

Cutting Planes by Projecting Interior Points onto Polytope Facets

Daniel Porumbel*

*CEDRIC CS Lab, CNAM, 292 rue Saint-Martin, F-75141 Paris, France

daniel.porumbel@cnam.fr

Abstract

Given a point \mathbf{x} inside a polytope \mathcal{P} and a direction $\mathbf{d} \in \mathbb{R}^n$, the projection of \mathbf{x} along \mathbf{d} asks to find the maximum step length t^* such that $\mathbf{x} + t^*\mathbf{d}$ is feasible; we say $\mathbf{x} + t^*\mathbf{d}$ is a pierce point because it belongs to the boundary of \mathcal{P} . In [13], we only explored the idea of projecting the origin $\mathbf{0}_n$ along integer directions, focusing on dual polytopes \mathcal{P} in **Column Generation** models. In this work, we address a more general projection (intersection) sub-problem, considering arbitrary interior points $\mathbf{x} \in \mathcal{P}$, arbitrary non-integer directions $\mathbf{d} \in \mathbb{R}^n$, and we solve problems beyond the scope of **Column Generation**, *e.g.*, robust optimization or Benders decomposition problems. Generalizing the standard separation sub-problem of the widely-used **Cutting-Planes**, the above projection sub-problem serves as the main building block for designing a new **Projective Cutting-Planes** algorithm to optimize over polytopes \mathcal{P} with prohibitively-many constraints. At each iteration, the **Projective Cutting-Planes** selects a point \mathbf{x}_{new} on the segment joining the points \mathbf{x} and $\mathbf{x} + t^*\mathbf{d}$ determined at the previous iteration. Then, it projects \mathbf{x}_{new} along the direction \mathbf{d}_{new} pointing towards the current optimal (outer) solution (of the current outer approximation of \mathcal{P}), so as to generate a new pierce point $\mathbf{x}_{\text{new}} + t_{\text{new}}^*\mathbf{d}_{\text{new}}$ and a new constraint of \mathcal{P} . By re-optimizing the linear program enriched with this new constraint, the algorithm finds a new current optimal (outer) solution and moves to the next iteration by updating $\mathbf{x} = \mathbf{x}_{\text{new}}$ and $\mathbf{d} = \mathbf{d}_{\text{new}}$. The **Projective Cutting-Planes** improves upon the standard **Cutting-Planes** in the sense that it generates a feasible inner solution $\mathbf{x} + t^*\mathbf{d}$ (a primal bound) at each iteration. These inner solutions converge to an optimal solution $\text{opt}(\mathcal{P})$ and, in this sense, the **Projective Cutting-Planes** is more similar to an interior point method than to the Simplex algorithm. Numerical experiments on four problems in different optimization settings confirm the potential of the proposed ideas.

1 Introduction

Optimizing Linear Programs (LP) with prohibitively many constraints has a long and rich history in mathematical programming. The well-established **Cutting-Planes** algorithm proceeds by iteratively removing infeasibility. It maintains at each iteration it an outer approximation \mathcal{P}_{it} of the feasible polytope \mathcal{P} , *i.e.*, a polytope \mathcal{P}_{it} defined only by a subset of the constraints of \mathcal{P} , so that $\mathcal{P}_{it} \supseteq \mathcal{P}$. The standard **Cutting-Planes** can be seen as an outer method in the sense that it converges towards an optimal solution $\text{opt}(\mathcal{P})$ through a sequence of outer (infeasible) solutions; as such, for a maximization problem, it generates a convergent sequence of upper bounds $\text{optVal}(\mathcal{P}_1) \geq \text{optVal}(\mathcal{P}_2) \geq \text{optVal}(\mathcal{P}_3) \geq \dots \geq \text{optVal}(\mathcal{P})$. The most canonical **Cutting-Planes** has no general built-in mechanism to generate a convergent sequence of feasible inner solutions. In contrast, an inner method constructs a convergent sequence of inner feasible solutions \mathbf{x}_{it} that converge towards $\text{opt}(\mathcal{P})$ along the iterations it . We can say the **Projective Cutting-Planes** algorithm proposed in this paper is both an inner and an outer method, in the sense that it generates a convergent sequence of both inner and outer solutions. We refer the reader to (Section 1.1 of) [13] for a review of existing work concerning inner methods, outer methods, the intersection sub-problem and related ideas.

The proposed **Projective Cutting-Planes** is based on an iterative operation of projecting an interior point onto facets of \mathcal{P} , as illustrated in Figure 1. At each iteration it , an inner solution $\mathbf{x}_{it} \in \mathcal{P}$ is projected towards the direction \mathbf{d}_{it} of the current optimal outer solution $\text{opt}(\mathcal{P}_{it-1})$, *i.e.*, setting $\mathbf{d}_{it} = \text{opt}(\mathcal{P}_{it-1}) - \mathbf{x}_{it}$. This is referred to as the projection (or intersection) sub-problem: determine $t_{it}^* =$

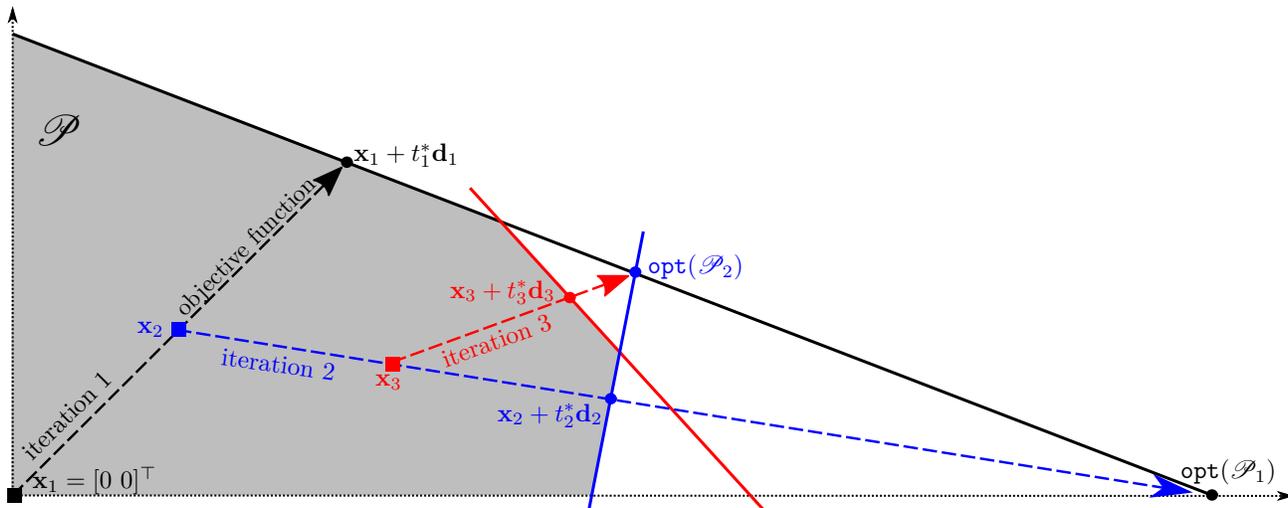


Figure 1: The first three iterations of the **Projective Cutting-Planes** on a linear program with 2 variables. At the first iteration, the projection sub-problem projects $\mathbf{x}_1 = \mathbf{0} = [0 \ 0]^\top$ along the objective function, as depicted by the black dashed arrow. At iteration $\text{it} = 2$, the midpoint \mathbf{x}_2 of this black arrow is projected towards the direction of the optimal outer solution $\text{opt}(\mathcal{P}_1)$ — at iteration 1, the outer approximation $\mathcal{P}_1 \supset \mathcal{P}$ only contains the largest triangle. This generates a second facet (blue solid line) that is added to the facets of \mathcal{P}_1 to construct \mathcal{P}_2 . The third sub-problem (in red) takes the midpoint \mathbf{x}_3 between the blue square and the blue circle (the last pierce point) and projects it towards $\text{opt}(\mathcal{P}_2)$.

$\max \{t : \mathbf{x}_{\text{it}} + t\mathbf{d}_{\text{it}} \in \mathcal{P}\}$. By solving this sub-problem, one determines a pierce (hit) point $\mathbf{x}_{\text{it}} + t_{\text{it}}^* \mathbf{d}_{\text{it}}$ and a new (first-hit) constraint of \mathcal{P} , which is added to the constraints of $\mathcal{P}_{\text{it}-1}$ to construct \mathcal{P}_{it} . At next iteration $\text{it} + 1$, the **Projective Cutting-Planes** takes a new interior point $\mathbf{x}_{\text{it}+1}$ on the segment joining \mathbf{x}_{it} and $\mathbf{x}_{\text{it}} + t_{\text{it}}^* \mathbf{d}_{\text{it}}$, and it projects $\mathbf{x}_{\text{it}+1}$ along $\mathbf{d}_{\text{it}+1} = \text{opt}(\mathcal{P}_{\text{it}}) - \mathbf{x}_{\text{it}+1}$.

To solve the intersection sub-problem and determine $t^* = \max \{t : \mathbf{x} + t\mathbf{d} \in \mathcal{P}\}$, one has to find a (first-hit) constraint satisfied with equality by $\mathbf{x} + t^*\mathbf{d}$. This implicitly solves the separation sub-problem for all points $\mathbf{x} + t\mathbf{d}$ with $t \in \mathbb{R}_+$, because this first-hit constraint separates all solutions $\mathbf{x} + t\mathbf{d}$ with $t > t^*$ while the points $\mathbf{x} + t\mathbf{d}$ with $t \in [0, t^*]$ can not be separated. A simplified version of the projection sub-problem limited to $\mathbf{x} = \mathbf{0}_n$ was already studied in our previous work related to **Column Generation** [13] or Benders decomposition models [14]. In the current work, we seek maximum generality in terms of projections: we will project arbitrary interior points $\mathbf{x} \in \mathcal{P}$ along arbitrary directions $\mathbf{d} \in \mathbb{R}^n$, and we will report numerical experiments on more problems than in [13] and [14] together.

In loose terms, the proposed algorithm is reminiscent of an Interior Point Method (IPM) because it produces a sequence of interior points that converge to the optimal solution. An IPM moves from solution to solution by advancing along a Newton direction at each iteration, in an attempt to solve first order optimality conditions [7]. In principle, performing a Newton step in this direction is similar to solving a projection sub-problem, although the projection is different because it executes a full step-length, *i.e.*, it advances along the given direction up to the pierce point where it intersects a first-hit constraint. On the other hand, an IPM does not advance on the Newton direction to fully solve the first order conditions at each iteration, since these conditions correspond to a primal objective function penalized by a barrier term (that only vanishes at the last iteration). A primal-dual **Column Generation** IPM generates well-centered dual solutions along the iterations, by keeping them in the proximity of a central path [8, § 3.3], which bears certain similarities to the construction of the feasible solutions $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \dots$ of the **Projective Cutting-Planes**. However, the central path in IPM consists of solutions which are interior with regards to the current outer approximation $\mathcal{P}_{\text{it}} \supset \mathcal{P}$ at iteration it , but they do not necessarily belong all to \mathcal{P} .

By generalizing the separation sub-problem, the projection sub-problem is inherently more difficult. It might even seem computationally far more expensive, but we will show this is not always the case. In fact,

we will present four techniques that can bring us very close to designing a projection algorithm that is as fast as the separation one.

- A first approach consists of generalizing the main ideas underlying the separation algorithm without increasing the computational complexity. This is used in Section 3.1 in the context of a robust optimization problem with prohibitively-many robust cuts [6]. In a nutshell, solving the separation sub-problem on a given $\mathbf{x} \in \mathbb{R}^n$ reduces to minimizing a difference of the form $c_a - (\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{x}$ over a set of nominal constraints $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$ and over all possible deviations $\hat{\mathbf{a}}$ of the nominal coefficients \mathbf{a} . The intersection sub-problem $\text{project}(\mathbf{x} \rightarrow \mathbf{d})$ reduces to minimizing a ratio $\frac{c_a - (\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{x}}{(\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{d}}$ over the same nominal constraints \mathcal{A}_{nom} and over the same deviations $\hat{\mathbf{a}}$ of \mathbf{a} . Both sub-problems can be solved by iterating over the nominal constraints \mathcal{A}_{nom} ; for each $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$, the separation sub-problem tries to minimize the above difference while the projection sub-problem tries to minimize the above ratio. So asking to minimize a ratio instead of a difference, one does not change the nature of the sub-problem iterative algorithm and does not significantly increase its running time.
- A second technique is applicable to the (numerous) problems in which the constraints of \mathcal{P} are given by the feasible solutions of an auxiliary LP. This is the case for the Benders decomposition models from Section 3.2 for which the separation sub-problem reduces to an LP. We will show that in such a case the intersection sub-problem can be formalized as a linear-fractional program: minimize a ratio of two linear functions subject to linear constraints. Using the Charnes–Cooper transformation [3], it is possible to translate this linear-fractional program to an equivalent LP of similar size. As such, both the separation and the projection sub-problems have the computational complexity of solving an LP.
- The above technique can be generalized to the (numerous) problems in which the constraints of \mathcal{P} are given by the feasible solutions of an *Integer* LP (ILP). We will develop this idea in Section 4.1, where \mathcal{P} is the dual polytope of a **Column Generation** model for graph coloring; the constraints of \mathcal{P} are given by the primal columns, which are associated to the 0–1 stables of the considered graph. The separation sub-problem can be formulated as an ILP with all constraints defined by edge inequalities. The projection sub-problem reduces to an integer linear-fractional program with the same constraints as the above ILP; by applying a new discrete version of the Charnes–Cooper transformation, this program can be translated to a Disjunctive LP. This latter Disjunctive LP has a discrete feasible area and can be solved with the same techniques as the separation ILP, *i.e.*, by a **Branch and Bound** algorithm with bounds determined from continuous relaxations. We will argue in Section 4.1.3.1 that there is no in-depth reason why such a Disjunctive LP should be harder in absolute terms than a similar-size ILP, especially when both are solved with very similar **Branch and Bound** methods.
- Finally, we used **Dynamic Programming** for the **Column Generation** model for *Multiple-Length Cutting-Stock* in Section 4.2. Typically, in **Column Generation**, if the separation sub-problem can be solved by **Dynamic Programming**, so can be the projection sub-problem. The main difference is that the projection sub-problem usually requires minimizing a ratio instead of a difference. 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 (subproblems respecting the Bellman’s principle of optimality). Recall that the most computationally difficult task in **Dynamic Programming** is to generate all states; once all states are generated, it is not difficult to return the state of minimum objective value (either for a linear objective function or for a fractional one).

The remainder is organized as follows. Section 2 is devoted to a detailed description of the proposed **Projective Cutting-Planes**. Section 3 illustrates the application of this algorithm on two problems in which \mathcal{P} is defined as a primal (master) polytope, *i.e.*, a robust linear program in Section 3.1 and a Benders reformulation model in Section 3.2. Section 4 presents the application of the **Projective Cutting-Planes** on two **Column Generation** models (for graph coloring and *Multiple-Length Cutting-Stock*), where \mathcal{P} is defined as a dual polytope. Section 5 is devoted to numerical results on all above problems, followed by conclusions in Section 6. Two appendices provide additional insight into the projection algorithms and describe the experimental settings (and the instances) in greater detail.

2 Algorithmic Description of the Projective Cutting-Planes

The proposed algorithm is designed to solve a large-scale LP of the following form:

$$\max \{ \mathbf{b}^\top \mathbf{x} : \mathbf{a}^\top \mathbf{x} \leq c_a, \forall (\mathbf{a}, c_a) \in \mathcal{A} \} = \max \{ \mathbf{b}^\top \mathbf{x} : \mathbf{x} \in \mathcal{P} \}, \quad (2.1)$$

where \mathcal{A} is a set of (unmanageably-many) constraints. In fact, we will also address a few generalizations and variations of (2.1). For instance, in the Benders reformulation model (3.2.2a)–(3.2.2c) from Section 3.2 we will use integer variables $\mathbf{x} \in \mathbb{Z}_+^n$. In Section 4, when (2.1) is a dual LP obtained after relaxing an integer **Column Generation** model, the goal is actually to find the best *rounded-up* objective value, *i.e.*, “ $\max \mathbf{b}^\top \mathbf{x}$ ” can be replaced by “ $\max \lceil \mathbf{b}^\top \mathbf{x} \rceil$ ”. In other cases, “ $\max \mathbf{b}^\top \mathbf{x}$ ” can be replaced by “ $\min \mathbf{b}^\top \mathbf{x}$ ”, but the proposed algorithm works exactly in the same manner regardless of the optimization direction.

Let us first briefly recall the standard **Cutting-Planes** for the above LP (2.1). This method maintains at each iteration it an outer approximation \mathcal{P}_{it} of \mathcal{P} obtained by restricting the constraint set \mathcal{A} to a subset \mathcal{A}_{it} , corresponding to a larger polytope $\mathcal{P}_{\text{it}} \supset \mathcal{P}$. To (try to) separate the current optimal solution $\mathbf{x}^{\text{out}} = \text{opt}(\mathcal{P}_{\text{it}})$ of this polytope \mathcal{P}_{it} , the most standard **Cutting-Planes** usually solves the separation sub-problem $\min_{(\mathbf{a}, c_a) \in \mathcal{A}} c_a - \mathbf{a}^\top \mathbf{x}^{\text{out}}$. If the optimum value of this sub-problem is less than 0 for some $(\bar{\mathbf{a}}, \bar{c}_a) \in \mathcal{A}$,

then \mathbf{x}^{out} is infeasible. In this case, the **Cutting-Planes** method inserts $\bar{\mathbf{a}}^\top \mathbf{x} \leq \bar{c}_a$ into the current constraint set (*i.e.*, it performs $\mathcal{A}_{\text{it}+1} = \mathcal{A}_{\text{it}} \cup \{(\bar{\mathbf{a}}, \bar{c}_a)\}$), so as to construct a new more refined outer approximation $\mathcal{P}_{\text{it}+1}$ and to separate $\mathbf{x}^{\text{out}} \notin \mathcal{P}_{\text{it}+1}$. The process is repeated by (re-)optimizing over $\mathcal{P}_{\text{it}+1}$ at the next iteration, until the current optimal outer solution \mathbf{x}^{out} becomes optimal (non-separable).

This work is devoted to a **Projective Cutting-Planes** method that replaces the above separation sub-problem with the following one.

Definition 1 (*Projection sub-problem*) *Given an interior point $\mathbf{x} \in \mathcal{P}$ and a direction $\mathbf{d} \in \mathbb{R}^n$, the projection sub-problem $\text{project}(\mathbf{x} \rightarrow \mathbf{d})$ asks to find:*

- *the maximum step length t^* such that $\mathbf{x} + t^* \mathbf{d}$ is feasible, *i.e.*, $t^* = \max \{ t \geq 0 : \mathbf{x} + t \mathbf{d} \in \mathcal{P} \}$. The solution $\mathbf{x} + t^* \mathbf{d}$ is referred to as the pierce point. If $\mathbf{x} + t \mathbf{d}$ is a ray of \mathcal{P} , the sub-problem returns $t^* = \infty$.*
- *a first-hit constraint $(\mathbf{a}, c_a) \in \mathcal{A}$ satisfied with equality by the pierce point, *i.e.*, such that $\mathbf{a}^\top (\mathbf{x} + t^* \mathbf{d}) = c_a$; such a constraint certainly exists if $t^* \neq \infty$.*

At the very first iteration, the **Projective Cutting-Planes** can start by performing a projection along $\mathbf{d}_1 = \mathbf{b}$, so as to directly advance along the direction with the fastest rate of objective function improvement, without forbidding using a problem-specific direction \mathbf{d}_1 . An initial feasible (inner) solution \mathbf{x}_1 is always needed because we do not focus on problems for which it is difficult to decide whether (2.1) is feasible or not. For instance, when possible, we initialize $\mathbf{x}_1 = \mathbf{0}_n$ as in the **Column Generation** models from Section 4.¹ In certain cases one can also provide a set of initial constraints \mathcal{A}_0 , for example, to impose simple bounds on the variables (*e.g.*, non-negativity constraints like $\mathbf{x} \geq \mathbf{0}_n$).

By solving $\text{project}(\mathbf{x}_1 \rightarrow \mathbf{d}_1)$ at iteration $\text{it} = 1$, the **Projective Cutting-Planes** determines the first pierce point $\mathbf{x}_1 + t_1^* \mathbf{d}_1$ and generates a first-hit constraint $(\mathbf{a}, c_a) \in \mathcal{A}$. After updating $\mathcal{A}_1 = \mathcal{A}_0 \cup \{(\mathbf{a}, c_a)\}$, the first outer approximation \mathcal{P}_1 is constructed. Then, the **Projective Cutting-Planes** executes the following steps at each iteration $\text{it} \geq 2$:

1. Select an inner solution \mathbf{x}_{it} , usually by taking a point on the segment joining $\mathbf{x}_{\text{it}-1}$ and $\mathbf{x}_{\text{it}-1} + t_{\text{it}-1}^* \mathbf{d}_{\text{it}-1}$, *i.e.*, on the segment between the previous inner solution and the last pierce point.
2. Consider the direction $\mathbf{d}_{\text{it}} = \text{opt}(\mathcal{P}_{\text{it}-1}) - \mathbf{x}_{\text{it}}$ pointing towards the current optimal (outer) solution $\text{opt}(\mathcal{P}_{\text{it}-1})$. Notice that the objective function value can not deteriorate by advancing along $\mathbf{x}_{\text{it}} \rightarrow \mathbf{d}_{\text{it}}$ because \mathbf{x}_{it} is a simple feasible solution and \mathbf{d}_{it} points to the optimal (outer) solution of $\mathcal{P}_{\text{it}-1} \supseteq \mathcal{P}$ such that we have $\text{optVal}(\mathcal{P}_{\text{it}-1}) \geq \text{optVal}(\mathcal{P}) \geq \mathbf{b}^\top \mathbf{x}_{\text{it}}$ for a maximization problem.

¹If $\mathbf{0}_n$ is infeasible, one has to find other methods to generate an initial feasible solution \mathbf{x}_1 . For example, one can take a feasible solution in a smaller LP (with fewer constraints) whose feasible area is a subset of \mathcal{P} , as in the case of the robust optimization problem from Section 3.1.2.

3. Solve the projection sub-problem $\text{project}(\mathbf{x}_{\text{it}} \rightarrow \mathbf{d}_{\text{it}})$ to determine the maximum step length t_{it}^* , the pierce point $\mathbf{x}_{\text{it}} + t_{\text{it}}^* \mathbf{d}_{\text{it}}$, and a first-hit constraint $(\bar{\mathbf{a}}, \bar{c}_a) \in \mathcal{A}$.

4. If $t_{\text{it}}^* \geq 1$, return $\text{opt}(\mathcal{P}_{\text{it}-1})$ as the optimal solution of the initial LP (2.1) over \mathcal{P} .

If $t_{\text{it}}^* < 1$, then current optimal solution $\text{opt}(\mathcal{P}_{\text{it}-1})$ can be separated, and so, the **Projective Cutting-Planes** performs the following:

- set $\mathcal{A}_{\text{it}} = \mathcal{A}_{\text{it}-1} \cup \{(\bar{\mathbf{a}}, \bar{c}_a)\}$ to obtain a new enlarged constraint set, corresponding to a more refined outer approximation \mathcal{P}_{it} that excludes $\text{opt}(\mathcal{P}_{\text{it}-1})$;
- determine a new current optimal outer solution $\text{opt}(\mathcal{P}_{\text{it}})$ by (re-)optimizing the polytope \mathcal{P}_{it} .
- if $\mathbf{x}_{\text{it}} + t_{\text{it}}^* \mathbf{d}_{\text{it}}$ and $\text{opt}(\mathcal{P}_{\text{it}})$ are close enough, stop and return $\text{opt}(\mathcal{P}_{\text{it}})$. For instance, if (2.1) is a relaxation of an integer program (as in **Column Generation**), the stopping condition is to reach the same rounded-up value of the lower and the upper bounds.
- repeat from Step 1 after updating $\text{it} \leftarrow \text{it} + 1$.

The above algorithm is finitely convergent because it implicitly solves a separation sub-problem on $\text{opt}(\mathcal{P}_{\text{it}-1})$ at each iteration it , generalizing the standard **Cutting-Planes**. As hinted at Step 4, if the intersection sub-problem returns $t_{\text{it}}^* < 1$ at iteration it , then the solution $\text{opt}(\mathcal{P}_{\text{it}-1})$ is certainly separated by the first-hit constraint $(\bar{\mathbf{a}}, \bar{c}_a)$. In pure theory, in the worst case, the proposed algorithm ends up enumerating all constraints of \mathcal{P} and it then eventually returns $\text{opt}(\mathcal{P})$. The fact that this convergence proof is very short is not completely fortuitous. Building on previous work [13, 14] with longer (convergence) theorems, the new **Projective Cutting-Planes** has been deliberately designed to simplify all proofs as much as possible. As in the case of the most standard **Cutting-Planes**, we do not envisage proving that the **Projective Cutting-Planes** converges in a polynomial number of iterations for any instantiation of (2.1).

