Projective Cutting-Planes for Robust Linear Programming and Cutting Stock Problems

Daniel Porumbel

To cite this version:
Daniel Porumbel. Projective Cutting-Planes for Robust Linear Programming and Cutting Stock Problems. INFORMS Journal on Computing, In press, 10.1287/ijoc.2022.1160. hal-03715458

HAL Id: hal-03715458
https://cnam.hal.science/hal-03715458
Submitted on 19 Jul 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Projective Cutting-Planes for robust linear programming and cutting stock problems

Daniel Porumbel
CEDRIC, CNAM (Conservatoire National des Arts et Métiers) Paris, daniel.porumbel@cnam.fr

We explore the Projective Cutting-Planes algorithm proposed in [14] from new angles, by applying it to two new problems, i.e., to robust linear programming and to a cutting-stock problem with multiple lengths. Projective Cutting-Planes is a generalization of the widely-used Cutting-Planes and it aims at optimizing a linear function over a polytope $\mathcal{P}$ with prohibitively-many constraints. The main new idea is to replace the well-known separation sub-problem with the following projection sub-problem: given an interior point $x \in \mathcal{P}$ and a direction $d$, find the maximum steplength $t$ such that $x + td \in \mathcal{P}$. This enables one to generate a feasible solution at each iteration, a feature that does not exist built-in in a standard Cutting-Planes algorithm. The practical success of this new algorithm does not mainly come from the higher level ideas already presented in [14]. Its success is significantly more dependent on the computation time needed to solve the projection sub-problem in practice. Thus, the main challenge addressed by the current paper is the design of new techniques for solving this sub-problem very efficiently for different polytopes $\mathcal{P}$. For the first addressed problem, robust linear programming, $\mathcal{P}$ is defined as a primal polytope. For the second addressed problem, Multiple-length Cutting-Stock, $\mathcal{P}$ is a dual polytope defined in a Column Generation model. Numerical experiments on both these new problems confirm the potential of the proposed ideas. This enables us to draw conclusions supported by numerical results from both the current paper and [14], while also gaining more insight into the dynamics of the algorithm.

Key words: interior points, projection sub-problem, column generation, robust linear programming

1. Introduction

We first shortly present the context of Projective Cutting-Planes that was introduced in [14]. We focus on a Linear Program (LP) of the form:

$$\text{opt} \left\{ b^\top x : a^\top x \leq c_a, \ \forall (a, c_a) \in \mathcal{A} \right\} = \text{opt} \left\{ b^\top x : x \in \mathcal{P} \right\}, \quad (1.1)$$

where $a, b \in \mathbb{R}^n$, $c_a \in \mathbb{R}$, $\mathcal{A}$ is a set of unmanageably-many constraints and “opt” stands for either “min” or “max”.

Given an interior point $x \in \mathcal{P}$ and a direction $d \in \mathbb{R}^n$, the projection sub-problem $\text{project}(x \rightarrow d)$ asks to advance from $x$ along $d$ up to the pierce point $x + t^*d$
Iteration 4 returns $t_4^* = 1$, i.e., $x_4 + d_4 = \text{opt}(P_3)$ is feasible, meaning $\text{opt}(P_3) = \text{opt}(P)$. Figure 1: Intuitive illustration of a Projective Cutting-Planes execution

where the boundary of $P$ is touched, i.e., to determine the maximum step length $t^* = \max \{t \geq 0: x + td \in P\}$. To solve this sub-problem, it is also necessary to find a constraint of $A$ satisfied with equality by $x + t^*d$. One may check a formal definition in [14, Def. 1], but the main idea is that the projection sub-problem requires determining: (i) a pierce point and (ii) a facet or constraint satisfied with equality by this pierce point. This projection sub-problem is the main building block of Projective Cutting-Planes.

1.1. Revisiting the main steps of the Projective Cutting-Planes described in [14]

In intuitive terms, one can trace in Figure 1 the evolution of a simple Projective Cutting-Planes execution for an LP with $n = 2$ variables. At iteration $i_t = 1$, the projection sub-problem projects $x_1 = 0 = [0 0]^T$ along the direction $d_1 = b$, see the black dashed arrow. This generates a pierce point $x_1 + t_1^*d_1$ and also the black facet (constraint) satisfied with equality by $x_1 + t_1^*d_1$. By integrating the black facet, Projective Cutting-Planes constructs the first outer approximation $P_1$ of $P$, i.e., $P_1$ is the largest triangle defined by this black facet (line) and the two axes; notice the optimal solution $\text{opt}(P_1)$ marked close to the bottom-right corner of the figure.

At the second iteration, Projective Cutting-Planes selects a (mid) point $x_2$ between $x_1$ and $x_1 + t_1^*d_1$ and projects $x_2$ towards the current optimal outer solution $\text{opt}(P_1)$. This generates a second pierce point $x_2 + t_2^*d_2$ and a second facet (blue solid line) that is added to the facets of $P_1$ to construct $P_2$. The process is repeated until the projection sub-problem certifies at iteration $i_t = 4$ that the outer solution $\text{opt}(P_3)$ cannot be separated; this means $\text{opt}(P_3)$ is an optimal solution over $P$. 
More generally, Projective Cutting-Planes generates a sequence of inner solutions $x_{it}$ as well as a sequence of outer solutions $\text{opt}(\mathcal{P}_{it})$ that both converge along the iterations $it$ to an optimal solution $\text{opt}(\mathcal{P})$. Each new inner solution $x_{it+1}$ is chosen as a point on the segment joining the previous inner solution $x_{it}$ and the last pierce point $x_{it} + t_{it}^* d_{it}$ returned by the last projection. The next projection $\text{project}(x_{it+1} \rightarrow d_{it+1})$ is chosen to point towards the current outer optimal solution $\text{opt}(\mathcal{P}_{it})$, i.e., $d_{it+1} = \text{opt}(\mathcal{P}_{it}) - x_{it+1}$.

Each call to the projection sub-problem also returns a (first-hit) constraint (of $\mathcal{A}$) satisfied with equality by the pierce point; this constraint is added to the constraints of $\mathcal{P}_{it}$ to construct $\mathcal{P}_{it+1}$. By adding one new constraint at each iteration $it$, the Projective Cutting-Planes algorithm constructs a sequence $\mathcal{P}_1 \supseteq \mathcal{P}_2 \supseteq \cdots \supseteq \mathcal{P}$ of outer approximations of $\mathcal{P}$, so that $\text{opt}(\mathcal{P}_1), \text{opt}(\mathcal{P}_2), \text{opt}(\mathcal{P}_3), \ldots$ converge to $\text{opt}(\mathcal{P})$, as in a standard Cutting-Planes.

At the first iteration $it = 1$, one can choose any starting feasible solution $x_1$, usually in a problem–specific manner. The first direction $d_1$ is often $d_1 = b$, to make the first projection $\text{project}(x_1 \rightarrow d_1)$ advance along the direction with the fastest rate of objective function improvement. At each subsequent iteration, the next interior point is selected using a formula of the form $x_{it+1} = x_{it} + \alpha t_{it}^* d_{it}$ for some $\alpha \in (0, 1]$. We will show that for $\alpha = 1$, each new interior solution $x_{it+1}$ is actually a boundary point of strictly higher quality than the previous one $x_{it}$. With such choice, the “grip” exerted by the lower and the upper bounds on $\text{optVal}(\mathcal{P})$ is guaranteed to strictly increase at each iteration, i.e., we have that $b^T x_{it} < b^T x_{it+1} \leq \text{opt}(\mathcal{P}) \leq b^T \text{opt}(\mathcal{P}_{it+1}) \leq b^T \text{opt}(\mathcal{P}_{it})$ at each iteration $it$ except at the very last one (considering a maximization setting).

We also warn the reader of an inherent deterrent to adopting the new method, citing the conclusions of [14]: “it can be more difficult to design a projection algorithm than a separation one, because the projection sub-problem is more general. As such, more work may be needed to make the Projective Cutting-Planes reach its full potential.” However, for both of the problems explored in this paper, the projection and the separation algorithms are of a similar nature because they rely on similar general techniques (e.g., they both use Dynamic Programming for Cutting-Stock); the main difference is that solving the projection requires a larger number of ad-hoc customizations that have to be tailored to the considered problem.
1.2. Related ideas from the literature

While the idea of constructing a converging sequence of interior points determined by an exact projection algorithm is new in standard Cutting-Planes, the idea of using an interior point $x_{in}$ to guide the separation of an exterior point $x_{out} = \text{opt}(\mathcal{P}_{it})$ is not new. For instance, [1] proposes to separate $x_{out}$ by first calling the separation oracle on a point $x_{sep}$ on the segment joining $x_{in}$ and $x_{out}$. Thus, Algorithm 1 from [1] defines $x_{sep} = \alpha x_{in} + (1 - \alpha) x_{out}$ for some $\alpha \in (0, 1]$. If $x_{sep} \notin \mathcal{P}$, we can expect that the cutting plane returned by separating $x_{sep}$ is more efficient; otherwise, the algorithm from [1] sets $x_{in} = x_{sep}$, updates $x_{sep}$ accordingly, and calls the separation oracle again. The process of separating $x_{out}$ is based on a repeated choice of a point $x_{sep}$ between $x_{in}$ and $x_{out}$ coupled with a repeated separation of $x_{sep}$. This approach does not determine the pierce point, i.e., it does not calculate the result of a projection from $x_{in}$ to $x_{out}$. This difference is quite deep because a major challenge in Projective Cutting-Planes is to design (for each considered problem) a projection algorithm that competes well in terms of computational speed with the separation one. In addition, citing [14], “this can not be simply achieved by repeated separation: such projection method would call the separation algorithm at least twice, or usually 3 or 4 times, i.e., it could become 3 or 4 times slower than the separation algorithm.”

The “in-out separation” from [10] is, despite its name, more remotely related. In this work, the notion of “in-point” or “out-point” is considered with regards to a feasible area that belongs to $\mathbb{R}^{n+1}$, being defined both by the elements $\pi$ of $\mathcal{P}$ and by their objective values $\eta$; such a feasible area is visible in [10, Fig. 1] using the above notations $\pi$ and $\eta$.

More generally, certain interior point methods for (dual) LPs with prohibitively-many constraints (in Column Generation) generate a sequence of interior points to calculate $\text{opt}(\mathcal{P}_{it})$ at each iteration $it$ [6]. However, these interior points are actually interior only to $\mathcal{P}_{it} \supset \mathcal{P}$, but not necessarily to $\mathcal{P}$.

A full literature review on other aspects related to the idea of advancing towards $\text{opt}(\mathcal{P})$ using either inner or outer solutions can be found in our previous work [12, §1.1.1].

1.3. New contributions

A challenging task in the implementation of a successful Projective Cutting-Planes is to design a fast projection algorithm, i.e., fast enough to compete well with the separation algorithm in terms of computational speed. While we already presented in [14] certain
methods that achieve this goal, we now introduce several other techniques that work in
different settings not addressed in [14].

We will show how to generalize the separation algorithm without increasing its compu-
tational complexity. We will first illustrate this on a well-known robust linear program-
ing problem. For this problem, the computational bottleneck of both the separation and
the projection sub-problem comes from the need of enumerating a set of nominal con-
straints (that generate prohibitively-many robust cuts). Regarding the second problem,
**Multiple-length Cutting-Stock**, we will show that if the separation sub-problem can
be solved by *Dynamic Programming*, then so can be the projection sub-problem. Both the
separation and the projection algorithms work with a set of *Dynamic Programming* states
and the main difference between the two algorithms is the following. The projection al-
gorithm has to return a state that minimizes a ratio of two state indicators, while the
separation algorithm has to return a state that minimizes a difference of the same indica-
tors. Such a change of objective function does not always induce an important slowdown
because it does not necessarily generate an explosion of the number of states (especially if
the interior points \(x_{it}\) are chosen carefully).

