Personal draft of paper accepted by *INFORMS Journal on Computing*

`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 $\mathscr{P}$ 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 $\mathbf{x} \in \mathscr{P}$ and a direction $\mathbf{d}$, find the maximum steplength t such that $\mathbf{x} + t\mathbf{d} \in \mathscr{P}$. 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 $\mathscr{P}$. For the first addressed problem, robust linear programming, $\mathscr{P}$ is defined as a primal polytope. For the second addressed problem, `Multiple-length Cutting-Stock`, $\mathscr{P}$ 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:

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

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

Given an interior point $\mathbf{x} \in \mathscr{P}$ and a direction $\mathbf{d} \in \mathbb{R}^n$, the projection sub-problem `project`$(\mathbf{x} \rightarrow \mathbf{d})$ asks to advance from $\mathbf{x}$ along $\mathbf{d}$ up to the *pierce point* $\mathbf{x} + t^*\mathbf{d}$

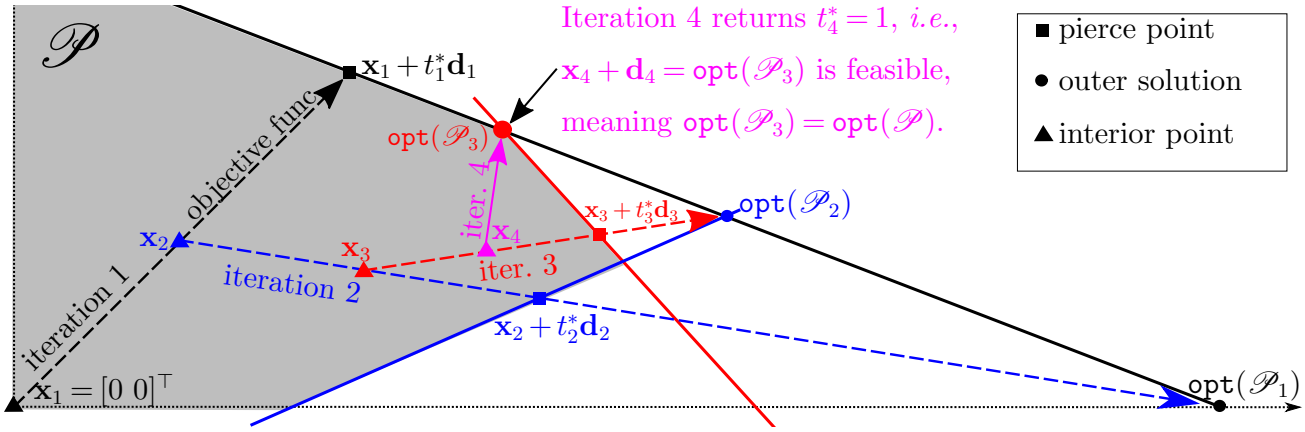

Figure 1 **Intuitive illustration of a** `Projective Cutting-Planes` **execution**

where the boundary of $\mathscr{P}$ is touched, *i.e.*, to determine the maximum step length $t^* = \max\{t \geq 0 :\ \mathbf{x} + t\mathbf{d} \in \mathscr{P}\}$. To solve this sub-problem, it is also necessary to find a constraint of $\mathcal{A}$ satisfied with equality by $\mathbf{x} + t^*\mathbf{d}$. 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 $\texttt{it} = 1$, the projection sub-problem projects $\mathbf{x}_1 = \mathbf{0} = [0\ 0]^\top$ along the direction $\mathbf{d}_1 = \mathbf{b}$, see the black dashed arrow. This generates a pierce point $\mathbf{x}_1 + t_1^*\mathbf{d}_1$ and also the black facet (constraint) satisfied with equality by $\mathbf{x}_1 + t_1^*\mathbf{d}_1$. By integrating the black facet, `Projective Cutting-Planes` constructs the first outer approximation $\mathscr{P}_1$ of $\mathscr{P}$, *i.e.*, $\mathscr{P}_1$ is the largest triangle defined by this black facet (line) and the two axes; notice the optimal solution $\texttt{opt}(\mathscr{P}_1)$ marked close to the bottom-right corner of the figure.

At the second iteration, `Projective Cutting-Planes` selects a (mid) point $\mathbf{x}_2$ between $\mathbf{x}_1$ and $\mathbf{x}_1 + t_1^*\mathbf{d}_1$ and projects $\mathbf{x}_2$ towards the current optimal outer solution $\texttt{opt}(\mathscr{P}_1)$. This generates a second pierce point $\mathbf{x}_2 + t_2^*\mathbf{d}_2$ and a second facet (blue solid line) that is added to the facets of $\mathscr{P}_1$ to construct $\mathscr{P}_2$. The process is repeated until the projection sub-problem certifies at iteration $\texttt{it} = 4$ that the outer solution $\texttt{opt}(\mathscr{P}_3)$ cannot be separated; this means $\texttt{opt}(\mathscr{P}_3)$ is an optimal solution over $\mathscr{P}$.

More generally, `Projective Cutting-Planes` generates a sequence of inner solutions $\mathbf{x}_{\mathtt{it}}$ as well as a sequence of outer solutions $\mathtt{opt}(\mathscr{P}_{\mathtt{it}})$ that both converge along the iterations $\mathtt{it}$ to an optimal solution $\mathtt{opt}(\mathscr{P})$. Each new inner solution $\mathbf{x}_{\mathtt{it}+1}$ is chosen as a point on the segment joining the previous inner solution $\mathbf{x}_{\mathtt{it}}$ and the last pierce point $\mathbf{x}_{\mathtt{it}} + t^*_{\mathtt{it}}\mathbf{d}_{\mathtt{it}}$ returned by the last projection. The next projection $\mathtt{project}(\mathbf{x}_{\mathtt{it}+1} \rightarrow \mathbf{d}_{\mathtt{it}+1})$ is chosen to point towards the current outer optimal solution $\mathtt{opt}(\mathscr{P}_{\mathtt{it}})$, *i.e.*, $\mathbf{d}_{\mathtt{it}+1} = \mathtt{opt}(\mathscr{P}_{\mathtt{it}}) - \mathbf{x}_{\mathtt{it}+1}$. 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 $\mathscr{P}_{\mathtt{it}}$ to construct $\mathscr{P}_{\mathtt{it}+1}$. By adding one new constraint at each iteration $\mathtt{it}$, the `Projective Cutting-Planes` algorithm constructs a sequence $\mathscr{P}_1 \supsetneq \mathscr{P}_2 \supsetneq \cdots \supset \mathscr{P}$ of outer approximations of $\mathscr{P}$, so that $\mathtt{opt}(\mathscr{P}_1),\ \mathtt{opt}(\mathscr{P}_2),\ \mathtt{opt}(\mathscr{P}_3), \ldots$ converge to $\mathtt{opt}(\mathscr{P})$, as in a standard `Cutting-Planes`.

At the first iteration $\mathtt{it} = 1$, one can choose any starting feasible solution $\mathbf{x}_1$, usually in a problem–specific manner. The first direction $\mathbf{d}_1$ is often $\mathbf{d}_1 = \mathbf{b}$, to make the first projection $\mathtt{project}(\mathbf{x}_1 \rightarrow \mathbf{d}_1)$ 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 $\mathbf{x}_{\mathtt{it}+1} = \mathbf{x}_{\mathtt{it}} + \alpha t^*_{\mathtt{it}}\mathbf{d}_{\mathtt{it}}$ for some $\alpha \in (0, 1]$. We will show that for $\alpha = 1$, each new interior solution $\mathbf{x}_{\mathtt{it}+1}$ is actually a boundary point of strictly higher quality than the previous one $\mathbf{x}_{\mathtt{it}}$. With such choice, the "grip" exerted by the lower and the upper bounds on $\mathtt{optVal}(\mathscr{P})$ is guaranteed to strictly increase at each iteration, *i.e.*, we have that $\mathbf{b}^\top\mathbf{x}_{\mathtt{it}} < \mathbf{b}^\top\mathbf{x}_{\mathtt{it}+1} \leq \mathtt{opt}(\mathscr{P}) \leq \mathbf{b}^\top\mathtt{opt}(\mathscr{P}_{\mathtt{it}+1}) \leq \mathbf{b}^\top\mathtt{opt}(\mathscr{P}_{\mathtt{it}})$ at each iteration $\mathtt{it}$ 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 $\mathbf{x}_{\text{in}}$ to guide the separation of an exterior point $\mathbf{x}^{\text{out}} = \texttt{opt}(\mathscr{P}_{\texttt{it}})$ is not new. For instance, [1] proposes to separate $\mathbf{x}^{\text{out}}$ by first calling the separation oracle on a point $\mathbf{x}_{\text{sep}}$ on the segment joining $\mathbf{x}_{\text{in}}$ and $\mathbf{x}^{\text{out}}$. Thus, Algorithm 1 from [1] defines $\mathbf{x}_{\text{sep}} = \alpha\mathbf{x}_{\text{in}} + (1-\alpha)\mathbf{x}^{\text{out}}$ for some $\alpha \in (0,1]$. If $\mathbf{x}_{\text{sep}} \notin \mathscr{P}$, we can expect that the cutting plane returned by separating $\mathbf{x}_{\text{sep}}$ is more efficient; otherwise, the algorithm from [1] sets $\mathbf{x}_{\text{in}} = \mathbf{x}_{\text{sep}}$, updates $\mathbf{x}_{\text{sep}}$ accordingly, and calls the separation oracle again. The process of separating $\mathbf{x}^{\text{out}}$ is based on a repeated choice of a point $\mathbf{x}_{\text{sep}}$ between $\mathbf{x}_{\text{in}}$ and $\mathbf{x}^{\text{out}}$ coupled with a repeated separation of $\mathbf{x}_{\text{sep}}$. This approach does not determine the pierce point, *i.e.*, it does not calculate the result of a projection from $\mathbf{x}_{\text{in}}$ to $\mathbf{x}^{\text{out}}$. 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 π of $\mathscr{P}$ and by their objective values η; such a feasible area is visible in [10, Fig. 1] using the above notations π and η.