2.1 Choosing the interior point \mathbf{x}_{it} at each iteration it

Just as the standard **Cutting-Planes**, the proposed **Projective Cutting-Planes** can actually be seen as a rather generic methodology that allows a number of problem-specific adaptations.

A key question for any implementation is the choice of the interior point \mathbf{x}_{it} at (Step 1 of) each iteration it . One might attempt to define \mathbf{x}_{it} as the best feasible solution found up to iteration it (the last pierce point), which actually reduces to assigning $\mathbf{x}_{\text{it}} = \mathbf{x}_{\text{it}-1} + t_{\text{it}-1}^* \mathbf{d}_{\text{it}-1}$. While this aggressive strategy does perform well in certain settings, for many problems it may also lead to poor results in the long run — partly because \mathbf{x}_{it} can fluctuate too much from iteration to iteration (this is referred to as the bang–bang effect, see Section 5.5 for examples). In practice, the best results have often been obtained with a formula of the form $\mathbf{x}_{\text{it}} = \mathbf{x}_{\text{it}-1} + \alpha t_{\text{it}-1}^* \mathbf{d}_{\text{it}-1}$, using $\alpha = 0.1$ for the robust optimization problem (Section 3.1.2), or $\alpha = 0.2$ for the Benders reformulation model (Section 3.2.2), or a value of α below 0.5 in the **Column Generation** model for *Multiple-Length Cutting-Stock*. This is reminiscent of interior point algorithms for linear programming that usually avoid touching the boundary of the polytope before fully converging [7].

The only problem for which we do use the most aggressive definition $\mathbf{x}_{\text{it}} = \mathbf{x}_{\text{it}-1} + t_{\text{it}-1}^* \mathbf{d}_{\text{it}-1}$ is the graph coloring **Column Generation** model from Section 4.1. This \mathbf{x}_{it} choice has the advantage of enabling the resulting **Projective Cutting-Planes** to improve the objective value $\mathbf{b}^\top \mathbf{x}_{\text{it}}$ at each new iteration it , because each projection $\mathbf{x}_{\text{it}} \rightarrow \mathbf{d}_{\text{it}}$ can only increase the objective value, as indicated at Step 2 above. This way, the lower bounds of this most aggressive **Projective Cutting-Planes** variant are monotonically increasing (see Figures 4–5), *i.e.*, they do not exhibit any “yo-yo” effect with ups and downs (as it happens in many **Column Generation** algorithms for example). Graph coloring also differs from the other three problems studied in this paper in the sense that the above aggressive \mathbf{x}_{it} definition does not make the inner solutions \mathbf{x}_{it} generated along the iterations it exhibit strong oscillations (*i.e.*, the bang–bang effects are reduced, see Section 5.5).

More generally, the difference between an aggressive choice (large α) and a “cautious” or well-centered choice (small α) is intuitively illustrated in Figure 2. The red circle represents an aggressive definition of \mathbf{x}_2 associated to a large α , so that \mathbf{x}_2 is very close to the last pierce point $\mathbf{x}_1 + t_1^* \mathbf{d}_1$. Such a choice enables the projection sub-problem at iteration 2 to easily exceed the objective value of the last pierce point $\mathbf{x}_1 + t_1^* \mathbf{d}_1$

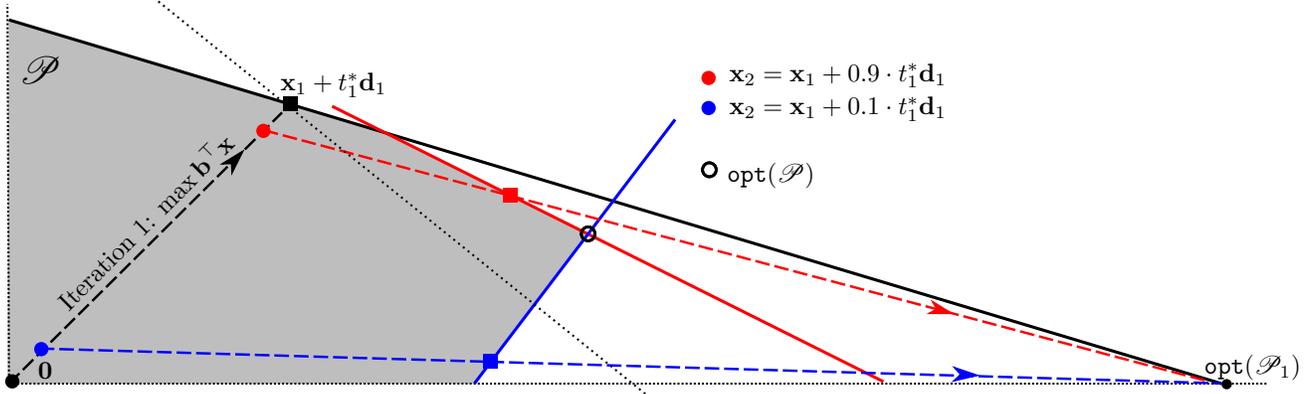


Figure 2: Intuitive illustration of two different choices of the interior point \mathbf{x}_2 at iteration 2. The red choice is more aggressive while the blue one is more cautious.

by only advancing a little from \mathbf{x}_2 towards $\text{opt}(\mathcal{P}_1)$ — see how rapidly the red dashed arrow crosses the black dotted line, *i.e.*, the level set of the last pierce point $\{\mathbf{x} \in \mathbb{R}_+^2 : \mathbf{b}^\top \mathbf{x} = \mathbf{b}^\top (\mathbf{x}_1 + t_1^* \mathbf{d}_1)\}$. The blue circle represents a definition of a point \mathbf{x}_2 closer to $\mathbf{0}_n$, so that it might be difficult to exceed the objective value of the last pierce point $\mathbf{x}_1 + t_1^* \mathbf{d}_1$ by advancing from this \mathbf{x}_2 towards $\text{opt}(\mathcal{P}_1)$, see the blue dashed arrow. The advantage of the blue projection is that it can lead to a stronger (blue) constraint, in the sense that the blue solid line cuts off a larger area of \mathcal{P}_1 (*i.e.*, of the largest triangle) than the red solid line.

2.2 Techniques for designing a fast projection algorithm

A challenging aspect when implementing the new method is the design of a fast projection algorithm, because the iterations of a successful **Projective Cutting-Planes** should not be significantly slower than the iterations of the standard **Cutting-Planes**. For instance, if the projection iterations were two–three times slower than the separation iterations, the overall **Projective Cutting-Planes** would probably be too slow, *i.e.*, it could remain slower than the standard **Cutting-Planes** even if it converged using half iterations. Given that the projection sub-problem generalizes the separation one, one could reasonably assume that any projection algorithm has to be (considerably) slower than the separation algorithm. However, we present below four techniques that can enable one to design a projection algorithm that competes (very) tightly with the separation algorithm in terms of computational speed.

Before presenting these four techniques, let us first explain how the projection sub-problem $\text{project}(\mathbf{x} \rightarrow \mathbf{d})$ reduces to minimizing the following fractional program (for any feasible $\mathbf{x} \in \mathcal{P}$ and for any $\mathbf{d} \in \mathbb{R}^n$):

$$t^* = \min \left\{ \frac{c_a - \mathbf{a}^\top \mathbf{x}}{\mathbf{a}^\top \mathbf{d}} : (\mathbf{a}, c_a) \in \mathcal{A}, \mathbf{d}^\top \mathbf{a} > 0 \right\}. \quad (2.2.1)$$

Given the value t^* that minimizes the above ratio, one can directly check that $\mathbf{a}^\top (\mathbf{x} + t\mathbf{d}) \leq c_a$ holds for all $t \in [0, t^*]$ and for all $(\mathbf{a}, c_a) \in \mathcal{A}$, independently on whether $\mathbf{a}^\top \mathbf{d} > 0$ or not. First, if $\mathbf{a}^\top \mathbf{d} \leq 0$, then $\mathbf{a}^\top (\mathbf{x} + t\mathbf{d}) \leq \mathbf{a}^\top \mathbf{x} \leq c_a$ actually holds for all $t \in [0, \infty]$, simply because $\mathbf{a}^\top \mathbf{x} \leq c_a$ follows from $\mathbf{x} \in \mathcal{P}$. Secondly, if $\mathbf{a}^\top \mathbf{d} > 0$, then $\mathbf{a}^\top (\mathbf{x} + t\mathbf{d}) \leq c_a$ is equivalent to $t \leq \frac{c_a - \mathbf{a}^\top \mathbf{x}}{\mathbf{a}^\top \mathbf{d}}$ which is true for any $t \leq t^*$, because t^* minimizes the above ratio in (2.2.1). This also shows that it is enough to focus only on the constraints $(\mathbf{a}, c_a) \in \mathcal{A}$ that satisfy $\mathbf{a}^\top \mathbf{d} > 0$ when designing the projection algorithm.

A *first projection technique* is based on the idea that the separation algorithm can sometimes be generalized to a projection algorithm *without* increasing the computation time by a (very) large factor. This can *not* be achieved by simply calling the separation algorithm multiple times because such approach would increase the computation time by a factor of at least 2, or often 3 or 4 in practice (see the last paragraph of Section 3.1.3.1). This technique is exemplified on the robust optimization problem from Section 3.1. The constraints \mathcal{A} of this problem are defined by extending an initial set of nominal constraints \mathcal{A}_{nom} as follows: define a robust cut $(\mathbf{a} + \hat{\mathbf{a}}, c_a)$ for each nominal constraint $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$ and for all deviations $\hat{\mathbf{a}}$ of \mathbf{a} , *i.e.*, for all vectors $\hat{\mathbf{a}} \in \mathbb{R}^n$

with at maximum Γ non-zero components such that $\hat{a}_i \in \{-0.01 \cdot a_i, 0, 0.01 \cdot a_i\} \forall i \in [1..n]$. To solve the separation sub-problem on a given $\mathbf{x} \in \mathbb{R}^n$, one has to minimize $c_a - (\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{x}$ over all $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$ and over all deviations $\hat{\mathbf{a}}$ of \mathbf{a} . The projection sub-problem (2.2.1) asks to minimize $\frac{c_a - (\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{x}}{(\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{d}}$ over the same (\mathbf{a}, c_a) and over the same $\hat{\mathbf{a}}$ as above. The two sub-problems become similar in the sense that the objective function change (minimize a ratio instead of a difference) does not substantially change the nature of the subproblem algorithm. More exactly, both sub-problems are essentially solved by iterating over all nominal constraints \mathcal{A}_{nom} , attempting at each element $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$ to decrease either the difference $c_a - (\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{x}$ or resp. the ratio $\frac{c_a - (\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{x}}{(\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{d}}$; this distinction (decrease a ratio instead of difference) is not important enough to drastically alter the running time of the iterative sub-problem algorithm.

The *second technique* to solve (2.2.1) applies when the constraints \mathcal{A} are associated to the (extreme) solutions of an auxiliary polytope \mathcal{P} . This is the case for most Benders decomposition models (Section 3.2) because the separation sub-problem of such models is often formulated as an LP over a Benders sub-problem polytope \mathcal{P} . In this case, (2.2.1) reduces to a linear-fractional program that can be reformulated as a pure LP using the Charnes–Cooper transformation [3]. This leads to an algorithm of the same complexity as the separation one, *i.e.*, they both have the complexity of solving an LP over \mathcal{P} .

A *third projection technique* actually generalizes the above one to the case in which \mathcal{A} is given by the integer feasible solutions of a polytope, as it happens in the **Column Generation** model for graph coloring from Section 4.1. In this model, each constraint $(\mathbf{a}, c_a) = (\mathbf{a}, 1) \in \mathcal{A}$ is associated to a primal column, which, in turn, is given by the incidence vector $\mathbf{a} \in \{0, 1\}^n$ of a stable in the considered graph. The stables of the graph can be seen as the integer solutions of a (stable set) polytope defined by edge inequalities. In this case, we can use a discrete Charnes-Cooper transformation to reduce (2.2.1) to a Disjunctive LP, *i.e.*, the integrality constraints $a_i \in \{0, 1\}$ are translated to disjunctive constraints of the form $\bar{a}_i \in \{0, \bar{\alpha}\}$, where $\bar{\alpha}$ is an additional decision variable. In Section 4.1.3.1, we will argue that there is no fundamental reason why such a Disjunctive LP should always be harder in absolute terms than the ILP used for the separation sub-problem, especially when both programs are solved with similar **Branch and Bound** methods (with bounds determined from continuous relaxations). To show this technique can work on more combinatorial optimization problems in which the columns are given by the solutions of an ILP, we will also provide brief experiments on a different coloring problem with different constraints.

The *fourth technique* can be useful when the constraints \mathcal{A} can be (implicitly) enumerated by **Dynamic Programming**, as in numerous **Column Generation** formulations of combinatorial optimization problems. In many such cases, if the separation sub-problem can be solved by **Dynamic Programming**, so can be the projection sub-problem. To solve the separation sub-problem on a given \mathbf{x} , the **Dynamic Programming** scheme has to enumerate all possible values of c_a and $\mathbf{a}^\top \mathbf{x}$, over all feasible $(\mathbf{a}, c_a) \in \mathcal{A}$. If all these values can be listed in reasonable time, so can be the values of the numerator and the denominator of the ratio $\frac{c_a - \mathbf{a}^\top \mathbf{x}}{\mathbf{a}^\top \mathbf{d}}$ from the intersection sub-problem (2.2.1). To reduce the potential values of the numerator for the (*Multiple-Length*) *Cutting-Stock* problems from Section 4.2, we will need however to use truncated solutions \mathbf{x} , *i.e.*, such that each component of \mathbf{x} is a multiple of 0.2 (see Section 4.2.3.2).

Finally, if the projection sub-problem is too difficult to be solved exactly in reasonable time, it could be enough (at certain iterations *it*) to determine an underestimated feasible step length $t_{\text{it}}^h \leq t_{\text{it}}^*$, so that $\mathbf{x}_{\text{it}} + t_{\text{it}}^h \mathbf{d}_{\text{it}}$ does not necessarily belongs to the boundary of \mathcal{P} . We will implicitly apply this approach in Section 4.1.4, where we will use an artificial, overly-constrained, graph coloring model, but with a simpler projection sub-problem. More generally, to certify that such t_{it}^h represents a feasible step length, one can simply call the separation sub-problem on $\mathbf{x}_{\text{it}} + t_{\text{it}}^h \mathbf{d}_{\text{it}}$, so that a feasible t_{it}^h can be seen as a heuristic step length. Given that the inner solutions \mathbf{x}_{it} are not usually chosen from the boundary points of \mathcal{P} (see Section 2.1 above), such an underestimated step length could be useful in certain problems, but such ideas lie outside the scope of the current paper.

3 Adapting the New Method for Robust Optimization and Benders Decompositions Models

This section presents two models that define \mathcal{P} as a primal (master) polytope, instantiating the general LP (2.1) with unmanageably-many constraints. The first model (Section 3.1) is devoted to a robust optimization problem and the second one explores the Benders' **Cutting-Planes** method (Section 3.2).

3.1 A robust optimization problem with prohibitively many cuts

The main idea in robust optimization is that one seeks an optimal solution that has to remain feasible if certain constraint coefficients deviate (reasonably) from their nominal values. The robust optimization literature is now constantly growing and the acceptable coefficient deviations can be defined in many ways, *e.g.*, using linear or ellipsoid uncertainty sets. However, to avoid unessential complication, we here focus only on the robustness model from [6]; 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 the nominal coefficients of a given constraint can not change all at the same time, always in an unfavorable manner.

3.1.1 The model with prohibitively-many constraints and their separation

Let us first consider a set \mathcal{A}_{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 $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$ a prohibitively-large set $\text{Dev}_\Gamma(\mathbf{a})$ of *deviation vectors* $\hat{\mathbf{a}}$, *i.e.*, vectors $\hat{\mathbf{a}} \in \mathbb{R}^n$ that have at maximum Γ non-zero components and that satisfy $\hat{a}_i \in \{-\delta a_i, 0, \delta a_i\} \forall i \in [1..n]$, using $\delta = 0.01$ in practice. Each such deviation vector $\hat{\mathbf{a}}$ yields a robust cut $(\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{x} \leq c_a$, so that we can state $(\mathbf{a} + \hat{\mathbf{a}}, c_a) \in \mathcal{A}$. In theory, each \hat{a}_i ($\forall i \in [1..n]$) might be allowed to take a fractional value in the interval $[-\delta a_i, \delta a_i]$, thus leading to infinitely-many robust cuts (semi-infinite programming); however, the strongest robust cuts are always obtained when each non-zero \hat{a}_i is either δa_i or $-\delta a_i$. One might find at most $\binom{n}{\Gamma} 2^\Gamma$ deviation vectors for each nominal constraint $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$, because there are $\binom{n}{\Gamma}$ ways to choose the non-zero components of $\hat{\mathbf{a}}$ and each one of them can be either positive or negative, hence the 2^Γ factor.

The general large-scale LP (2.1) is instantiated as the following robust optimization problem:

$$\min \left\{ \mathbf{b}^\top \mathbf{x} : (\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{x} \leq c_a \quad \forall (\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}} \quad \forall \hat{\mathbf{a}} \in \text{Dev}_\Gamma(\mathbf{a}); \quad a_i \in [\mathbf{lb}_i, \mathbf{ub}_i] \quad \forall i \in [1..n] \right\} \quad (3.1.1)$$

Although this problem has a minimization objective unlike general LP (2.1), the main steps of the (standard or new) **Cutting-Planes** method described in Section 2 remain exactly the same. The only difference is that the feasible solutions (pierce points) determined by **Projective Cutting-Planes** represent upper (primal) bounds instead of lower bounds. The last condition $a_i \in [\mathbf{lb}_i, \mathbf{ub}_i]$ of (3.1.1) constitutes the initial constraints \mathcal{A}_0 , most instances using $\mathbf{lb}_i = 0 \forall i \in [1..n]$, *i.e.*, the variables are most often non-negative.

We consider a canonical **Cutting-Planes** for the above (3.1.1), based on the following separation subproblem: given any $\mathbf{x} \in \mathbb{R}^n$, minimize $c_a - (\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{x}$ over all $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$ and over all $\hat{\mathbf{a}} \in \text{Dev}_\Gamma(\mathbf{a})$. For a fixed nominal constraint $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$, the strongest possible deviation $\hat{\mathbf{a}}_x^\top \mathbf{x}$ of (\mathbf{a}, c_a) with respect to \mathbf{x} is determined by maximizing $\hat{\mathbf{a}}_x^\top \mathbf{x} = \arg \max \{ \hat{\mathbf{a}}^\top \mathbf{x} : \hat{\mathbf{a}} \in \text{Dev}_\Gamma(\mathbf{a}) \}$. To find this $\hat{\mathbf{a}}_x^\top \mathbf{x}$, 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, $\hat{\mathbf{a}}_x^\top \mathbf{x}$ can be written as a sum of Γ terms of the form $\delta |a_i x_i|$. 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 $|a_1 x_1|, |a_2 x_2|, \dots, |a_n x_n|$ above) can be determined in $O(n)$ time. We use a partial-sorting algorithm essentially described as follows: iterate over $i \in [1..n]$ and attempt at each step to insert the i^{th} entry in the*

list of the highest Γ values; this operation takes constant time using the appropriate list data structure.² In practice, the repeated use of this algorithm takes around 15% of the total running time for $\Gamma \geq 10$. \square

Compared to our **Cutting-Planes** described above, the algorithm from [6] is slightly different because it returns multiple robust cuts at each separation call. This idea might be very effective in practice, both for the standard **Cutting-Planes** and the **Projective Cutting-Planes**. However, for now, the goal of this study is to compare the projection and the separation sub-problems in a standard setting, and so, we prefer a canonical approach with a unique (robust) cut per iteration.

3.1.2 The Projective Cutting-Planes for the robust optimization problem

Before presenting the projection algorithm for the intersection sub-problem (Section 3.1.3 next), let us first discuss the overall **Projective Cutting-Planes** for the robust optimization problem (3.1.1). In fact, if we consider the projection algorithm as a black-box component, the implementation of the remaining components of the **Projective Cutting-Planes** becomes rather straightforward.

A key question regards the selection of the interior point \mathbf{x}_{it} at each iteration $it \geq 1$. As with most problems studied in this work, experiments suggest that it is not very efficient to define \mathbf{x}_{it} as the best feasible solution found up to the iteration it (the last pierce point). Indeed, while the **Projective Cutting-Planes** is able to find better feasible solutions in the beginning by aggressively assigning $\mathbf{x}_{it} = \mathbf{x}_{it-1} + t_{it-1}^* \mathbf{d}_{it-1}$, this variant eventually needs *more iterations in the long run*. For best long-term results, it is certainly better to choose a more interior point \mathbf{x}_{it} , not too close to the boundary of \mathcal{P} , enabling the inner solutions $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \dots$ to follow a central path (a similar concept is used in some interior point algorithms). As such, we define \mathbf{x}_{it} using the formula $\mathbf{x}_{it} = \mathbf{x}_{it-1} + \alpha t_{it-1}^* \mathbf{d}_{it-1}$ with $\alpha = 0.1 \forall it > 1$.

To construct an initial feasible solution \mathbf{x}_1 , one could be tempted to try $\mathbf{x}_1 = \mathbf{0}_n$, but this is very often not possible because $\mathbf{0}_n$ is usually infeasible. However, it is not difficult to generate \mathbf{x}_1 by constructing a feasible solution in a relatively simple LP defined as follows: for each $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$, generate a constraint $\mathbf{a}^\top \mathbf{x} + 2\delta |\mathbf{a}|^\top \mathbf{x} \leq c_a$, where $|\mathbf{a}| = [|a_1| |a_2| \dots |a_n|]^\top$. If the variables \mathbf{x} are all non-negative (as in most instances), then any solution \mathbf{x} that satisfies $\mathbf{a}^\top \mathbf{x} + 2\delta |\mathbf{a}|^\top \mathbf{x} \leq c_a \forall (\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$ is feasible with regards to all robust cuts — because a robust cut uses a deviation vector $\hat{\mathbf{a}}$ that satisfies $\hat{\mathbf{a}} \leq \delta |\mathbf{a}|$, so that $(\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{x} \leq \mathbf{a}^\top \mathbf{x} + 2\delta |\mathbf{a}|^\top \mathbf{x} \leq c_a$. However, even for the instances that do allow some variables to be negative, the above LP still generated a feasible solution \mathbf{x}_1 in practice. The first direction \mathbf{d}_1 points to the solution of the nominal problem, *i.e.*, we can take $\mathbf{d}_1 = \text{opt}(\mathcal{P}_0) - \mathbf{x}_1$, where \mathcal{P}_0 is the polytope of the nominal problem with no robust cut.

Finally, we noticed that the above LP can remain feasible by replacing $\mathbf{a}^\top \mathbf{x} + 2\delta |\mathbf{a}|^\top \mathbf{x} \leq c_a$ with $\mathbf{a}^\top \mathbf{x} + 2\delta |\mathbf{a}|^\top \mathbf{x} + \Delta \leq c_a$, for some small $\Delta > 0$. The use of this parameter Δ makes the generated solutions \mathbf{x}_1 more 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 at the end of the solution process, when the barrier terms converge to zero.