A final contribution of the paper is to analyse the choice of the inner points \(x_{it}\) along
the iterations \(it\). While we always choose \(x_{it}\) using the formula \(x_{it} = x_{it-1} + \alpha t_{it-1} d_{it-1}\),
the value of \(\alpha\) can greatly vary. Comparing the best choices made throughout the current
paper and [14], sometimes it is better to use a small step length \(\alpha < 0.5\) and sometimes it is
better to use a large step length \(\alpha = 1\). This is actually only an empirical observation. We
will provide theoretical insights to explain it in Section 3.3; we will see that the best value
of \(\alpha\) is related to the (magnitude of the) oscillations of the interior solutions \(x_{it}\) along the
iterations \(it\).

The remainder is organized as follows. Section 2 presents the application of *Projective
Cutting-Planes* on: (i) a robust optimization problem and (ii) on a *Column Generation*
model for *Multiple-length Cutting-Stock*. Section 3 reports numerical results on these
problems, followed by conclusions in the fourth section. There are also two short append-
dices: the first one revisits the calculation of the Lagrangian bounds for *Multiple-length
Cutting-Stock* and the second one presents a fast data structure for efficiently manipu-
lating a Pareto frontier (here needed by the *Dynamic Programming* scheme used to solve
the sub-problems in the *Cutting-Stock* experiments).
2. Projective Cutting-Planes for robust linear programming and multiple-length cutting stock

We first recall [14, (2.2)] that for any feasible \( x \in \mathcal{P} \) and for any \( d \in \mathbb{R}^n \), the projection sub-problem \( \text{project}(x \to d) \) can be solved by minimizing the fractional program (2.1) below. Let us stress \( x \) stands for a parameter in the sub-problem (2.1) below, while in the overall Projective Cutting-Planes it is a variable representing the current inner solution of (1.1); the decision variable in (2.1) is the couple \((a, c_a) \in \mathcal{A}\) associated to the constraint \( a^\top x \leq c_a \) of the main program (1.1). We will instantiate (2.1) on a robust linear problem (Section 2.1) and on Multiple-length Cutting-Stock (Section 2.2).

\[
 t^* = \min \left\{ \frac{c_a - a^\top x}{a^\top d} : (a, c_a) \in \mathcal{A}, \ d^\top a > 0 \right\}.
\]

(2.1)

2.1. A robust optimization problem

The main idea in robust optimization is to seek an optimal solution that remains feasible if certain constraint coefficients deviate (reasonably) from their nominal values. The robust optimization literature is now constantly growing and there are many methods to define what coefficient deviations are acceptable, e.g., one can use linear or ellipsoid uncertainty sets. However, to avoid unessential complication, we here focus only on the robustness model from [4]; the reader may refer to this paper for more references, motivations and related ideas. There are two main principles behind this robustness model: (i) the deviation of a coefficient is at most \( \delta = 1\% \) of the nominal value (ii) there are at most \( \Gamma \) coefficients that are allowed to deviate in each nominal constraint. The underlying assumption is that in real life the nominal coefficients of a given constraint cannot change all at the same time, always in an unfavorable manner.

2.1.1. The model with prohibitively-many constraints and the standard Cutting-Planes

Let us first consider a set \( \mathcal{A}_{\text{nom}} \) of nominal constraints that is small enough to be enumerated in practice, i.e., there is no need of Cutting-Planes to solve the nominal version of the problem with no robustness. We then associate to each \((a, c_a) \in \mathcal{A}_{\text{nom}}\) a prohibitively-large set \( \text{Dev}_\Gamma(a) \) of deviation vectors \( \hat{a} \), i.e., vectors \( \hat{a} \in \mathbb{R}^n \) that have at maximum \( \Gamma \) non-zero components and that satisfy \( \hat{a}_i \in \{-\delta a_i, 0, \delta a_i\} \forall i \in [1..n] \), using \( \delta = 0.01 \) in practice. Each such deviation vector \( \hat{a} \) yields a robust cut \((a + \hat{a})^\top x \leq c_a\), so that we can state \((a + \hat{a}, c_a) \in \mathcal{A}\). In theory, each \( \hat{a}_i \) for any \( i \in [1..n] \) may be allowed to take a fractional value in the interval \([-\delta a_i, \delta a_i]\), thus leading to infinitely-many robust
cuts (semi-infinite programming); however, the strongest robust cuts are always obtained when each non-zero $\hat{a}_i$ is either $\delta a_i$ or $-\delta a_i$. There are at most $\binom{n}{\Gamma}2^\Gamma$ deviation vectors for each nominal constraint $(a, c_a) \in A_{\text{nom}}$, because there are $\binom{n}{\Gamma}$ ways to choose the non-zero components of $\hat{a}$ and each one of them can be either positive or negative, hence the $2^\Gamma$ factor.

The generic LP (1.1) is instantiated as follows:

$$\min \left\{ \mathbf{b}^\top \mathbf{x} : (\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{x} \leq c_a \ \forall (a, c_a) \in A_{\text{nom}} \ \forall \hat{\mathbf{a}} \in \text{Dev}_\Gamma(\mathbf{a}); \ x_i \in [1b_i, ub_i] \ \forall i \in [1..n] \right\} \quad (2.2)$$

The last condition $x_i \in [1b_i, ub_i]$ of (2.2) represents the initial constraints $A_0$, most instances using $1b_i = 0 \ \forall i \in [1..n]$, i.e., the variables are most often non-negative.

We consider a canonical Cutting-Planes algorithm for the above (2.2), based on the following separation sub-problem: given any $\mathbf{x} \in \mathbb{R}^n$, minimize $c_a - (\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{x}$ over all $(a, c_a) \in A_{\text{nom}}$ and over all $\hat{\mathbf{a}} \in \text{Dev}_\Gamma(\mathbf{a})$. For a fixed nominal constraint $(a, c_a) \in A_{\text{nom}}$, the strongest possible deviation $\hat{\mathbf{a}}_x^\top \mathbf{x}$ of $(a, c_a)$ with respect to $\mathbf{x}$ is determined by maximizing $\hat{\mathbf{a}}_x = \arg \max \left\{ \hat{\mathbf{a}}_x^\top \mathbf{x} : \hat{\mathbf{a}} \in \text{Dev}_\Gamma(\mathbf{a}) \right\}$. To find this $\hat{\mathbf{a}}_x$, one needs to determine the largest $\Gamma$ absolute values in the terms of the sum $\mathbf{a}^\top \mathbf{x} = \sum_{i=1}^{n} a_i x_i$; this way, $\hat{\mathbf{a}}_x^\top \mathbf{x}$ can be written as a sum of $\Gamma$ terms of the form $\delta|a_i x_i|$. We use absolute values because the strongest deviation of a term $a_i$ is either $\hat{a}_i = \delta a_i$ if $a_i x_i \geq 0$ or $\hat{a}_i = -\delta a_i$ if $a_i x_i < 0$. We next describe how these largest $\Gamma$ values can be determined by a partial-sorting algorithm of linear complexity.

Remark 1 If $\Gamma$ is a fixed parameter, the largest $\Gamma$ entries in a table of $n$ values (e.g., such as $|a_1 x_1|, |a_2 x_2|, \ldots |a_n x_n|$ above) can be determined in $O(n)$ time. We use a partial-sorting algorithm that essentially performs the following: iterate over $i \in [1..n]$ and attempt at each step $i$ to insert $|a_i x_i|$ in the list of the highest $\Gamma$ values known up to now. Considering $\Gamma$ is a fixed parameter, this operation would even take constant time when using a self-balancing binary search tree (as implemented in the C++ \texttt{std::multiset} data structure).

The most computationally demanding task is checking whether the new value $|a_i x_i|$ is larger than the minimum value $v_{\text{min}}$ recorded in the tree. If this is the case, the insertion of $|a_i x_i|$ may make the tree size exceed $\Gamma$, and so, $v_{\text{min}}$ has to be removed. Each insertion and each removal takes constant time with regards to $n$, when considering $\Gamma$ as a parameter. However, these operations can still lead to a non-negligible multiplicative constant factor (like $\log(\Gamma)$) in the complexity of the partial sorting algorithm, inducing a non-negligible overall slowdown. The repeated use of this algorithm takes around $15\%$ of the total running time for $\Gamma \geq 10$. 
Compared to the above Cutting-Planes, the algorithm from [4] is slightly different because it returns multiple robust cuts at each separation call. We will first design Projective Cutting-Planes in a standard setting considering a unique (robust) cut per iteration. However, we will also mention throughout the text how to handle multiple cuts per iteration (and numerical results in this sense will be presented in Section 3.1.3).

2.1.2. Solving the projection sub-problem Based on (2.1), the projection sub-problem reduces to minimizing \( \frac{c_a - (a + \hat{a})^T x}{(a + \hat{a})^T d} \) over all nominal constraints \( (a, c_a) \in A_{\text{nom}} \) and over all deviation vectors \( \hat{a} \in \text{Dev}_\Gamma(a) \) such that \( (a + \hat{a})^T d > 0 \). Just as the separation algorithm, the projection algorithm iterates over all nominal constraints \( A_{\text{nom}} \), in an attempt to reduce the above ratio – i.e., the step length – at each \( (a, c_a) \in A_{\text{nom}} \). Let \( t^*_i \) denote the optimal step length obtained after considering the robust cuts associated to the first \( i \) constraints from \( A_{\text{nom}} \). It is clear that \( t^*_i \) can only decrease as \( i \) grows. Starting with \( t_0 = 1 \), the projection algorithm determines \( t^*_i \) from \( t^*_{i-1} \) by applying the following five steps:

1. Set \( t = t^*_{i-1} \) and let \( (a, c_a) \) denote the \( i \)th constraint from \( A_{\text{nom}} \).
2. Determine the strongest deviation vector \( \hat{a}_t \) with respect to \( x + t d \) by maximizing:
   \[
   \hat{a}_t = \arg \max \{ \hat{a}^T (x + t d) : \hat{a} \in \text{Dev}_\Gamma(a) \}. \tag{2.3}
   \]
   For this, one has to extract the \( \Gamma \) largest absolute values from the terms of the sum \( a^T (x + td) \); we apply the partial-sorting algorithm used for the separation sub-problem in Remark 1.
3. If \( (a + \hat{a}_t)^T (x + td) \leq c_a \), then \( x + td \) is feasible with regards to the first \( i \) constraints from \( A_{\text{nom}} \) and the associated robust cuts, because any deviation vector \( \hat{a} \in \text{Dev}_\Gamma(a) \) satisfies \( \hat{a}^T (x + td) \leq \hat{a}_t^T (x + td) \). In this case, the final value \( t^*_i = t \) has been obtained for this value of \( i \). Otherwise, the robust cut \( (a + \hat{a}_t, c_a) \) leads to a smaller feasible step length:
   \[
   t' = \frac{c_a - (a + \hat{a}_t)^T x}{(a + \hat{a}_t)^T d} < t. \tag{2.4}
   \]
4. If \( t' = 0 \), then the overall projection algorithm returns \( t^* = 0 \) without checking the remaining nominal constraints, because it is not possible to return a step length below 0 since \( x \) is feasible.
5. Set \( t = t' \) and repeat from Step 2 (without incrementing \( i \)). The underlying idea is that the deviation vector \( \hat{a}_t \) determined via (2.3) is not the strongest one with regards to \( x + t'd \), because \( \hat{a}_t \) yields the highest deviation in (2.3) with regards to a different point \( i.e., x + td \). But there might exist a different robust cut \( (a + \hat{a}_t', c_a) \) for the same nominal
constraint such that $\hat{\mathbf{a}}_i^\top (\mathbf{x} + t'd) > \hat{\mathbf{a}}_i^\top (\mathbf{x} + t'd)$. This could further reduce the step length below $t'$, proving that $\mathbf{x} + t'd$ is infeasible.