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

A full literature review on other aspects related to the idea of advancing towards $\texttt{opt}(\mathscr{P})$ 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 computational complexity. We will first illustrate this on a well-known robust linear programming 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 constraints (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 algorithm 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 indicators. 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 $\mathbf{x}_{\mathtt{it}}$ are chosen carefully).

A final contribution of the paper is to analyse the choice of the inner points $\mathbf{x}_{\mathtt{it}}$ along the iterations `it`. While we always choose $\mathbf{x}_{\mathtt{it}}$ using the formula $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + \alpha t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}$, the value of α 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 α is related to the (magnitude of the) oscillations of the interior solutions $\mathbf{x}_{\mathtt{it}}$ 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 appendices: 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 manipulating 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 $\mathbf{x} \in \mathscr{P}$ and for any $\mathbf{d} \in \mathbb{R}^n$, the projection sub-problem $\texttt{project}(\mathbf{x} \rightarrow \mathbf{d})$ can be solved by minimizing the fractional program (2.1) below. Let us stress $\mathbf{x}$ 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 $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}$ associated to the constraint $\mathbf{a}^\top \mathbf{x} \leq c_{\mathbf{a}}$ 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).

$$t^* = \min \left\{ \frac{c_{\mathbf{a}} - \mathbf{a}^\top \mathbf{x}}{\mathbf{a}^\top \mathbf{d}} : \ (\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}, \ \mathbf{d}^\top \mathbf{a} > 0 \right\}. \tag{2.1}$$

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 Γ 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 $\mathcal{A}_{\texttt{nom}}$ 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 $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}_{\texttt{nom}}$ a prohibitively-large set $\texttt{Dev}_\Gamma(\mathbf{a})$ of *deviation vectors* $\widehat{\mathbf{a}}$, *i.e.*, vectors $\widehat{\mathbf{a}} \in \mathbb{R}^n$ that have at maximum Γ non-zero components and that satisfy $\widehat{a}_i \in \{-\delta a_i, \ 0, \ \delta a_i\} \ \forall i \in [1..n]$, using $\delta = 0.01$ in practice. Each such deviation vector $\widehat{\mathbf{a}}$ yields a robust cut $(\mathbf{a} + \widehat{\mathbf{a}})^\top \mathbf{x} \leq c_{\mathbf{a}}$, so that we can state $(\mathbf{a} + \widehat{\mathbf{a}}, c_{\mathbf{a}}) \in \mathcal{A}$. In theory, each $\widehat{a}_i$ 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 $\widehat{a}_i$ is either δa_i or $-\delta a_i$. There are at most $\binom{n}{\Gamma}2^\Gamma$ deviation vectors for each nominal constraint $(\mathbf{a}, c_\mathbf{a}) \in \mathcal{A}_{\texttt{nom}}$, because there are $\binom{n}{\Gamma}$ ways to choose the non-zero components of $\widehat{\mathbf{a}}$ and each one of them can be either positive or negative, hence the 2^Γ factor.

The generic LP (1.1) is instantiated as follows:

$$\min \left\{ \mathbf{b}^\top \mathbf{x} : (\mathbf{a}+\widehat{\mathbf{a}})^\top \mathbf{x} \leq c_\mathbf{a} \ \forall (\mathbf{a}, c_\mathbf{a}) \in \mathcal{A}_{\texttt{nom}} \ \forall \ \widehat{\mathbf{a}} \in \texttt{Dev}_\Gamma(\mathbf{a});\ x_i \in [\texttt{lb}_i, \texttt{ub}_i] \ \forall i \in [1..n] \right\} \tag{2.2}$$

The last condition $x_i \in [\texttt{lb}_i, \texttt{ub}_i]$ of (2.2) represents the initial constraints $\mathcal{A}_0$, most instances using $\texttt{lb}_i = 0 \ \forall i \in [1..n]$, *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 $c_\mathbf{a} - (\mathbf{a}+\widehat{\mathbf{a}})^\top \mathbf{x}$ over all $(\mathbf{a}, c_\mathbf{a}) \in \mathcal{A}_{\texttt{nom}}$ and over all $\widehat{\mathbf{a}} \in \texttt{Dev}_\Gamma(\mathbf{a})$. For a fixed nominal constraint $(\mathbf{a}, c_\mathbf{a}) \in \mathcal{A}_{\texttt{nom}}$, the strongest possible deviation $\widehat{\mathbf{a}}_\mathbf{x}^\top \mathbf{x}$ of $(\mathbf{a}, c_\mathbf{a})$ with respect to $\mathbf{x}$ is determined by maximizing $\widehat{\mathbf{a}}_\mathbf{x} = arg\ \max \left\{ \widehat{\mathbf{a}}^\top \mathbf{x} : \widehat{\mathbf{a}} \in \texttt{Dev}_\Gamma(\mathbf{a}) \right\}$. To find this $\widehat{\mathbf{a}}_\mathbf{x}$, one needs to determine the largest Γ absolute values in the terms of the sum $\mathbf{a}^\top \mathbf{x} = \sum_{i=1}^n a_i x_i$; this way, $\widehat{\mathbf{a}}_\mathbf{x}^\top \mathbf{x}$ can be written as a sum of Γ terms of the form $\delta|a_i x_i|$. We use absolute values because the strongest deviation of a term a_i is either $\widehat{a}_i = \delta a_i$ if $a_i x_i \geq 0$ or $\widehat{a}_i = -\delta a_i$ if $a_i x_i < 0$. We next describe how these largest Γ values can be determined by a partial-sorting algorithm of linear complexity.

Remark 1 *If Γ is a fixed parameter, the largest Γ 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 Γ values known up to now. Considering Γ is a fixed parameter, this operation would even take constant time when using a self-balancing binary search tree (as implemented in the `C++ 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_{min} recorded in the tree. If this is the case, the insertion of $|a_i x_i|$ may make the tree size exceed Γ, and so, v_{min} has to be removed. Each insertion and each removal takes constant time with regards to n, when considering Γ 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 $\frac{c_{\mathbf{a}} - (\mathbf{a}+\widehat{\mathbf{a}})^{\top}\mathbf{x}}{(\mathbf{a}+\widehat{\mathbf{a}})^{\top}\mathbf{d}}$ over all nominal constraints $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}_{\texttt{nom}}$ and over all deviation vectors $\widehat{\mathbf{a}} \in \texttt{Dev}_{\Gamma}(\mathbf{a})$ such that $(\mathbf{a}+\widehat{\mathbf{a}})^{\top}\mathbf{d} > 0$. Just as the separation algorithm, the projection algorithm iterates over all nominal constraints $\mathcal{A}_{\texttt{nom}}$, in an attempt to reduce the above ratio – *i.e.*, the step length – at each $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}_{\texttt{nom}}$. Let t_i^* denote the optimal step length obtained after considering the robust cuts associated to the first i constraints from $\mathcal{A}_{\texttt{nom}}$. 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 $(\mathbf{a}, c_{\mathbf{a}})$ denote the i^{th} constraint from $\mathcal{A}_{\texttt{nom}}$.
2. Determine the strongest deviation vector $\widehat{\mathbf{a}}_t$ with respect to $\mathbf{x}+t\mathbf{d}$ by maximizing:
$$\widehat{\mathbf{a}}_t = arg\ \max\left\{\widehat{\mathbf{a}}^{\top}(\mathbf{x}+t\mathbf{d}) : \widehat{\mathbf{a}} \in \texttt{Dev}_{\Gamma}(\mathbf{a})\right\}. \tag{2.3}$$
For this, one has to extract the Γ largest absolute values from the terms of the sum $\mathbf{a}^{\top}(\mathbf{x}+t\mathbf{d})$; we apply the partial-sorting algorithm used for the separation sub-problem in Remark 1.
3. If $(\mathbf{a}+\widehat{\mathbf{a}}_t)^{\top}(\mathbf{x}+t\mathbf{d}) \leq c_{\mathbf{a}}$, then $\mathbf{x}+t\mathbf{d}$ is feasible with regards to the first i constraints from $\mathcal{A}_{\texttt{nom}}$ and the associated robust cuts, because any deviation vector $\widehat{\mathbf{a}} \in \texttt{Dev}_{\Gamma}(\mathbf{a})$ satisfies $\widehat{\mathbf{a}}^{\top}(\mathbf{x}+t\mathbf{d}) \leq \widehat{\mathbf{a}}_t^{\top}(\mathbf{x}+t\mathbf{d})$. In this case, the final value $t_i^* = t$ has been obtained for this value of i. Otherwise, the robust cut $(\mathbf{a}+\widehat{\mathbf{a}}_t, c_{\mathbf{a}})$ leads to a smaller feasible step length:
$$t' = \frac{c_{\mathbf{a}} - (\mathbf{a}+\widehat{\mathbf{a}}_t)^{\top}\mathbf{x}}{(\mathbf{a}+\widehat{\mathbf{a}}_t)^{\top}\mathbf{d}} < t. \tag{2.4}$$
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 $\mathbf{x}$ is feasible.
5. Set $t = t'$ and repeat from Step 2 (without incrementing i). The underlying idea is that the deviation vector $\widehat{\mathbf{a}}_t$ determined via (2.3) is not the strongest one with regards to $\mathbf{x}+t'\mathbf{d}$, because $\widehat{\mathbf{a}}_t$ yields the highest deviation in (2.3) with regards to a different point (*i.e.*, $\mathbf{x}+t\mathbf{d}$). But there might exist a different robust cut $(\mathbf{a}+\widehat{\mathbf{a}}_{t'}, c_{\mathbf{a}})$ for the same nominal

constraint such that $\widehat{\mathbf{a}}_{t'}^{\top}(\mathbf{x}+t'\mathbf{d}) > \widehat{\mathbf{a}}_{t}^{\top}(\mathbf{x}+t'\mathbf{d})$. This could further reduce the step length below t', proving that $\mathbf{x}+t'\mathbf{d}$ is infeasible.