3.1.3 Solving the projection sub-problem

Recalling (2.2.1), the intersection sub-problem requires minimizing $\frac{c_a - (\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{x}}{(\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{d}}$ over all nominal constraints $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$ and over all deviation vectors $\hat{\mathbf{a}} \in \text{Dev}_\Gamma(\mathbf{a})$ such that $(\mathbf{a} + \hat{\mathbf{a}})^\top \mathbf{d} > 0$. Just as the separation algorithm, the projection algorithm iterates over all nominal constraints \mathcal{A}_{nom} , in an attempt to reduce the above ratio (the step length) at each $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$, *i.e.*, for each $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$ it is possible to

²This list of the largest Γ values is recorded in a self-balancing binary tree, as implemented in the C++ `std::multiset` data structure. At each iteration i , the partial-sorting algorithm has to check if the current value v_{new} is larger than the minimum value v_{min} recorded in the tree. If this is the case, the insertion of v_{new} 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 , by 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, hence this partial sorting can take 15% of the total running time of the overall **Cutting-Planes**.

find several increasingly stronger $\hat{\mathbf{a}}$ that gradually decrease the above ratio. Let t_i^* denote the optimal step length obtained after considering the robust cuts associated to the first i constraints from \mathcal{A}_{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_a) denote the i^{th} constraint from \mathcal{A}_{nom} .
2. Determine the strongest deviation vector $\hat{\mathbf{a}}_{t\mathbf{d}}$ with respect to $\mathbf{x} + t\mathbf{d}$ by maximizing:

$$\hat{\mathbf{a}}_{t\mathbf{d}} = \arg \max \{ \hat{\mathbf{a}}^\top (\mathbf{x} + t\mathbf{d}) : \hat{\mathbf{a}} \in \text{Dev}_\Gamma(\mathbf{a}) \}. \quad (3.1.2)$$

For this, one has to extract the largest Γ 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 $(\mathbf{a} + \hat{\mathbf{a}}_{t\mathbf{d}})^\top (\mathbf{x} + t\mathbf{d}) \leq c_a$, then $\mathbf{x} + t\mathbf{d}$ is feasible with regards to the first i constraints from \mathcal{A}_{nom} (and the associated robust cuts), because any deviation vector $\hat{\mathbf{a}} \in \text{Dev}_\Gamma(\mathbf{a})$ satisfies $\hat{\mathbf{a}}^\top (\mathbf{x} + t\mathbf{d}) \leq \hat{\mathbf{a}}_{t\mathbf{d}}^\top (\mathbf{x} + t\mathbf{d})$. In this case, we have obtained the final value $t_i^* = t$ and we terminate the algorithm (for this value of i). Otherwise, the robust cut $(\mathbf{a} + \hat{\mathbf{a}}_{t\mathbf{d}}, c_a)$ leads to a smaller feasible step length:

$$t' = \frac{c_a - (\mathbf{a} + \hat{\mathbf{a}}_{t\mathbf{d}})^\top \mathbf{x}}{(\mathbf{a} + \hat{\mathbf{a}}_{t\mathbf{d}})^\top \mathbf{d}} < t. \quad (3.1.3)$$

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. In practice, we used the condition “if $t < 10^{-6}$ ” because very small step lengths usually represent numerical computation errors.
5. Set $t = t'$ and repeat from Step 2 (without incrementing i). The underlying idea is that the deviation vector $\hat{\mathbf{a}}_{t\mathbf{d}}$ determined via (3.1.2) is not the strongest one with regards to $\mathbf{x} + t'\mathbf{d}$, because $\hat{\mathbf{a}}_{t\mathbf{d}}$ generates the highest deviation in (3.1.2) with regards to a different point (*i.e.*, $\mathbf{x} + t\mathbf{d}$). But there might exist a different robust cut $(\mathbf{a} + \hat{\mathbf{a}}_{t'\mathbf{d}}, c_a)$ for the same nominal constraint such that $\hat{\mathbf{a}}_{t'\mathbf{d}}^\top (\mathbf{x} + t'\mathbf{d}) > \hat{\mathbf{a}}_{t\mathbf{d}}^\top (\mathbf{x} + t'\mathbf{d})$. 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 $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$ one by one, the step length returned at the last constraint of \mathcal{A}_{nom} provides the sought t^* value.

3.1.3.1 Comparing the running times of the projection and the separation algorithms

In theory, the above projection algorithm could repeat many times the steps 2-5 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 (3.1.3) only a dozen of times at most, *for all* (thousands of) nominal constraints, *i.e.*, for *all* i . For many nominal constraints $(\mathbf{a}, c_a) \in \mathcal{A}_{\text{nom}}$, the above algorithm only concludes at Step 3 that $\mathbf{x} + t\mathbf{d}$ does respect all robust cuts associated to (\mathbf{a}, c_a) ; for such nominal constraints (for such i), the only needed calculations are the partial-sorting algorithm (called once at Step 2) and several simple `for` loops over $[1..n]$.

Furthermore, the intersection algorithm can even stop earlier without scanning all nominal constraints, by returning $t^* = 0$ at Step 4. An exact separation algorithm could not stop earlier, because $c_a - (\mathbf{a} + \hat{\mathbf{a}}_{\mathbf{x}})^\top \mathbf{x}$ can certainly decrease up to the last nominal constraint (\mathbf{a}, c_a) . As such, the projection algorithm can become even faster than the separation one in certain cases. Indeed, for the last (very large) instance from Table 1 with $\Gamma = 50$, a separation iteration takes around 0.62 seconds (in average), while the projection one takes 0.56 seconds (in average). At the other end of the spectrum, for an instance like `nesm` with $\Gamma = 50$, an intersection iteration can take about 30% more time than a separation one. All things considered, one can say that the running time of the above intersection algorithm is similar to that of the separation algorithm.

It would have been substantially less efficient to solve the intersection sub-problem by simply calling the separation sub-problem multiple times. More exactly, such approach would make the projection sub-problem

at least twice as slow as the separation one, *i.e.*, one would need to call the separation sub-problem a first time to find a first robust cut satisfied with equality by some $\mathbf{x} + t\mathbf{d}$, followed by *at least* a second call to check if $\mathbf{x} + t\mathbf{d}$ can be further separated to decrease t . And experiments suggest that a third or a fourth call might well be needed at many iterations in practice. More generally, one of the goals of the paper is to explore techniques that can bring us (very) close to designing a projection algorithm as fast as the separation one.

3.2 The Benders reformulation

3.2.1 The model with prohibitively-many constraints and their separation

First introduced in the 1960s [2], the Benders' method has nowadays become a widely used **Cutting-Planes** approach to solve Integer Linear Program (ILP) of the following form:

$$\min \{ \mathbf{b}^\top \mathbf{x} : \mathbf{B}\mathbf{x} + \mathbf{A}\mathbf{y} \geq \mathbf{c}, \mathbf{x} \in \mathbb{Z}_+^n, \mathbf{y} \geq \mathbf{0} \}. \quad (3.2.1)$$

Generally speaking, \mathbf{x} can encode the main decisions and they are also referred to as design variables. The variables \mathbf{y} could quantify flows in network design/loading problems [5], goods delivered to customers in facility location problems, second-stage uncertain events in two-stage stochastic LPs, etc. The goal is to minimize the cost $\mathbf{b}^\top \mathbf{x}$ of the design decisions \mathbf{x} which have to allow \mathbf{y} to receive feasible values in $\mathbf{B}\mathbf{x} + \mathbf{A}\mathbf{y} \geq \mathbf{c}$, to enable the underlying system to work properly. The integrality condition $\mathbf{x} \in \mathbb{Z}_+^n$ can be lifted, for instance when solving a linear relaxation of (3.2.1), as needed in a **Branch and Bound** algorithm. More generally, one can not rule out that $\mathbf{x} \in \mathbb{R}_+^n$ could represent fractional amounts of installed technology in certain problems. For notational consistency throughout the paper, we actually swapped (the classical interpretation of) notations \mathbf{x} and \mathbf{y} , *i.e.*, in most Benders decomposition papers, \mathbf{y} represents the design decisions and \mathbf{x} are the secondary-level (flow) variables [5].

The Benders' method can actually address even more general programs, introducing a term like $\mathbf{f}^\top \mathbf{y}$ in the objective function of (3.2.1), *e.g.*, to introduce some costs associated to flows. We do not (yet) propose a projection algorithm for this most general Benders model. However, the above Benders LP (3.2.1) is still rather general, because it is often realistic to consider (zero flow costs) $\mathbf{f} = \mathbf{0}_n$, *e.g.*, there is virtually no (volume-based) cost for operations such as: transmitting data along a cable, sending fluids along water pipes, transporting electricity along power lines, etc.

Considering a fixed \mathbf{x} , the inner condition of (3.2.1) reduces to a system of inequalities $\mathbf{A}\mathbf{y} \geq \mathbf{c} - \mathbf{B}\mathbf{x}$, in variables \mathbf{y} . This system admits a feasible solution \mathbf{y} if and only if we can state that $\min \{ \mathbf{0}^\top \mathbf{y} : \mathbf{A}\mathbf{y} \geq \mathbf{c} - \mathbf{B}\mathbf{x}, \mathbf{y} \geq \mathbf{0} \} = 0$. Writing the dual of this LP, any dual feasible solution \mathbf{u} has to belong to $\mathcal{P} = \{ \mathbf{u} \geq \mathbf{0}_m : \mathbf{A}^\top \mathbf{u} \leq \mathbf{0}_n \}$, where m is the number of inequalities in the system $\mathbf{A}\mathbf{y} \geq \mathbf{c} - \mathbf{B}\mathbf{x}$. The dual objective value associated to \mathbf{u} has to be less than or equal to 0, and so, we obtain that $(\mathbf{c} - \mathbf{B}\mathbf{x})^\top \mathbf{u} \leq 0$. This condition can also be derived using the Farkas' lemma (see Footnote 4 of [14]); more generally, we refer the reader to [14, § 2.1] or [5] for more details into the steps of the general Benders reformulation. However, using standard algebraic manipulations of above formulae, (3.2.1) can be equivalently written in the following Benders decomposition form:

$$\min \mathbf{b}^\top \mathbf{x} \quad (3.2.2a)$$

$$\mathcal{P} \left\{ \begin{array}{l} \mathbf{c}^\top \mathbf{u} - (\mathbf{B}\mathbf{x})^\top \mathbf{u} \leq 0 \quad \forall \mathbf{u} \in \mathcal{P} \text{ s. t. } \mathbf{1}_m^\top \mathbf{u} = 1 \\ \mathbf{x} \in \mathbb{Z}_+^n, \end{array} \right. \quad (3.2.2b)$$

$$\mathbf{x} \in \mathbb{Z}_+^n, \quad (3.2.2c)$$

where

$$\mathcal{P} = \{ \mathbf{u} \geq \mathbf{0}_m : \mathbf{A}^\top \mathbf{u} \leq \mathbf{0}_n \} \quad (3.2.3)$$

is the *Benders sub-problem polytope* that does not depend on the current \mathbf{x} . This (3.2.2a)–(3.2.2c) program is an instantiation of the general large-scale LP (2.1); even if \mathcal{P} is now a discrete set, the Benders' method applies the **Cutting-Planes** algorithm exactly as described in Section 2, although it now generates a sequence of discrete sets $\mathcal{P}_1 \supseteq \mathcal{P}_2 \supseteq \mathcal{P}_3 \supseteq \dots \supseteq \mathcal{P}$. At each iteration it , we say \mathcal{P}_{it} corresponds to a relaxed master associated to (3.2.2a)–(3.2.2c), obtained by only keeping a subset of the constraints (3.2.2b). In fact, the only

difference compared to (2.1) is that \mathbf{x} is integer in (3.2.2c), so that $\text{opt}(\mathcal{P}_{\text{it}})$ needs to be determined using an ILP solver instead of an LP solver. Given the current optimal solution $\mathbf{x} = \text{opt}(\mathcal{P}_{\text{it}})$, the separation algorithm solves the following LP to (try to) exclude \mathbf{x} from \mathcal{P} .

$$\max \left\{ \mathbf{c}^\top \mathbf{u} - (\mathbf{B}\mathbf{x})^\top \mathbf{u} : \mathbf{u} \in \mathcal{P}, \mathbf{1}_m^\top \mathbf{u} = 1 \right\} \quad (3.2.4)$$

The condition $\mathbf{1}_m^\top \mathbf{u} = 1$ first arising in (3.2.2b) could be considered superfluous in theory because all positive multiples of $\mathbf{u} \in \mathcal{P}$ belong to \mathcal{P} and they all produce the same inequality (3.2.2b), *i.e.*, the status of this inequality does not change by multiplying all its terms by a positive constant. However, this condition is useful to avoid numerical issues in practice because it enables the separation algorithm to return only normalized constraints (3.2.2b), with no exceedingly large term. Furthermore, this condition removes the extreme rays from (3.2.4), which is also useful because the extreme rays with their unbounded objective values are very difficult to compare, *e.g.*, for the (Simplex) algorithm solving (3.2.4).

3.2.2 The Projective Cutting-Planes for the Benders reformulation

Before presenting the projection algorithm, let us discuss the overall **Projective Cutting-Planes** for the above Benders reformulation (3.2.2a)–(3.2.2c). It essentially executes the steps indicated in Section 2, the main difference being that \mathbf{x} is integer in (3.2.2a)–(3.2.2c). An important consequence of this integrality is that one needs to solve an (NP-hard) ILP instead of an LP to find $\text{opt}(\mathcal{P}_{\text{it}})$ at each iteration it . Because of this, the iterative call to the ILP solver for determining $\text{opt}(\mathcal{P}_{\text{it}})$ becomes the most important computational bottleneck of the overall **Projective Cutting-Planes**. This may be seen as an encouraging factor for adopting the **Projective Cutting-Planes**: the projection sub-problem (an LP) is computationally far less expensive than the master problem, *i.e.*, than the above ILP for determining $\text{opt}(\mathcal{P}_{\text{it}})$.

However, we will also examine the linear relaxation of the Benders reformulation (3.2.2a)–(3.2.2c), replacing $\mathbf{x} \in \mathbb{Z}_+^n$ with $\mathbf{x} \in \mathbb{R}_+^n$ in (3.2.2c). For this problem variant, the only difference compared to the general large-scale LP (2.1) is that the objective function is minimized instead of maximized, which does not essentially change our **Cutting-Planes** algorithms.

For both the integer and the relaxed problem variant, a key question concerns the choice of the interior point \mathbf{x}_{it} at each iteration $\text{it} \geq 1$. The very first feasible solution \mathbf{x}_1 is determined using a problem-specific routine we will describe later in Section 3.2.4.2. For $\text{it} > 1$, experiments suggest it is preferable to choose an interior point \mathbf{x}_{it} relatively far from the boundary — as with the robust optimization problem previously studied. As in Section 3.1.2, we apply the formula $\mathbf{x}_{\text{it}} = \mathbf{x}_{\text{it}-1} + \alpha t_{\text{it}-1}^* \mathbf{d}_{\text{it}-1}$, using a value of α significantly below 1. More exactly, we set $\alpha = 0.2$ for the linear relaxation of the Benders reformulation. For the original integer model, we set $\alpha = 0.2$ only during the first 100 iterations; after that, we set $\alpha = 0.4$, *i.e.*, the algorithm becomes slightly more aggressive in the second part of the search.

Remark 2 *A second consequence of the integrality condition $\mathbf{x} \in \mathbb{Z}_+^n$ is that the pierce point $\mathbf{x}_{\text{it}} + t_{\text{it}}^* \mathbf{d}_{\text{it}}$ returned by the projection algorithm is not necessarily integer. However, depending on the underlying problem, one can usually build an integer feasible solution by simply rounding up all components of $\mathbf{x}_{\text{it}} + t_{\text{it}}^* \mathbf{d}_{\text{it}}$. At least when \mathbf{x} encodes design decisions to install (transmission) facilities, there is generally no reason to forbid an increase (by rounding) of the number of these facilities. This is formally proved for the application problem example from Section 3.2.4 in Observation 4 of [14].*

3.2.3 The intersection sub-problem algorithm

Consider an interior point \mathbf{x} satisfying all constraints (3.2.2b) and a direction $\mathbf{d} \in \mathbb{R}^n$. Based on Definition 1 (p. 4), the intersection sub-problem $\text{project}(\mathbf{x} \rightarrow \mathbf{d})$ requires finding:

- (i) the maximum step length $t^* \geq 0$ such that $\mathbf{x} + t^* \mathbf{d}$ satisfies all constraints (3.2.2b);
- (ii) a vector $\mathbf{u} \in \mathcal{P}$ such that the associated constraint (3.2.2b) is respected with equality by the pierce point $\mathbf{x} + t^* \mathbf{d}$. We do not ask the returned \mathbf{u} to be normalized, *i.e.*, it is not necessary to multiply \mathbf{u} by some factor to make it satisfy a condition like $\mathbf{1}_m^\top \mathbf{u} = 1$ from (3.2.2b).

Substituting $\mathbf{x} + t^*\mathbf{d}$ for \mathbf{x} in (3.2.2b), the intersection sub-problem requires finding the maximum value t^* such that $\mathbf{c}^\top \mathbf{u} - (\mathbf{B}(\mathbf{x} + t^*\mathbf{d}))^\top \mathbf{u} \leq 0 \forall \mathbf{u} \in \mathcal{P}$, equivalent to $-t^*(\mathbf{B}\mathbf{d})^\top \mathbf{u} \leq (\mathbf{B}\mathbf{x})^\top \mathbf{u} - \mathbf{c}^\top \mathbf{u} \forall \mathbf{u} \in \mathcal{P}$. The right-hand side of this last inequality is always non-negative because \mathbf{x} is feasible and satisfies all constraints (3.2.2b). Furthermore, any $\mathbf{u} \in \mathcal{P}$ associated to a non-positive $-(\mathbf{B}\mathbf{d})^\top \mathbf{u} \leq 0$ would allow t^* to be arbitrarily large. As such, hereafter we focus only on the vectors $\mathbf{u} \in \mathcal{P}$ such that $-(\mathbf{B}\mathbf{d})^\top \mathbf{u} > 0$, and so, t^* can be determined by solving the following linear-fractional program:

$$t^* = \min \left\{ \frac{(\mathbf{B}\mathbf{x} - \mathbf{c})^\top \mathbf{u}}{-(\mathbf{B}\mathbf{d})^\top \mathbf{u}} : \mathbf{u} \in \mathcal{P}, -(\mathbf{B}\mathbf{d})^\top \mathbf{u} > 0 \right\} \quad (3.2.5)$$

This program can be translated to a standard LP using the Charnes–Cooper transformation [3]. More exactly, writing $\bar{\mathbf{u}} = \frac{\mathbf{u}}{-(\mathbf{B}\mathbf{d})^\top \mathbf{u}}$, one can show that (3.2.5) is completely equivalent to:

$$t^* = \min (\mathbf{B}\mathbf{x} - \mathbf{c})^\top \bar{\mathbf{u}} \quad (3.2.6a)$$

$$\mathbf{A}^\top \bar{\mathbf{u}} \leq \mathbf{0}_n \quad (3.2.6b)$$

$$-(\mathbf{B}\mathbf{d})^\top \bar{\mathbf{u}} = 1 \quad (3.2.6c)$$

$$\bar{\mathbf{u}} \geq \mathbf{0}_m \quad (3.2.6d)$$

It is not difficult to check that the above change of variable $\mathbf{u} \rightarrow \bar{\mathbf{u}}$ transforms a feasible solution of (3.2.5) into a feasible solution of (3.2.6a)–(3.2.6d) with the same objective value. Conversely, a feasible solution $\bar{\mathbf{u}}$ of (3.2.6a)–(3.2.6d) is itself feasible in (3.2.5) and it has the same objective value in both programs.

The algorithm for solving the LP (3.2.6a)–(3.2.6d) has clearly the same asymptotic running time as the one for the separation sub-problem (3.2.4), *i.e.*, both sub-problems have the complexity of solving an LP with m variables and n or $n + 1$ constraints.

3.2.4 From the general Benders model to a network design problem

We introduced the **Projective Cutting-Planes** so far in a rather generic Benders model (3.2.2a)–(3.2.2c); we did not want to impair the understanding of the main ideas with the particularities of a specific problem. However, we will hereafter use a network design problem to experimentally compare the proposed **Projective Cutting-Planes** with the standard Benders’s **Cutting-Planes**.

3.2.4.1 The Benders reformulation model for a network design problem

We consider the network design problem from [14, § 4] which asks to install multiple times a technology (*e.g.*, cables or other telecommunication links) on the edges E of a graph $G = (V, E)$. The installed transmission facilities (links) should allow one to transfer data from a source or origin $O \in V$ towards a set of terminals $T \subsetneq V$, each $i \in T$ having a flow (data) demand of f_i . We propose to use variables $\mathbf{x} \in \mathbb{Z}_+^n$ to represent the number of links installed on each edge (so that $|E| = n$) and $\mathbf{y} \geq \mathbf{0}$ to encode data flows along edges. The goal is to find the minimum total number of links needed to accommodate a *one-to-many* flow from O to T . The initial Benders ILP (3.2.1) is instantiated as follows:

$$\min \sum_{\{i,j\} \in E} x_{ij} \left(= \mathbf{1}_n^\top \mathbf{x} \right) \quad (3.2.7a)$$

$$\sum_{\{i,j\} \in E} y_{ji} - \sum_{\{i,j\} \in E} y_{ij} \geq 0, \forall i \notin T \cup \{O\} \quad (3.2.7b)$$

$$\sum_{\{i,j\} \in E} y_{ji} - \sum_{\{i,j\} \in E} y_{ij} \geq f_i, \forall i \in T \quad (3.2.7c)$$

$$b_{\text{wd}} x_{ij} - y_{ij} - y_{ji} \geq 0, \forall \{i, j\} \in E, i < j \quad (3.2.7d)$$

$$x_{ij} \in \mathbb{Z}_+, y_{ij}, y_{ji} \geq 0, \forall \{i, j\} \in E, i < j \quad (3.2.7e)$$

The principles underpinning this model are the following. For each edge $\{i, j\} \in E$, the design variable x_{ij} indicates the number of installed links between i and j , y_{ij} encodes the flow from i to j and y_{ji} indicates the flow from j to i . The objective function contains no \mathbf{y} term because we assume the flow costs are zero, which is realistic when the cost of sending data along a cable is virtually zero. The inequalities (3.2.7b)–(3.2.7c) represent modified flow conservation constraints, *i.e.*, notice they allow flow losses but forbid any flow creation, except at the source O . We prefer such inequalities to standard flow conservation equalities, to make (3.2.7a)–(3.2.7e) more similar to the initial Benders ILP (3.2.1). However, any feasible solution that generates some flow loss in (3.2.7a)–(3.2.7e) can be transformed into a solution with no flow loss, by decreasing some entering flows in (3.2.7b) or (3.2.7c). Constraints (3.2.7d) indicate that the flow transmitted in either sense on any edge $\{i, j\}$ can not exceed the number of links mounted on $\{i, j\}$ multiplied by the bandwidth b_{wd} of each individual link. The condition $i < j$ from (3.2.7d) and (3.2.7e) only arises because there is a unique design variable x_{ij} for each $\{i, j\} \in E$.

To build the Benders reformulation of above (3.2.7a)–(3.2.7e), one has to instantiate the general steps from Section 3.2.1. Accordingly, we consider (3.2.7b)–(3.2.7e) as an inner LP (a system of inequalities) and we will dualize the decision variables \mathbf{y} . Recalling how we generated the dual variables $\mathbf{u} \in \mathbb{R}_+^m$ of (3.2.3), we here obtain a vector \mathbf{u} of $|E| + |V| - 1$ dual variables, *i.e.*, one variable u_{ij} for each $\{i, j\} \in E$ with $i < j$ from (3.2.7d) and one variable u_i for each $i \in V \setminus \{O\}$ from (3.2.7b)–(3.2.7c). The dual constraints are built from the coefficients of the columns of y_{ij} and y_{ji} , for all $\{i, j\} \in E$ with $i < j$. After moving $b_{\text{wd}}x_{ij}$ in the right-hand side of (3.2.7d), the dual objective function is built from the right-hand side terms in (3.2.7c)–(3.2.7d). Following the development that led to (3.2.2b), the dual objective value has to be no larger than 0, *i.e.*, we obtain $\sum_{i \in T} f_i u_i - \sum_{\{i, j\} \in E} b_{\text{wd}} x_{ij} u_{ij} \leq 0$. Referring the reader to [14, § 3.2] for full exact details, the Benders reformulation of (3.2.7a)–(3.2.7e) can be written as below, obtaining an instance of (3.2.2a)–(3.2.2c):

$$\min \mathbf{1}_n^\top \mathbf{x} \tag{3.2.8a}$$

$$\mathcal{P} \left\{ \begin{array}{l} \sum_{i \in T} f_i u_i - \sum_{\{i, j\} \in E} b_{\text{wd}} x_{ij} u_{ij} \leq 0 \quad \forall \mathbf{u} \in \mathcal{P} \text{ s. t. } \mathbf{1}^\top \mathbf{u} = 1 \\ \mathbf{x} \in \mathbb{Z}_+^n, \end{array} \right. \tag{3.2.8b}$$

$$\tag{3.2.8c}$$

where \mathcal{P} is given by (3.2.9a)–(3.2.9c) below, *i.e.*, by the dual constraints associated to the columns of y_{ij} and y_{ji} from (3.2.7b)–(3.2.7e). Since there is no constraint (3.2.7b) or (3.2.7c) associated to the origin O , we use the convention that the term u_i (resp. u_j) vanishes in (3.2.9a)–(3.2.9b) when i (resp. j) equals O .

$$\mathcal{P} \left\{ \begin{array}{l} y_{ij} : -u_{ij} - u_i + u_j \leq 0 \quad \forall \{i, j\} \in E, i < j \\ y_{ji} : -u_{ij} - u_j + u_i \leq 0 \quad \forall \{i, j\} \in E, i < j \\ \mathbf{u} \geq \mathbf{0} \end{array} \right. \tag{3.2.9a}$$

$$\tag{3.2.9b}$$

$$\tag{3.2.9c}$$

Remark 3 We will also provide numerical results for the linear relaxation of the above (3.2.8a)–(3.2.8c), replacing $\mathbf{x} \in \mathbb{Z}_+^n$ with $\mathbf{x} \in \mathbb{R}_+^n$. This amounts to installing fractional amounts of transmission capacities along the edges, which could be very realistic in certain applications, *e.g.*, if the transmission capacities are obtained by acquiring bandwidth from a telecommunication carrier. Furthermore, even if only the integer model is relevant, the linear relaxation could be needed by a **Branch and Bound** algorithm, at different nodes of the branching tree.