By sequentially applying the above steps to all constraints $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$ one by one, the step length returned at the last constraint of $\mathcal{A}_{\text{nom}}$ provides the sought $t^*$ value. It is not difficult to adapt this algorithm to switch to a multi-cut variant: it is enough to return all robust cuts generated for all values of $i$ that produced a step length decrease in (2.4). The robust cuts associated to some $i$ that could not decrease the step length via (2.4) may be too weak to be useful and there is no need to return such cuts.

2.1.2.1. The speed of the projection and the separation algorithms

In theory, the above projection algorithm could repeat Steps 2-5 many times for each $i$, iteratively decreasing $t$ in a long loop. However, experiments suggest that long loops arise only rarely in practice; the value of $t$ is typically decreased via (2.4) only a dozen of times at most for all (thousands of) nominal constraints, i.e., for all $i$. For many nominal constraints $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$, the above algorithm only concludes at Step 3 that $\mathbf{x} + td$ satisfies all robust cuts associated to $(\mathbf{a}, c_a)$, and, in such cases, the most computationally expensive task is the partial-sorting algorithm (called once at Step 2).

Furthermore, the overall projection algorithm can even stop earlier without scanning all nominal constraints, by returning $t^* = 0$ at Step 4. An exact separation algorithm could never stop earlier, because $c_a - (\mathbf{a} + \hat{\mathbf{a}}_x)^\top \mathbf{x}$ can certainly decrease up to the last nominal constraint $(\mathbf{a}, c_a)$. In a few cases, the projection algorithm can become even faster than the separation one. Indeed, for the last (very large) instance from Table 1 (p. 19) with $\Gamma = 50$, a separation iteration takes around 0.62 seconds (on average), while the projection one takes 0.56 seconds (on average). At the other end of the spectrum, for an instance like nesm with $\Gamma = 50$, a projection iteration can take about 30% more time than a separation one. All things considered, one can say that the running time of the above projection algorithm is similar to that of the separation algorithm.

We are skeptical that it is possible to compete with the above algorithm by simply calling the separation algorithm multiple times. An approach based on repeated separation would make the projection algorithm at least twice as slow as the separation one: a first separation call would find a first robust cut satisfied with equality by some $\mathbf{x} + td$ and then one needs at least a second call to check if $\mathbf{x} + td$ can be further separated to decrease $t$. Experiments suggest that a third or a fourth call is often needed in practice. More generally, a goal of
this work is to develop techniques that can lead to designing a projection algorithm as fast as the separation one; this is the most fruitful endeavour in the long run.

2.1.3. The overall Projective Cutting-Planes

If we consider the above projection algorithm as a black-box component, the design of Projective Cutting-Planes is rather straightforward. It is essentially enough to follow the guidelines from Section 1.1 or more exactly the steps 1-4 specified in [14, § 2].

2.1.3.1. Choosing $x_{it}$ for $it > 1$

The only non-trivial part is choosing the interior points $x_{it}$. As with most problems studied in this work and [14], experiments suggest that it is not very efficient to define $x_{it}$ as the best feasible solution found up to the iteration $it$ (i.e., the last pierce point $x_{it} = x_{it-1} + t_{it-1}^* d_{it-1}$). Although such an aggressive Projective Cutting-Planes variant could find better feasible solutions in the beginning, it may eventually need more iterations in the long run. For best long-term results, it is certainly better to choose a more interior point $x_{it}$, not too close to the boundary of $P$, enabling the inner solutions $x_1, x_2, x_3, \ldots$ to follow a central path (a similar concept is used in some interior point algorithms). We thus define $x_{it} = x_{it-1} + \alpha t_{it-1}^* d_{it-1}$ with $\alpha = 0.1 \forall it > 1$.

2.1.3.2. Determining the first interior point $x_1$

To construct an initial feasible solution $x_1$, one could be tempted to try $x_1 = 0_n$, but $0_n$ may be infeasible. We propose to generate $x_1$ as a feasible solution in a relatively simple LP whose feasible area stays (deeply) inside the feasible area of (2.2). We construct this “deep” inner LP as follows: for each $(a, c_a) \in A_{nom}$, we insert a constraint $a^\top x + \delta |a|^\top x \leq c_a$, where $|a| = [|a_1| \ldots |a_n|]^\top$. If $x$ is non-negative (as in most instances), than any solution $x$ that satisfies $a^\top x + \delta |a|^\top x \leq c_a \forall (a, c_a) \in A_{nom}$ is feasible with regards to all robust cuts — because a robust cut uses a deviation vector $\hat{a}$ that satisfies $\hat{a} \leq \delta |a|$, so that $(a + \hat{a})^\top x \leq a^\top x + \delta |a|^\top x \leq c_a$.

We also noticed that this “deep” inner LP can remain feasible by replacing $a^\top x + \delta |a|^\top x \leq c_a$ with $a^\top x + 2\delta |a|^\top x \leq c_a - \Delta$, for some small $\Delta > 0$. The use of this parameter $\Delta$ makes the generated solutions $x_1$ even more deeply interior, pushing them away from the boundary; experiments suggest it is usually better to start from such (well-centered) solutions rather than from a boundary point. This is in line with similar ideas in interior point algorithms for standard LP, i.e., it is better to start out with very interior points.
associated to high barrier terms and to converge towards the boundary only towards the end of the solution process, when the barrier terms converge to zero.

In fact, the above procedure worked perfectly well in practice even for the instances that do contain negative variables; if this ever fails, one can still generate a feasible interior point by reducing \( \delta |a|^\top x \) (up to zero in the worst case). Finally, the first direction \( d_1 \) points to the solution of the nominal problem, i.e., we take \( d_1 = \text{opt}(P_0) - x_1 \), where \( P_0 \) is the polytope of the nominal problem with no robust cut. This is consistent with the general choice \( d_{it} = \text{opt}(P_{it-1}) - x_{it} \) that we will also use at all subsequent iterations \( it \geq 1 \), following an idea from Section 1.

2.2. Multiple-Length Cutting-Stock

2.2.1. The model with prohibitively-many constraints and the pure Cutting-Planes

Cutting-stock is one of the most celebrated problems usually solved by Column Generation, as first proposed in the pioneering work of Gilmore and Gomory in the 1960s. Given a stock of standard-size input pieces (e.g., of paper or metal), the goal is to cut these input pieces into smaller pieces (items) to fulfill a given demand. The pattern-oriented formulation of Cutting-Stock consists of a primal program with prohibitively-many variables, using one variable for each feasible (cutting) pattern – see program (A.2) from Appendix A. After applying a linear relaxation, we obtain the following dual of this primal program.

\[
\begin{align*}
\max \quad & b^\top x \\
\text{subject to} \quad & y_a : a^\top x \leq c_a, \quad \forall (a, c_a) \in A \\
& x \geq 0_n.
\end{align*}
\]

(2.5)

The notations from (2.5) can be directly interpreted in Cutting-Stock terms. Each constraint \( (a, c_a) \in A \) is associated to a primal column representing a (cutting) pattern \( a \in \mathbb{Z}^n_+ \) such that \( a_i \) is the number of items \( i \) to be produced from an input piece (for any item \( i \in [1..n] \)). Considering a vector \( w \in \mathbb{Z}^n_+ \) of item lengths, all feasible patterns \( a \in \mathbb{Z}^n_+ \) have to satisfy \( w^\top a \leq W \), assuming \( W \) is the unique length of all given standard-size pieces. The vector \( b \in \mathbb{Z}^n_+ \) represents the demands for the \( n \) items. Writing the primal LP – see (A.2) from Appendix A – associated to (2.5), one can see how the primal objective function asks to minimize the total cost of the selected patterns.
In pure Cutting-Stock, all feasible patterns \((a, c_a) \in A\) have a fixed unitary cost \(c_a = 1\), but we will focus on the more general Multiple-length Cutting-Stock in which the standard-size pieces can actually have different lengths of different costs. While all discussed algorithms could address an arbitrary number of lengths, we will focus on the case of two lengths \(0.7W\) and \(W\) of costs 0.6 and 1, respectively. The cost of a pattern \(a\) is thus the cost of the smallest standard-size piece that can accommodate \(a\), e.g., if \(w^\top a \leq 0.7W\) then \(c_a = 0\), otherwise \(c_a = 1\).

The standard Column Generation method is equivalent to a Cutting-Planes algorithm that optimizes the above LP (2.5) by iteratively solving the separation subproblem \(\min_{(a, c_a) \in A} c_a - a^\top x\) on the current optimal outer solution \(x = \text{opt}(\mathcal{P}_{it})\) at each iteration \(it\). In Multiple-length Cutting-Stock, this sub-problem is typically solved by Dynamic Programming. In a nutshell, the main idea is to assign for each length \(\ell \in [1..W]\) a state \(s_\ell\) represented by a pattern \(a_\ell \in \mathbb{Z}_+^n\) of length \(\ell\) that minimizes \(c_\ell - a_\ell^\top x = \min \{c_\ell - a^\top x: a^\top w = \ell\}\); this pattern gives the objective value of \(s_\ell\), i.e., \(\text{obj}(s_\ell) = c_\ell - a_\ell^\top x\). One can ignore all non-available lengths \(\ell \in [1..W]\) for which there is no pattern \(a\) such that \(a^\top w = \ell\). The Dynamic Programming scheme generates transitions among such states, and, after calculating them all, it returns \(\min_{\ell \in [1..W]} c_\ell - a_\ell^\top x\).

### 2.2.2. Adapting Projective Cutting-Planes for Multiple-Length Cutting-Stock

Projective Cutting-Planes is not meant to be a rigid algorithm, but it was deliberately designed as a framework that can naturally allow a certain flexibility. To make Projective Cutting-Planes reach its full potential on Multiple-length Cutting-Stock, we need a slightly different approach to choose \(x_{it}\) at each iteration \(it\).

As with other problems explored all along this work and the initial paper [14], a key observation is that defining \(x_{it}\) as the best solution ever found up to iteration \(it\) is not efficient in the long run, partly because \(x_{it}\) could fluctuate too much from iteration to iteration. Furthermore, we will also see in Section 2.2.3.2 that the projection sub-problem \(\text{project}(x \rightarrow d)\) can be solved more rapidly when \(x\) is a “truncated” solution, e.g., when \(x_i\) is a multiple of \(\gamma = 0.2\) for each \(i \in [1..n]\). For these reasons, we propose a slightly different Projective Cutting-Planes variant in which the choice of \(x_{it}\) is performed as follows.