By sequentially applying the above steps to all constraints $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}_{\texttt{nom}}$ one by one, the step length returned at the last constraint of $\mathcal{A}_{\texttt{nom}}$ 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 $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}_{\texttt{nom}}$, the above algorithm only concludes at Step 3 that $\mathbf{x}+t\mathbf{d}$ satisfies all robust cuts associated to $(\mathbf{a}, c_{\mathbf{a}})$, 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 $c_{\mathbf{a}} - (\mathbf{a}+\widehat{\mathbf{a}}_{\mathbf{x}})^{\top}\mathbf{x}$ can certainly decrease up to the last nominal constraint $(\mathbf{a}, c_{\mathbf{a}})$. 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 $\mathbf{x}+t\mathbf{d}$ and then one needs *at least* a second call to check if $\mathbf{x}+t\mathbf{d}$ 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* $\mathbf{x}_{it}$ *for* $\mathtt{it} > 1$

The only non-trivial part is choosing the interior points $\mathbf{x}_{\mathtt{it}}$. As with most problems studied in this work and [14], experiments suggest that it is not very efficient to define $\mathbf{x}_{\mathtt{it}}$ as the best feasible solution found up to the iteration `it` (*i.e.*, the last pierce point $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}$). 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 $\mathbf{x}_{\mathtt{it}}$, not too close to the boundary of $\mathscr{P}$, enabling the inner solutions $\mathbf{x}_1,\ \mathbf{x}_2,\ \mathbf{x}_3, \ldots$ to follow a central path (a similar concept is used in some interior point algorithms). We thus define $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + \alpha t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}$ with $\alpha = 0.1\ \forall \mathtt{it} > 1$.

2.1.3.2. *Determining the first interior point* $\mathbf{x}_1$

To construct an initial feasible solution $\mathbf{x}_1$, one could be tempted to try $\mathbf{x}_1 = \mathbf{0}_n$, but $\mathbf{0}_n$ may be infeasible. We propose to generate $\mathbf{x}_1$ 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 $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}_{\mathtt{nom}}$, we insert a constraint $\mathbf{a}^\top\mathbf{x} + \delta|\mathbf{a}|^\top\mathbf{x} \leq c_{\mathbf{a}}$, where $|\mathbf{a}| = [|a_1|\ |a_2|\ \ldots |a_n|]^\top$. If $\mathbf{x}$ is non-negative (as in most instances), than any solution $\mathbf{x}$ that satisfies $\mathbf{a}^\top\mathbf{x} + \delta|\mathbf{a}|^\top\mathbf{x} \leq c_{\mathbf{a}}\ \forall(\mathbf{a},\ c_{\mathbf{a}}) \in \mathcal{A}_{\mathtt{nom}}$ is feasible with regards to all robust cuts — because a robust cut uses a deviation vector $\widehat{\mathbf{a}}$ that satisfies $\widehat{\mathbf{a}} \leq \delta|\mathbf{a}|$, so that $(\mathbf{a}+\widehat{\mathbf{a}})^\top\mathbf{x} \leq \mathbf{a}^\top\mathbf{x} + \delta|\mathbf{a}|^\top\mathbf{x} \leq c_{\mathbf{a}}$.

We also noticed that this "deep" inner LP can remain feasible by replacing $\mathbf{a}^\top\mathbf{x} + \delta|\mathbf{a}|^\top\mathbf{x} \leq c_{\mathbf{a}}$ with $\mathbf{a}^\top\mathbf{x} + 2\delta|\mathbf{a}|^\top\mathbf{x} \leq c_{\mathbf{a}} - \Delta$, for some small $\Delta > 0$. The use of this parameter Δ makes the generated solutions $\mathbf{x}_1$ 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 $\delta|\mathbf{a}|^\top\mathbf{x}$ (up to zero in the worst case). Finally, the first direction $\mathbf{d}_1$ points to the solution of the nominal problem, *i.e.*, we take $\mathbf{d}_1 = \texttt{opt}(\mathscr{P}_0) - \mathbf{x}_1$, where $\mathscr{P}_0$ is the polytope of the nominal problem with no robust cut. This is consistent with the general choice $\mathbf{d}_{\texttt{it}} = \texttt{opt}(\mathscr{P}_{\texttt{it}-1}) - \mathbf{x}_{\texttt{it}}$ that we will also use at all subsequent iterations $\texttt{it} \geq 1$, 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.

$$\left.\begin{array}{l} \max\ \mathbf{b}^\top\mathbf{x} \\ y_a : \mathbf{a}^\top\mathbf{x} \leq c_{\mathbf{a}},\ \forall(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A} \\ \mathbf{x} \geq \mathbf{0}_n. \end{array}\right\} \mathscr{P} \tag{2.5}$$

The notations from (2.5) can be directly interpreted in `Cutting-Stock` terms. Each constraint $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}$ is associated to a primal column representing a (cutting) pattern $\mathbf{a} \in \mathbb{Z}^n_+$ 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 $\mathbf{w} \in \mathbb{Z}^n_+$ of item lengths, all feasible patterns $\mathbf{a} \in \mathbb{Z}^n_+$ have to satisfy $\mathbf{w}^\top\mathbf{a} \leq W$, assuming W is the unique length of all given standard-size pieces. The vector $\mathbf{b} \in \mathbb{Z}^n_+$ 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 $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}$ have a fixed unitary cost $c_{\mathbf{a}} = 1$, 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 $\mathbf{a}$ is thus the cost of the smallest standard-size piece that can accommodate $\mathbf{a}$, *e.g.*, if $\mathbf{w}^\top \mathbf{a} \leq 0.7W$ then $c_{\mathbf{a}} = 0.6$, otherwise $c_{\mathbf{a}} = 1$.

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 $\min_{(\mathbf{a},c_{\mathbf{a}})\in\mathcal{A}} c_{\mathbf{a}} - \mathbf{a}^\top \mathbf{x}$ on the current optimal outer solution $\mathbf{x} = \texttt{opt}(\mathscr{P}_{\texttt{it}})$ 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_ℓ represented by a pattern $\mathbf{a}_\ell \in \mathbb{Z}^n_+$ of length ℓ that minimizes $c_\ell - \mathbf{a}_\ell^\top \mathbf{x} = \min\left\{c_\ell - \mathbf{a}^\top \mathbf{x} :\ \mathbf{a}^\top \mathbf{w} = \ell\right\}$; this pattern gives the objective value of s_ℓ, *i.e.*, $\texttt{obj}(s_\ell) = c_\ell - \mathbf{a}_\ell^\top \mathbf{x}$. One can ignore all non-available lengths $\ell \in [1..W]$ for which there is no pattern $\mathbf{a}$ such that $\mathbf{a}^\top \mathbf{w} = \ell$. The `Dynamic Programming` scheme generates transitions among such states, and, after calculating them all, it returns $\min\limits_{\ell\in[1..W]} c_\ell - \mathbf{a}_\ell^\top \mathbf{x}$.

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 $\mathbf{x}_{\texttt{it}}$ at each iteration `it`.

As with other problems explored all along this work and the initial paper [14], a key observation is that defining $\mathbf{x}_{\texttt{it}}$ as the best solution ever found up to iteration `it` is not efficient in the long run, partly because $\mathbf{x}_{\texttt{it}}$ could fluctuate too much from iteration to iteration. Furthermore, we will also see in Section 2.2.3.2 that the projection sub-problem $\texttt{project}(\mathbf{x} \rightarrow \mathbf{d})$ can be solved more rapidly when $\mathbf{x}$ 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 $\mathbf{x}_{\texttt{it}}$ is performed as follows. Let us first introduce the operator $\lfloor \mathbf{x} \rfloor_\gamma$ that truncates $\mathbf{x}$ 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 $\mathbf{x}^{\text{bst}}_\gamma$ denote the best

truncated feasible solution generated up to the current iteration; $\mathbf{x}^{\text{bst}}_{\gamma}$ can be determined as follows: start with $\mathbf{x}^{\text{bst}}_{\gamma} = \mathbf{0}_n$ at iteration $\texttt{it} = 1$, and replace $\mathbf{x}^{\text{bst}}_{\gamma}$ with $\left\lfloor \mathbf{x}_{\texttt{it}} + t^*_{\texttt{it}} \mathbf{d}_{\texttt{it}} \right\rfloor_{\gamma}$ at each iteration $\texttt{it} > 1$ where $\mathbf{b}^{\top} \left\lfloor \mathbf{x}_{\texttt{it}} + t^*_{\texttt{it}} \mathbf{d}_{\texttt{it}} \right\rfloor_{\gamma} > \mathbf{b}^{\top} \mathbf{x}^{\text{bst}}_{\gamma}$. We propose to choose the inner solution $\mathbf{x}_{\texttt{it}}$ at each iteration $\texttt{it}$ based on the following rules:

- set $\mathbf{x}_{\texttt{it}} = \mathbf{0}_n$ in half of the cases (half of the iterations);
- set $\mathbf{x}_{\texttt{it}} = \mathbf{x}^{\text{bst}}_{\gamma}$ in 25% of the cases;
- set $\mathbf{x}_{\texttt{it}} = \left\lfloor \frac{1}{2} \mathbf{x}^{\text{bst}}_{\gamma} \right\rfloor_{\gamma}$ in 25% of the cases.