3.2.4.2 The Projective Cutting-Planes and the initial feasible solution

The above model (3.2.8a)–(3.2.8c) is an instantiation of the general Benders reformulation (3.2.2a)–(3.2.2c) from Section 3.2.1; it also fits well the most general large-scale LP (2.1). The **Projective Cutting-Planes** described in Section 3.2.2 in a general Benders context can be directly applied to solve the above (3.2.8a)–(3.2.8c).

The only detail that remains to be filled concerns the very first feasible solution \mathbf{x}_1 . We construct it by assigning to each edge $\{i, j\} \in E$ the value $\left\lceil \frac{\sum_{i \in T} f_i}{b_{\text{wd}}} \right\rceil$, so that each edge has enough capacity to transfer

all the demands, making this \mathbf{x}_1 certainly feasible. The first direction is $\mathbf{d}_1 = -\mathbf{1}_n$, *i.e.*, the direction with the fastest rate of objective function improvement. We also tried to define \mathbf{x}_1 by assigning the above value $\left\lceil \frac{\sum_{i \in T} f_i}{b_{\text{wd}}} \right\rceil$ only to the edges of a spanning tree of G . This latter solution is also feasible, but it assigns the value 0 to the numerous edges outside the spanning tree, and so, it can not be qualified as “well-centered”. This would reduce the effectiveness of the overall **Projective Cutting-Planes** in the long run because it is preferable to start from a more interior solution \mathbf{x}_1 , as also described in the last paragraph of Section 3.1.2, confirming ideas used in interior point algorithms.

3.2.4.3 The projection sub-problem

Consider an (interior) solution $\mathbf{x} \in \mathbb{R}_+^n$ that satisfies all constraints (3.2.8b) and a random direction $\mathbf{d} \in \mathbb{R}^n$. To solve the intersection sub-problem **project**($\mathbf{x} \rightarrow \mathbf{d}$), we will instantiate the general linear-fractional program (3.2.5), following the development from Section 3.2.3. Accordingly, notice that the numerator of (3.2.5) contains a term $(\mathbf{B}\mathbf{x})^\top \mathbf{u}$ that was build from the terms involving \mathbf{x} in the constraint (3.2.2b) of the general Benders model (3.2.2a)–(3.2.2c). Since (3.2.2b) has been instantiated to the above (3.2.8b), one can check that $(\mathbf{B}\mathbf{x})^\top \mathbf{u}$ corresponds to $\sum_{\{i,j\} \in E} b_{\text{wd}} x_{ij} u_{ij}$. Using the fact that \mathbf{d} is defined in the same space as \mathbf{x} , one can also check that $(\mathbf{B}\mathbf{d})^\top \mathbf{u}$ becomes $\sum_{\{i,j\} \in E} b_{\text{wd}} d_{ij} u_{ij}$. Finally, $\mathbf{c}^\top \mathbf{u}$ represents the free terms (without \mathbf{x}) from (3.2.2b) that correspond to $\sum_{i \in T} f_i u_i$. We thus obtain that (3.2.5) can be instantiated as follows:

$$t^* = \min \left\{ \frac{\sum_{\{i,j\} \in E} b_{\text{wd}} x_{ij} u_{ij} - \sum_{i \in T} f_i u_i}{-\sum_{\{i,j\} \in E} b_{\text{wd}} d_{ij} u_{ij}} : \mathbf{u} \in \mathcal{P}, -\sum_{\{i,j\} \in E} b_{\text{wd}} d_{ij} u_{ij} > 0 \right\} \quad (3.2.10)$$

Recalling how we translated the linear-fractional program (3.2.5) to the standard LP (3.2.6a)–(3.2.6d), we apply the same Charnes–Cooper transformation to reformulate (3.2.10) as a pure LP. Accordingly, after writing $\bar{\mathbf{u}} = \frac{\mathbf{u}}{-\sum_{\{i,j\} \in E} b_{\text{wd}} d_{ij} u_{ij}}$, (3.2.10) becomes equivalent to:

$$t^* = \min \sum_{\{i,j\} \in E} b_{\text{wd}} x_{ij} \bar{u}_{ij} - \sum_{i \in T} f_i \bar{u}_i \quad (3.2.11a)$$

$$-\bar{u}_{ij} - \bar{u}_i + \bar{u}_j \leq 0 \quad \forall \{i,j\} \in E, i < j \quad (3.2.11b)$$

$$-\bar{u}_{ij} - \bar{u}_j + \bar{u}_i \leq 0 \quad \forall \{i,j\} \in E, i < j \quad (3.2.11c)$$

$$-\sum_{\{i,j\} \in E} b_{\text{wd}} d_{ij} \bar{u}_{ij} = 1 \quad (3.2.11d)$$

$$\bar{\mathbf{u}} \geq \mathbf{0}, \quad (3.2.11e)$$

where we used the convention that if i (resp. j) equals O then the term u_i (resp. u_j) vanishes in (3.2.11b)–(3.2.11c), as we did for the constraints (3.2.9a)–(3.2.9b) defining \mathcal{P} .

4 The Projective Cutting-Planes in Column Generation

In **Column Generation**, the generic LP (2.1) is instantiated as the dual of a relaxed primal LP. The prohibitively-many constraints of \mathcal{P} are given by an unmanageably-large set \mathcal{A} of primal columns. These columns can represent stables in graph coloring, cutting patterns in *(Multiple-Length) Cutting-Stock*, routes in vehicle routing problems, assignments of courses to timeslots in timetabling, or any specific subsets in the most general set-covering problem. Given a column $(\mathbf{a}, c_a) \in \mathcal{A}$ of such a problem, c_a is the objective function coefficient and $\mathbf{a} \in \mathbb{Z}_+^n$ is often an incidence vector such that a_i indicates how many times an element $i \in [1..n]$ is covered by \mathbf{a} . We use a primal decision variable y_a to encode the number of selections of each column $(\mathbf{a}, c_a) \in \mathcal{A}$. The **Column Generation** model 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. After relaxing $y_a \in \mathbb{Z}_+$ into $y_a \in \mathbb{R}_+$, the primal program becomes:

$$\mathbf{x} : \begin{cases} \min \sum c_a y_a \\ \sum a_i y_a \geq b_i \quad \forall i \in [1..n] \\ y_a \geq 0 \quad \forall (\mathbf{a}, c_a) \in \mathcal{A} \end{cases} \quad (4.1)$$

The dual LP is:

$$\mathcal{P} \begin{cases} \max \mathbf{b}^\top \mathbf{x} \\ \mathbf{a}^\top \mathbf{x} \leq c_a \quad \forall (\mathbf{a}, c_a) \in \mathcal{A} \\ \mathbf{x} \geq \mathbf{0}_n \end{cases} \quad (4.2)$$

The **Column Generation** method can be seen as a **Cutting-Planes** algorithm (*e.g.*, Kelley’s method) acting on the above dual program (4.2). This program fits very well the general LP (2.1) and we will hereafter use (4.2) to show how to adapt the **Projective Cutting-Planes** to different **Column Generation** models.

4.1 Graph coloring

4.1.1 The model(s) with prohibitively-many constraints and their separation

Standard graph coloring belongs to a large class of coloring problems that involve assignments of colors (labels) to vertices, *e.g.*, multi-coloring, defective coloring, list coloring, sum coloring, etc. Besides their intrinsic interest, such problems enjoy widespread applications in various fields of science and engineering, such as frequency assignment, register allocation in compilers, timetabling or scheduling. Standard graph coloring can be directly formulated as a set covering problem: determine the minimum number of stables (independent sets) of a given graph $G(V, E)$ needed to cover (color) each vertex of V once. Focusing on the dual LP (4.2), each constraint $(\mathbf{a}, c_a) \in \mathcal{A}$ corresponds to the incidence vector \mathbf{a} of a stable of G and we always consider $c_a = 1$ because each color counts once.

The **Column Generation** method optimizes (4.2) by solving at each iteration the separation sub-problem $\min_{(\mathbf{a}, 1) \in \mathcal{A}} 1 - \mathbf{a}^\top \mathbf{x}$, where \mathbf{x} is the current optimal (outer) solution $\text{opt}(\mathcal{P}_{\text{it}})$ at iteration it . In standard graph coloring, the constraints \mathcal{A} are given by the standard stables of G , so that the above separation sub-problem reduces to the maximum weight stable problem with weights \mathbf{x} (a well-known NP-hard problem). This **Column Generation** coloring model has been widely-studied (see [12, 9] and references therein); besides popularity reasons, this also comes from the fact that graph coloring is a rather generic problem with no particularly skewed constraints. More exactly, there seems to be little potential in analyzing, reformulating or reinterpreting the simple constraints of graph coloring; progress can rather be expected from focusing on (more general) optimization aspects. According to the abstract of [9], **Column Generation** is also the “best method known for determining lower bounds on the vertex coloring number”.

The most standard coloring variant considers a unique color per vertex, so that $b_i = 1 \quad \forall i \in V = [1..n]$, *i.e.*, the objective function coefficients in (4.2) are given by $\mathbf{b} = \mathbf{1}_n$. This can be easily generalized to a multi-coloring problem with $\mathbf{b} \neq \mathbf{1}_n$, in which one has to assign multiple colors per vertex. Multiply-colored vertices could naturally arise in many applications of graph coloring. For example, a frequency allocation problem might ask to assign multiple frequencies per station; in university timetabling, one might require multiple timeslots per course; in scheduling, certain jobs might need several resources, etc. An interesting feature of this multi-coloring variant is that the maximum clique size is no longer a lower bound, while all the lower bounds $\mathbf{b}^\top \mathbf{x}_1, \mathbf{b}^\top \mathbf{x}_2, \mathbf{b}^\top \mathbf{x}_3, \dots$ generated by **Projective Cutting-Planes** remain valid.

4.1.2 The Projective Cutting-Planes for Graph Coloring

The main steps of the **Projective Cutting-Planes** from Section 2 can be applied on the dual **Column Generation** program (4.2) with very few customizations. The choice of \mathbf{x}_{it} is given by $\mathbf{x}_{\text{it}} = \mathbf{x}_{\text{it}-1} + t_{\text{it}-1}^* \mathbf{d}_{\text{it}-1}$ at each iteration $\text{it} > 1$, so that \mathbf{x}_{it} becomes the best feasible solution found so far (the last

pierce point); graph coloring is the only problem studied in the paper for which this choice leads to good results in the long run. Regarding the very first iteration, we take simply $\mathbf{x}_1 = \mathbf{0}_n$ because $\mathbf{0}_n$ is feasible in (4.2). The first direction \mathbf{d}_1 is constructed by assigning to each component $v \in V = [1..n]$ the value $\frac{1}{|\text{stab}(v)|}$, where $\text{stab}(v) \ni v$ is the stable containing v in a given initial feasible coloring (determined heuristically as stated in Section 5.3.2, Footnote 8, p. 33). The use of an initial feasible coloring is useful for warm starting reasons and it offers several advantages to both the standard **Column Generation** and the **Projective Cutting-Planes**:

- The heuristic solution provides an initial set of stables or initial constraints \mathcal{A}_0 in (4.2), so as to start from the very first iteration with a reasonable outer approximation $\mathcal{P}_0 \supseteq \mathcal{P}$.
- The first outer approximation \mathcal{P}_0 obtained as above leads to a first upper bound $\mathbf{b}^\top \text{opt}(\mathcal{P}_0)$ that is equal to the number of colors used by the heuristic coloring. This way, a quality upper bound is known from the very first iteration; one can easily refer to it to (more clearly) evaluate the gap of all lower bounds reported at subsequent iterations. Without this initial heuristic coloring, one might need dozens or hundreds of iterations to obtain an upper bound of the same quality.
- If we had started by projecting $\mathbf{0}_n \rightarrow \mathbf{1}_n$, we would have obtained a very first pierce point $t_1^* \mathbf{1}_n = \frac{1}{\alpha(G)} \mathbf{1}_n$ that might correspond to multiple constraints of (4.2) associated to multiple stables of maximum size $\alpha(G)$. If one then takes $\mathbf{x}_2 = \mathbf{x}_1 + t_1^* \mathbf{1}_n = \frac{1}{\alpha(G)} \mathbf{1}_n$, the second projection can return $t_2^* = 0$ because of a second stable of size $\alpha(G)$. If this repeats a third or a fourth time, the iterative solution process could stall for too many iterations, generating a form of degeneracy.

We will also apply the **Projective Cutting-Planes** on a second graph coloring model in which the constraints \mathcal{A} of (4.2) are defined using a new (broader) notion of reinforced relaxed stables (RR-stables). In this new model, each element of \mathcal{A} is associated to a solution of an auxiliary polytope \mathcal{P} that does contain the standard stables, so that $(\mathbf{a}, 1) \in \mathcal{A} \iff \mathbf{a} \in \mathcal{P}$. The advantage of this second model is that it has a considerably simpler projection sub-problem that can be formulated (Section 4.1.4) as a pure LP. The disadvantage is that the new (4.2) model becomes overly-constrained, *i.e.*, the new (4.2) contains artificial constraints, because \mathcal{P} contain many other elements besides the legitimate standard stables. However, each feasible inner solution $\mathbf{x}_{i\tau}$ generated by **Projective Cutting-Planes** on the new (4.2) model is also feasible in the original (4.2) model. We will see that the lower bounds $\mathbf{b}^\top \mathbf{x}_{i\tau}$ calculated on the second (overly-constrained) model can be very fast and compete rather well with the bounds obtained using the original model. Section 4.1.3 presents the projection sub-problem in the original model; Section 4.1.4 is devoted to the projection sub-problem in the new model.

4.1.3 The Projection Sub-Problem in the Original Model with Standard Stables

Following the steps that led to the general intersection sub-problem formulation (2.2.1), the intersection sub-problem $\text{project}(\mathbf{x} \rightarrow \mathbf{d})$ for standard graph coloring can be written as follows:

$$t^* = \min \frac{1 - \mathbf{x}^\top \mathbf{a}}{\mathbf{d}^\top \mathbf{a}} \tag{4.1.1a}$$

$$\mathbf{d}^\top \mathbf{a} > 0 \tag{4.1.1b}$$

$$\mathcal{P}_{0-1} \begin{cases} a_i + a_j \leq 1, & \forall \{i, j\} \in E \\ a_i \in \{0, 1\} & \forall i \in [1..n] \end{cases} \tag{4.1.1c}$$

This is an integer linear-fractional program that will be translated to a Disjunctive LP in which the integrality constraints $a_i \in \{0, 1\} \forall i \in [1..n]$ are reformulated as disjunctive constraints of the form $\bar{a}_i \in \{0, \bar{\alpha}\} \forall i \in [1..n]$. We will see that a disjunctive constraint breaks the continuity in the same manner as integrality constraint, so that the resulting program has a discrete feasible area and can be optimized with similar **Branch and Bound** methods as an ILP. We recall that the standard separation sub-problem reduces to solving the ILP $\min \{1 - \mathbf{x}^\top \mathbf{a} : \mathbf{a} \in \mathcal{P}_{0-1}\}$, where $\mathbf{x} \in \mathbb{R}_+^n$ is the current optimal outer solution. The

constraints $a_i + a_j \leq 1 \forall \{i, j\} \in E$ from (4.1.1.c) are referred to as edge inequalities [10, 12]; the convex closure $\text{conv}(\mathcal{P}_{0-1})$ of the set \mathcal{P}_{0-1} of standard stabes is referred to as the stable set polytope.

To reformulate (4.1.1.a)–(4.1.1.c) as a Disjunctive LP, we now apply a discrete version of the Charnes–Cooper transformation initially proposed for standard LPs [3]. Accordingly, let us consider a change of variables $\bar{\mathbf{a}} = \frac{\mathbf{a}}{\mathbf{d}^\top \mathbf{a}}$ and $\bar{\alpha} = \frac{1}{\mathbf{d}^\top \mathbf{a}}$; we will prove that (4.1.1.a)–(4.1.1.c) is completely equivalent to:

$$t^* = \min \bar{\alpha} - \mathbf{x}^\top \bar{\mathbf{a}} \tag{4.1.2a}$$

$$\bar{a}_i + \bar{a}_j \leq \bar{\alpha} \quad \forall \{i, j\} \in E \tag{4.1.2b}$$

$$\mathbf{d}^\top \bar{\mathbf{a}} = 1 \tag{4.1.2c}$$

$$\bar{a}_i \in \{0, \bar{\alpha}\} \quad \forall i \in [1..n] \tag{4.1.2d}$$

$$\bar{\alpha} \geq 0 \tag{4.1.2e}$$

To prove this equivalence, let us first show that the change of variables $\mathbf{a} \rightarrow \bar{\mathbf{a}}, \bar{\alpha}$ maps a feasible solution \mathbf{a} of (4.1.1.a)–(4.1.1.c) to a feasible solution of (4.1.2.a)–(4.1.2.e) with the same objective value. To check this, it is actually enough to directly substitute $\bar{a}_i = \frac{a_i}{\mathbf{d}^\top \mathbf{a}} \forall i \in [1..n]$ and $\bar{\alpha} = \frac{1}{\mathbf{d}^\top \mathbf{a}} > 0$ to notice that the initial constraints of (4.1.1.a)–(4.1.1.c) are transformed into the constraints of the Disjunctive LP. Conversely, a feasible solution of (4.1.2.a)–(4.1.2.e) can be reversely mapped to $\mathbf{a} = \frac{1}{\bar{\alpha}} \cdot \bar{\mathbf{a}}$, which is a feasible solution of the initial program. To show this, first notice that the expression $\frac{1}{\bar{\alpha}} \cdot \bar{\mathbf{a}}$ is consistent in the sense that $\bar{\alpha} \neq 0$ — otherwise, $\bar{\alpha} = 0$ would make $\bar{\mathbf{a}} = \mathbf{0}_n$ via (4.1.2.d), so that (4.1.2.c) would certainly be infeasible. One can directly check that the resulting \mathbf{a} satisfies all constraints in the initial program. The equality of the objective values follows from $\bar{\alpha} - \mathbf{x}^\top \bar{\mathbf{a}} = \frac{\bar{\alpha}}{\mathbf{d}^\top \bar{\mathbf{a}}} - \frac{\mathbf{x}^\top \bar{\mathbf{a}}}{\mathbf{d}^\top \bar{\mathbf{a}}} = \frac{1}{\mathbf{d}^\top \mathbf{a}} - \frac{\mathbf{x}^\top \mathbf{a}}{\mathbf{d}^\top \mathbf{a}}$.

4.1.3.1 The resulting Disjunctive LP can be in theory as hard as the associated ILP

We have just formulated the projection sub-problem as the Disjunctive LP (4.1.2.a)–(4.1.2.e). The integrality constraints $a_i \in \{0, 1\}$ have been transformed into disjunctive constraints of the form $\bar{a}_i \in \{0, \bar{\alpha}\}$. The integrality and the disjunctive constraints break the continuity in a similar manner; we find *no* deep meaningful difference that would make one much easier to handle than the other. Both the ILP (for the separation sub-problem) and the Disjunctive LP (projection sub-problem) are solved by a similar **Branch and Bound** algorithm, constructing in both cases a branching tree in which each node corresponds to a relaxation that lifts certain (integer or resp. disjunctive) constraints.

However, in practice, we solve both programs using the tools from the optimization software package `cplex`, the disjunctive constraints being implemented as logical constraints. As such, we implicitly use a larger arsenal on the ILP. For instance, `cplex` can generate well-studied valid inequalities (mixed-integer rounding cuts, Gomory cuts, etc) on the ILP; although many such ILP cuts could in theory be transformed into Disjunctive LP cuts,³ `cplex` does not “realize” this and it does not generate such cuts on the Disjunctive LP.

This comes from the fact that `cplex` does *not* have the notion of valid inequalities satisfied by all “discrete” solutions $\bar{\mathbf{a}}$ of the Disjunctive LP (*i.e.*, discrete in the sense that $\bar{a}_i \in \{0, \bar{\alpha}\} \ i \in [1..n]$); it does not see $\bar{a}_i \in \{0, \bar{\alpha}\}$ somehow similar to an integrality constraint (in the sense that $\frac{\bar{a}_i}{\bar{\alpha}}$ is integer). In fact, it does not even have the information that relaxing (lifting) a disjunctive constraint $\bar{a}_i \in \{0, \bar{\alpha}\}$ allows \bar{a}_i to take a fractional value in the interval $[0, \bar{\alpha}]$. As such, `cplex` can not use very evolved branching rules because it does not “realize” that any fractional $\bar{a}_i \in (0, \bar{\alpha})$ is actually a value between two “discrete” bounds 0 and $\bar{\alpha}$. The situation is completely different for the ILP, because `cplex` can use elaborately-tuned branching rules based on evaluating the distance from each fractional variable to its bounds 0 and 1.

³The logs show that `cplex` uses many “zero-half cuts” on the ILP. According to the documentation, these cuts are simply “based on the observation that when the lefthand side of an inequality consists of integral variables and integral coefficients, then the righthand side can be rounded down to produce a zero-half cut.” For instance, it can sum up different constraints to obtain $a_1 + a_2 \leq 3.5$ which is reduced to the zero-half cut $a_1 + a_2 \leq 3$. In theory, the same operations could apply perfectly well on a disjunctive LP and $\bar{a}_1 + \bar{a}_2 \leq 3.5\bar{\alpha}$ could be reduced to $\bar{a}_1 + \bar{a}_2 \leq 3\bar{\alpha}$.

We did attempt to compensate the above advantage of the `cplex` (arsenal of) ILP cuts by adding cuts to the Disjunctive LP, but one should be aware that the `cplex` ILP solver has benefited from decades of experience and research in valid inequalities for ILPs. However, to (try to) reinforce the Disjunctive LP (4.1.2.a)–(4.1.2.e), we propose to insert k -clique inequalities with size $k = 4$ at the root node of the branching tree, *i.e.*, before starting to branch (on the disjunctive constraints). At this root node, all constraints (4.1.2.d) are completely lifted and the problem reduces to a pure LP. This pure LP is solved by **cut generation** as described in Section 4.1.4.2, searching at each iteration to separate the current solution using a k -clique inequality of the form $\sum_{i \in \mathcal{C}} a_i \leq 1$ (associated to a clique \mathcal{C} of size k), equivalent to $\sum_{i \in \mathcal{C}} \bar{a}_i \leq \bar{\alpha}$ in the Charnes-Cooper reformulation. Whenever such cuts are used, the value of k will be indicated in the numerical results (Table 4).

We could have further accelerated the standard **Column Generation** by solving the maximum weight stable problem (at each separation call) using a purely-combinatorial algorithm particularly tuned to this well-studied problem. However, such algorithm would only bring a fortuitous advantage to the standard **Column Generation** but it would remain limited to standard graph coloring, *i.e.*, it would no longer work if one added a single new constraint to the sub-problem, as illustrated by the problem example from the next section.