Let us first introduce the operator \([x]_\gamma\) that truncates \(x\) down to multiples of some \(\gamma \in \mathbb{R}_+\) (we used \(\gamma = 0.2\), i.e., \(x_i\) becomes \(\gamma \cdot \left\lfloor \frac{1}{\gamma} x_i \right\rfloor\) for any \(i \in [1..n]\). Let \(x_{it}^{\text{bst}}\) denote the best
truncated feasible solution generated up to the current iteration; \( x_{\text{bst}} \gamma \) can be determined as follows: start with \( x_{\text{bst}} \gamma = 0_n \) at iteration \( i = 1 \), and replace \( x_{i\gamma} \) with \( \lfloor x_{i\gamma} + t_{i\gamma} d_{i\gamma} \rfloor \) at each iteration \( i > 1 \) where \( b^\top \lfloor x_{i\gamma} + t_{i\gamma} d_{i\gamma} \rfloor > b^\top x_{\text{bst}} \gamma \). We propose to choose the inner solution \( x_{i\gamma} \) at each iteration \( i \) based on the following rules:

\begin{itemize}
  \item set \( x_{i\gamma} = 0_n \) in half of the cases (half of the iterations);
  \item set \( x_{i\gamma} = x_{\text{bst}} \gamma \) in 25\% of the cases;
  \item set \( x_{i\gamma} = \lfloor \frac{1}{2} x_{\text{bst}} \gamma \rfloor \) in 25\% of the cases.
\end{itemize}

The advantage of the first choice \( x_{i\gamma} = 0_n \) is that the associated projection sub-problem can be solved more rapidly. Projecting from \( 0_n \) is always easier. We will have more to say about this in Section 2.2.3.2, but, for now, you can already check how (2.6) below is greatly simplified by using \( x = 0_n \); in such a case, (2.6) could even reduce to a very knapsack-like problem that mainly asks to maximize the denominator \( d^\top a \). The second choice \( x_{i\gamma} = x_{\text{bst}} \gamma \) is useful because the projection \( x_{\text{bst}} \gamma \rightarrow d_{i\gamma} \) may lead to a higher-quality pierce point. The last choice is a trade-off between the first two choices.

2.2.3. Solving the Projection Sub-problem

Numerous Column Generation algorithms for cutting and packing problems rely on Dynamic Programming (DP) to solve the separation sub-problem. And, in many such cases, if the separation sub-problem can be solved by Dynamic Programming, then so can be the projection one.

Given a feasible \( x \in \mathcal{P} \) in (2.5) and a direction \( d \in \mathbb{R}^n \), recall that the projection subproblem \( \text{project}(x \rightarrow d) \) asks to minimize (2.1). For Multiple-length Cutting-Stock, (2.1) is instantiated as follows:

\[
t^* = \min_a \left\{ \frac{f(w^\top a) - a^\top x}{d^\top a} : a \in \mathbb{Z}^n_+, \ w^\top a \leq W, \ d^\top a > 0 \right\}, \quad (2.6)
\]

where the function \( f : [0,W] \rightarrow \mathbb{R}_+ \) maps each \( \ell \in [0,W] \) to the cost of the cheapest (shortest) standard-size input piece of length at least \( \ell \). The DP scheme proposed next can work for any function \( f \) that is non-decreasing, i.e., encoding the natural assumption that shorter pieces are cheaper than longer pieces. Many different Cutting-Stock variants (e.g., Variable-Sized Bin-Packing or Elastic Cutting Stock) can be formulated using an appropriate choice of such a function \( f \) [12, §4.1.1].
2.2.3.1. The main Dynamic Programming scheme and the states

We consider a set $S_\ell$ of DP states for every feasible length $\ell \in [0..W]$. Each state $s \in S_\ell$ is associated to all patterns $a \in A$ of:

1. length $s_{\text{len}} = w^\top a = \ell$;
2. profit $s_p = d^\top a$.
3. cost $s_c = f(w^\top a) - a^\top x = f(\ell) - a^\top x = c_a - a^\top x$;

All states in $S_\ell$ have the same length $\ell$ but their costs and profits can vary. Under this cost/profit interpretation, (2.6) reduces to minimizing the cost/profit ratio $\text{obj}(s) = \frac{s_c}{s_p}$ over all states $s$ ever generated, i.e., $\min \left\{ \text{obj}(s) = \frac{s_c}{s_p} : s \in S_\ell, \ \ell \in [0..W] \right\}$. Notice any feasible pattern can be associated to a state, although we will see that some of these states are dominated and do not need to be recorded. Finally, the above cost $s_c = f(w^\top a) - a^\top x = c_a - a^\top x$ is always non-negative because $x \in P$ satisfies all constraints of (2.5).

The proposed DP algorithm starts only with an initial null state of length 0, cost 0 and profit 0. It then performs a DP iteration for each item $i \in [1..n]$; if $b_i > 1$, this iteration is performed $b_i$ times because a pattern can contain up to $b_i$ copies of item $i$. Each such DP iteration generates transitions from the current states to produce new states or to update the existing ones. A state transition $s \rightarrow s'$ associated to an item $i$ leads to a state $s'$ such that:

(a) $s'_{\text{len}} = s_{\text{len}} + w_i$, i.e., the length simply increases by adding a new item;
(b) $s'_p = s_p + d_i$, i.e., add the profit of item $i$;
(c) $s'_c = s_c + f(s'_{\text{len}}) - f(s_{\text{len}}) - x_i$, i.e., the term $f(s'_{\text{len}}) - f(s_{\text{len}})$ updates the cost of the pattern whose size increased from $s_{\text{len}}$ to $s'_{\text{len}}$, and $-x_i$ comes from the “$-a^\top x$” term used in the state cost definition $f(\ell) - a^\top x$ from the above point (3).

Algorithm 1 provides the pseudo-code executed for each item $i \in [1..n]$ considered $b_i$ times. The most complex operation arises at Step 5, where one needs to check that the new state $s'$ is not dominated by an existing state in $S_{\ell+w_i}$ before inserting it in $S_{\ell+w_i}$; the efficient implementation of this step is described in Section 2.2.3.2.

We can say this pseudo-generalizes the DP algorithm for the separation sub-problem (which asks to minimize $s_c - s_p$ instead of $\frac{s_c}{s_p}$). Indeed, the separation DP scheme may easily be described using the same framework. Recall its goal is to solve the (knapsack-like) sub-problem $\min \{ f(w^\top a) - d^\top a : a \in \mathbb{Z}^n, \ w^\top a \leq W \}$ for some $d \in \mathbb{R}^n$. For this purpose, it is enough to consider only singleton sets $S_\ell = \{ s \}$, where $s$ is a state defined by a pattern...
Algorithm 1 The Dynamic Programming steps executed $b_i$ times for each item $i$

1. for $\ell = W - w_i$ to 0:

2. for each $s \in S_\ell$: $\triangleright$ for each state with length $\ell$

3. initialize state $s'$ with $s'_{\text{len}} = \ell + w_i$, according to above formula (a)

4. calculate $s'_c$, $s'_p$ with above formulae (b) and (c)

5. if $s'$ is not dominated by an existing state in $S_{\ell+w_i}$ (Section 2.2.3.2) then

   - $S_{\ell+w_i} \leftarrow S_{\ell+w_i} \cup \{s'\}$
   - record the transition $s \rightarrow s'$ (to reconstruct an optimal pattern in the end)

The projection sub-problem is more difficult because it is no longer enough to record a unique state per length as above. To illustrate this, notice that a state with a cost/profit ratio of $\frac{5}{4}$ does not necessarily dominate a state with a cost/profit ratio of $\frac{3}{2}$ only because $\frac{5}{4} < \frac{3}{2}$. Indeed, the $\frac{5}{4}$ state can evolve to a sub-optimal state by following a transition that decreases the cost by 1 and increases the profit by 4 because $\frac{5-1}{4+4} = \frac{4}{8} < \frac{3-1}{2+4} = \frac{2}{6}$. This cannot happen in the (knapsack-like) separation sub-problem, i.e., the relative order of two states defined by cost–profit differences would never change because all transitions induce linear (additive) changes to such differences.

2.2.3.2. Reducing the number of DP states to accelerate the DP projection algorithm. To accelerate the projection, we need to reduce the number of recorded states. First, let us show it is enough to record a unique maximum-profit state for each feasible cost of a state of fixed length $\ell$ (in $S_\ell$). For this, consider two states $s^*, s \in S_\ell$ such that $s^*_c = s_c$ and $s^*_p > s_p$. The state $s$ is dominated and can be ignored because any transition(s) equally applied on $s^*$ and $s$ would lead to the same cost $s^*_c + \Delta_c = s_c + \Delta_c > 0$ and to profits $s^*_p + \Delta_p > s_p + \Delta_p$; this way, it is easy to check that $\frac{s^*_c + \Delta_c}{s^*_p + \Delta_p} < \frac{s_c + \Delta_c}{s_p + \Delta_p}$ always holds when the denominators are positive. And these denominators are always positive for any final state (that could ever be returned), because of the condition $d^Ta > 0$ from (2.6).
Let us now compare $s^*$ to a state $s \in S_\ell$ that satisfies $s_c > s^*_c$ and $s_p \leq s^*_p$. Such state $s$ is also dominated by $s^*$ because it can only lead via transitions to $\frac{s_c + \Delta_c}{s^*_c + \Delta_c} < \frac{s_p + \Delta_p}{s^*_p + \Delta_p}$.

As such, a state $s \in S_\ell$ with a higher cost than an existing state $s^* \in S_\ell$ (i.e., $s_c > s^*_c$) must have a higher profit to be non-dominated, i.e., a state $s$ such that $s_c > s^*_c$ has to satisfy $s_p > s^*_p$ to be non-dominated. This can be seen as a formalization of a very natural principle “pay a higher cost only when you gain a higher profit”. The cost and the profits of all non-dominated states in $S_\ell$ can thus be ordered using a (Pareto dominance) relation of the form:

$$c_1 < c_2 < c_3 < \ldots \quad (2.7a)$$
$$p_1 < p_2 < p_3 < \ldots \quad (2.7b)$$

Let us now investigate how long these lists (2.7.a)–(2.7.b) can be for each $S_\ell \forall \ell \in [0..W]$. If there are fewer potential costs values, these lists have to be shorter, and so, the total number of states is reduced. Accordingly, if all pattern costs $f(\ell)$ ($\forall \ell \in [0,W]$) are multiples of $\gamma = 0.2$ and if all selected interior points $x$ satisfy $x_i \in \{0,0.2,0.4,0.6,0.8,1\}$, then the maximum number of feasible costs values is 6, i.e., any state cost has the form $f(\ell) - a^\top x$ for some $a \in \mathbb{Z}^n_+$ and, thus, it has to be a number from the set $\{0,0.2,0.4,0.6,0.8,1\}$. This way, the resulting DP algorithm might often need to record only a few states per length; this means it is not necessarily much slower than a separation DP algorithm that records a unique state per length.

Finally, we need a fast data structure to manipulate lists of cost/profit pairs satisfying (2.7.a)–(2.7.b), because it is important to accelerate the following two operations executed by Algorithm 1:

1. iterate over all elements of $S_\ell$ to implement the for loop at Line 2;
2. insert a new state at Line 5 after checking that it is not dominated.

A list of cost/profit values satisfying (2.7.a)–(2.7.b) can be seen as a Pareto frontier with two objectives (minimize the cost and maximize the profit). It is not difficult to scan the elements of such a frontier to implement the above operation (i). The most computationally-demanding task is to insert a new state for the above operation (ii), because this requires checking if the new state is dominated by an existing state. Checking this by naively scanning the whole list of cost/profit values is not the most efficient approach. We will see it is better to record this list in a self-balancing binary search tree [8, § 6.2.3] that can perform
many look-up operations in logarithmic time. A further complication comes from the fact
that the insertion of a new non-dominated state can lead to the removal of other existing
states that become dominated. This actually explains the need for a specific, more refined,
self-balancing binary tree data structure described in Appendix B.

