	`It`	`Sp`	`Cl`	`T`
Generalized Assignment	1.87	1.87	1.74	1.25	1.22	1.22	1.12	0.84	`SmoothExpl`	200	1.30	1.31	1.15	0.25
Multi-Echelon Lot Sizing	1.89	1.89	1.81	1.83	1.58	1.62	1.36	1.16	`SmoothCurv`	2	0.86	0.81	0.98	0.57
Bin Packing	2.06	2.06	1.61	0.79	1.34	1.33	1.18	0.42						

Table 7: Comparing Simplex based and Barrier based approaches with and without automatic smoothing.

The results show that using Barrier instead of Simplex brings fewer iterations whether one uses smoothing or not. However, the overall speed-up gained by using smoothing is less important for Barrier than for Simplex and it depends largely on the application. The biggest speed-up is achieved for Lot-Sizing, as it is an application in which most of the time is spent in solving pricing subproblems. When smoothing is used, the overall running time is better for the Simplex based approach, except for Lot-Sizing. Although the Table does not show this, the relative performance of Barrier in comparison with Simplex depends a lot on the instance. This performance is very good for instances that require a "heavy" stabilization. However, for other instances, changing the LP solver to the Barrier algorithm may degrade the running time severely. Therefore, if one searches for a "consistent" approach, using Simplex seems to be the best approach.

We have also tested using penalty function combined with smoothing under the Barrier based approach. However, this option did not perform better (usually significantly worse) than the combined stabilization with the

Simplex algorithm for all the instances of these selected applications. The third column of Table 7 presents these comparative results, expect for bin packing where results are worse with the combined approach (for both Barrier and Simplex). We observed that even for instances where Barrier combined with smoothing yielded better results than Simplex with smoothing, i.e., even on instances that require a "heavy" stabilization, the Simplex based approach with combined stabilization has given rise to the best running times.

Aknowledments: This research was supported by the associated team program of INRIA through the SAMBA project. The remarks of J-P. Richard were of great value to improve our draft. The comments of the referees have been very constructive. They lead in particular to enlarging the discussion beyond Simplex based approaches.