4.1.3.2 The discrete Charnes-Cooper transformation beyond graph coloring

For the sake of a fair comparison between the **Projective Cutting-Planes** and the standard **Column Generation**, we will solve their respective sub-problems with similar mathematical programming tools, based on **Branch and Bound**. The advantage of these tools is that they can easily extend to address more (diverse) constraints beyond the edge inequalities $a_i + a_j \leq 1 \forall \{i, j\} \in E$. For instance, we can consider the defective coloring problem, in which each vertex is allowed to have a maximum number of $d \geq 0$ neighbors of the same color. When $d = 0$, this problem reduces to standard graph coloring. A mathematical programming method can simply replace the edge inequalities from (4.1.1.a)–(4.1.1.c) with the constraints (4.1.3a) below — notice how such constraint is active only when $a_i = 1$. Applying the discrete Charnes-Cooper transformation, (4.1.3a) could be directly translated to (4.1.3b).

$$n(a_i - 1) + \sum_{\{i,j\} \in E} a_j \leq d \quad \forall i \in [1..n] \quad (4.1.3a)$$

$$n(\bar{a}_i - \bar{\alpha}) + \sum_{\{i,j\} \in E} \bar{a}_j \leq d\bar{\alpha} \quad \forall i \in [1..n] \quad (4.1.3b)$$

An important consequence of the above development concerns the potential extensions and generalizations of this discrete Charnes-Cooper transformation. If the sub-problem had other constraints instead of $\bar{a}_i + \bar{a}_j \leq 1 \forall \{i, j\} \in E$, the transformation (4.1.1.a)–(4.1.1.c) \rightarrow (4.1.2.a)–(4.1.2.e) would have worked in the same manner, *i.e.*, any linear constraint can be reformulated using the Charnes-Cooper transformation, as also exemplified by the above translation (4.1.3a) \rightarrow (4.1.3b). This suggests that the proposed approach could be applied on many **Column Generation** programs in which the columns \mathcal{A} represent the integer solutions of an LP. In such cases, the separation sub-problem reduces to an ILP and the intersection sub-problem can reduce to a Disjunctive LP. Recall this amounts to changing integrality constraints like $a_i \in \{0, 1\}$ into disjunctive constraints of the form $\bar{a}_i \in \{0, \bar{\alpha}\}$, *i.e.*, a similar form of continuity breaking constraints. We focused on standard graph coloring only because it is a problem without complex problem-specific constraints whose presentation could impair the understanding of the more general algorithmic ideas.

4.1.4 The Projection Sub-Problem in the New Coloring Model with RR-Stables

Hereafter, we focus on a second coloring model that defines the columns (or the dual constraints \mathcal{A}) using a broader notion of reinforced relaxed stables (RR-stables), formally introduced by Definition 2 below. In a nutshell, the constraints \mathcal{A} are characterized by $(\mathbf{a}, 1) \in \mathcal{A} \iff \mathbf{a} \in \mathcal{P}$, where \mathcal{P} is an auxiliary polytope of RR-stables, constructed from a set \mathcal{R} of constraints defined by (4.1.5) further below. This Section 4.1.4 is devoted to this second coloring model with RR stables and it is structured as follows:

- We develop the Charnes-Cooper LP (re)formulation of the projection subproblem in Section 4.1.4.1;

- We propose a **cut generation** algorithm to solve the above LP in Section 4.1.4.2;
- Section 4.1.4.3 exposes a special degenerate case (null step length) that can arise in this new model. We will see we need to define $\mathbf{x}_{it} = \alpha(\mathbf{x}_{i-1} + t_{i-1}^* \mathbf{d}_{i-1})$ with $\alpha = 0.9999$ to avoid such degeneracy.

Definition 2 *The reinforced relaxed (RR) stables are the (extreme) solutions of an auxiliary polytope \mathcal{P} representing an outer approximation of the stable set polytope $\text{conv}(\mathcal{P}_{0-1})$. We construct $\mathcal{P} \supseteq \text{conv}(\mathcal{P}_{0-1})$ by adding reinforcing cuts to $\text{conv}(\mathcal{P}_{0-1})$ in several stages. At first, we only simply consider the edge inequalities $a_i + a_j \leq 1 \forall \{i, j\} \in E$, so that the first-stage outer approximation of $\text{conv}(\mathcal{P}_{0-1})$ only contains simple relaxed stables (solutions of the fractional stable set polytope). We then continue building the outer approximation by adding stronger and stronger cuts that become more and more difficult to generate. The more cuts we add, the better the outer approximation \mathcal{P} . However, generating more cuts comes with a greater computational cost and a trade-off has to be found. We will use six cut families briefly presented in Section 4.1.4.2 and then described in full detail in Appendix A.2.*

4.1.4.1 The LP formulation of the projection sub-problem with RR stables

Using the same ideas that led to formulating (4.1.1.a)–(4.1.1.c) in Section 4.1.3 above, the intersection sub-problem $\text{project}(\mathbf{x} \rightarrow \mathbf{d})$ for the coloring model with RR-stables can be written as:

$$t^* = \min \left\{ \frac{1 - \mathbf{x}^\top \mathbf{a}}{\mathbf{d}^\top \mathbf{a}} : \mathbf{a} \in \mathcal{P}, \mathbf{d}^\top \mathbf{a} > 0 \right\}, \quad (4.1.4)$$

where \mathcal{P} from Definition 2 is formalized by (4.1.5) below. As stated in Definition 2, \mathcal{P} is defined by six classes of cuts \mathcal{R} that will be later presented in Section 4.1.4.2.

$$\mathcal{P} = \{ \mathbf{a} \geq \mathbf{0}_n : \mathbf{e}^\top \mathbf{a} \leq 1 \forall (\mathbf{e}, 1) \in \mathcal{R}, \mathbf{f}^\top \mathbf{a} \leq 0 \forall (\mathbf{f}, 0) \in \mathcal{R} \} \quad (4.1.5)$$

This intersection sub-problem (4.1.4) is a linear-fractional program that can be translated to a standard LP using the Charnes–Cooper transformation [3]. More exactly, writing $\bar{\mathbf{a}} = \frac{\mathbf{a}}{\mathbf{d}^\top \mathbf{a}}$ and $\bar{\alpha} = \frac{1}{\mathbf{d}^\top \mathbf{a}}$, one can show that (4.1.4) is completely equivalent to:

$$t^* = \min \bar{\alpha} - \mathbf{x}^\top \bar{\mathbf{a}} \quad (4.1.6a)$$

$$\mathbf{e}^\top \bar{\mathbf{a}} \leq \bar{\alpha}, \mathbf{f}^\top \bar{\mathbf{a}} \leq 0 \quad \forall (\mathbf{e}, 1) \in \mathcal{R}, \forall (\mathbf{f}, 0) \in \mathcal{R} \quad (4.1.6b)$$

$$\mathbf{d}^\top \bar{\mathbf{a}} = 1 \quad (4.1.6c)$$

$$\bar{\mathbf{a}} \geq \mathbf{0}_n, \bar{\alpha} \geq 0 \quad (4.1.6d)$$

The equivalence of the two formulations can be checked by following (and slightly generalizing) the proof of the equivalence between (4.1.1.a)–(4.1.1.c) and (4.1.2.a)–(4.1.2.e) used for the model with standard stables (in Section 4.1.3). Following this proof, the only difference is that the above new programs have a continuous feasible area and we no longer have disjunctive or integrality constraints (of the form $\bar{a}_i \in \{0, \bar{\alpha}\}$ or resp. $a_i \in \{0, 1\}$). Even without these constraints, all feasible solutions of (4.1.6a)–(4.1.6d) still satisfy $\bar{\alpha} \neq 0$, because $\bar{\alpha} = 0$ would also impose $\bar{\mathbf{a}} = \mathbf{0}_n$ — recall \mathcal{R} contains all edge inequalities and they are reformulated in (4.1.6b) under the form $\bar{a}_i + \bar{a}_j \leq \bar{\alpha} \forall \{i, j\} \in E$. However, even if such edge inequalities did not exist, this Charnes-Cooper transformation is very general and it would still work for any constraints (4.1.6b).⁴

⁴For any constraints (4.1.6b), any feasible solution $(\bar{\mathbf{a}}, \bar{\alpha})$ with $\bar{\alpha} = 0$ of the LP (4.1.6a)–(4.1.6d) can always be associated to an extreme ray of feasible solutions in the initial linear-fractional program. More exactly, one can take any $\mathbf{a} \in \mathcal{P}$ and construct a ray $\mathbf{a} + z\bar{\mathbf{a}}$ of \mathcal{P} , *i.e.*, $\mathbf{a} + z\bar{\mathbf{a}}$ is feasible in (4.1.5) for all $z \geq 0$. To check this, notice that $\mathbf{e}^\top (\mathbf{a} + z\bar{\mathbf{a}}) \leq 1 \forall (\mathbf{e}, 1) \in \mathcal{R}$ follows from $\mathbf{a} \in \mathcal{P}$ and $\mathbf{e}^\top \bar{\mathbf{a}} \leq 0 \forall (\mathbf{e}, 1) \in \mathcal{R}$, which holds because $\bar{\alpha} = 0$ in (4.1.6b); a similar argument proves $\mathbf{f}^\top \mathbf{a} \leq 0 \forall (\mathbf{f}, 0) \in \mathcal{R}$. The objective value of $\mathbf{a} + z\bar{\mathbf{a}}$ in (4.1.4) converges to

$$\lim_{z \rightarrow \infty} \frac{1 - \mathbf{x}^\top \mathbf{a} - z\mathbf{x}^\top \bar{\mathbf{a}}}{\mathbf{d}^\top \mathbf{a} + z\mathbf{d}^\top \bar{\mathbf{a}}} = \lim_{z \rightarrow \infty} \frac{-z\mathbf{x}^\top \bar{\mathbf{a}}}{\mathbf{d}^\top \mathbf{a} + z} = -\mathbf{x}^\top \bar{\mathbf{a}}.$$

The above LP (4.1.6a)–(4.1.6d) is solved by **cut generation** because enumerating all reinforcing cuts \mathcal{R} is computationally very exhausting, if not impossible. Notice that these reinforcing cuts \mathcal{R} are slightly modified when they are inserted in the above LP (re-)formulation, *i.e.*, we use $\mathbf{e}^\top \bar{\mathbf{a}} \leq \bar{\alpha}$ in (4.1.6b) instead of $\mathbf{e}^\top \mathbf{a} \leq 1$ as in the \mathcal{P} definition from (4.1.5). However, the difficulty of the separation sub-problem for (4.1.6b) does not depend on the right-hand side $\bar{\alpha}$, but on the structure of the cuts \mathcal{R} . To make the overall **Projective Cutting-Planes** reach its full potential, it is important to have a fast algorithm for the separation sub-problem of (4.1.6a)–(4.1.6d); we next describe the **cut generation** algorithm for this (4.1.6a)–(4.1.6d) LP (re-)formulation of the projection sub-problem.

4.1.4.2 The cut generation for the LP formulation (4.1.6a)–(4.1.6d) of the projection sub-problem

A positive distinguishing characteristic of the above LP formulation (4.1.6a)–(4.1.6d) is that the prohibitively-many reinforcing cuts \mathcal{R} from (4.1.6b) do not depend on \mathbf{x} or \mathbf{d} ; these constraints remain the same along all iterations of the overall **Projective Cutting-Planes**. As such, after solving an intersection sub-problem $\text{project}(\mathbf{x}_{\text{it}} \rightarrow \mathbf{d}_{\text{it}})$ at some iteration it of the overall **Projective Cutting-Planes**, one can keep all generated cuts (4.1.6b) and only update (4.1.6c) to move to the next iteration $\text{it} + 1$.

The **cut generation** algorithm for (4.1.6a)–(4.1.6d) has to separate the reinforcing cuts \mathcal{R} , which are composed of six classes of inequalities (a)–(f) presented in greater detail in Appendix A.2.1. Their design has been inspired by research in valid inequalities for the maximum stable problem [10]. However, our context is different compared to the existing literature: we will need to generate too many clique inequalities (see cuts (f) below) to afford using cliques of arbitrary size, and so, we will only use cliques of bounded size k . However, we use several other cuts besides these clique inequalities (f). The first four cut classes (a)–(d) from \mathcal{R} are actually static and they are inserted in (4.1.6a)–(4.1.6d) at the very first iteration of the overall **Cutting-Planes**; they are then implicitly re-used at all subsequent iterations because they are never removed from (4.1.6a)–(4.1.6d).

Only the cuts (e)–(f) are dynamically generated one by one using a specific **cut generation** algorithm, by repeatedly solving a separation sub-problem. This sub-problem reduces to finding the maximum of $\max_{(\mathbf{e},1) \in \mathcal{R}} \mathbf{e}^\top \bar{\mathbf{a}} - \bar{\alpha}$ and $\max_{(\mathbf{f},0) \in \mathcal{R}} \mathbf{f}^\top \bar{\mathbf{a}}$, for the current optimal solution $(\bar{\mathbf{a}}, \bar{\alpha})$ at each **cut generation** iteration. The **cut generation** algorithm is described in greater detail in Appendix A.2.2. The cuts (e) are referred to as odd hole inequalities in the literature of valid inequalities for the stable set polytope. They can be separated in polynomial time by applying Dijkstra’s algorithm on a bipartite graph with $2n + 2$ vertices.

The class (f) represents k -clique cuts of the form $\sum_{v \in \mathcal{C}} a_v \leq 1$, or $\sum_{v \in \mathcal{C}} \bar{a}_v \leq \bar{\alpha}$ in (4.1.6b), where \mathcal{C} is a clique with k vertices. The separation of these cuts is computationally rather expensive because it requires solving a maximum weight clique problem with bounded size k , which is NP-Hard when k is not a constant parameter. We propose in Appendix A.2.3 a dedicated **Branch & Bound with Bounded Size (BBBS)** algorithm to solve this problem. The repeated call to this algorithm becomes the most important computational bottleneck for the overall **Cutting-Planes**, especially when k is not very small. We also tried to generate the cuts (f) by solving the maximum weight clique problem with no size restriction ($k = \infty$). This is the well-known maximum clique problem for which there exist elaborately-tuned off-the-shelf software (*e.g.*, we used the well-known **Cliquer**, due to S. Niskanen and P. Östergård, see users.aalto.fi/~pat/cliquer.html), but our BBBS algorithm is faster when k is not too large.

The value of k can be used to control a trade-off between speed and efficiency, between the total computation time of the **Projective Cutting-Planes** and the reported optimal value (of the new (4.2) model with RR stables). Experiments confirm that the above maximum weight clique problem with bounded size k can be solved more rapidly (*i.e.*, BBBS becomes faster) when k is lower.⁵ On the other hand, by lowering k , the outer approximation $\mathcal{P} \supseteq \text{conv}(\mathcal{P}_{0-1})$ becomes coarser, in the sense that \mathcal{P} contains more artificial RR stables that are not standard stables. This leads to more artificial constraints in the new (4.2) model, so that the lower bound reported in the end becomes smaller.

⁵This was actually observed both for the BBBS algorithm developed in this work and for the **Cplex** ILP solver applied on the same problem. In theory, a very small k can even be seen as a parameter, so that the maximum weight clique problem with bounded size k is no longer NP-Hard (it becomes polynomial by enumerating all such cliques in $O(n^k)$ time).

On sparser graphs, the resulting **Projective Cutting-Planes** with **RR-stables** is naturally faster than the classical **Column Generation** with standard stables. Sparser graphs have smaller cliques and larger stables, so that the maximum weight clique problem with bounded size (for the **Projective Cutting-Planes**) becomes easier and the maximum weight stable problem (for the standard **Column Generation**) becomes harder.

Remark 4 *The optimum of the **Column Generation** model with **RR stables** can be greater than the maximum clique size ω . This comes from the fact that the cuts (d) from Appendix A.2 can exclude $[\frac{1}{\omega} \frac{1}{\omega} \dots \frac{1}{\omega}]^\top$ from \mathcal{P} . As such, the new **Column Generation** model (4.2) does not necessarily contain a constraint of the form $[\frac{1}{\omega} \frac{1}{\omega} \dots \frac{1}{\omega}]^\top \mathbf{x} \leq 1$, and so, the dual objective function value does not necessarily satisfy $\mathbf{1}_n^\top \mathbf{x} \leq \omega$. Notice that $[\frac{1}{\omega} \frac{1}{\omega} \dots \frac{1}{\omega}]^\top$ could not be excluded from \mathcal{P} by a k -clique cut of class (f) associated to a clique of size $k = \omega$. Experiments suggest that the **Column Generation** model with **RR-stables** can often have an optimal value above k , in part because of these cuts (d). Furthermore, the **Projective Cutting-Planes** could work perfectly well for any (dual) objective function $\mathbf{b} \neq \mathbf{1}_n$, e.g., for the multi-coloring problem proposed at the end of Section 4.1.1. In such a multi-coloring context, ω is no longer be a valid lower bound, while each lower bound $\mathbf{b}^\top \mathbf{x}_{it}$ reported by the **Projective Cutting-Planes** remains perfectly valid. \square*

4.1.4.3 Projecting a boundary point \mathbf{x}_{it} can lead to a null step length t_{it}^* in the model with **RR stables**

For the model with **RR-stables**, the **Projective Cutting-Planes** variant that chooses $\mathbf{x}_{it} = \mathbf{x}_{it-1} + t_{it-1}^* \mathbf{d}_{it-1}$ is not very effective because projecting boundary points is prone to a certain form of degeneracy. More exactly, the projection sub-problem might return $t_{it}^* = 0$ in this case because a boundary point can belong to multiple facets; as such, the projection operation can reduce to finding a new facet that touches \mathbf{x}_{it} . This could make **Projective Cutting-Planes** stagnate like a Simplex algorithm that performs degenerate iterations without improving the objective value (even if new constraints can enter the basis).

Technically, this is explained by the following phenomenon. First, notice that $\mathbf{x}_{it} = \mathbf{x}_{it-1} + t_{it-1}^* \mathbf{d}_{it-1}$ belongs to the first-hit constraint/facet $\mathbf{a}^\top \mathbf{x} \leq 1$ returned by the projection sub-problem at iteration $it - 1$, so that $\mathbf{a}^\top \mathbf{x}_{it} = 1$. Furthermore, the current optimal outer solution $\text{opt}(\mathcal{P}_{it-1})$ also belongs to the above first-hit facet, and so, by taking $\mathbf{d}_{it} = \text{opt}(\mathcal{P}_{it-1}) - \mathbf{x}_{it}$ as indicated by Step 2 from Section 2, we also obtain $\mathbf{a}^\top \mathbf{d}_{it} = 0$. Now recall that \mathbf{a} can be seen as a feasible solution (an **RR-stable**) of the polytope \mathcal{P} from (4.1.5), so that there might exist a continuous set of **RR stables** $\hat{\mathbf{a}} \in \mathcal{P}$ very close to \mathbf{a} . There might exist multiple first-hit facets (of \mathcal{P}) that touch \mathbf{x}_{it} because a part of these $\hat{\mathbf{a}} \in \mathcal{P}$ can satisfy $\hat{\mathbf{a}}^\top \mathbf{x}_{it} = 1$ — recall that $\mathbf{a} = \frac{\bar{\mathbf{a}}}{\alpha}$ is *not* an extreme solution determined by optimizing the LP (4.1.6a)–(4.1.6d) in the direction of \mathbf{x}_{it} . As such, it is often possible to find some $\hat{\mathbf{a}} \in \mathcal{P}$ such that $\hat{\mathbf{a}}^\top \mathbf{x}_{it} = 1$ and $\hat{\mathbf{a}}^\top \mathbf{d} = \epsilon > 0$, which leads to a step length of $t_{it}^* = \frac{1 - \hat{\mathbf{a}}^\top \mathbf{x}_{it}}{\hat{\mathbf{a}}^\top \mathbf{d}_{it}} = \frac{0}{\epsilon} = 0$.

We thus propose to define \mathbf{x}_{it} as a strictly interior point using $\mathbf{x}_{it} = \alpha(\mathbf{x}_{i-1} + t_{i-1}^* \mathbf{d}_{i-1})$ with $\alpha = 0.9999$. This way, the above **RR stables** $\hat{\mathbf{a}}$ very close to \mathbf{a} no longer cause any problem: they lead to $1 - \hat{\mathbf{a}}^\top \mathbf{x}_{it} > 0$ and to a small (“ ϵ -sized”) $\hat{\mathbf{a}}^\top \mathbf{d}_{it}$, so that $\frac{1 - \hat{\mathbf{a}}^\top \mathbf{x}_{it}}{\hat{\mathbf{a}}^\top \mathbf{d}_{it}}$ becomes very large. Generally speaking, this 0.9999 factor is reminiscent of the “fraction-to-the-boundary stepsize factor” used in (some) interior point algorithms to prevent them from touching the boundary — see the parameter $\alpha_0 = 0.99$ in the pseudo-code above Section 3 in [7].

4.2 Multiple-Length Cutting-Stock

4.2.1 The model with prohibitively-many constraints and their separation

Cutting-Stock is one of the most celebrated problems typically solved by **Column Generation**, as first proposed in the pioneering work of Gilmore and Gomory in the 1960s. Given a stock of large standard-size pieces (e.g., of wood or paper), this problem asks to cut these standard pieces into smaller pieces (items) to fulfill a given demand. The pattern-oriented formulation of *Cutting-Stock* consists of a program with prohibitively-many variables, using one variable for each feasible (cutting) pattern. After applying a linear relaxation as in

the case of graph coloring, the dual of this program takes the form of the generic **Column Generation** dual LP (4.2), recalled below for the reader’s convenience.

$$\mathcal{P} \begin{cases} \max \mathbf{b}^\top \mathbf{x} \\ y_a : \mathbf{a}^\top \mathbf{x} \leq c_a, \quad \forall (\mathbf{a}, c_a) \in \mathcal{A} \\ \mathbf{x} \geq \mathbf{0}_n \end{cases} \quad (4.2.1)$$

The notations from (4.2.1) can be directly interpreted in *(Multiple-Length) Cutting-Stock* terms. Each constraint $(\mathbf{a}, c_a) \in \mathcal{A}$ 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 cut from a large standard-size 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. Recalling the primal LP (4.1) corresponding to the above (4.2.1), one can check that the primal objective function asks to minimize the total cost of the selected cutting-patterns. On the account of the industrial origins of the problem, the lengths W and \mathbf{w} are also referred to as (paper) roll widths.

In pure *Cutting-Stock*, all feasible patterns $(\mathbf{a}, c_a) \in \mathcal{A}$ have a fixed unitary cost $c_a = 1$, but we will rather focus on the more general *Multiple-Length Cutting-Stock* in which the large standard-size pieces can have different lengths of different costs. While all discussed algorithms could address an arbitrary number of lengths, we prefer to avoid unessential complication and to generally consider two lengths $0.7W$ and W of costs 0.6 and resp. 1 . The cost of a cutting 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 $c_a = 0.6$, else $c_a = 1$.

Recall that the standard **Column Generation** actually optimizes the above LP (4.2.1) by **Cutting-Planes**, iteratively solving the separation subproblem $\min_{(\mathbf{a}, c_a) \in \mathcal{A}} c_a - \mathbf{a}^\top \mathbf{x}$ on the current optimal outer solution $\mathbf{x} = \text{opt}(\mathcal{P}_{\text{it}})$ 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 a state s_ℓ for each feasible length $\ell \in [1..W]$; all patterns of length ℓ have the same cost c_ℓ and the pattern $\mathbf{a}_\ell \in \mathbb{Z}_+^n$ that minimizes $c_\ell - \mathbf{a}_\ell^\top \mathbf{x}$ gives the objective value of s_ℓ , *i.e.*, $\text{obj}(s_\ell) = c_\ell - \mathbf{a}_\ell^\top \mathbf{x}$. The **Dynamic Programming** scheme generates transitions among such states, and, after calculating them all, returns $\min_{\ell \in [1..W]} c_\ell - \mathbf{a}_\ell^\top \mathbf{x}$ in the end.

4.2.2 The Projective Cutting-Planes for Multiple-Length Cutting-Stock

The **Projective Cutting-Planes** was presented in Section 2 (as a rather generic methodology) and we now need a few customizations to make it reach its full potential on *Multiple-Length Cutting-Stock*. As with other problems explored in this work, a key observation is that defining \mathbf{x}_{it} as the best solution ever found up to iteration it is not efficient in the long run, partly because \mathbf{x}_{it} could fluctuate too much from iteration to iteration (recall also Section 2.1). We will also see in Section 4.2.3.2 that, in *(Multiple-Length) Cutting-Stock*, the projection sub-problem $\text{project}(\mathbf{x} \rightarrow \mathbf{d})$ 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 $i \in [1..n]$.