Finally, to further accelerate the DP, experiments suggest it can be useful (in practice)
to sort the items \( i \in [1..n] \) in descending order of the value \( \frac{w_i}{1+x^\text{bst}_i} \). Precisely, Algorithm 1 is
executed for each of the items \([1..n]\) considered in this order. In a loose sense, this amounts
to considering that it is better to start with longer items that did not contribute too much
to the best truncated inner solution \( x^\text{bst} \) ever found.

3. Numerical Experiments

3.1. Robust linear programming

We use the Netlib instances from [4], considering \( \Gamma \in \{1,10,50\} \). In fact, we discarded
all instances that are infeasible for \( \Gamma = 50 \), since our methods are not designed to detect
infeasibilities. We also ignored all instances solved by the algorithm from [4] in less than 5
iterations (i.e., seba, shell and woodw) because they are too small to produce meaningful
comparisons; they are also the only instances that Projective Cutting-Planes can solve
in very few iterations. We thus remain with a test bed of 21 instances with between \( n = 1000 \)
and \( n = 15695 \) variables. Most instances have between \( n = 1000 \) and \( n = 5000 \) variables
and a number of constraints between 500 and 3000. We refer to [4, Table 1] for the
nominal objective value of each instance. We mention that stocfor3 is an exceptionally
large instance with \( n = 15695 \) and more than 15000 constraints. For even greater detail
on their characteristics, the instances are publicly available on-line in a human-readable
format (the original MPS files are difficult to parse) at cedric.cnam.fr/~porumbel/
projcutplanes/instances-robust.zip.

Recall our robust optimization problem has a minimization objective, so that the inner
solutions \( x_{it} \) determined by Projective Cutting-Planes along the iterations \( it \) generate
upper bounds \( b^\top x_{it} \).

3.1.1. The running profile

Figure 2 plots the running profile of the standard Cutting-Planes compared to that of the Projective Cutting-Planes on two instances. The standard Cutting-Planes needed 83 iterations to fully converge on the first instance, while Projective Cutting-Planes reported a feasible solution with a proven low gap of
0.06% after only 24 iterations. On the second instance, Cutting-Planes needed 207 iterations to fully converge, while Projective Cutting-Planes reported a feasible solution with a proven low gap of 3.1% after 108 iterations, as indicated by the arrows in the figure.

3.1.2. The main results in tabular form Table 1 compares the total computing effort (iterations and CPU time) needed to fully solve each instance using the new and the standard method. For Projective Cutting-Planes, we also provide the computing effort needed to reach a gap of 1% between the lower and the upper bounds; this may often require a very short time. For example, the standard Cutting-Planes needed between 45 minutes and one hour (depending on \( \Gamma \)) to determine the optimal solution for the last instance stocfor3, while the Projective Cutting-Planes reported in less than 3 seconds a feasible solution with a proven gap below 1% (see columns “gap 1%” in bold in the last row). In many practical settings, this could represent a satisfactory feasible solution. It is also true that the robust optimal solution is often less than 3% higher than the nominal optimum (see Column 2 of Table 1). The value of the starting solution \( x_1 \) might be only a few percentage points higher than the nominal optimum.
<table>
<thead>
<tr>
<th>Instance</th>
<th>25fv47</th>
<th>bn12</th>
<th>czprob</th>
<th>ganges</th>
<th>gfrd-pnc</th>
<th>maros</th>
<th>nesm</th>
<th>pilotja</th>
<th>pilotnov</th>
<th>pilotea</th>
<th>scfxm2</th>
<th>scfxm3</th>
<th>sctap3</th>
<th>ship08l</th>
<th>ship08s</th>
<th>ship12l</th>
<th>ship12s</th>
<th>sierra</th>
<th>stocfor2</th>
<th>stocfor3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>2.538</td>
<td>1.847</td>
<td>0.6401</td>
<td>0.4736</td>
<td>0.0649</td>
<td>12.12</td>
<td>0.8752</td>
<td>4.877</td>
<td>8.51</td>
<td>6.109</td>
<td>2.114</td>
<td>2.142</td>
<td>2.844</td>
<td>3.04</td>
<td>0.1244</td>
<td>0.1396</td>
<td>0.3528</td>
<td>0.3898</td>
<td>0.0239</td>
<td>1.522</td>
</tr>
<tr>
<td>new method (+%)</td>
<td>146</td>
<td>486</td>
<td>61</td>
<td>56</td>
<td>64</td>
<td>272</td>
<td>56</td>
<td>121</td>
<td>96</td>
<td>102</td>
<td>93</td>
<td>139</td>
<td>185</td>
<td>145</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>gap 1%</td>
<td>1.618</td>
<td>13.66</td>
<td>0.485</td>
<td>&lt;0.001</td>
<td>0.1404</td>
<td>2.418</td>
<td>0.579</td>
<td>2.181</td>
<td>4.227</td>
<td>0.97</td>
<td>0.387</td>
<td>0.957</td>
<td>1.946</td>
<td>2.649</td>
<td>0.111</td>
<td>0.006</td>
<td>&lt;0.001</td>
<td>0.015</td>
<td>0.001</td>
<td>0.022</td>
</tr>
<tr>
<td>full converg. iters time</td>
<td>149</td>
<td>491</td>
<td>734</td>
<td>25</td>
<td>64</td>
<td>278</td>
<td>80</td>
<td>161</td>
<td>120</td>
<td>118</td>
<td>139</td>
<td>196</td>
<td>242</td>
<td>239</td>
<td>29</td>
<td>32</td>
<td>48</td>
<td>63</td>
<td>54</td>
<td>437</td>
</tr>
<tr>
<td>full converg. iters time</td>
<td>1.191</td>
<td>13.81</td>
<td>32.3</td>
<td>0.059</td>
<td>0.141</td>
<td>2.458</td>
<td>0.798</td>
<td>3.042</td>
<td>4.615</td>
<td>1.102</td>
<td>0.584</td>
<td>1.353</td>
<td>2.567</td>
<td>4.55</td>
<td>0.171</td>
<td>0.122</td>
<td>0.442</td>
<td>0.377</td>
<td>5047</td>
<td>7777</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
<th>50</th>
<th>10</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>254.8</td>
<td>1.847</td>
<td>0.6401</td>
</tr>
<tr>
<td>new method (+%)</td>
<td>146</td>
<td>486</td>
<td>61</td>
</tr>
<tr>
<td>gap 1%</td>
<td>1.618</td>
<td>13.66</td>
<td>0.485</td>
</tr>
<tr>
<td>full converg. iters time</td>
<td>149</td>
<td>491</td>
<td>734</td>
</tr>
<tr>
<td>full converg. iters time</td>
<td>1.191</td>
<td>13.81</td>
<td>32.3</td>
</tr>
</tbody>
</table>

Table 1: Results of Projective Cutting-Planes and standard Cutting-Planes on the robust optimization instances. The columns OPT indicate the increase in percentage of the robust objective value with respect to the nominal one (with no robustness). Columns “gap 1%” indicate the computing effort needed to reach the iteration \( \text{it} \) when the gap between the upper bound \( b^T x_{it} \) and the lower bound \( \text{optVal}(P_{1t}) \) is below 1%, i.e., either \( 0 < \text{optVal}(P_{1t}) \leq b^T x_{it} \leq 1.01 \text{optVal}(P_{1t}) \) or \( \text{optVal}(P_{1t}) \leq b^T x_{it} \leq 0.99 \text{optVal}(P_{1t}) < 0. \)
For sctap2 and sctap3 with $\Gamma = 50$, the standard Cutting-Planes is seriously slowed down by degeneracy issues, \textit{i.e.}, it performs too many Simplex pivots that only change the Simplex basis without improving the objective value. It thus needs significantly more iterations than normally expected — see the italic font figures in the rows of sctap2 and sctap3. We suppose that such degeneracy phenomena are also visible for czprob with $\Gamma = 50$ in Table 1 of [4], because their algorithm takes 100 times more time for $\Gamma = 50$ than for $\Gamma = 10$, which is unusual.

**Remark 2** Except for the above experiments, the degeneracy issues of the standard Cutting-Planes are not very visible in other Cutting-Planes implementations from this work (including [14]). Yet such problems are well acknowledged in the Cutting-Planes literature, especially in Column Generation. As [9, §4.2.2] put it, “When the master problem is a set partitioning problem, large instances are difficult to solve due to massive degeneracy [...] Then, the value of the dual variables are no meaningful measure for which column to adjoin to the Reduced Master Problem”. In Projective Cutting-Planes, the inner-outer solutions $x_{it}$ and $\text{opt}(P_{it-1})$ represent together a more “meaningful measure” for selecting a new constraint, avoiding iterations that keep the objective value constant. In fact, as hinted at point 2 of [14, §2], a projection cannot keep the objective value constant when $x_{it}$ is strictly interior (which is surely the case when $\alpha < 1$). This comes from the fact that the objective value cannot deteriorate or remain constant by advancing along $x_{it} \rightarrow d_{it}$, because $x_{it} + d_{it} = \text{opt}(P_{it-1})$ and $x_{it}$ belongs to the strict interior of $P_{it-1} \supseteq \mathcal{P}$.

### 3.1.3. Beyond the standard Cutting-Planes and the standard Projective Cutting-Planes

We here consider a multi-cut version of both Projective Cutting-Planes and the standard Cutting-Planes, \textit{i.e.}, we enable both algorithms to return multiple cuts at each iteration; recall the approach from [4] also returns multiple cuts per iteration.

The most straightforward multi-cut separation one can imagine works in two steps: (1) determine for each nominal constraint the strongest robust cut with regards to the current outer optimal solution $x^{\text{out}} \notin \mathcal{P}_{it}$, (2) return all cuts determined at Step (1) that are violated by $x^{\text{out}}$. However, numerical experiments show this may generate an important computational bottleneck: the polytopes $\mathcal{P}_{it}$ constructed along the iterations $it$ may become too “heavy”, containing too many constraints. As such, we use a more practical multi-cut separation that only returns the five most violated robust cuts. If there are less
than five robust cuts violated by $x^{\text{out}}$, then the separation algorithm returns only these less than five cuts. For comparison, recall that the mono-cut standard Cutting-Planes returns only the most violated cut. As a further protection against computational explosions, we stop returning multiple cuts (i.e., we swap to mono-cut separation) if we ever detect that the outer approximation $P_{1t}$ contains 10000 (new) robust cuts.

The multi-cut projection is designed as follows: start from the mono-cut algorithm from Steps 1–5 of Section 2.1.2, but return all robust cuts that generate a step decrease in (2.4) at Step 3. This is a simple generalization of the mono-cut projection algorithm that executes the same five steps (for each nominal constraint) but only returns the very best robust cut that minimizes the final step length. We also use the protection against computational explosions used for the separation: we switch to the mono-cut algorithm if we ever detect that $P_{1t}$ contains 10000 (new) robust constraints.

Table 2 compares the multi-cut Projective Cutting-Planes and the multi-cut Cutting-Planes using almost the same format and the same columns as in Table 1. In fact, we removed the “OPT” columns and we added the new columns “cuts” that report the total number of new robust cuts ever generated to fully converge.

Comparing Table 2 to the previous Table 1, it is quite clear that, for both algorithms, the multi-cut variant is superior to the mono-cut one. Focusing on Table 2 only, the multi-cut Projective Cutting-Planes requires (far) less iterations than the multi-cut Cutting-Planes; there are only three exceptions to this rule (instance ganges for $\Gamma = 50$ and sierra for $\Gamma \in \{10,50\}$). On roughly half of the instances, the multi-cut Projective Cutting-Planes requires a one-digit number of iterations while this happens only rarely for any other algorithm considered in this paper. In terms of CPU time, the multi-cut Projective Cutting-Planes is at least 10 times faster than the Cutting-Planes for roughly a third of the instances (in such cases, the CPU time is reported in italic font).