The advantage of the first choice $\mathbf{x}_{\texttt{it}} = \mathbf{0}_n$ is that the associated projection sub-problem can be solved more rapidly. Projecting from $\mathbf{0}_n$ 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 $\mathbf{x} = \mathbf{0}_n$; in such a case, (2.6) could even reduce to a very knapsack-like problem that mainly asks to maximize the denominator $\mathbf{d}^{\top}\mathbf{a}$. The second choice $\mathbf{x}_{\texttt{it}} = \mathbf{x}^{\text{bst}}_{\gamma}$ is useful because the projection $\mathbf{x}^{\text{bst}}_{\gamma} \rightarrow \mathbf{d}_{\texttt{it}}$ 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 $\mathbf{x} \in \mathscr{P}$ in (2.5) and a direction $\mathbf{d} \in \mathbb{R}^n$, recall that the projection sub-problem $\texttt{project}(\mathbf{x} \rightarrow \mathbf{d})$ asks to minimize (2.1). For `Multiple-length Cutting-Stock`, (2.1) is instantiated as follows:

$$t^* = \min_{\mathbf{a}} \left\{ \frac{f(\mathbf{w}^{\top}\mathbf{a}) - \mathbf{a}^{\top}\mathbf{x}}{\mathbf{d}^{\top}\mathbf{a}} : \ \mathbf{a} \in \mathbb{Z}^n_+, \ \mathbf{w}^{\top}\mathbf{a} \leq W, \ \mathbf{d}^{\top}\mathbf{a} > 0 \right\}, \tag{2.6}$$

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 ℓ. 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 $\mathcal{S}_\ell$ of DP states for every feasible length $\ell \in [0..W]$. Each state $\mathtt{s} \in \mathcal{S}_\ell$ is associated to all patterns $\mathbf{a} \in \mathcal{A}$ of:

(1) length $\mathtt{s}_{\mathtt{len}} = \mathbf{w}^\top \mathbf{a} = \ell$;

(2) profit $\mathtt{s}_p = \mathbf{d}^\top \mathbf{a}$.

(3) cost $\mathtt{s}_c = f(\mathbf{w}^\top \mathbf{a}) - \mathbf{a}^\top \mathbf{x} = f(\ell) - \mathbf{a}^\top \mathbf{x} = c_{\mathbf{a}} - \mathbf{a}^\top \mathbf{x}$;

All states in $\mathcal{S}_\ell$ have the same length ℓ but their costs and profits can vary. Under this cost/profit interpretation, (2.6) reduces to minimizing the cost/profit ratio $\mathtt{obj}(\mathtt{s}) = \dfrac{\mathtt{s}_c}{\mathtt{s}_p}$ over all states $\mathtt{s}$ ever generated, *i.e.*, $\min\left\{\mathtt{obj}(\mathtt{s}) = \frac{\mathtt{s}_c}{\mathtt{s}_p} : s \in \mathcal{S}_\ell,\ \ell \in [0..W]\right\}$. 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 $\mathtt{s}_c = f(\mathbf{w}^\top \mathbf{a}) - \mathbf{a}^\top \mathbf{x} = c_{\mathbf{a}} - \mathbf{a}^\top \mathbf{x}$ is always non-negative because $\mathbf{x} \in \mathscr{P}$ 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 $\mathtt{s} \rightarrow \mathtt{s}'$ associated to an item i leads to a state $\mathtt{s}'$ such that:

(a) $\mathtt{s}'_{\mathtt{len}} = \mathtt{s}_{\mathtt{len}} + w_i$, *i.e.*, the length simply increases by adding a new item;

(b) $\mathtt{s}'_p = \mathtt{s}_p + d_i$, *i.e.*, add the profit of item i;

(c) $\mathtt{s}'_c = \mathtt{s}_c + f(\mathtt{s}'_{\mathtt{len}}) - f(\mathtt{s}_{\mathtt{len}}) - x_i$, *i.e.*, the term $f(\mathtt{s}'_{\mathtt{len}}) - f(\mathtt{s}_{\mathtt{len}})$ updates the cost of the pattern whose size increased from $\mathtt{s}_{\mathtt{len}}$ to $\mathtt{s}'_{\mathtt{len}}$, and $-x_i$ comes from the "$-\mathbf{a}^\top \mathbf{x}$" term used in the state cost definition $f(\ell) - \mathbf{a}^\top \mathbf{x}$ 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 $\mathtt{s}'$ is not dominated by an existing state in $\mathcal{S}_{\ell+w_i}$ before inserting it in $\mathcal{S}_{\ell+w_i}$; 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 $\mathtt{s}_c - \mathtt{s}_p$ instead of $\frac{\mathtt{s}_c}{\mathtt{s}_p}$). Indeed, the separation DP scheme may easily be described using the same framework. Recall its goal is to solve the (knapsack-like) sub-problem $\min\{f(\mathbf{w}^\top \mathbf{a}) - \mathbf{d}^\top \mathbf{a} :\ \mathbf{a} \in \mathbb{Z}^n,\ \mathbf{w}^\top \mathbf{a} \leq W\}$ for some $\mathbf{d} \in \mathbb{R}^n$. For this purpose, it is enough to consider only singleton sets $\mathcal{S}_\ell = \{\mathtt{s}\}$, where $\mathtt{s}$ 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** $\mathtt{s} \in \mathcal{S}_\ell$: ▷ for each state with length ℓ
3. initialize state $\mathtt{s}'$ with $\mathtt{s}'_{\mathtt{len}} = \ell + w_i$, according to above formula (a)
4. calculate $\mathtt{s}'_p$, $\mathtt{s}'_c$ with above formulae (b) and (c)
5. **if** $\mathtt{s}'$ is not dominated by an existing state in $\mathcal{S}_{\ell+w_i}$ (Section 2.2.3.2) **then**
   – $\mathcal{S}_{\ell+w_i} \leftarrow \mathcal{S}_{\ell+w_i} \cup \{\mathtt{s}'\}$
   – record the transition $\mathtt{s} \rightarrow \mathtt{s}'$ (to reconstruct an optimal pattern in the end)

$\mathbf{a}$ of cost $\mathtt{s}_c = f(\mathbf{w}^\top \mathbf{a}) = f(\ell)$ and maximum profit $\mathtt{s}_p = \mathbf{d}^\top \mathbf{a}$. Any other pattern $\mathbf{a}'$ having the same length (*i.e.*, $\mathbf{w}^\top \mathbf{a}' = \ell$) but a smaller profit (*i.e.*, $\mathbf{d}^\top \mathbf{a}' < \mathbf{d}^\top \mathbf{a}$) can never be part of the optimal solution. Thus, it is enough to record for each length $\ell \in [0, W]$ only the maximum profit state, the cost being fixed to $f(\ell)$. In the end, the separation algorithm simply returns $\min\{\mathtt{s}_c - \mathtt{s}_p : \ \mathtt{s} \in \mathcal{S}_\ell, \ \ell \in [0..W]\}$.

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 $\frac{5-1}{4+4} = \frac{4}{8} \not\leq \frac{3-1}{2+4} = \frac{2}{6}$. 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 ℓ (in $\mathcal{S}_\ell$). For this, consider two states $\mathtt{s}^*, \mathtt{s} \in \mathcal{S}_\ell$ such that $\mathtt{s}^*_c = \mathtt{s}_c$ and $\mathtt{s}^*_p > \mathtt{s}_p$. The state $\mathtt{s}$ is dominated and can be ignored because any transition(s) equally applied on $\mathtt{s}^*$ and $\mathtt{s}$ would lead to the same cost $\mathtt{s}^*_c + \Delta_c = \mathtt{s}_c + \Delta_c > 0$ and to profits $\mathtt{s}^*_p + \Delta_p > \mathtt{s}_p + \Delta_p$; this way, it is easy to check that $\dfrac{\mathtt{s}^*_c + \Delta_c}{\mathtt{s}^*_p + \Delta_p} < \dfrac{\mathtt{s}_c + \Delta_c}{\mathtt{s}_p + \Delta_p}$ 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 $\mathbf{d}^\top \mathbf{a} > 0$ from (2.6).

Let us now compare $\mathtt{s}^*$ to a state $\mathtt{s} \in \mathcal{S}_\ell$ that satisfies $\mathtt{s}_c > \mathtt{s}^*_c$ and $\mathtt{s}_p \leq \mathtt{s}^*_p$. Such state $\mathtt{s}$ is also dominated by $\mathtt{s}^*$ because it can only lead via transitions to $\frac{\mathtt{s}^*_c + \Delta_c}{\mathtt{s}^*_p + \Delta_p} < \frac{\mathtt{s}_c + \Delta_c}{\mathtt{s}_p + \Delta_p}$. As such, a state $\mathtt{s} \in \mathcal{S}_\ell$ with a higher cost than an existing state $\mathtt{s}^* \in \mathcal{S}_\ell$ (*i.e.*, $\mathtt{s}_c > \mathtt{s}^*_c$) must have a higher profit to be non-dominated, *i.e.*, a state $\mathtt{s}$ such that $\mathtt{s}_c > \mathtt{s}^*_c$ has to satisfy $\mathtt{s}_p > \mathtt{s}^*_p$ 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 $\mathcal{S}_\ell$ can thus be ordered using a (Pareto dominance) relation of the form:

$$c_1 < c_2 < c_3 < \ldots \tag{2.7a}$$

$$p_1 < p_2 < p_3 < \ldots \tag{2.7b}$$

Let us now investigate how long these lists (2.7.a)–(2.7.b) can be for each $\mathcal{S}_\ell$ $\forall \ell \in [0..W]$. 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 $\mathbf{x}$ 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 $f(\ell) - \mathbf{a}^\top \mathbf{x}$ for some $\mathbf{a} \in \mathbb{Z}^n_+$ 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:

(i) iterate over all elements of $\mathcal{S}_\ell$ to implement the `for` loop at Line 2;

(ii) 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 $\frac{w_i}{1+\mathbf{x}_i^{\text{bst}}}$. 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 $\mathbf{x}^{\text{bst}}$ 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/~porumbed/projcutplanes/instances-robust.zip`.