Based on above observations, we define \mathbf{x}_{it} using the following approach. Let us first introduce the operator $\lfloor \mathbf{x} \rfloor$ that truncates \mathbf{x} down to multiples of some $\gamma \in \mathbb{R}_+$ (we used $\gamma = 0.2$), *i.e.*, x_i becomes $\gamma \cdot \lfloor \frac{1}{\gamma} x_i \rfloor$ for any $i \in [1..n]$. Let \mathbf{x}^{bst} denote the best truncated feasible solution generated up to the current iteration; \mathbf{x}^{bst} can be determined as follows: start with $\mathbf{x}^{\text{bst}} = \mathbf{0}_n$ at iteration $\text{it} = 1$, and replace \mathbf{x}^{bst} with $\lfloor \mathbf{x}_{\text{it}} + t_{\text{it}}^* \mathbf{d}_{\text{it}} \rfloor$ at each iteration $\text{it} > 1$ where $\mathbf{b}^\top \lfloor \mathbf{x}_{\text{it}} + t_{\text{it}}^* \mathbf{d}_{\text{it}} \rfloor > \mathbf{b}^\top \mathbf{x}^{\text{bst}}$. The **Projective Cutting-Planes** chooses the inner solution \mathbf{x}_{it} at each iteration it based on the following rules:

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

We did try to let \mathbf{x}_{it} take the value of the best feasible solution (last pierce point) found up to iteration it , using the formula $\mathbf{x}_{\text{it}} = \mathbf{x}_{\text{it}-1} + t_{\text{it}-1}^* \mathbf{d}_{\text{it}-1}$. This leads to a more “aggressive” **Projective Cutting-Planes**

variant that starts very well by strictly increasing the lower bound at each iteration it , *i.e.*, check that $\mathbf{b}^\top \mathbf{x}_{it} = \mathbf{b}^\top (\mathbf{x}_{it-1} + t_{it-1}^* \mathbf{d}_{it-1}) \geq \mathbf{b}^\top \mathbf{x}_{it-1}$ is surely satisfied because the objective function does not deteriorate by advancing along $\mathbf{x}_{it-1} \rightarrow \mathbf{d}_{it-1}$ (see arguments at Step 2 from Section 2). Accordingly, this **Projective Cutting-Planes** variant brings the advantage that the lower bound $\mathbf{b}^\top \mathbf{x}_{it}$ becomes constantly increasing along the iterations it , eliminating the infamous “yo-yo” effect often appearing in **Column Generation**. However, our preliminary experiments (available on-line, see footnote 10, p. 39) suggest that this aggressive choice leads to more iterations in the long run; furthermore, these iterations are also computationally more expensive because the above \mathbf{x}_{it} is not truncated. For this reasons, we decided not to use this **Projective Cutting-Planes** variant when reporting the main results.

Regarding the iterations $it = 1$ and $it = 2$, let us choose $\mathbf{x}_1 = \mathbf{0}_n$ and $\mathbf{d}_1 = \frac{1}{W} \mathbf{w}$, and resp. $\mathbf{x}_2 = \mathbf{0}_n$ and $\mathbf{d}_2 = \mathbf{b}$. The choice of projecting along $\mathbf{0}_n \rightarrow \frac{1}{W} \mathbf{w}$ at the very first iteration is inspired by research in dual feasible functions for *Cutting-Stock* problems [4], which shows that $\frac{1}{W} \mathbf{w}$ is often a dual-feasible solution (in pure *Cutting-Stock*) of very high quality. The choice at iteration 2 is a standard one, following ideas from Section 2. By solving these two sub-problems, the **Projective Cutting-Planes** also generates a few initial constraints in (4.2.1). To ensure an unbiased comparison, our standard **Column Generation** algorithm also generates such initial constraints in the beginning, *i.e.*, it solves the separation sub-problem on \mathbf{b} and $\frac{1}{W} \mathbf{w}$ before launching the standard iterations.

4.2.3 Solving the Projection Sub-problem

Most **Column Generation** algorithms for cutting and packing problems can use **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**, so can be the projection one.

Given a feasible $\mathbf{x} \in \mathcal{P}$ in (4.2.1) and a direction $\mathbf{d} \in \mathbb{R}^n$, recall that the projection subproblem $\text{project}(\mathbf{x} \rightarrow \mathbf{d})$ asks to minimize (2.2.1). For *Multiple-Length Cutting-Stock*, (2.2.1) is instantiated as follows:

$$t^* = \min \left\{ \frac{f(\mathbf{w}^\top \mathbf{a}) - \mathbf{a}^\top \mathbf{x}}{\mathbf{d}^\top \mathbf{a}} : \mathbf{a} \in \mathbb{Z}_+^n, \mathbf{w}^\top \mathbf{a} \leq W, \mathbf{d}^\top \mathbf{a} > 0 \right\}, \quad (4.2.2)$$

where the function $f : [0, W] \rightarrow \mathbb{R}_+$ maps each $\ell \in [0, W]$ to the cost of the cheapest (shortest) standard-size piece of length at least ℓ available in stock. The DP scheme proposed next can work for any non-decreasing function f , *i.e.*, under the natural assumption that shorter pieces are cheaper than longer pieces. Such functions f can encode many different *Cutting-Stock* variants, like variable-sized bin-packing or elastic cutting stock, as in the examples from [13, §4.1.1].

4.2.3.1 The main DP algorithm, the state definition and the state transitions

We consider a set \mathcal{S}_ℓ of DP states for every feasible length $\ell \in [0..W]$. Each state $\mathbf{s} \in \mathcal{S}_\ell$ is associated to all patterns $\mathbf{a} \in \mathcal{A}$ of length $\mathbf{s}_{\text{len}} = \mathbf{w}^\top \mathbf{a} = \ell$, of cost $\mathbf{s}_c = f(\mathbf{w}^\top \mathbf{a}) - \mathbf{a}^\top \mathbf{x} = f(\ell) - \mathbf{a}^\top \mathbf{x}$ and of profit $\mathbf{s}_p = \mathbf{d}^\top \mathbf{a}$. Using this cost/profit interpretation, we can say that (4.2.2) asks to find a state $s \in \{\mathcal{S}_\ell : \ell \in [0..W]\}$ that minimizes the cost/profit ratio $\text{obj}(\mathbf{s}) = \frac{\mathbf{s}_c}{\mathbf{s}_p}$, *i.e.*, if possible, minimize the cost and maximize the profit.

Notice that any cutting pattern can be associated to a state, although we will see that certain states are dominated and do not need to be recorded. The above cost $\mathbf{s}_c = f(\mathbf{w}^\top \mathbf{a}) - \mathbf{a}^\top \mathbf{x}$ is always non-negative because $\mathbf{x} \in \mathcal{P}$.

The DP algorithm starts out 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 cut up to b_i copies of item i (and there is no use in exceeding the item demand b_i). Each such DP iteration generates transitions from the current states to update other (or produce new) states. A state transition $\mathbf{s} \rightarrow \mathbf{s}'$ associated to an item i leads to a state \mathbf{s}' such that:

- (a) $\mathbf{s}'_{\text{len}} = \mathbf{s}_{\text{len}} + w_i$, *i.e.*, the length simply increases by adding a new item;
- (b) $\mathbf{s}'_p = \mathbf{s}_p + d_i$, *i.e.*, we add the profit of item i ;

- (c) $\mathbf{s}'_c = \mathbf{s}_c + f(\mathbf{s}'_{1\text{en}}) - f(\mathbf{s}_{1\text{en}}) - x_i$, *i.e.*, the term $f(\mathbf{s}'_{1\text{en}}) - f(\mathbf{s}_{1\text{en}})$ updates the cost of the standard-size stock piece from which the pattern is cut, and $-x_i$ comes from the $-\mathbf{a}^\top \mathbf{x}$ term from the cost definition $f(\ell) - \mathbf{a}^\top \mathbf{x}$ presented above.

Algorithm 1 The main Dynamic Programming steps executed for each item i b_i times

1. **for** $\ell = W - w_i$ **to** 0:
 2. **for each** $\mathbf{s} \in \mathcal{S}_\ell$: ▷ for each state with length ℓ
 3. initialize state \mathbf{s}' with $\mathbf{s}'_{1\text{en}} = \ell + w_i$, according to above formula (a)
 4. calculate \mathbf{s}'_p , \mathbf{s}'_c with above formulae (b) and (c)
 5. **if** \mathbf{s}' is not dominated by an existing state in $\mathcal{S}_{\ell+w_i}$ (see Section 4.2.3.2) **then**
 - $\mathcal{S}_{\ell+w_i} \leftarrow \mathcal{S}_{\ell+w_i} \cup \{\mathbf{s}'\}$
 - record the transition $\mathbf{s} \rightarrow \mathbf{s}'$ (to reconstruct an optimal pattern in the end)
-

Algorithm 1 provides the pseudo-code executed for each item $i \in [1..n]$ considered b_i times. In the end, the overall DP algorithm returns the best state ever generated, *i.e.*, $\min \left\{ \text{obj}(\mathbf{s}) = \frac{\mathbf{s}_c}{\mathbf{s}_p} : \mathbf{s} \in \mathcal{S}_\ell, \ell \in [0..W] \right\}$. The most complex operation arises at Step 5, where one needs to check that the new state \mathbf{s}' is not dominated by an existing state in $\mathcal{S}_{\ell+w_i}$ before inserting it in $\mathcal{S}_{\ell+w_i}$. If we only had to solve a separation sub-problem (*i.e.*, $\min \{ \mathbf{s}_c - \mathbf{s}_p : \mathbf{s} \in \mathcal{S}_\ell, \ell \in [0..W] \}$), it would have been enough to make each set \mathcal{S}_ℓ contain a unique state, the one of maximum profit (as all states in \mathcal{S}_ℓ have the same cost). The above pseudo-code would reduce to a DP knapsack algorithm, calculating only the maximum profit for each length $\ell \in [0..W]$.

The projection sub-problem is more difficult because recording a unique state per length is no longer enough. 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 could never happen in a (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.

Hereafter, we focus on how to reduce the number of states recorded in each set \mathcal{S}_ℓ , so as to accelerate the DP algorithm for the projection sub-problem. First, let us show it is enough to record a unique maximum-profit state for each feasible cost value of a state in \mathcal{S}_ℓ . For this, let us consider two states $\mathbf{s}^*, \mathbf{s} \in \mathcal{S}_\ell$ such that $\mathbf{s}^*_c = \mathbf{s}_c$ and $\mathbf{s}^*_p > \mathbf{s}_p$; we will show that \mathbf{s} is dominated and can be ignored. This comes from the fact that any transition(s) applied on such states would lead to the same cost $\mathbf{s}^*_c + \Delta_c = \mathbf{s}_c + \Delta_c > 0$ and to profits $\mathbf{s}^*_p + \Delta_p > \mathbf{s}_p + \Delta_p$; this way, it is easy to check that $\frac{\mathbf{s}^*_c + \Delta_c}{\mathbf{s}^*_p + \Delta_p} < \frac{\mathbf{s}_c + \Delta_c}{\mathbf{s}_p + \Delta_p}$ holds — as long as we are only interested in positive profit states (positive denominators) as requested by the condition $\mathbf{d}^\top \mathbf{a} > 0$ from (4.2.2).

Let us now compare \mathbf{s}^* to a state $\mathbf{s} \in \mathcal{S}_\ell$ of higher cost and no better profit, *i.e.*, such that $\mathbf{s}^*_c < \mathbf{s}_c$ and $\mathbf{s}^*_p \geq \mathbf{s}_p$. Such state \mathbf{s} is also dominated by \mathbf{s}^* because it can only lead via transitions to $\frac{\mathbf{s}^*_c + \Delta_c}{\mathbf{s}^*_p + \Delta_p} < \frac{\mathbf{s}_c + \Delta_c}{\mathbf{s}_p + \Delta_p}$. As such, if a state $\mathbf{s} \in \mathcal{S}_\ell$ has a higher cost than an existing state $\mathbf{s}^* \in \mathcal{S}_\ell$ (*i.e.*, $\mathbf{s}^*_c < \mathbf{s}_c$), then it can only be non-dominated if it has a better profit as well, *i.e.*, only if $\mathbf{s}^*_p < \mathbf{s}_p$. This can be seen as a formalization of a very natural principle “pay a higher cost only when you gain a higher profit”. However, the cost and the profits of all non-dominated states in \mathcal{S}_ℓ can thus be ordered using a relation of the form:

$$c_1 < c_2 < c_3 < \dots \tag{4.2.3a}$$

$$p_1 < p_2 < p_3 < \dots \tag{4.2.3b}$$

4.2.3.2 Reducing the number of states to accelerate the DP

We now propose several ideas to reduce the number of states and to accelerate the DP projection algorithm. First, let us focus on the length of the lists (4.2.3.a)–(4.2.3.b) that have to be recorded 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 0.2 and if we only use truncated solutions \mathbf{x}_{it} such that all components of \mathbf{x}_{it} are also multiples of 0.2, the maximum number of feasible costs values is 6, because 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 belong to $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$. This way, the resulting DP algorithm might often need to record only a few states per length, and so, it is not necessarily significantly slower than a separation DP algorithm recording a unique state per length.

Secondly, we need a fast data structure to manipulate the list of cost/profit pairs satisfying the above conditions (4.2.3.a)–(4.2.3.b). Such a fast data structure aims at speeding-up the projection algorithm by accelerating the following two operations executed by Algorithm 1:

- (i) iterating over all elements of \mathcal{S}_ℓ , as needed by the `for` loop at Step 2;
- (ii) inserting a new state at Step 5 after checking that it is not dominated.

Remark 5 *A list of cost/profit values satisfying (4.2.3.a)–(4.2.3.b) can be seen as a Pareto frontier with two objectives (minimize the cost and maximize the profit). Iterating over the elements of such a frontier for the above operation (i) is relatively simple. On the other hand, achieving a very fast insertion for the above operation (ii) is significantly more difficult because the new state mentioned at (ii) could be dominated. The difficulty comes from the fact that one should **not** scan the whole list to check whether the new state is dominated or not. Furthermore, the insertion of a new non-dominated state can result in the removal of other existing states that become dominated. We propose in Appendix A.3.1 a fast data structure (relying on a self-balancing binary tree) designed to perform such operations as rapidly as possible. \square*

Finally, experiments suggest it is possible to further accelerate the DP scheme (in practice) by sorting 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}^{bst} ever found.

5 Numerical Evaluations

Sections 5.1 and 5.2 provide numerical results for the (minimization) problems discussed in Section 3, in which \mathcal{P} is defined as a primal polytope. Section 5.3 and 5.4 are devoted to the problems from Section 4 in which \mathcal{P} is the dual polytope in a **Column Generation** model with a maximization (dual) objective. We will finish with a brief Section 5.5 explaining why the **Projective Cutting-Planes** can be more successful on certain problems than on others, also investigating why choosing $\mathbf{x}_{\text{it}} = \mathbf{x}_{\text{it}-1} + \alpha t_{\text{it}-1}^* \mathbf{d}_{\text{it}-1}$ with $\alpha < 0.5$ is often better than choosing $\alpha = 1$ (except in graph coloring). The C++ source code for all four considered problems is publicly available on-line at cedric.cnam.fr/~porumbed/projcutplanes/; we refer the reader to (the beginning of) Appendix B for information on the technical configuration (compilation, CPU, memory, etc.).

5.1 A Robust optimization problem

This section compares the standard and the new **Cutting-Planes** on the robust optimization instances from [6], considering $\Gamma \in \{1, 10, 50\}$; these instances originate in the **Netlib** or the **Miplib** libraries. In fact, we discarded all instances that are infeasible for $\Gamma = 50$, since our methods are not designed to find infeasibilities. We also ignored all instances that require less than 5 cuts in *loc. cit.* (i.e., **seba**, **shell** and **woodw**) because they are too small to yield a meaningful comparison. As such, we remain with a test bed of 21 instances with between $n = 1000$ and $n = 15000$ variables, as indicated in Appendix B.1. Recall that the

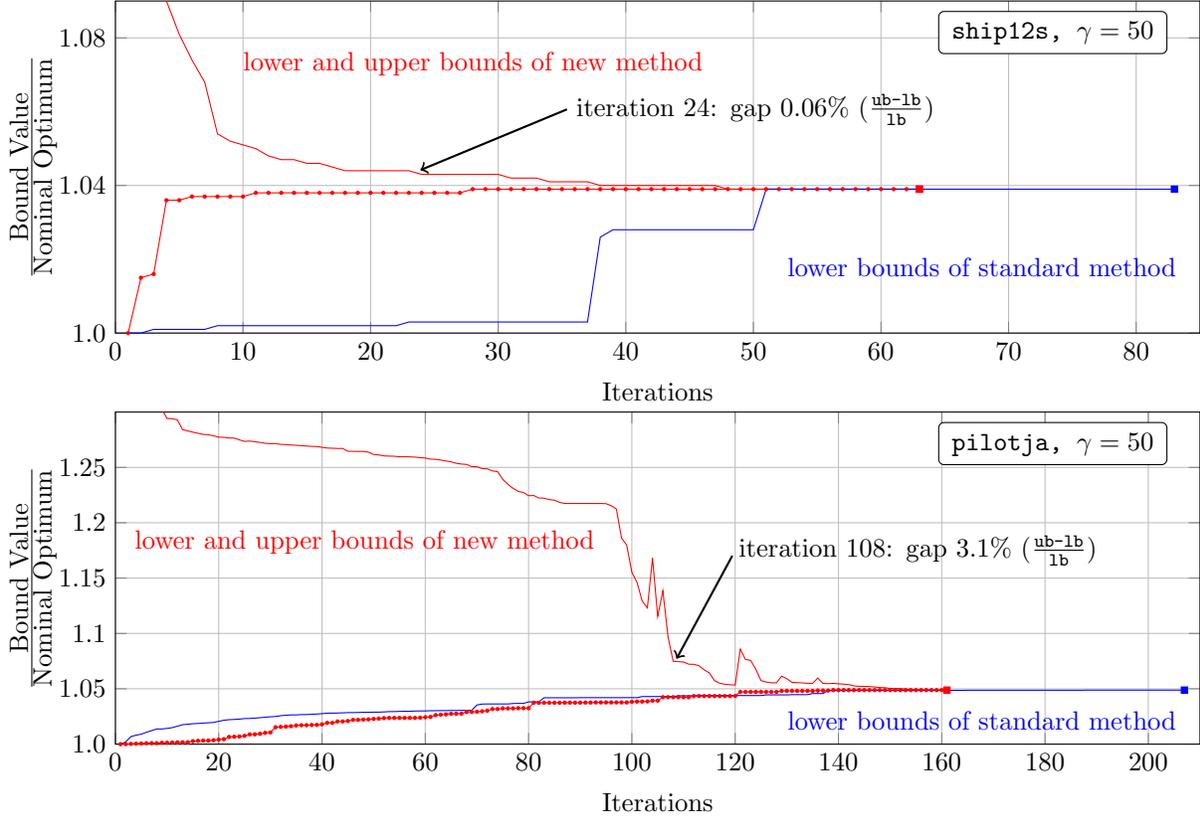


Figure 3: The progress over the iterations of the lower and upper bounds reported by the **Projective Cutting-Planes** (in red), compared to those of the standard **Cutting-Planes** (lower bounds only, in blue).

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

Figure 3 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 and resp. 207 iterations to fully converge on these two instances. After only half this number of iterations, the **Projective Cutting-Planes** reported a feasible solution with a proven low gap of 0.06% or resp. 3.1% — see the arrows at the center of Figure 3.

Table 1 next page compares the total computing effort (iterations and CPU time) needed to fully solve each instance; for the **Projective Cutting-Planes**, this table also indicates the computing effort needed to reach a gap of 1% between the lower and the upper bounds. In average, the new method reduced the total number of iterations by 26% for $\gamma = 50$, by 15% for $\gamma = 10$ and by 7% for $\gamma = 1$.

More important than the total computational effort to fully converge, we notice that the upper bounds generated by the **Projective Cutting-Planes** can be of good quality, even since the beginning of the search in some cases. For example, the standard **Cutting-Planes** needed between one and two hours (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% — remark the columns “gap 1%” in the last row of Table 1, where the time appears in bold. In many practical settings, this could represent a satisfactory feasible solution.⁶

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 basis without improving the objective value. It thus needs significantly more iterations than normally expected — see the figures in bold

⁶It is also true that the robust optimal solution is always within 103% of the nominal optimum. The value of the starting solution \mathbf{x}_1 might be only a few percents higher than the nominal optimum.

Instance	$\gamma = 50$						$\gamma = 10$						$\gamma = 1$					
	OPT (+%)		new method		std. method		OPT (+%)		new method		std. method		OPT (+%)		new method		std. method	
	gap 1% iters	time	full converg. iters	time	full converg. iters	time	gap 1% iters	time	full converg. iters	time	full converg. iters	time	gap 1% iters	time	full converg. iters	time	full converg. 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	147	1.11	149	1.12
bn12	1.847	486	13.66	491	13.81	1927	48.13	1.84	703	20.92	708	21.08	1295	31.31	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	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	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	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	200	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	58	0.647	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	110	1.522	135	1.908
pilotnov	8.51	96	4.227	120	4.615	119	3.714	8.51	103	6.12	139	6.912	141	4.915	94	3.355	120	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	98	0.853	115	1.066
scfmx2	2.114	93	0.387	139	0.584	146	0.537	2.113	101	0.401	152	0.603	150	0.498	88	0.357	131	0.486
scfmx3	2.142	139	0.957	196	1.353	215	1.27	2.141	142	0.955	227	1.575	222	1.309	91	0.605	197	1.352
sctap2	2.844	185	1.946	242	2.567	6545	147.3	2.814	332	3.685	696	8.4	954	10.62	191	2.035	353	3.88
sctap3	3.04	145	2.649	239	4.55	9463	366.1	2.995	180	3.45	773	15.49	1168	20.22	213	3.785	406	7.394
ship081	0.1244	1	0.002	20	0.111	29	0.171	0.1157	1	0.002	19	0.128	23	0.134	1	0.002	19	0.127
ship08s	0.1396	2	0.006	32	0.122	42	0.139	0.129	2	0.006	34	0.134	35	0.123	1	0.001	32	0.123
ship121	0.3528	1	<0.001	48	0.442	65	0.576	0.3462	1	<0.001	48	0.418	65	0.555	1	0.004	45	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	4	0.015	58	0.305
sierra	0.0239	1	0.001	54	0.414	61	0.567	0.0239	1	0.001	54	0.412	61	0.569	1	0.004	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	3	0.054	438	7.573
stocfor3	1.482	29	2.192	3777	2125	4329	2701	1.482	32	1.862	3781	2029	4330	2851	1	0.99	3720	3023

Table 1: Comparison between the Projective Cutting-Planes and the 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). The columns “gap 1%” indicate the computing effort needed to reach the iteration it when the gap between the upper bound $\mathbf{b}^\top \mathbf{x}_{it}$ and the lower bound $\text{optVal}(\mathcal{P}_{it})$ is below 1%, *i.e.*, either $0 < \text{optVal}(\mathcal{P}_{it}) \leq \mathbf{b}^\top \mathbf{x}_{it} \leq 1.01 \text{optVal}(\mathcal{P}_{it})$ or $\text{optVal}(\mathcal{P}_{it}) \leq \mathbf{b}^\top \mathbf{x}_{it} \leq 0.99 \text{optVal}(\mathcal{P}_{it}) < 0$.

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 [6], because their algorithm takes 100 more time for $\Gamma = 50$ than for $\Gamma = 10$, which is unusual.

Remark 6 *Except for the above experiments, the degeneracy issues of the standard **Cutting-Planes** are not very visible in other **Cutting-Planes** implementations from this paper. However, they can generally arise in many problems, and, for instance, **Column Generation** algorithms are often prone to degeneracy phenomena; as [11, §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 Reduces Master Problem”. In **Projective Cutting-Planes**, we can say that the inner-outer solutions \mathbf{x}_{it} and $\text{opt}(\mathcal{P}_{it-1})$ represent together a more “meaningful measure” for selecting a new constraint, avoiding iterations that keep the objective value constant; projecting along $\mathbf{x}_{it} \rightarrow \mathbf{d}_{it}$ does not normally keep the objective value constant (given that $\mathbf{b}^\top \mathbf{d}_{it} > 0$, recall Step 2 from Section 2).*

To summarize, the proposed **Projective Cutting-Planes** brings two advantages at a rather low computational cost: (i) it generates feasible solutions of provable quality along the iterations and (ii) it avoids the degeneracy issues of the standard **Cutting-Planes**. We can not claim that the new algorithm achieves a very spectacular acceleration in terms of iterations or CPU time for full convergence on this problem. However, we will see that it can reduce the number of iterations by factors of 3 or 4 on the Benders decomposition experiments from the next section, as well as on certain graph coloring instances from Section 5.3.1.

5.2 The Benders reformulation

We here evaluate the potential of the intersection sub-problem to improve the standard Benders’ **Cutting-Planes** on the network design problem from Section 3.2.4, reformulated using the model (3.2.8a)–(3.2.8c). In fact, we will first report results on the linear relaxation of this problem, replacing $\mathbf{x} \in \mathbb{Z}_+^n$ from (3.2.8c) with $\mathbf{x} \in \mathbb{R}_+^n$, *i.e.*, this reduces to solving the problem from Remark 3 (p. 14). We use a test bed of 14 existing instances from [14] and 7 new instances (see exact details in Appendix B.2). We consider a bandwidth of $b_{\text{wd}} = 3$ for all instances and the demands have always been generated uniformly at random from an interval $[0, \text{dem}_{\text{max}}]$. We will present statistical results over 10 runs,⁷ reporting the average, the standard deviation and the minimum value of the main performance indicators, *i.e.*, the number of iterations and the CPU time.