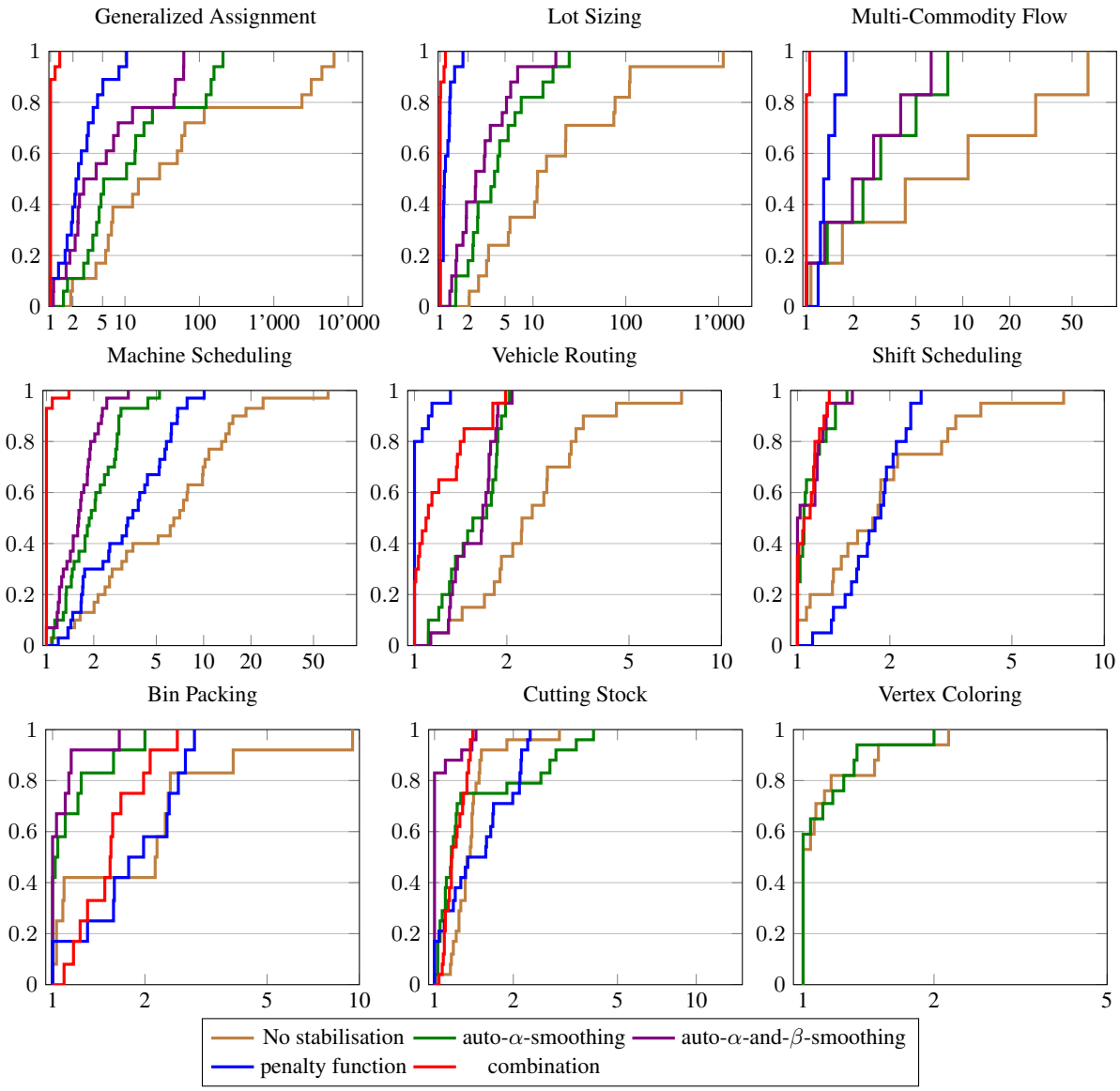


Fig. 11: Performance profile graphs

References

1. Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283, 2011.
2. Francisco Barahona and Ranga Anbil. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, 87(3):385–399, 2000.
3. J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, 41(11):1069–1072, 1990.
4. Walid Ben-Ameur and José Neto. Acceleration of cutting-plane and column generation algorithms: Applications to network design. *Networks*, 49(1):3–17, 2007.
5. O. Briant, C. Lemaréchal, Ph. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck. Comparison of bundle and classical column generation. *Mathematical Programming*, 113(2):299–344, 2008.
6. Marie-Claude Côté, Bernard Gendron, and Louis-Martin Rousseau. Grammar-based column generation for personalized multi-activity shift scheduling. *INFORMS Journal on Computing*, 25(3):461–474, 2013.
7. W. de Oliveira and C. SagastizÃąbal. Level bundle methods for oracles with on-demand accuracy. *Optimization Methods and Software*, 0(0):1–30, 0.
8. Sophie Demassey, Gilles Pesant, and Louis-Martin Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006.
9. Olivier du Merle, Daniel Villeneuve, Jacques Desrosiers, and Pierre Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1-3):229–237, 1999.
10. Issmail Elhallaoui, Abdelmoutalib Metrane, Guy Desaulniers, and FranÃğois Soumis. An improved primal simplex algorithm for degenerate linear programs. *INFORMS Journal on Computing*, 23(4):569–577, 2011.
11. Matteo Fischetti and Domenico Salvagnin.
12. Jean-Louis Goffin and Jean-Philippe Vial. Convex nondifferentiable optimization: A survey focused on the analytic center cutting plane method. *Optimization Methods and Software*, 17(5):805–867, 2002.
13. Jacek Gondzio, Pablo GonzÃąlez-Brevis, and Pedro Munari. New developments in the primalâĂŞdual column generation technique. *European Journal of Operational Research*, 224(1):41 – 51, 2013.
14. David J. Johnson and Michael A. Trick, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993*. American Mathematical Society, Boston, MA, USA, 1996.
15. Chungmok Lee and Sungsoo Park. Chebyshev center based column generation. *Discrete Applied Mathematics*, 159(18):2251 – 2265, 2011.
16. Claude LemarÃľchal, Arkadii Nemirovskii, and Yurii Nesterov. New variants of bundle methods. *Mathematical Programming*, 69(1-3):111–147, 1995.
17. R. E. Marsten, W. W. Hogan, and J. W. Blankenship. The boxstep method for large-scale optimization. *Operations Research*, 23(3):389–405, 1975.
18. Pedro Munari and Jacek Gondzio. Using the primal-dual interior point algorithm within the branch-price-and-cut method. *Computers and Operations Research*, 40(8):2026 – 2036, 2013.
19. Philip James Neame. *Nonsmooth Dual Methods in Integer Programming*. PhD thesis, University of Melbourne, Department of Mathematics and Statistics, 2000.
20. Patric R. J. Östergård. A new algorithm for the maximum-weight clique problem. *Nordic Journal of Computing*, 8:424–436, December 2001.
21. Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and Francois Vanderbeck.
22. Artur Pessoa, Eduardo Uchoa, Marcus Poggi de Aragão, and Rosiane Rodrigues. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, 2:259–290, 2010.
23. David Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45(5):758–767, 1997.
24. Chris N. Potts and Luk N. Van Wassenhove. A Branch and Bound Algorithm for the Total Weighted Tardiness Problem. *Operations Research*, 33(2):363–377, 1985.

25. Ruslan Sadykov, Alexander A. Lazarev, Vitaliy Shiryaev, and Alexey Stratonnikov. In *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, Dagstuhl, Germany.
26. François Vanderbeck. Implementing mixed integer column generation. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, *Column Generation*, pages 331–358. Springer US, 2005.
27. François Vanderbeck and Martin W. P. Savelsbergh. A generic view of Dantzig-Wolfe decomposition in mixed integer programming. *Operations Research Letters*, 34(3):296–306, 2006.
28. Paul Wentges. Weighted dantzig–wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, 4(2):151–162, 1997.
29. Mutsunori Yagiura, Toshihide Ibaraki, and Fred Glover. A path relinking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research*, 169(2):548 – 569, 2006.
30. Williard I. Zangwill. A backlogging model and a multi-echelon model of a dynamic economic lot size production system-a network approach. *Management Science*, 15(9):506–527, 1969.