Recall our robust optimization problem has a minimization objective, so that the inner solutions $\mathbf{x}_{\texttt{it}}$ determined by `Projective Cutting-Planes` along the iterations `it` generate *upper* bounds $\mathbf{b}^\top \mathbf{x}_{\texttt{it}}$.

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

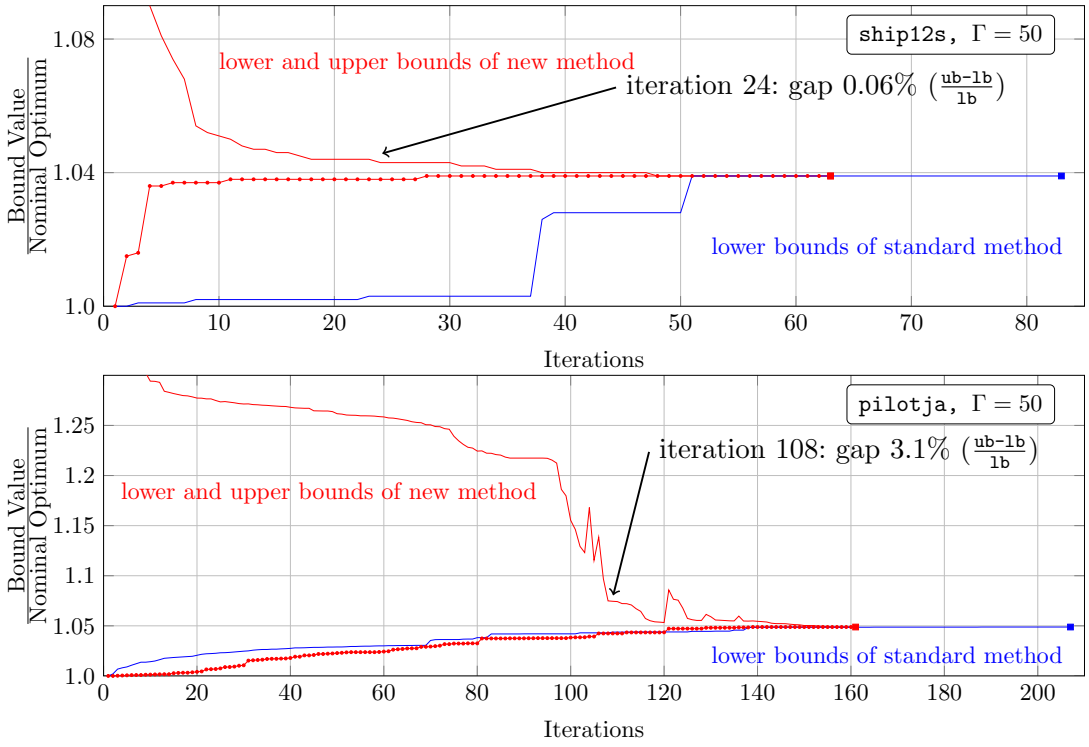

Figure 2 **The progress over the iterations of the lower and upper bounds reported by the `Projective Cutting-Planes` (in red), compared to those of the standard `Cutting-Planes` (lower bounds only, in blue).**

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 Γ) 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 $\mathbf{x}_1$ might be only a few percentage points higher than the nominal optimum.

	$\Gamma = 50$							$\Gamma = 10$							$\Gamma = 1$						
	OPT	new method				std. method		OPT	new method				std. method		OPT	new method				std. method	
	(+%)	gap 1%		full converg.		full converg.		(+%)	gap 1%		full converg.		full converg.		(+%)	gap 1%		full converg.		full converg.	
Instance		iters	time	iters	time	iters	time		iters	time	iters	time	iters	time		iters	time	iters	time	iters	time
25fv47	2.548	146	1.168	149	1.191	199	1.265	2.541	158	1.188	169	1.273	204	1.455	1.457	135	1.023	147	1.11	149	1.12
bnl2	1.847	486	13.66	491	13.81	1927	48.13	1.84	703	20.92	708	21.08	1295	31.31	0.7903	501	14.47	504	14.56	552	12.66
czprob	0.6401	61	0.485	734	32.3	1293	58.52	0.3749	32	0.181	125	0.899	170	1.302	0.1223	14	0.0935	25	0.196	25	0.124
ganges	0.4736	1	<0.001	25	0.059	25	0.059	0.4302	1	<0.001	31	0.085	33	0.074	0.0531	1	<0.001	25	0.063	25	0.049
gfrd-pnc	0.0649	64	0.1404	64	0.141	64	0.092	0.0649	64	0.1086	64	0.109	64	0.090	0.0592	67	0.1415	67	0.142	67	0.100
maros	12.12	272	2.402	278	2.458	379	3.518	12.11	281	2.508	300	2.683	395	3.486	5.76	200	1.812	219	1.983	227	1.714
nesm	0.8752	56	0.579	80	0.798	80	0.659	0.8752	56	0.604	80	0.824	80	0.658	0.4515	58	0.647	82	0.880	82	0.639
pilotja	4.877	121	2.418	161	3.042	207	3.701	4.815	125	2.316	172	3.074	179	3.418	2.344	110	1.522	135	1.908	143	1.693
pilotnov	8.51	96	4.227	120	4.615	119	3.714	8.51	103	6.12	139	6.912	141	4.915	4.402	94	3.355	120	3.805	119	1.776
pilotwe	6.109	102	0.97	118	1.102	143	1.204	6.108	102	1.045	119	1.19	144	1.308	3.193	98	0.853	115	1.005	124	1.066
scfxm2	2.114	93	0.387	139	0.584	146	0.537	2.113	101	0.401	152	0.603	150	0.498	0.9889	88	0.357	131	0.536	142	0.486
scfxm3	2.142	139	0.957	196	1.353	215	1.27	2.141	142	0.955	227	1.575	222	1.309	0.977	91	0.605	197	1.352	213	1.216
sctap2	2.844	185	1.946	242	2.567	*6545*	147.3	2.814	332	3.685	696	8.4	954	10.62	1.533	191	2.035	353	3.88	302	2.644
sctap3	3.04	145	2.649	239	4.55	*9463*	366.1	2.995	180	3.45	773	15.49	1168	20.22	1.602	213	3.785	406	7.394	347	4.799
ship08l	0.1244	1	0.002	20	0.111	29	0.171	0.1157	1	0.002	19	0.128	23	0.134	0.0300	1	0.002	19	0.127	24	0.147
ship08s	0.1396	2	0.006	32	0.122	42	0.139	0.129	2	0.006	34	0.134	35	0.123	0.0317	1	0.001	32	0.123	38	0.127
ship12l	0.3528	1	<0.001	48	0.442	65	0.576	0.3462	1	<0.001	48	0.418	65	0.555	0.0600	1	0.004	45	0.451	56	0.483
ship12s	0.3898	4	0.015	63	0.377	83	0.376	0.3857	5	0.019	64	0.387	86	0.398	0.0617	4	0.015	58	0.305	63	0.281
sierra	0.0239	1	0.001	54	0.414	61	0.567	0.0239	1	0.001	54	0.412	61	0.569	0.0223	1	0.004	51	0.538	51	0.483
stocfor2	1.522	6	0.022	437	5.047	484	6.67	1.522	7	0.025	438	5.387	486	6.562	0.7588	3	0.054	438	7.573	712	10.3
stocfor3	1.482	29	**2.192**	3777	2125	4329	2701	1.482	32	**1.862**	3781	2029	4330	2851	0.7327	1	**0.99**	3720	3023	6069	3482

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 `it` when the gap between the upper bound $\mathbf{b}^\top \mathbf{x}_{\texttt{it}}$ and the lower bound $\texttt{optVal}(\mathscr{P}_{\texttt{it}})$ is below 1%, *i.e.*, either $0 < \texttt{optVal}(\mathscr{P}_{\texttt{it}}) \leq \mathbf{b}^\top \mathbf{x}_{\texttt{it}} \leq 1.01\texttt{optVal}(\mathscr{P}_{\texttt{it}})$ or $\texttt{optVal}(\mathscr{P}_{\texttt{it}}) \leq \mathbf{b}^\top \mathbf{x}_{\texttt{it}} \leq 0.99\texttt{optVal}(\mathscr{P}_{\texttt{it}}) < 0$.