Tables 2 and resp. Table 3 compare the new and the standard method on the linear relaxation of (3.2.8a)–(3.2.8c) and resp. on the original integer Benders model (3.2.8a)–(3.2.8c). The first five columns of both tables represent the same instance information: the instance class in Column 1, the instance ID (number) in column 2, the number of edges $n = |E|$ in Column 3, the number of vertices in Column 4 and the maximum demand dem_{max} in Column 5. Column 6 reports either the optimum of the linear relaxation (LP OPT in Table 2) or the optimum of the original integer model (IP OPT in Table 3). In both tables and for both methods, we report statistics on the number of iterations and on the total CPU time in the column groups “avg (std) min”. The columns “Time solve master” report the percentage of CPU time spent on solving master problems along the iterations, *i.e.*, to determine $\text{opt}(\mathcal{P}_1)$, $\text{opt}(\mathcal{P}_2)$, $\text{opt}(\mathcal{P}_3)$, etc.

Table 2 contains an additional Column 7 (Best IP Sol) that reports the best integer upper bound reported by the **Projective Cutting-Planes** along all runs on the LP relaxation. The **Projective Cutting-Planes** determines an *integer* feasible solution at each iteration using Remark 2 (p. 12). We calculated that the LP optimum in Column 6 represents in average 94.5% of the above integer solution from Column 7, *i.e.*, the average integrality gap is 5.5%. If we only consider the last 9 instances with $\text{dem}_{\text{max}} \geq 100$, the average integrality gap is below 1%. These instances are so large that we are skeptical their integer optimum could be found using (the new or the standard) **Cutting-Planes**. As such, one could attempt to tackle them by **Branch and Bound**, solving the above LP relaxation at the root node of the branching tree. It well-known that the effectiveness of a **Branch and Bound** substantially depends on the gap at this root node. For the above 9 largest instances, we can say the **Projective Cutting-Planes** reports a gap below 1% at the root

⁷By default, the Benders algorithms from Section 3.2 have no random component. However, we could randomize them by inserting 10 random cut-set constraints in the beginning of the solution process, as in the experimental section of [14].

Graph class	Instance				Projective Cutting-Planes				Standard Cutting-Planes				
	ID	$ E $	$ V $	α_{\max}	LP OPT	Best IP	Sol	Iterations avg (std) min	Time total [secs] avg (std) min	Time solve master	Iterations avg (std) min	Time total [secs] avg (std) min	Time solve master
a	1	100	20	10	42.333	48	22.8 (1) 22	0.06 (0.002) 0.06	0.06 (0.01) 0.07	4.4%	35 (4.9) 31	0.09 (0.01) 0.07	5.5%
	1	105	60	10	245.67	265	73.8 (2.7) 72	0.2 (0.006) 0.2	0.2 (0.04) 0.3	6.1%	131 (11.8) 115	0.4 (0.04) 0.3	8.7%
	1	110	50	10	204.33	220	56.5 (1.5) 56	0.2 (0.004) 0.2	0.2 (0.05) 0.2	4.9%	78.5 (16) 68	0.2 (0.05) 0.2	5.8%
	1	120	60	10	299.33	317	67.5 (3) 63	0.2 (0.01) 0.2	0.4 (0.02) 0.4	4.3%	104 (4.3) 101	0.4 (0.02) 0.4	6.1%
	1	120	30	10	67.333	77	35.4 (0.8) 35	0.1 (0.006) 0.1	0.1 (0.02) 0.1	4.2%	39.5 (5.5) 34	0.1 (0.02) 0.1	5.5%
	1	600	90	10	281.33	306	150 (35.3) 121	6.2 (1.6) 4.8	6.2 (1.1) 6.3	2.5%	251 (28.8) 199	8.1 (1.1) 6.3	3.5%
	1	1000	100	10	284.33	312	147 (52.1) 107	14.3 (6.1) 9.7	23.4 (1.9) 20.2	1.3%	310 (19.6) 278	23.4 (1.9) 20.2	3.3%
	rnd-10	1	70	25	10	81.667	89	22.3 (0.5) 22	0.04 (<0.001) 0.04	0.04 (<0.001) 0.04	5.3%	24 (0) 24	0.04 (<0.001) 0.04
2		70	25	10	64.667	74	19 (0) 19	0.04 (<0.001) 0.04	0.04 (<0.001) 0.04	4.8%	25 (0) 25	0.04 (<0.001) 0.04	5.9%
1		80	30	10	94	102	32.2 (1.5) 31	0.07 (0.002) 0.07	0.07 (0.002) 0.07	5.6%	35.2 (1.5) 34	0.07 (0.002) 0.07	6.9%
2		80	30	10	82	91	29.5 (1.5) 29	0.06 (0.003) 0.06	0.07 (0.003) 0.07	5.1%	37.6 (1.2) 37	0.07 (0.003) 0.07	6.3%
rnd-100	1	90	35	10	124.67	137	43.4 (1.2) 43	0.1 (0.003) 0.1	0.2 (0.02) 0.2	5.8%	78.5 (7.5) 71	0.2 (0.02) 0.2	7.4%
	1	600	90	100	2918.7	2946	153 (20.7) 129	6.2 (1) 5	9.3 (0.9) 7.9	2.2%	278 (27.7) 242	9.3 (0.9) 7.9	4.5%
	2	600	90	100	2782	2810	141 (37.4) 113	5.5 (1.7) 4.3	7.7 (0.7) 6.9	2.4%	229 (17.9) 208	7.7 (0.7) 6.9	3.6%
	1	1000	110	100	3414.7	3452	229 (70.2) 122	23.5 (8.1) 11.5	28.8 (2.3) 23.7	2%	360 (23) 306	28.8 (2.3) 23.7	4.2%
rnd-300	1	1500	130	100	3922	3956	382 (182) 193	19.3 (8.2) 12.6	30.6 (3.7) 22.2	2.3%	389 (38.2) 313	30.6 (3.7) 22.2	3.6%
	2	1500	130	100	4033.7	4073	282 (77.4) 190	89.6 (45.9) 41.1	74.8 (4.7) 65.9	3%	466 (20.8) 430	74.8 (4.7) 65.9	2.9%
	1	2000	150	100	4638	4684	259 (106) 166	103 (50.7) 59.1	90.4 (7.8) 74.4	2.2%	536 (42.5) 446	90.4 (7.8) 74.4	4.2%
	2	2000	150	300	13314	13365	302 (187) 164	122 (92.6) 58.1	215 (29.1) 175	1%	744 (65.8) 638	215 (29.1) 175	4.9%
rnd-300	2	2000	150	300	13358	13404	295 (172) 168	115 (76.6) 57.5	202 (14.7) 184	1.9%	688 (74.8) 634	202 (14.7) 184	4.6%

Table 2: Statistical comparison of the Projective Cutting-Planes and the standard Cutting-Planes on the *linear relaxation* of the Benders reformulation (3.2.8a)–(3.2.8c), reporting averages over ten runs. Recall (Appendix B.2) that the instances **rnd-100** and **rnd-300** correspond to the instances **random-100-bnd3** and **random-300-bnd3** from Table 4 of [14]. The stabilized Cutting-Planes reported in [14, Table 4] the following numbers of iterations for the seven **rnd-100** instances: 235, 224, 320, 328, 408, 529, 563; for the two **rnd-300** instances, it reported 537 and 545 iterations. The number of iterations reported by Projective Cutting-Planes in this table (last 9 row) is systematically better.

Graph class	Instance			Projective Cutting-Planes				Standard Cutting-Planes			
	ID	$ E $	$ V $	$\overline{\text{dep}}_{\text{max}}$	IP OPT	Iterations avg (std) min	Time total [secs] avg (std) min	Time solve master	Iterations avg (std) min	Time total [secs] avg (std) min	Time solve master
a	1	100	20	10	46	174 (27.4) 141	7.4 (5.8) 3.3	89.5%	229 (44.6) 146	9.6 (3) 4.6	95%
	1	105	60	10	260	824 (206) 464	1073 (636) 379	99.5%	2987 (375) 2427	4129 (819) 2971	99.8%
	1	110	50	10	214	242^{-1} (27.1) 201	99 (31.6) 40.2	98.4%	526 (58.6) 442	378 (70.8) 284	99.6%
	1	120	60	10	313	336 (53.4) 251	321 (103) 156	99.2%	1315 (184) 1049	2367 (469) 1837	99.8%
	1	120	30	10	74	1336 (138) 1020	4907 (1640) 2086	99.8%	2250 (450) 1292	6703 (2857) 2450	99.9%
b	1	70	25	10	87	102 (11.6) 81	11.7 (5.7) 2.4	97.2%	138 (15.7) 111	12.7 (6.9) 3.2	98.4%
	2	70	25	10	72	154 (31) 121	24 (16.8) 5.9	98%	182 (30.5) 142	38.3 (19.3) 17	99.3%
	1	80	30	10	100	188 (37.8) 126	65.7 (28) 15.6	98.9%	368 (87.9) 236	183 (87.8) 45.2	99.6%
	2	80	30	10	88	214 (43.6) 105	96.5 (33.1) 60.4	99.1%	369 (58.4) 317	156 (50.4) 93.5	99.5%
	1	90	35	10	134	722^{-5} (65.9) 664	790 (83) 629	99.5%	1962^{-2} (188) 1748	2314 (365) 1803	99.8%

Table 3: Statistical comparison of the Projective Cutting-Planes and the standard Cutting-Planes on the original integer Benders model (3.2.8a)–(3.2.8c), reporting average results over ten runs. We used the smallest Benders instances from Table 2, because we no longer solve the linear relaxation, but the integer problem which is far more difficult.

Comparing Columns 7 and 14 (labelled “avg”), we notice the new method can require roughly 4 times less iterations (row d), 3 times less iterations (row b) or 2 times less iterations (ante-penultimate row) than the standard method. Such reductions (factors of 3 or 4) can not be achieved by applying enhancement techniques (or stabilization) on the standard Cutting-Planes. The last rows of Table 2 from [14] show that such enhancement techniques could lead to, respectively, 116, 165, 276, 244 or 1128 iterations for the last five rows above, *i.e.*, they could reduce the average number of iterations by a factor between 1.2 and 1.75.

The notation $^{-\kappa}$ in Columns 7 or 14 indicates that κ runs out of 10 failed due to numerical difficulties, as described in Appendix A.1.

node; thus the integration in a **Branch and Bound** seems promising, but this lies outside the scope of this paper.

We now turn our attention to the total computing effort needed to fully converge to the LP optimum. In Table 2, the average number of iterations of the **Projective Cutting-Planes** is often better than the best number of iterations of the standard **Cutting-Planes**. The new method can roughly reduce the (average) number of iterations by a factor of almost 3 (row 3 from bottom to top), or by a factor of about 2 for roughly a third of the instances; the average reduction is $\frac{2}{3}$, *i.e.*, the new method requires $\frac{1}{3}$ fewer iterations in average. This shows there is no need for a statistical test to confirm there is a statistically significant difference between the numbers of iterations reported by the two methods. The average reduction of the CPU time is however only $\frac{3}{4}$ because the projection sub-problem is computationally (slightly) more expensive than the separation one.

Regarding the original integer Benders problem, Table 3 confirms that both the separation and the projection sub-problem can be solved significantly faster than an ILP master problem. The columns “Time solve master” indicate that 97% of the total running time is spent on solving ILP master problems, which clearly become the major computational bottleneck of the (standard or new) **Cutting-Planes**. This confirms theoretical expectations (recall Section 3.2.2) since both sub-problems are reformulated as a pure LP which can be solved significantly faster than a (master) ILP.

Table 3 suggests that, for the original integer Benders model, the **Projective Cutting-Planes** can reduce the average number of iterations by factors of 3 or 4 (*e.g.*, see instances **b** and **d**). Such a performance could not be achieved by simply improving or stabilizing the standard **Cutting-Planes** — see comparative results in the caption of Table 3. The new method can also halve the average running time on four instances out of ten (see rows 2, 3, 4, or 7), although it can also fail to solve two instances with a 100% success rate. The running time is not perfectly proportional to the number of iterations because the structure of the ILP master problems generated along the iterations can be very different from method to method, from instance to instance.

5.3 Graph Coloring

This section is devoted to two different experimental comparisons on two different coloring models.

- Section 5.3.1 compares the **Projective Cutting-Planes** and the classical **Column Generation** on the original graph coloring model with (columns defined by) standard stables, *i.e.*, each dual constraint $\mathbf{a}^\top \mathbf{x} \leq 1$ in \mathcal{A} is associated to the incidence vector \mathbf{a} of a standard stable from the considered graph. The projection sub-problem algorithm for this model was described in Section 4.1.3.
- In Section 5.3.2 we will apply the **Projective Cutting-Planes** on a second coloring model using (columns defined by) a broader notion of reinforced relaxed (RR) stables from Definition 2 (p. 20). The classical **Column Generation** will always be applied on the original coloring model. We will thus compare the new **Projective Cutting-Planes** variant working on the second coloring model with the classical **Column Generation** that uses standard stables. The projection sub-problem is solved as described in Section 4.1.4.

We use 15 coloring instances generated during the second DIMACS implementation challenge (in the 1990s) and widely-used afterwards for benchmarking many different coloring algorithms. This test bed contains random graphs, random geometrical graphs or particular graphs with hidden cliques as described in Appendix B.3.

5.3.1 The Projective Cutting-Planes and the Column Generation on standard graph coloring

We here consider the **Projective Cutting-Planes** from Section 4.1.2. The projection sub-problem is formulated (in Section 4.1.3) using the Disjunctive LP (4.1.2.a)–(4.1.2.e), that was obtained via a discrete Charnes–Cooper reformulation. Recall that the separation sub-problem of the standard **Column Generation** algorithm reduces to the maximum stable problem, that can be formulated as an ILP over the set \mathcal{P}_{0-1} of standard stables from (4.1.1.c). We prefer to solve both the (separation) ILP and the (projection) Disjunctive LP with similar mathematical programming tools based on **Branch and Bound**, as provided in the `cplex`

software package. Had we used two (very) different methods for these sub-problems, we would have skewed the results in the favor of the most refined of the two methods.

To have an additional marker for evaluating the quality of the reported lower bounds, we will also refer to an upper bound determined from a heuristic coloring.⁸ As indicated in Section 4.1.2, this heuristic coloring is also used to construct an initial direction \mathbf{d}_1 and a set of initial constraints $\mathcal{A}_0 \subsetneq \mathcal{A}$; all these initial constraints take the form $\mathbf{a}^\top \mathbf{x} \leq 1$, where $\mathbf{a} \in \mathbb{Z}_+^n$ is the incidence vector of a stable from the heuristic coloring. While the **Projective Cutting-Planes** starts out by solving $\text{project}(\mathbf{0}_n \rightarrow \mathbf{d}_1)$ with \mathbf{d}_1 defined as above, the standard **Column Generation** starts out by separating the same \mathbf{d}_1 at the first iteration, so as to ensure similar starting conditions for the two methods.

By instantiating (4.1.1.a)–(4.1.1.c), the first projection sub-problem can be written in the form $t_1^* = \min \left\{ \frac{1}{\mathbf{d}_1^\top \mathbf{a}} : \mathbf{a} \in \mathcal{P}_{0-1}, \mathbf{d}_1^\top \mathbf{a} > 0 \right\}$, where recall \mathcal{P}_{0-1} is the set of standard stables from (4.1.1.c). The first pierce point is $t_1^* \cdot \mathbf{d}_1$ and the associated first lower bound is $\mathbf{b}^\top (t_1^* \cdot \mathbf{d}_1) = t_1^* \cdot \mathbf{1}_n^\top \mathbf{d}_1$, equivalent to:

$$\frac{\mathbf{1}_n^\top \mathbf{d}_1}{\max \{ \mathbf{d}_1^\top \mathbf{a} : \mathbf{a} \in \mathcal{P}_{0-1} \}} \quad (5.3.1)$$

The standard **Column Generation** method relies on the separation sub-problem $\min \{ 1 - \mathbf{x}^\top \mathbf{a} : \mathbf{a} \in \mathcal{P}_{0-1} \}$, where \mathbf{x} is the current optimal outer solution $\text{opt}(\mathcal{P}_{it})$ at iteration it . A well-known Lagrangean bound for problems with $c_a = 1 \ \forall (\mathbf{a}, c_a) \in \mathcal{A}$ is the Farley bound (A.3.1) from Appendix A.3.2 — see also [1, § 2.2], [16, § 3.2] or [11, § 2.1] for interesting descriptions and proofs. Instantiating the notations from (A.3.1) to our problem, this first bound becomes $\frac{\mathbf{1}_n^\top \mathbf{x}}{1 - \min \{ 1 - \mathbf{x}^\top \mathbf{a} : \mathbf{a} \in \mathcal{P}_{0-1} \}}$; for $\mathbf{x} = \mathbf{d}_1$, this reduces to

$\frac{\mathbf{1}_n^\top \mathbf{d}_1}{\max \{ \mathbf{d}_1^\top \mathbf{a} : \mathbf{a} \in \mathcal{P}_{0-1} \}}$. Thus, the very first lower bound reported by the standard **Column Generation** is equal to the first bound (5.3.1) of the **Projective Cutting-Planes**.

Figure 4 depicts the progress over the (first 150) iterations of the lower bounds reported by the **Projective Cutting-Planes** compared to those of the standard **Column Generation**. As expected from the above theoretical arguments, the two methods start from the same lower bound (5.3.1) at the very first iteration. However, the lower bounds of the **Projective Cutting-Planes** increase monotonically, while those of the standard **Column Generation** method demonstrate the “yo-yo” effect. This (infamous) effect is due to the strong oscillations of the optimal solutions $\text{opt}(\mathcal{P}_{it})$ along the **Column Generation** iterations it (which is also referred to as the “bang-bang” behavior). By stabilizing the **Column Generation**, one can reduce such effects, but we are not aware of any other **Column Generation** work in which the “yo-yo” effect could be completely eliminated.

The **Projective Cutting-Planes** from this section eliminates this “yo-yo” effect completely, mainly because each current inner solution $\mathbf{x}_{it} = \mathbf{x}_{it-1} + t_{it-1}^* \mathbf{d}_{it-1}$ is better than the previous one \mathbf{x}_{it-1} . Indeed, we have $\mathbf{b}^\top \mathbf{x}_{it} > \mathbf{b}^\top \mathbf{x}_{it-1}$ since the objective value can not decrease by advancing along $\mathbf{x}_{it-1} \rightarrow \mathbf{d}_{it-1}$, *i.e.*, a projection can only improve the objective value, based on the arguments indicated at Step 2 from Section 2.

Table 4 reports three lower bounds determined by the classical **Column Generation** (Columns 2-4), followed by 3 bounds of the **Projective Cutting-Planes** (last three columns). In loose terms, for both methods, the three reported bounds respectively correspond to the beginning (Columns 2 and 6), to a mid-point (Columns 3 and 7) and to the to end (Columns 4 and 8) of the solution process. In fact, we tried to make Columns 2 and 6, 3 and 7, and resp. 4 and 8 correspond to equal rounded-up bound values; this explains why Column 2 might actually report more than 100 iterations, *i.e.*, the **Column Generation** might need hundreds of iterations to reach the bound value obtained by **Projective Cutting-Planes** in a dozen of iterations (in Column 6). For each bound, we indicate the number of iterations **iter** needed to reach it, the bound value **lb** and the CPU time **tm** in seconds. A digit in Column 5 indicates that we used k -clique inequalities to accelerate the projection sub-problem algorithm (see Section 4.1.3.1). We excluded graphs

⁸Such feasible colorings have been determined by heuristic algorithms developed in our previous work on graph coloring (almost a decade ago). They are publicly available on-line at cedric.cnam.fr/~porumbed/graphs/evodiv/ or cedric.cnam.fr/~porumbed/graphs/tsdivint/. The upper bound value for each instance is provided in Column 4 of Table 5.

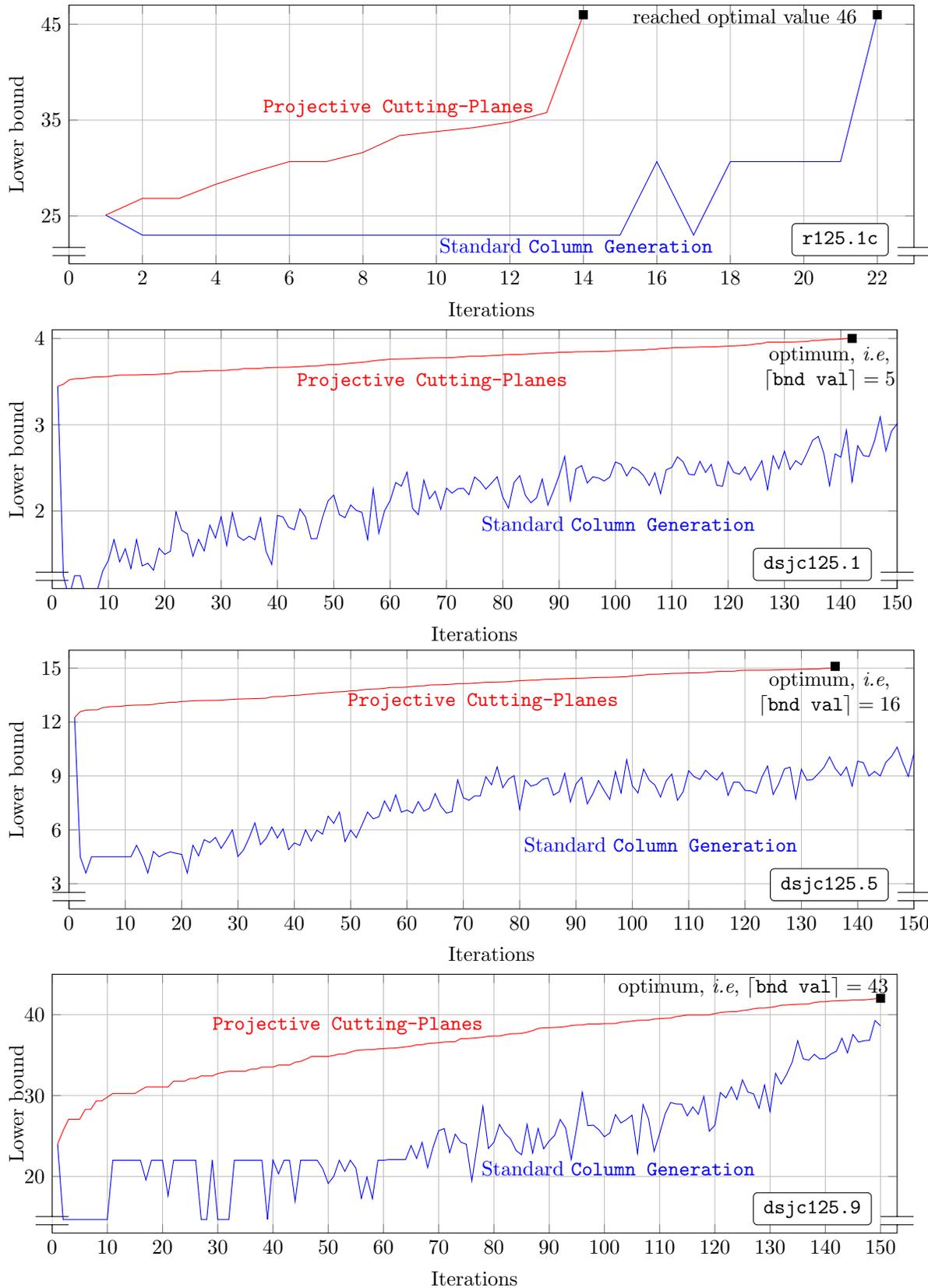


Figure 4: The running profile (lower bounds) of the Projective Cutting-Planes and of the classical Column Generation on 4 standard coloring instances. The new method does *not* exhibit the infamous “yo-yo” effect arising in most (if not all) classical Column Generation algorithms.

Instance	Classical Column Generation			clique cut sz.	Projective Cutting-Planes		
	beginning	mid iter	last iter		beginning	mid iter	last iter
	iter:lb/tm	iter:lb/tm	iter:lb/tm		k	iter:lb/tm	iter:ca lb/tm
dsjc125.1	283:3.44/29.2	380:3.8/53.8	544:4.01/135.6	4	3:3.52/33.0	78:3.80/1225	142:4.01/2809
dsjc125.5	253:13.08/365	306:14.27/634	378:15.08/1101	—	16:13.04/213	62:14.001/1288	136:15.003/4077
dsjc125.9	70:25.67/24.4	134:34.13/78	171:42.11/136	—	2:25.67/7.3	44:34.11/109	150:42.03/486
dsjc250.9	254:51.6/2221	275:54.98/2702	437:70.09/7757	—	5:51.06/487	34:54.01/3013	67:70.01/50185
r125.1	34:2.35/0.06	36:2.62/0.06	47:5/0.08	4*	5:2.35/0.18	17:2.61/0.68	20:5/0.80
r125.1c	15:25.09/8.18	17:30.06/8.45	22:46/9.8	—	2:26.83/4.7	6:30.06/10.4	14:46/21.6
r125.5	82:21.39/7.3	100:24.59/9.2	121:36/11.2	4*	3:21.21/4.2	67:24.01/74.3	116:36/136.7

Table 4: The **Projective Cutting-Planes** compared to the classical **Column Generation** on all standard graph coloring instances that could be solved *by either method* in less than 10000 seconds.