In no few cases, the multi-cut Projective Cutting-Planes variant from this section is a real success. For example, it solved the very large instance stocfor3 for $\Gamma = 1$ in three iterations and roughly 8 seconds, while all other algorithms studied in this paper needed thousands of iterations and thousands of seconds for the same instance. Let us discuss how these three iterations proceeded to determine the optimal value. The first projection generated 7853 robust cuts; at the second iteration, Projective Cutting-Planes stopped at 2147 cuts because it reached the limit $7853 + 2147 = 10000$. This forced it to switch to
Table 2: Results of Projective Cutting-Planes and Standard Cutting-Planes with multiple cuts per round. The CPU times are reported in table form for all instances on which Projective Cutting-Planes is at least ten times faster.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
<td>iters</td>
<td>time</td>
<td>cuts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
a mono-cut projection mode. The third and last iteration simply confirmed the feasibility of the outer solution obtained after the first two iterations.\(^1\) It is clear here that the robust cuts determined using a projection logic are much stronger than the ones discovered using the classical separation logic.

### 3.2. Multiple–Length Cutting-Stock

Let us consider a Multiple-length Cutting-Stock variant with two types of standard-size input pieces: one of length \(W\) and cost 1, and one of length \(0.7W\) and cost 0.6. Preliminary experiments confirm that introducing a third type of standard-size piece lead to similar experimental conclusions. We prefer Multiple-length Cutting-Stock over the standard Cutting-Stock because it is more general: (i) the constraints \((a, c_a) \in A\) of the Column Generation dual LP (2.5) do not satisfy all \(c_a = 1\), and (ii) it is not possible to generate lower bounds using the Dual Feasible Functions that proved so effective in standard Cutting-Stock [3].

Let us mention that we warm-start Projective Cutting-Planes by executing the first two projections in a problem-specific (ad-hoc) manner. More exactly, let us choose \(x_1 = 0_n\) and \(d_1 = \frac{1}{W}w\) for the first iteration and \(x_2 = 0_n\) and \(d_2 = b\) for the second one. The choice of projecting along \(0_n \rightarrow \frac{1}{W}w\) for \(i_t = 1\) is inspired by research in dual feasible functions for Cutting-Stock problems [3], which shows that \(\frac{1}{W}w\) is often a dual-feasible solution (in pure Cutting-Stock) of very high quality. The choice for \(i_t = 2\) is a rather standard one: the projection towards \(b\) makes Projective Cutting-Planes advance along the direction with the fastest rate of objective function improvement. This enables Projective Cutting-Planes to determine two lower bounds and two initial constraints.

To reduce any bias, we also warm-start the standard Column Generation in a similar manner, i.e., before launching the standard iterations, we solve the separation sub-problem on \(b\) and \(\frac{1}{W}w\), generating two initial constraints. However, in Column Generation, these two sub-problems can only generate one Lagrangian lower bound associated to \(b\). We cannot calculate such a lower bound for \(\frac{1}{W}w\) because the Lagrangian bound does not hold for a (dual) feasible solution like \(\frac{1}{W}w \in P\) associated to a non-negative reduced cost, see full details in Remark 3 of Appendix A.

\(^1\) The source code is available on-line at the github repository of INFORMS Journal on Computing [11], for both problems considered in the paper, i.e., robust linear programming and (multiple-length) cutting-stock.
3.2.1. The standard Projective Cutting-Planes

We consider ten well-studied benchmark instance sets [3, 16] and we take the first 3 instances from each set. For each set, the number (ID) of each individual instance is indicated by a suffix, e.g., we write m01-1, m01-2, m01-3 to refer to the first, second and third instance respectively from the benchmark set m01. The characteristics of the instances (i.e., the values of $n$, $W$, $b$, etc) and their origins are described in Table 1 from [15].

Table 3 compares the Projective Cutting-Planes (from Section 2.2.2) to the standard Column Generation on these instances. Column 1 represents the instance, Column 2 indicates the optimal value of (2.5), Columns 3–6 report the results of the new method, and Columns 7–11 provide the results of the standard Column Generation. For both methods, Table 3 first indicates the computing effort (iterations and CPU time) needed to reach a gap of 20% (i.e., so that $ub \leq 1.2\cdot lb$) and then the total computing effort needed to fully converge. All reported CPU times are smaller than those reported in the companion paper (Section 2, p. 6) of [15], for both the new method and the standard Column Generation. This cannot only be explained by the hardware evolution, but also by a better implementation. In Column 5 we indicate in parentheses the number of Projective Cutting-Planes iterations as a percentage of the number (CG-Std) of Column Generation iterations (i.e., as a percentage of Column 9).

The last column reports the minimum and the maximum value (over ten runs) of the ratio $\frac{CG-Stab}{CG-Std}$, where CG-Stab is the number of iterations needed by the stabilized Column Generation from [15, § 6.1.2] and CG-Std is the number of iterations of the standard Column Generation. The ten considered runs were randomized by choosing at each iteration an arbitrary optimal solution of value $\text{optVal}(\mathcal{P}_i)$; this artificial randomization is obtained by optimizing a random objective function while keeping the value of the original legitimate objective function at $\text{optVal}(\mathcal{P}_{it})$. Referring again to [15, § 6.1.2] for full details, we mention that we used the best stabilization techniques for Cutting-Stock from [10]: dual solution smoothing and adding a step-wise penalty to the (dual) objective function. The best parameters for these stabilization techniques were chosen on an instance by instance basis, exactly as in [15, § 6.1.2].

Table 3 demonstrates that Projective Cutting-Planes reaches the 20% gap three or four times more rapidly than the standard Column Generation (compare Columns 3–4 to Columns 7–8). This is mostly due to the fact that Projective Cutting-Planes can
<table>
<thead>
<tr>
<th>Instance</th>
<th>OPT</th>
<th>Projective Cutting-Planes</th>
<th>Standard Column Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>gap 20% full convergence</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>iters time[s]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>iters time[s]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stabilized</td>
</tr>
<tr>
<td>m01-1</td>
<td>49.3</td>
<td>90 0.02 166 (86%) 0.05</td>
<td>187 0.07 194 0.08</td>
</tr>
<tr>
<td>m01-2</td>
<td>53</td>
<td>82 0.02 140 (69%) 0.04</td>
<td>171 0.06 202 0.07</td>
</tr>
<tr>
<td>m01-3</td>
<td>48.2</td>
<td>70 0.02 134 (63%) 0.04</td>
<td>180 0.07 212 0.08</td>
</tr>
<tr>
<td>m20-1</td>
<td>56.6</td>
<td>79 0.02 101 (69%) 0.03</td>
<td>101 0.03 148 0.04</td>
</tr>
<tr>
<td>m20-2</td>
<td>58.7</td>
<td>73 0.02 103 (59%) 0.02</td>
<td>123 0.04 175 0.05</td>
</tr>
<tr>
<td>m20-3</td>
<td>64.8</td>
<td>61 0.01 116 (85%) 0.02</td>
<td>118 0.03 136 0.03</td>
</tr>
<tr>
<td>m35-1</td>
<td>73.9</td>
<td>61 0.01 61 (95%) 0.01</td>
<td>64 0.01 64 0.01</td>
</tr>
<tr>
<td>m35-2</td>
<td>71.5</td>
<td>125 0.02 125 (87%) 0.02</td>
<td>143 0.02 143 0.02</td>
</tr>
<tr>
<td>m35-3</td>
<td>73.7</td>
<td>67 0.01 67 (82%) 0.01</td>
<td>82 0.01 82 0.01</td>
</tr>
<tr>
<td>vb50c1-1</td>
<td>866.3</td>
<td>46 0.8 82 (73%) 2.2</td>
<td>83 5.5 113 8.3</td>
</tr>
<tr>
<td>vb50c1-2</td>
<td>842.5</td>
<td>39 1.6 86 (71%) 2.5</td>
<td>91 7.6 121 9.6</td>
</tr>
<tr>
<td>vb50c1-3</td>
<td>860.2</td>
<td>37 1.5 85 (74%) 3.1</td>
<td>87 6.9 115 9.5</td>
</tr>
<tr>
<td>vb50c2-1</td>
<td>672.3</td>
<td>55 2.2 114 (90%) 9.8</td>
<td>82 13.1 127 20.2</td>
</tr>
<tr>
<td>vb50c2-2</td>
<td>593.1</td>
<td>40 1.9 80 (58%) 5.1</td>
<td>88 11 139 21.1</td>
</tr>
<tr>
<td>vb50c2-3</td>
<td>480.048</td>
<td>36 3.5 181 (84%) 47.2</td>
<td>75 20.6 216 76.3</td>
</tr>
<tr>
<td>vb50c3-1</td>
<td>282</td>
<td>37 11.7 122 (68%) 57.6</td>
<td>67 36.1 179 105</td>
</tr>
<tr>
<td>vb50c3-2</td>
<td>239.398</td>
<td>37 16.8 115 (79%) 64.6</td>
<td>60 30.6 145 85.1</td>
</tr>
<tr>
<td>vb50c3-3</td>
<td>271.398</td>
<td>36 12.9 132 (76%) 65.3</td>
<td>68 38.2 173 109</td>
</tr>
<tr>
<td>vb50c4-1</td>
<td>579.548</td>
<td>40 3.5 115 (73%) 17.5</td>
<td>73 12.5 158 35.5</td>
</tr>
<tr>
<td>vb50c4-2</td>
<td>551.01</td>
<td>36 3 123 (74%) 21.9</td>
<td>73 18.5 166 46.6</td>
</tr>
<tr>
<td>vb50c4-3</td>
<td>700.039</td>
<td>40 2.3 111 (76%) 9.9</td>
<td>81 11.9 147 24.8</td>
</tr>
<tr>
<td>vb50c5-1</td>
<td>337.8</td>
<td>40 8.7 133 (58%) 51.9</td>
<td>61 24.8 228 109</td>
</tr>
<tr>
<td>vb50c5-2</td>
<td>349.799</td>
<td>30 4.8 130 (63%) 44.1</td>
<td>64 21 207 81.4</td>
</tr>
<tr>
<td>vb50c5-3</td>
<td>295.775</td>
<td>36 11 115 (65%) 53.6</td>
<td>71 28.4 177 83.9</td>
</tr>
<tr>
<td>wäscher-1</td>
<td>24.0648</td>
<td>71 0.2 319 (66%) 4.2</td>
<td>294 2.3 483 4.7</td>
</tr>
<tr>
<td>wäscher-2</td>
<td>22.0003</td>
<td>69 0.2 501 (103%) 8.6</td>
<td>158 1 481 6.7</td>
</tr>
<tr>
<td>wäscher-3</td>
<td>12.1219</td>
<td>31 0.03 110 (65%) 0.3</td>
<td>110 0.3 170 0.5</td>
</tr>
<tr>
<td>hard-sch-1</td>
<td>51.4254</td>
<td>112 14.7 345 (48%) 69.2</td>
<td>345 48.1 712 115</td>
</tr>
<tr>
<td>hard-sch-2</td>
<td>51.4426</td>
<td>116 15.1 339 (49%) 67</td>
<td>365 50.9 685 110</td>
</tr>
<tr>
<td>hard-sch-3</td>
<td>50.5957</td>
<td>110 15.1 295 (47%) 58.6</td>
<td>357 52.8 630 107</td>
</tr>
</tbody>
</table>

Table 3 Projective Cutting-Planes compared to the standard Column Generation on Multiple-length Cutting-Stock. The value in parentheses in Column 5 reports the ratio between the number of Projective Cutting-Planes iterations and the number of Cutting-Planes iterations (Column 5 divided by Column 9). The Projective Cutting-Planes reduced the number of iterations by at least a third (i.e., to less than 66%) on more than a third of the instances, see the bold figures in Columns 5 in parentheses.
generate high-quality lower bounds from the very first iterations, as we will also see in the running profiles from Figure 3.