For `sctap2` and `sctap3` with $\Gamma = 50$, the standard `Cutting-Planes` is seriously slowed down by degeneracy issues, *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* $\mathbf{x}_{it}$ *and* $\texttt{opt}(\mathscr{P}_{it-1})$ *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* $\mathbf{x}_{it}$ *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* $\mathbf{x}_{it} \rightarrow \mathbf{d}_{it}$*, because* $\mathbf{x}_{it} + \mathbf{d}_{it} = \texttt{opt}(\mathscr{P}_{it-1})$ *and* $\mathbf{x}_{it}$ *belongs to the strict interior of* $\mathscr{P}_{it-1} \supseteq \mathscr{P}$*.*

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`, *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 $\mathbf{x}^{\text{out}} \notin \mathscr{P}_{\texttt{it}}$, (2) return all cuts determined at Step (1) that are violated by $\mathbf{x}^{\text{out}}$. However, numerical experiments show this may generate an important computational bottleneck: the polytopes $\mathscr{P}_{\texttt{it}}$ 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 $\mathbf{x}^{\text{out}}$, 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 $\mathscr{P}_{\texttt{it}}$ 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 $\mathscr{P}_{\texttt{it}}$ 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

	$\Gamma = 50$								$\Gamma = 10$								$\Gamma = 1$							
	new method					std. method			`Projective Cutting-Planes`					`Cutting-Planes`			new method					std. method		
	gap 1%		full converg.			full converg.			gap 1%		full convergence			full convergence			gap 1%		full converg.			full converg.		
Instance	iters	time	iters	time	cuts	iters	time	cuts	iters	time	iters	time	cuts	iters	time	cuts	iters	time	iters	time	cuts	iters	time	cuts
25fv47	4	0.07754	4	*0.08033*	1239	151	1.04	753	7	0.1274	9	0.1552	2780	162	1.145	809	5	0.09952	6	0.1126	1853	52	0.3682	259
bnl2	20	0.721	20	*0.7284*	10001	1481	40.75	7320	115	3.963	115	4.779	10097	1017	26.42	4979	21	0.02333	22	0.7335	10003	217	4.808	1025
czprob	20	0.2835	144	*11.74*	6219	2183	694.7	9569	7	0.07266	57	1.072	2405	78	0.5975	226	3	0.02158	5	0.03708	206	7	0.04048	30
ganges	1	<0.001	25	0.05736	377	6	0.01624	29	1	0.0005068	5	0.01564	139	11	0.02738	55	1	0.0004746	2	0.008084	55	6	0.01655	30
gfrd-pnc	3	0.005698	3	*0.009146*	210	56	0.0796	64	3	0.005027	3	*0.007078*	210	56	0.08037	64	3	0.005187	3	*0.007217*	212	56	0.08032	70
maros	52	0.2544	52	1.381	10012	360	3.721	1794	52	0.3149	53	1.383	10015	358	3.613	1780	15	0.3423	15	0.3451	5872	119	0.9702	591
nesm	2	0.03056	2	*0.03367*	547	60	0.4452	256	2	0.03072	2	*0.03388*	547	60	0.4483	256	2	0.03089	2	0.03403	551	31	0.2368	117
pilotja	3	0.3578	4	*0.3868*	1137	177	4.673	826	3	0.5271	4	0.5452	1139	150	4.172	709	4	0.1665	4	0.1698	1133	96	1.181	479
pilotnov	6	1.082	6	*1.086*	1675	146	16.16	716	6	1.007	12	1.511	3341	147	14.09	734	3	0.2071	3	0.2106	836	84	2.076	420
pilotwe	5	0.165	5	0.168	721	120	0.9665	600	5	0.1483	5	0.1513	718	117	1.019	585	3	0.065	3	*0.06795*	431	87	0.6904	433
scfxm2	4	0.02459	4	*0.0273*	1176	93	0.3011	444	7	0.04271	8	0.05065	2344	94	0.3126	450	4	0.02217	5	0.02949	1453	55	0.1676	257
scfxm3	5	0.04749	6	*0.05992*	2148	146	0.8557	678	5	0.04725	11	0.1142	4312	146	0.8244	679	4	0.03446	6	0.05546	2133	89	0.5102	388
sctap2	18	0.3629	28	*0.5688*	10011	4959	164.3	12893	110	2.267	320	7.774	10304	923	13.26	4343	26	0.159	169	3.438	10152	227	1.967	837
sctap3	11	0.3315	24	*0.6951*	10012	14460	1082	22406	54	1.178	547	17.34	10535	1186	25.73	5678	43	0.8201	230	6.677	10218	269	3.632	1055
ship08l	1	0.003371	3	0.02582	247	27	0.1628	135	1	0.003318	3	0.02561	247	21	0.1235	105	1	0.003323	2	0.01682	167	17	0.09824	84
ship08s	2	0.00796	2	*0.01088*	165	40	0.1431	200	2	0.008142	3	0.01608	249	33	0.1133	165	1	0.00181	2	0.01056	167	30	0.1014	148
ship12l	1	0.0002038	43	0.4873	2307	62	0.5652	310	1	0.0005575	28	0.307	1604	62	0.575	309	1	0.005452	3	0.03818	327	43	0.3671	215
ship12s	2	0.01041	4	*0.02909*	336	81	0.3941	404	4	0.02286	5	*0.03394*	343	81	0.3911	404	3	0.01728	4	0.02819	434	55	0.2455	274
sierra	1	0.001649	53	0.6052	10019	51	0.4582	163	1	0.001635	53	0.604	10019	51	0.4572	163	1	0.005382	18	0.1804	6787	44	0.3969	143
stocfor2	6	0.02236	437	6.526	10384	484	6.582	1976	7	0.02591	438	7.564	10384	486	6.623	1977	2	0.07735	2	*0.08385*	2034	435	5.824	1688
stocfor3	29	1.701	3777	2146	13731	4329	2729	11951	32	1.891	3780	1949	13732	4330	2806	11993	1	1.085	3	*6.849*	10001	4123	2391	11273

Table 2: Results of `Projective Cutting-Planes` and standard `Cutting-Planes` with multiple cuts per round. The CPU times are reported in italic font for all instances on which `Projective Cutting-Planes` is at least ten times faster.

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 $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}$ of the `Column Generation` dual LP (2.5) do not satisfy all $c_{\mathbf{a}} = 1$, 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 $\mathbf{x}_1 = \mathbf{0}_n$ and $\mathbf{d}_1 = \frac{1}{W}\mathbf{w}$ for the first iteration and $\mathbf{x}_2 = \mathbf{0}_n$ and $\mathbf{d}_2 = \mathbf{b}$ for the second one. The choice of projecting along $\mathbf{0}_n \rightarrow \frac{1}{W}\mathbf{w}$ for $\texttt{it} = 1$ is inspired by research in dual feasible functions for `Cutting-Stock` problems [3], which shows that $\frac{1}{W}\mathbf{w}$ is often a dual-feasible solution (in pure `Cutting-Stock`) of very high quality. The choice for $\texttt{it} = 2$ is a rather standard one: the projection towards $\mathbf{b}$ 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 $\mathbf{b}$ and $\frac{1}{W}\mathbf{w}$, generating two initial constraints. However, in `Column Generation`, these two sub-problems can only generate one Lagrangian lower bound associated to $\mathbf{b}$. We cannot calculate such a lower bound for $\frac{1}{W}\mathbf{w}$ because the Lagrangian bound does not hold for a (dual) feasible solution like $\frac{1}{W}\mathbf{w} \in \mathscr{P}$ 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, $\mathbf{b}$, 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 $\texttt{ub} \leq 1.2 \cdot \texttt{lb}$) 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 $\frac{\texttt{CG-Stab}}{\texttt{CG-Std}}$, 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 `it` an arbitrary optimal solution of value $\texttt{optVal}(\mathscr{P}_i)$; this artificial randomization is obtained by optimizing a random objective function while keeping the value of the original legitimate objective function at $\texttt{optVal}(\mathscr{P}_{\texttt{it}})$. 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

Instance	OPT	Projective Cutting-Planes				Standard Column Generation				
		gap 20%		full convergence		gap 20%		full converg.		stabilized
		iters	time[s]	iters	time[s]	iters	time[s]	iters	time[s]	iters
m01-1	49.3	90	0.02	166 (86%)	0.05	187	0.07	194	0.08	86%–89%
m01-2	53	82	0.02	140 (69%)	0.04	171	0.06	202	0.07	85%–86%
m01-3	48.2	70	0.02	134 (**63%**)	0.04	180	0.07	212	0.08	81%–91%
m20-1	56.6	79	0.02	101 (69%)	0.03	101	0.03	148	0.04	78%–90%
m20-2	58.7	73	0.02	103 (**59%**)	0.02	123	0.04	175	0.05	84%–92%
m20-3	64.8	61	0.01	116 (85%)	0.02	118	0.03	136	0.03	85%–91%
m35-1	73.9	61	0.01	61 (95%)	0.01	64	0.01	64	0.01	90%–92%
m35-2	71.5	125	0.02	125 (87%)	0.02	143	0.02	143	0.02	50%–54%
m35-3	73.7	67	0.01	67 (82%)	0.01	82	0.01	82	0.01	45%–60%
vb50c1-1	866.3	46	0.8	82 (73%)	2.2	83	5.5	113	8.3	86%–90%
vb50c1-2	842.5	39	1.6	86 (71%)	2.5	91	7.6	121	9.6	77%–87%
vb50c1-3	860.2	37	1.5	85 (74%)	3.1	87	6.9	115	9.5	87%–90%
vb50c2-1	672.3	55	2.2	114 (90%)	9.8	82	13.1	127	20.2	81%–82%
vb50c2-2	593.1	40	1.9	80 (**58%**)	5.1	88	11	139	21.1	85%–94%
vb50c2-3	480.048	36	3.5	181 (84%)	47.2	75	20.6	216	76.3	90%–97%
vb50c3-1	282	37	11.7	122 (68%)	57.6	67	36.1	179	105	79%–94%
vb50c3-2	239.398	37	16.8	115 (79%)	64.6	60	30.6	145	85.1	89%–91%
vb50c3-3	271.398	36	12.9	132 (76%)	65.3	68	38.2	173	109	87%–92%
vb50c4-1	579.548	40	3.5	115 (73%)	17.5	73	12.5	158	35.5	82%–90%
vb50c4-2	551.01	36	3	123 (74%)	21.9	73	18.5	166	46.6	95%–95%
vb50c4-3	700.039	40	2.3	111 (76%)	9.9	81	11.9	147	24.8	83%–89%
vb50c5-1	337.8	40	8.7	133 (**58%**)	51.9	61	24.8	228	109	86%–91%
vb50c5-2	349.799	30	4.8	130 (**63%**)	44.1	64	21	207	81.4	96%–99%
vb50c5-3	295.775	36	11	115(**65%**)	53.6	71	28.4	177	83.9	89%–89%
wäscher-1	24.0648	71	0.2	319 (**66%**)	4.2	294	2.3	483	4.7	67%–71%
wäscher-2	22.0003	69	0.2	501 (103%)	8.6	158	1	481	6.7	70%–75%
wäscher-3	12.1219	31	0.03	110(**65%**)	0.3	110	0.3	170	0.5	72%–86%
hard-sch-1	51.4254	112	14.7	345 (**48%**)	69.2	345	48.1	712	115	81%–86%
hard-sch-2	51.4426	116	15.1	339 (**49%**)	67	365	50.9	685	110	85%–87%
hard-sch-3	50.5957	110	15.1	295 (**47%**)	58.6	357	52.8	630	107	80%–83%

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 $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}$, *i.e.*, $\mathbf{x}_{\mathtt{it}}$ 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 $\mathbf{b}^\top\mathbf{x}_{\mathtt{it}} = \mathbf{b}^\top\left(\mathbf{x}_{\mathtt{it}-1} + t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}\right) \geq \mathbf{b}^\top\mathbf{x}_{\mathtt{it}-1}$ is surely satisfied because the objective function does not deteriorate by advancing along $\mathbf{x}_{\mathtt{it}-1} \rightarrow \mathbf{d}_{\mathtt{it}-1}$ (see also Step 2 from [14, § 2]). In fact, the inequality is always strict except at the very last iteration when $t^*_{\mathtt{it}-1} = 0$. This way, the lower bound $\mathbf{b}^\top\mathbf{x}_{\mathtt{it}}$ 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 osculations 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

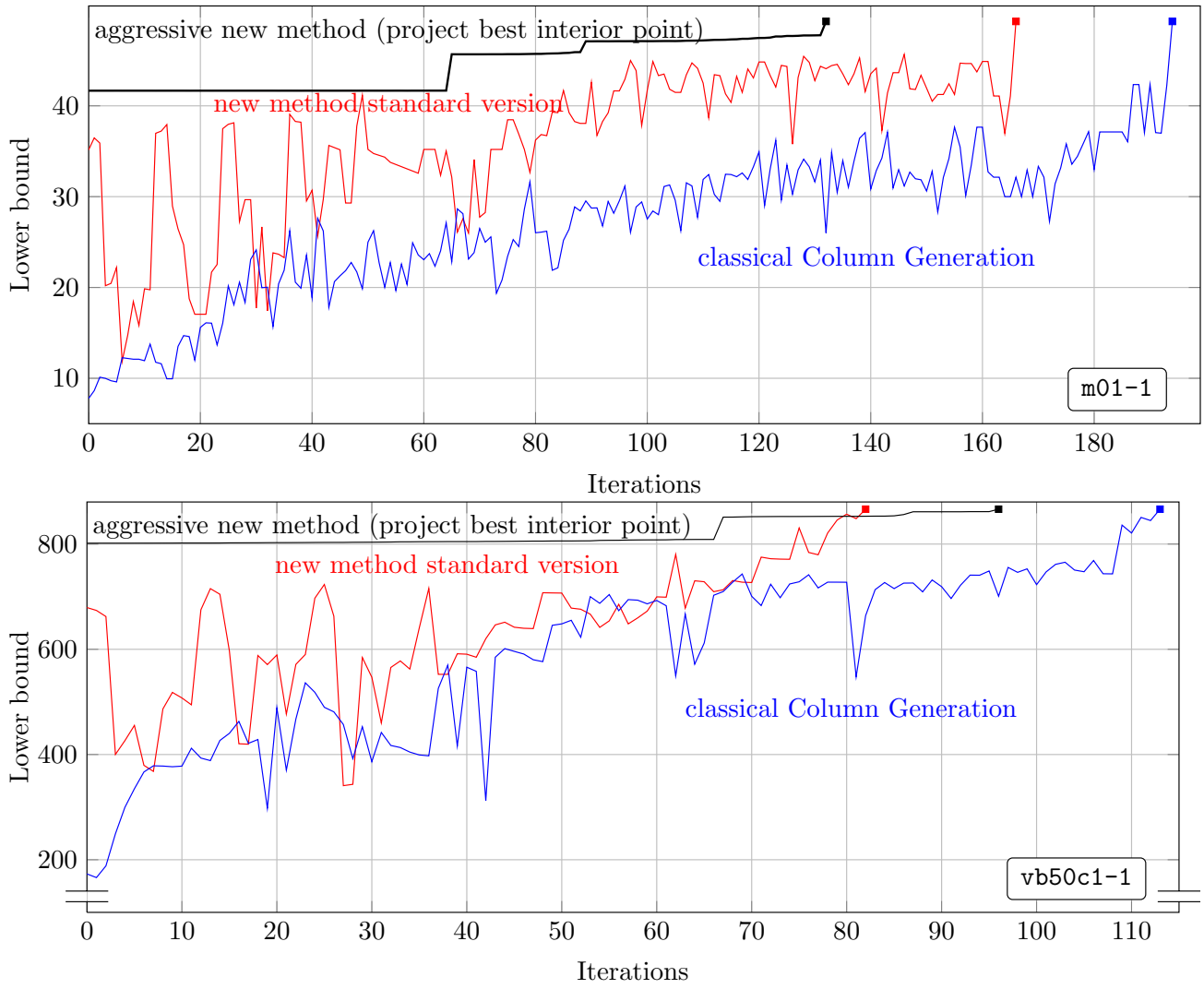

Figure 3 **Two representative running profiles, comparing the aggressive and the standard** `Projective Cutting-Planes` **against** `Column Generation`**. While the aggressive** `Projective Cutting-Planes` **starts very well (the black curves show no "yo-yo" effect), it converges rather slowly in terms of CPU time.**

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 $\mathbf{x}_{\texttt{it}}$, 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 $\mathbf{x}_{\mathtt{it}}$ like $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}$ 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 $\mathbf{x}_{\mathtt{it}}$ along the iterations. We report below the values of the first 15 components of $\mathbf{x}_{\mathtt{it}+1} = \mathbf{x}_{\mathtt{it}} + t^*_{\mathtt{it}}\mathbf{d}_{\mathtt{it}}$ for $\mathtt{it} \in \{1,\ 11,\ 21,\ 31,\ 41\}$, *i.e.*, as generated by `Projective Cutting-Planes` using the above aggressive $\mathbf{x}_{\mathtt{it}}$ 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 $\mathbf{x}_{\mathtt{it}}$. Keep in mind to interpret these oscillations not only in absolute values but also in relative values.

```
The benders reformulation (IP version):
2.76  2.76  2.76  2.76  2.76  2.76  2.76  2.76  2.76  2.76  2.76  2.76  2.76  2.76  2.76
0.37  0.37  0.37  0.37  0.37  0.37  4.08  0.37  0.37  0.37  0.37  0.37  0.37  0.37  0.37
0.112 0.112 0.112 0.112 0.112 0.112 1.93  0.112 0.112 1.64  0.112 0.112 0.112 0.112 0.112
0.026 0.026 0.026 0.026 0.026 0.026 1.62  0.026 0.026 1.62  0.026 0.026 0.026 0.026 0.026
0.018 0.024 0.018 0.018 0.018 0.029 1.67  0.03  0.018 1.12  0.018 0.079 0.018 0.029 0.018
The robust optimization problem:
0     37.36 0     59.62 0     69.77 0     97    199.2 0     0     417   4403  0     65.66
20.76 22.81 0     49.76 0     45.46 0     65.86 236.4 0     136.3 254.6 3500  0     64.43
27.38 18.04 0     46.28 0     37.49 0     55.68 248.7 0     180.8 201.3 3205  0     64.03
33.26 13.8  0     43.21 0     30.41 0     46.66 259.6 0     220.7 154.1 2942  0     63.67
36.22 11.68 0     41.63 0     26.86 0     42.14 265.1 0     240.7 130.3 2811  0     63.49
Standard graph coloring:
0.025 0.021 0.036 0.021 0.033 0.021 0.021 0.021 0.029 0.029 0.021 0.025 0.029 0.025 0.033
```

```
0.04  0.064 0.033 0.028 0.084 0.028 0.035 0.019 0.04  0.059 0.019 0.073 0.054 0.025 0.05
0.04  0.063 0.033 0.029 0.085 0.028 0.044 0.019 0.04  0.058 0.019 0.072 0.056 0.025 0.051
0.038 0.062 0.032 0.03  0.089 0.027 0.045 0.018 0.039 0.056 0.018 0.07  0.054 0.025 0.051
0.037 0.06  0.032 0.031 0.088 0.026 0.044 0.018 0.038 0.055 0.018 0.068 0.053 0.025 0.051
Multiple length cutting stock:
0.28  0.43  0.72  0.79  0.23  0.7   0.55  0.39  0.69  0.01  0.41  0.4   0.05  0.25  0.95
0.27  0.43  0.72  0.79  0.24  0.7   0.55  0.39  0.69  0.01  0.41  0.4   0.05  0.24  0.95
0.28  0.43  0.72  0.79  0.23  0.7   0.55  0.39  0.69  0.01  0.41  0.4   0.05  0.24  0.95
0.28  0.44  0.72  0.79  0.23  0.7   0.55  0.4   0.69  0.01  0.41  0.41  0.05  0.24  0.95
0.28  0.43  0.72  0.79  0.23  0.7   0.55  0.39  0.69  0.01  0.42  0.39  0.05  0.24  0.95
```