Regarding the complete convergence, the “upgrade” from the standard Cutting-Planes to Projective Cutting-Planes resulted in an average reduction of the number of iterations to 72% and of the CPU time to 61%. For the last three (most difficult) instances, the Projective Cutting-Planes reduced the number of iterations to roughly 50%.

Let us also compare this iteration speed-up to the one that could be achieved by stabilizing the Column Generation. Focusing on the minimum value reported in the last column of Table 3, we observe that none of the stabilized Column Generation runs could reduce the number of iterations to less than 85% for roughly half of the instances. In contrast, the values in parentheses in Column 5 show that almost all Projective Cutting-Planes runs managed to reduce the number of iterations to less than 85% (there are six exceptions to this rule). Confirming [15, § 6.1.2], the stabilization is really very successful only on the m35 instances that can be solved in 0.01 seconds. Still, we can not claim that the number of iterations reported by Projective Cutting-Planes is systematically smaller than the (minimum) number of iterations a stabilized Column Generation can reach.

3.2.2. An aggressive Projective Cutting-Planes Let us now consider an aggressive Projective Cutting-Planes that chooses \( x_{it} = x_{it-1} + t^*_{it-1}d_{it-1} \), i.e., \( x_{it} \) is the best feasible solution discovered up to now (the last pierce point). This aggressive Projective Cutting-Planes starts very well by strictly increasing the lower bound with each iteration \( it \), i.e., check that \( b^\top x_{it} = b^\top (x_{it-1} + t^*_{it-1}d_{it-1}) \geq b^\top x_{it-1} \) is surely satisfied because \( t^*_{it-1} = 0 \). This way, the lower bound \( b^\top x_{it} \) becomes constantly increasing along the iterations \( it \). This eliminates the infamous “yo-yo” effect appearing very often (if not always) in Column Generation—i.e., the “yo-yo” up and down oscillations of the lower bound values reported along the iterations.

Figure 3 presents the lower bounds calculated along the iterations by three methods: the above aggressive Projective Cutting-Planes (in black), the standard Projective Cutting-Planes (in red) and the standard Column Generation (Lagrangian lower bounds in blue). This figure demonstrates that the aggressive Projective Cutting-Planes starts very well by strictly increasing the lower bound at each iteration (no “yo-yo” effect); at
certain iterations, the increase is actually too small to be visible in the figure, but we can certify it is real in the actual data. However, this aggressive variant needs significantly more CPU time than the standard Projective Cutting-Planes to fully convergence. This comes from the fact that the aggressive algorithm does not use truncated interior points \( \mathbf{x}_{it} \), so that its iterations are significantly slower. For example, on the first instance m01-1 from Figure 3, even if the aggressive Projective Cutting-Planes needs 20% less iterations, its total convergence time is approximately three times larger than that of the standard Projective Cutting-Planes. For the second instance vb50c1-1 in Figure 3, the aggressive algorithm needs 9 times more (CPU) time. More generally, preliminary
experiments indicate that even the original non-aggressive Projective Cutting-Planes would be a few times slower without the truncation feature from Section 2.2.2.

### 3.3. The oscillations of the inner solutions and the “bang-bang” effects

The goal of this section is to (try to) gain more insight into why an “aggressive” definition of $x_{it}$ like $x_{it} = x_{it-1} + t^*_it_{it-1}$ leads to poor results in the long run on certain problems and to reasonable results on others. Considering all problems addressed in this work and [14], we can safely say that: (i) this aggressive definition was really not useful for the Benders decomposition ([14, § 4.1]) and the robust optimization (Section 3.1) problems; (ii) it led to quality results on the remaining two problems, i.e., it proved very successful for standard graph coloring in [14, § 4.2] and it led to a reasonable total number of iterations in Section 3.2.2 just above.

A possible explanation is related to the oscillations of the inner solutions $x_{it}$ along the iterations. We report below the values of the first 15 components of $x_{it+1} = x_{it} + t^*_it_{it}$ for $it \in \{1, 11, 21, 31, 41\}$, i.e., as generated by Projective Cutting-Planes using the above aggressive $x_{it}$ definition. For each problem, we selected the very first instance from the main table of results, i.e., from the second group of rows of [14, Table 1], from Table 1, from [14, Table 3], and then from Table 3. The addressed problems are listed (sorted) below in descending order of the strength of the oscillations of these inner solutions $x_{it}$. Keep in mind to interpret these oscillations not only in absolute values but also in relative values.

**The benders reformulation (IP version):**

<table>
<thead>
<tr>
<th></th>
<th>2.76</th>
<th>2.76</th>
<th>2.76</th>
<th>2.76</th>
<th>2.76</th>
<th>2.76</th>
<th>2.76</th>
<th>2.76</th>
<th>2.76</th>
<th>2.76</th>
<th>2.76</th>
<th>2.76</th>
<th>2.76</th>
<th>2.76</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td></td>
<td>0.112</td>
<td>0.112</td>
<td>0.112</td>
<td>0.112</td>
<td>0.112</td>
<td>1.93</td>
<td>0.112</td>
<td>0.112</td>
<td>1.64</td>
<td>0.112</td>
<td>0.112</td>
<td>0.112</td>
<td>0.112</td>
<td>0.112</td>
</tr>
<tr>
<td></td>
<td>0.026</td>
<td>0.026</td>
<td>0.026</td>
<td>0.026</td>
<td>0.026</td>
<td>1.62</td>
<td>0.026</td>
<td>0.026</td>
<td>1.62</td>
<td>0.026</td>
<td>0.026</td>
<td>0.026</td>
<td>0.026</td>
<td>0.026</td>
</tr>
<tr>
<td></td>
<td>0.018</td>
<td>0.018</td>
<td>0.018</td>
<td>0.018</td>
<td>0.029</td>
<td>1.67</td>
<td>0.03</td>
<td>0.018</td>
<td>1.12</td>
<td>0.018</td>
<td>0.079</td>
<td>0.018</td>
<td>0.029</td>
<td>0.018</td>
</tr>
</tbody>
</table>

**The robust optimization problem:**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>37.36</th>
<th>0</th>
<th>59.62</th>
<th>0</th>
<th>69.77</th>
<th>0</th>
<th>97199.2</th>
<th>0</th>
<th>0</th>
<th>417</th>
<th>4403</th>
<th>0</th>
<th>65.66</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20.76</td>
<td>22.81</td>
<td>0</td>
<td>49.76</td>
<td>0</td>
<td>45.46</td>
<td>0</td>
<td>65.86</td>
<td>236.4</td>
<td>0</td>
<td>136.3</td>
<td>254.6</td>
<td>3500</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>27.38</td>
<td>18.04</td>
<td>0</td>
<td>46.28</td>
<td>0</td>
<td>37.49</td>
<td>0</td>
<td>55.68</td>
<td>248.7</td>
<td>0</td>
<td>180.8</td>
<td>201.3</td>
<td>3205</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>33.26</td>
<td>13.8</td>
<td>0</td>
<td>43.21</td>
<td>0</td>
<td>30.41</td>
<td>0</td>
<td>46.66</td>
<td>259.6</td>
<td>0</td>
<td>220.7</td>
<td>154.1</td>
<td>2942</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>36.22</td>
<td>11.68</td>
<td>0</td>
<td>41.63</td>
<td>0</td>
<td>26.86</td>
<td>0</td>
<td>42.14</td>
<td>265.1</td>
<td>0</td>
<td>240.7</td>
<td>130.3</td>
<td>2811</td>
<td>0</td>
</tr>
</tbody>
</table>

**Standard graph coloring:**

|         | 0.025 | 0.021 | 0.036 | 0.021 | 0.033 | 0.021 | 0.021 | 0.021 | 0.029 | 0.029 | 0.029 | 0.021 | 0.029 | 0.025 | 0.025 | 0.033 |
These results may explain why setting $x_{it} = x_{it-1} + t_{it-1}^* d_{it-1}$ leads to poor results on the first two problems and to reasonable results on graph coloring or Multiple-length Cutting-Stock. We used on purpose a cautious formulation “reasonable results on graph coloring or Multiple-length Cutting-Stock”, because the lack of strong oscillations does not guarantee the superiority of the aggressive version. The aggressive variant proved clearly superior only on graph coloring; recall (Remark 3 of [14]) that it could even report new lower bounds in the competitive graph coloring literature [7]. On the Multiple-length Cutting-Stock tests from Section 3.2.2, we can only cautiously say that the aggressive version yielded “reasonable results”, i.e., it needs less iterations than the standard Column Generation in Figure 3, but the CPU time is too large, mainly for reasons not related to oscillations.

On the other hand, it is quite clear that when the oscillations are strong, the aggressive version is most likely not very efficient. This is why the best settings for the first two problems (Benders decomposition and robust optimization) take a form $x_{it} = x_{it-1} + \alpha t_{it-1}^* d_{it-1}$ with $\alpha < 0.5$.

4. Conclusions

We explored the Projective Cutting-Planes algorithm from [14] in greater detail and we applied it to two new problems, i.e., robust linear programming and Multiple-length Cutting-Stock. Recall that main new proposed feature is the use of a new projection sub-problem instead of the well-known and widely-used separation sub-problem. A key step to make Projective Cutting-Planes really effective is to develop new techniques to solve this projection sub-problem very rapidly, if possible (almost) as rapidly as the separation sub-problem. Thus, an important goal of the paper was to develop fast projection algorithms for the two new considered problems.
The main advantage of Projective Cutting-Planes is that it has a built-in mechanism to generate feasible inner solutions along the iterations; these inner solutions converge towards $\text{opt}(\mathcal{P})$ similarly to the solutions of the central path in interior point algorithms. The standard Cutting-Planes does not contain such a built-in feature. In fact, even if some ad-hoc methods could sometimes be used in Cutting-Planes to construct feasible solutions (along the iterations), these inner solutions generally represent merely a by-product of the Cutting-Planes algorithm; they are not usually a very determining factor in the Cutting-Planes evolution.

Projective Cutting-Planes can offer a number of advantages beyond the reduction of the computing effort needed to fully converge:

- An aggressive Projective Cutting-Planes that chooses $x_{i\text{it}}$ as the best solution discovered up to now (i.e., $x_{i\text{it}} = x_{i\text{it}-1} + t_{i\text{it}-1}^* d_{i\text{it}-1}$) eliminates the infamous “yo-yo” effect that appears very often (if not always) in Column Generation. This was already presented in [14, Figure 3] for graph coloring and it is confirmed by the new Multiple-length Cutting-Stock experiments from Figure 3 (p. 27). However, this aggressive choice of $x_{i\text{it}}$ may also lead to poor results in the long run, partly because $x_{i\text{it}}$ may oscillate too much from iteration to iteration. To gain more insight into this (bang-bang) phenomenon, Section 3.3 characterizes the cases where it is better to choose $x_{i\text{it}} = x_{i\text{it}-1} + \alpha \cdot t_{i\text{it}-1}^* d_{i\text{it}-1}$ with some $\alpha < 1$ instead of $x_{i\text{it}} = x_{i\text{it}-1} + t_{i\text{it}-1}^* d_{i\text{it}-1}$ (i.e., instead of $\alpha = 1$). We thus addressed one of the questions raised in the conclusions of [14]: “It remains rather difficult to explain why $\alpha < 0.5$ is often better than $\alpha = 1$ when choosing the inner solution $x_{i\text{it}}$”.

- Implementing Projective Cutting-Planes takes more time than implementing a basic Cutting-Planes. Yet, a basic Cutting-Planes implementation may also require certain hidden costs in terms of working time if one wants to make it cover a part of the built-in features of Projective Cutting-Planes. For example, Projective Cutting-Planes can avoid degeneracy issues more easily than a standard Cutting-Planes. In standard Cutting-Planes, the separation sub-problem uses only one guide-point to determine a new constraint, i.e., the current optimal (outer) solution. The Projective Cutting-Planes determines each new constraint by taking into account a pair of inner-outer solutions such that each projection direction $x_{i\text{it}} \rightarrow d_{i\text{it}}$ satisfies $b^T d_{i\text{it}} > 0$. This way, Projective Cutting-Planes has a built-in feature to avoid degeneracy (iterations that keep the objective value constant). Although our experiments exhibit such standard Cutting-Planes
degeneracy issues only in Section 3.1 (Remark 2, p. 20), it is well-known that they do arise quite frequently in Column Generation as well.\(^2\)

- The robust linear programming experiments from Table 1 (Columns “gap 1\%”) demonstrate that Projective Cutting-Planes needs a very short time (less than 5\% of the total convergence time) to produce a feasible solution with a provable optimality gap below 1\%, i.e., an acceptable solution in practice. We also tested on this problem the idea of returning multiple cuts per round. The multiple cuts determined using a projection logic seem stronger than the ones obtained using a separation logic. In the best case (instance stocfor\(3\) for \(\Gamma = 1\)), the multi-cuts Projective Cutting-Planes from Section 3.1.3 fully converged in roughly 8 seconds, while the multi-cuts Cutting-Planes needed thousands of seconds.

- The projection sub-problem may generally lead to stronger constraints than the separation one. We can cite one point from the conclusions of [14] that applies to the current paper as well: “As described in Section 2.4.1 of [13], when \(\mathbf{x} = \mathbf{0}_n\), the projection sub-problem \(\text{project}(\mathbf{x} \to \mathbf{d})\) is equivalent to normalizing all constraints (to make them all have the same right-hand side value) and then choosing one by separating \(\mathbf{x} + \mathbf{d}\). Even if this paper uses \(\mathbf{x} \neq \mathbf{0}_n\), the projection sub-problem can still generate stronger (normalized) constraints than the separation sub-problem.”.

We hope that the ideas presented throughout this work and [14] may shed useful light on solving other LPs with prohibitively-many constraints.

Acknowledgements We thank two referees for the time they spent to read the paper twice.

References


\(^2\) As [2, §4] put it, “Column generation processes are known to have a slow convergence and degeneracy problems”. Section 4.2.2 of [9] explains that “large instances are difficult to solve due to massive degeneracy” — see also the references from loc. cit. for more detailed explanations of the mechanisms that lead to degeneracy issues.


Appendix or On-line supplement

A. The detailed Column Generation model and its Lagrangian bounds

The Column Generation model optimized in Section 3.2 of [14] (graph coloring) and in Section 2.2 of the current paper (Cutting-Stock) is

\[
\begin{align*}
\max b^\top x \\
y_a : a^\top x &\leq c_a, \forall (a,c_a) \in \mathcal{A} \\
x &\geq 0_n
\end{align*}
\]

(A.1)

All proposed algorithms related to Column Generation were presented from the standpoint of this LP, both for graph coloring in [14, (3.12)] and for Multiple-length Cutting-Stock in (2.5). This is actually the dual of the master primal LP below.

\[
\begin{align*}
\min \sum_{(a,c_a) \in \mathcal{A}} c_a y_a \\
x : \sum_{(a,c_a) \in \mathcal{A}} a_i y_a &\geq b_i, \forall i \in [1..n] \\
y_a &\geq 0, \forall (a,c_a) \in \mathcal{A}
\end{align*}
\]

(A.2)

The above LP was obtained after relaxing \(y_a \in \mathbb{Z}_+\) into \(y_a \geq 0\); in the very initial formulation, \(y_a\) is an integer variable that encodes the number of selections of each column \((a,c_a) \in \mathcal{A}\). These columns \(\mathcal{A}\) may represent stables in graph coloring, cutting patterns in (Multiple-Length) Cutting-Stock, or, more generally routes in vehicle routing problems, assignments of courses to timeslots in timetabling, or any specific subsets in the most general set-covering problem. The number of columns may be enormous and they cannot usually be enumerated in reasonable time. For each column \((a,c_a) \in \mathcal{A}\), \(a \in \mathbb{Z}_+^n\) is an incidence vector such that \(a_i\) indicates how many times an element \(i \in [1..n]\) is covered by \(a\).

The objective of (A.2) asks to minimize the total cost of the selected columns, under the (set-covering) constraint that each element \(i \in [1..n]\) has to be covered at least \(b_i\) times.

On several occasions, we referred to the Lagrangian lower bounds of the standard Column Generation. When all columns have equal unitary costs (i.e., \(c_a = 1, \forall (a,c_a) \in \mathcal{A}\) as in graph coloring), we simply used the Farley lower bound

\[
\mathcal{L}(x) = \frac{b^\top x}{1 - m_{rdc}(x)},
\]

(A.3)

where \(m_{rdc}(x)\) is the minimum reduced cost with regards to the optimal (dual) values \(x = \text{opt}(\mathcal{P}_{it})\) at the current iteration \(it\), i.e., \(m_{rdc}(x) = \min_{(a,c_a) \in \mathcal{A}} c_a - a^\top x\).
In **Multiple-length Cutting-Stock**, the column costs are no longer unitary, but we can still apply the Farley bound (A.3) after normalizing all columns in $A$. More exactly, we replace $(a, c_{a})$ with $(\frac{a}{c_{a}}, 1)$ for each $(a, c_{a}) \in A$ and we obtain a normalized model (A.2) that has the same objective value as the original model because the variables $y$ are continuous. Let $c_{\text{min}} = \min \{c_{a} : (a, c_{a}) \in A\}$. The normalized minimum reduced cost $m_{\text{rdc}}^{\text{norm}}(x)$ satisfies $m_{\text{rdc}}^{\text{norm}}(x) \geq \frac{1}{c_{\text{min}}} m_{\text{rdc}}(x)$ when $m_{\text{rdc}}(x) \leq 0$, because any $(a, c_{a}) \in A$ associated to some $m_{\text{rdc}}(x) = c_{a} - a^{\top}x \leq 0$ satisfies $\frac{1}{c_{a}} (c_{a} - a^{\top}x) \geq \frac{1}{c_{\text{min}}} (c_{a} - a^{\top}x)$. The Farley bound evolves to $\mathcal{L}(x)$ below.

$$\frac{b^{\top}x}{1 - m_{\text{rdc}}^{\text{norm}}(x)} \geq \mathcal{L}(x) = \frac{b^{\top}x}{1 - \frac{1}{c_{\text{min}}} m_{\text{rdc}}(x)}$$ (A.4)

The above $\mathcal{L}(x)$ is a valid lower bound when $m_{\text{rdc}}(x) \leq 0$, but not necessarily when $m_{\text{rdc}}(x) > 0$, because we used $m_{\text{rdc}}(x) \leq 0$ in the proof. An example can simply confirm this. Consider an instance with two standard-size pieces in stock: a piece of length 0.7 and cost 0.6 and a piece of length 1 and cost 1. The demand consists of two small items of lengths $w_{1} = 0.7$ and $w_{2} = 0.3$. Taking $x_{1} = 0.5$ and $x_{2} = 0.4$, one obtains $m_{\text{rdc}}(x) = 0.6 - 0.5 = 1 - 0.5 - 0.4 = 0.1$ and we get $\mathcal{L}(x) = \frac{0.9}{1 - \frac{1}{0.6}0.1} = 1.08$ which is not a valid lower bound, since the optimum for this instance is 1 (cut both items from a standard-size piece of length 1).

**Remark 3** Recall (Section 3.2) that the first two iterations of **Projective Cutting-Planes for Multiple-length Cutting-Stock** solve the projection sub-problems $\text{project}(0_{n} \rightarrow \frac{1}{W}w)$ and $\text{project}(0_{n} \rightarrow b)$, obtaining two initial lower bounds. For the standard **Column Generation**, however, $\mathcal{L}(\frac{1}{W}w)$ is not necessarily a valid lower bound because we may have $m_{\text{rdc}}(\frac{1}{W}w) > 0$. As such, even if we also (warm-)start the **Column Generation** by solving two initial separation sub-problems on $\frac{1}{W}w$ and $b$, this offers a unique lower bound $\mathcal{L}(b)$ for the standard **Column Generation**.

**B. A fast data structure to manipulate a Pareto frontier**

We explained in Section 2.2.3 how the **Dynamic Programming** projection algorithm needs to handle a list of states $I$ whose cost and profits $c_{i}/p_{i} \forall i \in I$ satisfy the Pareto dominance relation (2.7.a)–(2.7.b), recalled below for convenience (and also to obtain a stand-alone appendix).

$$c_{1} < c_{2} < c_{3} < \ldots$$
\[ p_1 < p_2 < p_3 < \ldots \]

We now present an effective data structure to manipulate such pairs of values \( c_i/p_i \) (with \( i \in I \)). This data structure is not essentially linked to Projective Cutting-Planes: it can manipulate any Pareto frontier with two objectives.

We focus on the following key task that has a particularly high risk of introducing a useless computational bottleneck: the insertion of a new pair \( c^+/p^+ \) in the list \( I \) (at Step 5 of Algorithm 1). It can be very inefficient and impractical to scan the whole list \( I \) only to check if \( c^+/p^+ \) is dominated or not. As such, we propose to record \( I \) using a self-balancing binary search tree \([8, \S 6.2.3]\), which is a data structure designed to manipulate ordered lists, \( e.g. \), it performs a lookup, an insertion and a removal in logarithmic time with respect to \( |I| \). The order of the states in the tree is given by the simple comparison of costs, \( i.e., \), if \( c_i < c_j \), then \( c_i/p_i \) is ordered before \( c_j/p_j \).

Given a new pair \( c^+/p^+ \), we first insert it into the self-balancing binary search tree and then we will compare it to the elements before and after it to check for dominance relations. Let \( c^*/p^* \) be the element before \( c^+/p^+ \) after it has been added to the tree, \( i.e., \), \( c^*/p^* \) is the pair with the highest cost \( c^* \) no larger than \( c^+ \) so that \( c^* = \max\{c_i: c_i \leq c^+, i \in I\} \).

Once \( c^*/p^* \) is determined, we check for dominance relations as follows. First, if \( p^* \leq p^+ \), then the new pair \( c^+/p^+ \) is directly removed because it is dominated by definition. Otherwise, if \( p^+ > p^* \), then \( c^*/p^+ \) has to remain in the tree but it may dominate other recorded pairs that have to be removed. For instance, if \( c^* = c^+ \) and \( p^* < p^+ \), then \( c^*/p^* \) is immediately removed from the tree. Furthermore, our insertion routine enumerates one by one all next recorded pairs \( c^#/p^# \) ordered after \( c^*/p^* \) (and after \( c^+/p^+ \)) that satisfy \( p^# \leq p^+ \) and removes them all. Indeed, such pairs \( c^#/p^# \) are certainly dominated by \( c^+/p^+ \), given that \( p^# \leq p^+ \) and \( c^# > c^+ \); the latter inequality follows from the fact that \( c^+/p^+ \) was inserted before \( c^#/p^# \) in the tree.