These results may explain why setting $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}$ 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 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 $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + \alpha t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}$ 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 $\texttt{opt}(\mathscr{P})$ 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 $\mathbf{x}_{\texttt{it}}$ as the best solution discovered up to now (*i.e.*, $\mathbf{x}_{\texttt{it}} = \mathbf{x}_{\texttt{it}-1} + t^*_{\texttt{it}-1}\mathbf{d}_{\texttt{it}-1}$) 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 $\mathbf{x}_{\texttt{it}}$ may also lead to poor results in the long run, partly because $\mathbf{x}_{\texttt{it}}$ 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 $\mathbf{x}_{\texttt{it}} = \mathbf{x}_{\texttt{it}-1} + \alpha \cdot t^*_{\texttt{it}-1}\mathbf{d}_{\texttt{it}-1}$ with some $\alpha < 1$ instead of $\mathbf{x}_{\texttt{it}} = \mathbf{x}_{\texttt{it}-1} + t^*_{\texttt{it}-1}\mathbf{d}_{\texttt{it}-1}$ (*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 $\mathbf{x}_{\texttt{it}}$".

– 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 $\mathbf{x}_{\texttt{it}} \rightarrow \mathbf{d}_{\texttt{it}}$ satisfies $\mathbf{b}^\top \mathbf{d}_{\texttt{it}} > 0$. 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 `stocfor3` 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 $\texttt{project}(\mathbf{x} \rightarrow \mathbf{d})$ 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

[1] W. Ben-Ameur and J. Neto., *Acceleration of cutting-plane and column generation algorithms: Applications to network design*, Networks, 49 (2007), pp. 3–17.

[2] H. Ben Amor and J. M. V. de Carvalho, *Cutting stock problems*, in Column Generation, G. Desaulniers, J. Desrosiers, and M. M. Solomon, eds., vol. 5, Springer, 2005, pp. 131–161.

[3] F. Clautiaux, C. Alves, and J. M. V. de Carvalho, *A survey of dual-feasible and superadditive functions*, Annals of Operations Research, 179 (2009), pp. 317–342.

[4] M. Fischetti and M. Monaci, *Cutting plane versus compact formulations for uncertain (integer) linear programs*, Mathematical Programming Computation, 4 (2012), pp. 239–273.

[5] J. Gondzio, *Interior point methods 25 years later*, European Journal of Operational Research, 218 (2012), pp. 587–601.

[6] J. Gondzio, P. González-Brevis, and P. Munari, *Large-scale optimization with the primal-dual column generation method*, Math. Prog. Comp., 8 (2016), pp. 47–82.

[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.

[7] S. Held, W. Cook, and E. C. Sewell, *Maximum-weight stable sets and safe lower bounds for graph coloring*, Mathematical Programming Computation, 4 (2012), pp. 363–381.

[8] D. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, Addison-Wesley, 1998

[9] M. E. Lübbecke and J. Desrosiers, *Selected topics in column generation*, Operations Research, 53 (2005), pp. 1007–1023.

[10] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. *In-out separation and column generation stabilization by dual price smoothing*, *12th International Symposium on Experimental Algorithms (2013)*, pp. 354–365, Springer.

[11] D. Porumbel. *Projective cutting-planes for robust linear programming and cutting-stock problems* URL `http://dx.doi.org/10.5281/zenodo.5745335`, https://github.com/INFORMSJoC/2022.1160.

[12] D. Porumbel. *Ray projection for optimizing polytopes with prohibitively many constraints in set-covering column generation*, *Mathematical Programming*, 155 (2016), pp. 147–197.

[13] D. Porumbel. *From the separation to the intersection subproblem for optimizing polytopes with prohibitively many constraints in a Benders decomposition context*, Discrete Optimization, 29 (2018), pp. 148–173.

[14] D. Porumbel. `Projective Cutting-Planes`, SIAM Journal on Optimization, 30(2020), pp. 1007-1032

[15] D. Porumbel and F. Clautiaux. *Constraint aggregation in column generation models for resource-constrained covering problems*, INFORMS JoC, 29 (2017), pp. 170–184.

[16] F. Vanderbeck. *Computational study of a column generation algorithm for bin packing and cutting stock problems*, Mathematical Programming, 86 (1999), pp. 565–594.

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

$$\left.\begin{array}{c} \max \mathbf{b}^\top \mathbf{x} \\ y_a : \mathbf{a}^\top \mathbf{x} \leq c_{\mathbf{a}},\ \forall (\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A} \\ \mathbf{x} \geq \mathbf{0}_n \end{array}\right\} \mathscr{P} \tag{A.1}$$

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.

$$\begin{array}{c} \min \sum_{(\mathbf{a},c_{\mathbf{a}})\in\mathcal{A}} c_{\mathbf{a}} y_a \\ \mathbf{x}: \sum_{(\mathbf{a},c_{\mathbf{a}})\in\mathcal{A}} a_i y_a \geq b_i \ \ \forall i \in [1..n] \\ y_a \geq 0 \qquad \forall (\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A} \end{array} \tag{A.2}$$

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 $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}$. 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 can not usually be enumerated in reasonable time. For each column $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}$, $\mathbf{a} \in \mathbb{Z}_+^n$ is an incidence vector such that a_i indicates how many times an element $i \in [1..n]$ is covered by $\mathbf{a}$. 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.*, $c_{\mathbf{a}} = 1\ \forall (\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}$ as in graph coloring), we simply used the Farley lower bound

$$\mathscr{L}(\mathbf{x}) = \frac{\mathbf{b}^\top \mathbf{x}}{1 - m_{rdc}(\mathbf{x})}, \tag{A.3}$$

where $m_{rdc}(\mathbf{x})$ is the minimum reduced cost with regards to the optimal (dual) values $\mathbf{x} = \mathtt{opt}(\mathscr{P}_{\mathtt{it}})$ at the current iteration `it`, *i.e.*, $m_{rdc}(\mathbf{x}) = \min_{(\mathbf{a},c_{\mathbf{a}})\in\mathcal{A}} c_{\mathbf{a}} - \mathbf{a}^\top \mathbf{x}$.

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 $\mathcal{A}$. More exactly, we replace $(\mathbf{a}, c_{\mathbf{a}})$ with $(\frac{\mathbf{a}}{c_{\mathbf{a}}}, 1)$ for each $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}$ and we obtain a normalized model (A.2) that has the same objective value as the original model because the variables $\mathbf{y}$ are continuous. Let $c_{\min} = \min\{c_{\mathbf{a}} : (\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}\}$. The normalized minimum reduced cost $m^{\text{norm}}_{rdc}(\mathbf{x})$ satisfies $m^{\text{norm}}_{rdc}(\mathbf{x}) \geq \frac{1}{c_{\min}} m_{rdc}(\mathbf{x})$ when $m_{rdc}(\mathbf{x}) \leq 0$, because any $(\mathbf{a}, c_{\mathbf{a}}) \in \mathcal{A}$ associated to some $m_{rdc}(\mathbf{x}) = c_{\mathbf{a}} - \mathbf{a}^\top \mathbf{x} \leq 0$ satisfies $\frac{1}{c_{\mathbf{a}}}\left(c_{\mathbf{a}} - \mathbf{a}^\top \mathbf{x}\right) \geq \frac{1}{c_{\min}}\left(c_{\mathbf{a}} - \mathbf{a}^\top \mathbf{x}\right)$. The Farley bound evolves to $\mathscr{L}(\mathbf{x})$ below.

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

The above $\mathscr{L}(\mathbf{x})$ is a valid lower bound when $m_{rdc}(\mathbf{x}) \leq 0$, but not necessarily when $m_{rdc}(\mathbf{x}) > 0$, because we used $m_{rdc}(\mathbf{x}) \leq 0$ 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 $m_{rdc}(\mathbf{x}) = 0.6 - 0.5 = 1 - 0.5 - 0.4 = 0.1$ and we get $\mathscr{L}(x) = \frac{0.9}{1 - \frac{1}{0.6} 0.1} = 1.08$ 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* ***project***$(\mathbf{0}_n \rightarrow \frac{1}{W}\mathbf{w})$ *and* ***project***$(\mathbf{0}_n \rightarrow \mathbf{b})$*, obtaining two initial lower bounds. For the standard* ***Column Generation****, however,* $\mathscr{L}(\frac{1}{W}\mathbf{w})$ *is not necessarily a valid lower bound because we may have* $m_{rdc}\left(\frac{1}{W}\mathbf{w}\right) > 0$*. As such, even if we also (warm-)start the* ***Column Generation*** *by solving two initial separation sub-problems on* $\frac{1}{W}\mathbf{w}$ *and* $\mathbf{b}$*, this offers a unique lower bound* $\mathscr{L}(\mathbf{b})$ *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).

$$c_1 < c_2 < c_3 < \ldots$$

$$p_1 < p_2 < p_3 < \dots$$

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, § 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.