* For these graphs, we added the cuts (c) from Appendix A.2.1. We also protect the algorithm from generating zero step lengths and stagnating: if the intersection sub-problem returns $t_{it}^* < 10^{-6}$ at some iteration it , we switch to a new formula to determine future inner solutions: $\mathbf{x}_{it} = 0.99(\mathbf{x}_{it-1} + t^*_{it-1} \mathbf{d}_{it-1})$. We thus avoid the degeneracy-like issues from Section 4.1.4.3.

like `1e450_25c`, `1e450_25d`, `1e450_15c`, `1e450_15d`, `dsjc500.1`, simply because they require more than 10000 seconds for both methods, at least when the sub-problem is solved by `cplex` (see also Remark 7, p. 38).

The first conclusion that can be drawn from Table 4 is the following. For half of the instances, the classical **Column Generation** might need hundreds of iterations to reach the lower bounds generated by **Projective Cutting-Planes** in less than 20 iterations (compare Columns 2 and 6 labeled “beginning”). This mainly comes from the fact that the lower bounds of the **Projective Cutting-Planes** start on a rapidly increasing slope. Regarding the final convergence, the **Projective Cutting-Planes** is systematically faster in terms of iterations, up to reducing the number of iterations to half (`dsjc125.5`, `r125.1`) or to less than a third (on `dsjc125.1`). However, it can be slower in terms of absolute CPU times. As discussed in Section 4.1.3.1, this is mostly due to the speed of the sub-problem solvers provided by the `cplex` software package: the solver for the (separation) ILP is faster than one for the (projection) Disjunctive LP. By changing these solvers, one could easily obtain very different results in terms of CPU time.

More generally, although the sub-problem algorithm can be seen as a black-box component in the overall design, the relative CPU performance of all discussed coloring algorithms depends substantially on the running time of this sub-problem algorithm. Focusing only on the standard **Column Generation**, the total running time can completely change if we replace the current `cplex` pricing (Table 4) by a BBBS pricing (Table 5 in Section 5.3.2). On the low-density graph `dsjc125.1`, the **Column Generation** with `cplex` pricing needs 135 seconds, while the BBBS version needs 5431 seconds. The situation is inverted on a high-density graph like `dsjc125.9`: the `cplex` version needed 136 seconds and the BBBS version needed 1.61 seconds. Similar phenomena arise for the **Projective Cutting-Planes**; for instance, we could double the running time for `dsjc125.1` by simply changing the implementation of $\bar{a}_i \in \{0, \bar{\alpha}\}$ from “ $\bar{a}_i \leq 0$ or $\bar{a}_i \geq \bar{\alpha}$ ” to “ $\bar{a}_i = 0$ or $\bar{a}_i = \bar{\alpha}$ ” — both these equivalent constraints are implemented as logical “or” constraints in `cplex`.

Further research could explore other algorithms specifically devoted to this class of Disjunctive LPs, as needed by the projection sub-problem. Based on the arguments from Section 4.1.3.1, we see no in-depth reason why a **Branch and Bound** algorithm for such a Disjunctive LP should always be fundamentally slower in absolute terms than any **Branch and Bound** algorithm for the associated ILP.

The proposed approach beyond standard graph coloring

As described in Section 4.1.3.2, the discrete Charnes-Cooper transformation has the advantage that it can actually work on (numerous) problems in which the separation sub-problem can be expressed as an ILP. The reformulation of the edge inequalities used in graph coloring can well apply to other inequalities. To give *only one example*, if instead of the edge inequalities $a_i + a_j \leq 1 \forall \{i, j\} \in E$ we consider defective coloring inequalities (4.1.3a), the discrete Charnes-Cooper can perform the translation (4.1.3a) \rightarrow (4.1.3b). We here only present a brief experiment. Figure 5 plots the lower bounds generated by the two methods along the first (150) iterations on two defective coloring instances. This figure confirms the trends observed on Figure 4

on standard graph coloring.

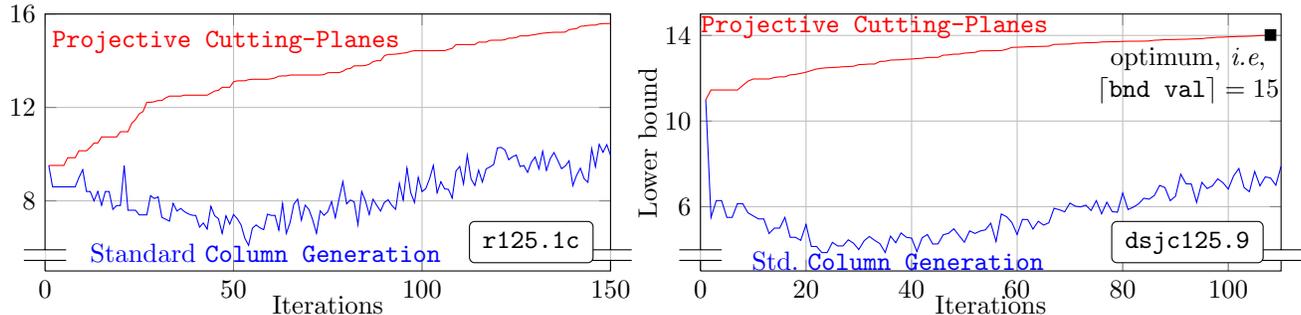


Figure 5: The running profile (lower bounds) of the **Projective Cutting-Planes** and of the **standard Column Generation** on two instances of the defective coloring problem, allowing $d = 3$ neighbors of the same color (recall Section 4.1.3.2).

5.3.2 The new method with RR-stables against the standard method with standard stables

We now focus on a second coloring model in which the constraints \mathcal{A} from (4.2) are no longer associated to standard stables but to a new, broader, notion of RR-stables. Accordingly, the constraints \mathcal{A} are defined by the extreme solutions of the polytope \mathcal{P} from Definition 2 (p. 20), see also (4.1.5). Since \mathcal{P} does contain all original stables, any feasible solution of this second (4.2) model with RR-stables is also feasible for the original (4.2) model; as such, any lower bound for the new model is also a lower bound for the original model. We will compare the lower bounds reported by the **Projective Cutting-Planes** on this second model to the Lagrangian bounds reported by the **standard Column Generation** on the original model. We will see (Remark 7) that the **Projective Cutting-Planes** can find in less than one hour certain lower bounds that the **standard Column Generation** may not reach in days of computations (on the original model).

The projection sub-problem now reduces to the pure LP (4.1.6a)–(4.1.6d) obtained via the continuous Charnes–Cooper transformation as described in Section 4.1.4.1. This LP is solved by **cut generation** as indicated in Section 4.1.4.2. We will report results for two values of the parameter k that controls the size of the k -cliques used to generate reinforcing cuts (f) when constructing the polytope \mathcal{P} (by **cut generation**). Confirming theoretical arguments from Section 4.1.4.2, a higher k generates stronger lower bounds but at the expense of a lower speed.

For the sake of an unbiased comparison, we always prefer to solve the intersection and the separation sub-problems with similar techniques. In Section 5.3.1, both sub-problems have been solved with mathematical programming tools (based on **Branch and Bound** and continuous relaxations), as provided by the **cplex** software package. In the current section, we determine the maximum weight stables (for the separation sub-problem) using the same **Branch & Bound with Bounded Size (BBBS)** algorithm used by the **Projective Cutting-Planes** to find k -cliques when constructing \mathcal{P} . For the **standard Column Generation**, the maximum stable of the considered graph (Column 5 in Table 5) is given as input to the BBBS algorithm, *i.e.*, it constitutes the bounded size given as input to BBBS.

Table 5 compares the standard and the new method, placing a special emphasis on three lower bounds reported along the iterations. The first four columns describe the instance: the density in Column 1, the graph name in Column 2, the number of vertices in Column 3 and the heuristic upper bound in Column 4. Columns 6–8 report three lower bounds determined by the **standard Column Generation** along the iterations (each table cell in these columns indicates the bound value and the required CPU time). For the **Projective Cutting-Planes**, we consider in Column 9 two values of the parameter k used to generate k -cliques to construct \mathcal{P} . Columns 10–12 provide three lower bounds of the **Projective Cutting-Planes** in the same format as in the Columns 6–8.

The last column of Table 5 reports the result of a final additional iteration: take the last pierce point reported by the **Projective Cutting-Planes** with RR-stables (next-to-last column), multiply it with $\alpha = 0.9999$ (for the reasons indicated in Section 4.1.4.3), and project it towards $\mathbf{1}_n$ in the original model with

Density	Instance	n	Upper bound	Optimal		Standard Column Generation with standard stables		Projective Cutting-Planes with RR stables				one last iter with std. stab. [lb]/tm
				max stable	lb/tm	lb/tm	lb/tm	clq size	k	lb/tm	lb/tm	
0.094	dsjc125.1	125	5	35	3.45/1.5	3.52/760	4.01/5431	3	2.95/0.43	3.01/2.77	4/15.13	4/21.45
0.17	1e450_25c	450	25	47	2.5/5.03	5.2/195	6.87/2535	4	2.95/0.53	3.01/1.22	4/15.51	5/21.06
0.17	1e450_25d	450	25	43	2.77/5.26	5.45/409	6.15/2440	5	9.11/2.34	10.01/41	25/1242	optim ended
0.24	p_hat300-1	300	19	39	2.11/3.4	7.36/206	10.19/5418	25	9.11/2.28	10.01/43	25/1412	optim ended
0.50	dsjc125.5	125	18	35	4.5/2.25	10.01/2.44	15.06/4.88	5	10.21/3.28	13.01/224	25/1364	optim ended
0.90	dsjc125.9	125	44	4	24/1.54	40.03/1.60	42.26/1.61	25	10.21/3.14	13.01/210	25/1393	optim ended
0.10	dsjc250.1	250	9	70	2.01/2.28	3.05/270	3.70/9180	3	4.54/1.58	6.01/86	8/292	9/8479
0.50	dsjc250.5	250	28	12	5.6/4.93	15.04/10.34	25.01/225	8	6.01/33	7.01/160	8/267	9/3212
0.90	dsjc250.9	250	73	5	34.3/10.4	60.01/11.18	70.09/11.6	7	7.91/0.66	8.01/0.89	9/8.11	14/76
0.10	dsjc500.1	500	12	122	2.25/5.14	2.50/66	3.06/3376	10	8.12/2.50	8.95/13	10/13.3	11/62
0.03	r125.1	125	5	49	2.31/0.02	5/0.2	5/0.2	10	16.25/2.68	24/52	25/259	26/265
0.97	r125.1c	125	46	7	23/2.66	34.5/2.67	46/2.67	3	24/7.75	29.06/747	31.01/5497	32/5812
0.50	r125.5	125	36	5	20.57/0.09	25.2/0.22	36/0.25	3	3.5/0.35	3.75/10.9	3.81/18.2	5*/1076
0.17	1e450_15c	450	15	49	3.07/6.23	4.22/145	5.32/1172	4	3.5/0.46	3.7/6.1	3.75/9.8	5*/91
0.17	1e450_15d	450	15	49	3.07/6.33	4.23/145	5.33/1172	6	8.24/7.03	9.01/13.36	10.001/119	21/5921
								∞	9.98/83	10.01/86	12/391	13/2492
								10	10/3.17	10/3.17	10/3.17	50/586
								∞	time out (>10000)			
								3	3.55/4.19	4.01/272	5/340	6*/5514
								5	4.08/79	4.50/295	5/682	6*/5857
								2	2.31/0.23	2.51/5.67	5/17.7	optim ended
								5	2.31/0.26	2.55/2.61	5/10.6	optim ended
								10	10.55/0.42	11.27/0.75	16/2.56	17/6.1
								∞	25.01/1.85	34.2/13.26	46/28.64	optim ended
								30	20.57/0.74	25.04/7.80	30/59.45	30/157.8
								36	20.57/0.74	25.01/7.84	36/15.57	optim ended
								3	8.34/6.74	10.01/308	15/1751	optim ended
								15	9.19/14.52	13.01/210	15/3331	optim ended
								3	8.21/5.53	10.01/359	15/2326	optim ended
								15	9.04/11.6	12.01/2425	15/3786	optim ended

Table 5: Comparison of (three) lower bounds determined by the standard Cutting-Planes (on the original coloring model) and by the Projective Cutting-Planes (on the coloring model with RR-stables) along the iterations. For the Projective Cutting-Planes part, the last column reports the rounded-up bound obtained by performing a last projection in the model with standard 0-1 stables; project the last pierce point multiplied by $\alpha = 0.9999$ towards $\mathbf{1}_n$; * indicates that we could only compute a lower bound on this last step length (with cplex). The lower bounds that equal the value of the given heuristic (Column 4) are marked in bold; when the Projective Cutting-Planes closes the gap this way, the last column indicates “optim ended” because performing an additional iteration would become useless.

standard stables. This last projection may lead to an even better lower bound. The following remark summarizes the main conclusions that can be drawn from Table 5.

Remark 7 *One of the state-of-the-art Column Generation algorithms for graph coloring [9] could not converge in less than three days for instances like le450_25c, le450_25d, le450_15c, le450_15d and dsjc500.1. Our numerical experiments confirm that a standard Column Generation can indeed “stall” on such instances, exactly for the reason indicated in [9], namely, “the maximum-weight stable-set problems that need to be solved exactly become too numerous and too difficult.” More generally, low-density graphs like the above ones are often quite difficult for the standard Column Generation because they have very large stables that can be really hard to generate (to solve the separation sub-problem). For such graphs, the Projective Cutting-Planes from this section could lead to certain successes:*

1. For le450_25c, le450_25d, le450_15c and le450_15d, the Projective Cutting-Planes reported a lower bound that matches the chromatic number in less than one hour, which seems out of reach for the standard Column Generation. Although these first four instances are not very hard in absolute terms because they can be solved with external methods (using a lower bound given by the maximum clique⁹), the lower bounds of the Projective Cutting-Planes are more general. These bounds could be determined in the same manner for any (dual) objective function $\mathbf{b} \neq \mathbf{1}_n$, as in a multi-coloring problem (see Section 4.1.1) for which the above external methods would fail (because the maximum clique is no longer a lower bound).
2. For dsjc500.1, the last projection in the model with standard stables (last column of Table 5) reports a (rounded up) lower bound of 6; to the best of our knowledge [12, 9], this is the first time anyone has found a feasible solution of such quality in the dual Column Generation LP (4.2). As in the case of the four graphs at point 1 above, the bound value in itself has been already discovered, but only using external means (based on constructing a reduced induced subgraph in [9]). We can even describe this feasible solution of (4.2): assign $0.9999 + 0.00000101$ to the vertices 4, 30, 47, 361, 475 and 0.00000101 to all remaining 495 vertices; the associated objective value is $5 \cdot 0.9999 + 500 \cdot 0.00000101 = 4.9995 + 0.000505 = 5.0005$. Without any risk of numerical errors, we can prove this solution is feasible because there is no stable of size 100 that contains any of the vertices 4, 30, 47, 361, or 475 (these vertices form a clique). Indeed, cplex showed in less than 2 hours that the size of such a stable is upper bounded by 99; the above solution is thus feasible because $0.9999 + 99 \cdot 0.00000101 = 0.99999999 < 1$.
3. For dsjc250.1, the last column of Table 5 indicates that the (cplex solver for the) last projection needed at most 1000 seconds to show that the last step length t_{last}^* is large enough to prove $\lceil 3.80585222 + t_{large}^* \cdot 250 \rceil = 6$, where 3.80585222 is the value of the last pierce point multiplied by $\alpha = 0.9999$. Allowing even more CPU time to this last projection sub-problem, cplex could report after roughly 36 hours (using up to 20 threads and 40GB of RAM on a stronger, multi-core CPU) a lower bound of 0.0088 on the last step length, i.e., it proved $t_{last}^* > 0.0088$. The last projection with standard stables thus proves a lower bound of $\lceil 3.80585222 + t_{last}^* \cdot 250 \rceil \geq \lceil 3.80585222 + 0.0088 \cdot 250 \rceil \geq \lceil 6.005 \rceil = 7$. We see no risk of numerical errors because after 56 hours, cplex could even prove $t_{last}^* > 0.01$. To the best of our knowledge [12, 9], this is the first time a lower bound of 7 has even been reported on this graph.

5.4 Multiple-Length Cutting-Stock

This section is devoted to *Multiple-Length Cutting-Stock*, generally considering two types of standard-size pieces in stock: one of length W and cost 1, and one of length $0.7W$ and cost 0.6. We have two reasons to prefer this problem variant over the standard *Cutting-Stock*: (i) the constraints $(\mathbf{a}, c_a) \in \mathcal{A}$ of the Column Generation dual LP (4.2.1) do not satisfy all $c_a = 1$, and (ii) one cannot compute lower bounds using the Dual Feasible Functions that proved so effective for the standard *Cutting-Stock* [4]. We use a test bed of 30 *Cutting-Stock* instances whose characteristics (i.e., the values of n , W , \mathbf{b} , etc) are described in Appendix B.4, more exactly in Table 9.

⁹One can use a meta-heuristic to find an upper bound and any algorithm to determine the maximum clique size. Since for such graphs the upper bound is often equal to the maximum clique size, this maximum clique size gives the chromatic number.

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

Table 6: The Projective Cutting-Planes compared to the classical Column Generation on *Multiple-Length Cutting-Stock*. The Projective Cutting-Planes needs 40% fewer iterations on almost a $\frac{1}{4}$ of the instances (in bold in Column 5). Notice the CPU times are always smaller in absolute terms than those reported in the companion paper (Section 2, p. 6) of [15], for both the new method and the standard Column Generation. This can not only be explained by the hardware evolution, but also by a better implementation.

Table 6 compares the Projective Cutting-Planes (from Section 4.2.2) to the standard Column Generation. Column 1 indicates the instance, Column 2 presents the optimal value of (4.2.1), Columns 3–6 report the results of the new method, and Columns 7–10 provide the results of the standard Column Generation. For both methods, Table 6 first indicates the computing effort (iterations and CPU time) needed to reach a gap of 20% (*i.e.*, so that $\text{ub} \leq 1.2 \cdot \text{lb}$) and then the total computing effort needed to fully converge.

We notice in Table 6 that the 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 the lower bounds of the Projective Cutting-Planes start on a more upward trend¹⁰ than the (Lagrangian) lower bounds of the standard Column Generation determined via (A.3.2).

Regarding the complete convergence, the standard Column Generation requires in average 44% more iterations than the Projective Cutting-Planes. For the last three (most difficult) instances, the Projective Cutting-Planes reduced the number of iterations by half. By applying stabilization techniques on the clas-

¹⁰The evolution of these lower bounds along the iterations are publicly available on-line at cedric.cnam.fr/~porumbed/projcutplanes/cutstock/. These bounds are also compared to those of a so-called “aggressive” Projective Cutting-Planes that defines \mathbf{x}_{it} via $\mathbf{x}_{it} = \mathbf{x}_{it} + t_{it-1}^* \mathbf{d}_{it-1}$. This Projective Cutting-Planes version eliminates the yo-yo effect (as in Figures 4–5) but it is slower in the long run.

sical **Column Generation**, the reduction of the number of iterations would generally be between 10% and 20%, for all instances except the (very easy) **m20** and **m35** [15, Table 2].

Table 7 provides experimental evidence that the above experimental conclusions are supported by statistical results over 10 runs. For both the new and the standard method, Table 7 reports the average, the standard deviation and the minimum/maximum number of iterations needed by both methods to fully converge. To randomize the two methods, we determine each optimal solution $\text{opt}(\mathcal{P}_{\text{it}})$ by randomly breaking ties in case of equality (at each iteration **it**). The maximum number of iterations needed by the **Projective Cutting-Planes** is usually lower than the minimum number of iterations of the standard **Column Generation**, and so, there is no need for statistical tests to confirm this difference is statistically significant. In addition, all standard deviations are usually rather limited for both methods, generally representing less than 5% of the average value. Other preliminary experiments confirm that similar trends show up across all instances from each instance set, *e.g.*, the instances **m01-1**, **m01-2**, **m01-3** lead to similar results.

Instance	OPT	Projective Cutting-Planes		standard Column Generation	
		avg (std. dev)	min/max	avg (std. dev)	min/max
m01-1	49.3	159 (4.7)	152/168	191 (8.2)	173/202
m20-1	56.6	98.8 (3.2)	91/102	162 (6.9)	152/175
m35-1	73.9	63.3 (3.2)	61/69	66.3 (2)	64/69
vb50c1-1	866.3	82 (0)	82/82	113 (0)	113/113
vb50c2-1	672.3	114 (0)	114/114	127 (0)	127/127
vb50c3-1	281.949	173 (18.2)	119/180	196 (7.2)	174/198
vb50c4-1	579.548	115 (0)	115/115	158 (0)	158/158
vb50c5-1	337.675	201 (22.8)	133/209	238 (3.3)	228/239
wäscher-1	24.0648	308 (13)	287/328	485 (7.5)	466/494
zhard-sch-1	51.4253	356 (7.7)	346/370	707 (11.4)	691/726

Table 7: Statistical comparison (over ten runs) of the number of iterations needed by the two methods to fully converge on the first instance from each instance set.

Finally, the above experimental conclusions are further supported by the results on a second *Multiple-Length Cutting-Stock* variant that introduces a third standard-size piece of length $0.5W$ and cost 0.4. Table 8 compares the new and the standard method on this second problem variant, using the same format (and same columns) as in Table 6. Compared to the standard **Column Generation**, the **Projective Cutting-Planes** requires roughly half iterations for 4 instances out of 10. For another 3 instances, the **Projective Cutting-Planes** requires 40% fewer iterations.

Instance	OPT	Projective Cutting-Planes				Classical Column Generation			
		gap 20%		full convergence		gap 20%		full convergence	
		iters	time[s]	iters	time[s]	iters	time[s]	iters	time[s]
m01-1	49.1	98	0.03	108	0.03	179	0.07	201	0.08
m20-1	56.4	77	0.02	91	0.02	101	0.03	150	0.04
m35-1	72.6	53	0.009	53	0.009	64	0.01	122	0.02
vb50c1-1	832.6	45	1.1	87	2.3	87	6.6	136	10.8
vb50c2-1	622.8	42	2.6	83	4.6	91	15.5	150	21.6
vb50c3-1	263.243	35	7.4	204	95.4	59	31.9	286	184
vb50c4-1	562.3	40	3	80	10	73	13.1	174	41.1
vb50c5-1	315.488	36	7.7	145	52.1	70	29.4	171	83.4
wäscher-1	23.075	77	0.3	212	1.8	283	2.1	450	3.9
zhard-sch-1	47.9	124	17	281	48.3	394	57.4	714	122

Table 8: The **Projective Cutting-Planes** compared to the standard **Column Generation** on a *Multiple-Length Cutting-Stock* variant with 3 types of standard-size pieces in stock: one of length W and cost 1, one of length $0.7W$ and cost 0.6, and one of length $0.5W$ and cost 0.4.

5.5 The oscillations of the inner solutions and the “bang-bang” effects

We here (try to) gain more insight into why an “aggressive” definition of \mathbf{x}_{it} like $\mathbf{x}_{it} = \mathbf{x}_{it-1} + t_{it-1}^* \mathbf{d}_{it-1}$ leads to poor results (in the long run) on certain problems and to reasonable results on others. A possible explanation is related to the oscillations of the inner solutions \mathbf{x}_{it} along the iterations it . The above aggressive \mathbf{x}_{it} definition generates stronger oscillations (“bang-bang” effects) for the first two problems (Sections 5.1 and 5.2) than for the last two (Sections 5.3 and 5.4).

We provide below the values of the first 15 components of $\mathbf{x}_{it+1} = \mathbf{x}_{it} + t_{it}^* \mathbf{d}_{it}$ for $it \in \{1, 11, 21, 31, 41\}$, *i.e.*, as generated by **Projective Cutting-Planes** using the above aggressive \mathbf{x}_{it} definition. For each problem, we selected the very first instance from the main table of results, *i.e.*, from Table 1, Table 3, Table 5, and resp. Table 6. It is clear that these values exhibit stronger oscillations for the first two problems than for the last two. This explains why setting $\mathbf{x}_{it} = \mathbf{x}_{it-1} + t_{it-1}^* \mathbf{d}_{it-1}$ is appropriate for graph coloring, while the best settings for the first two problems take a form $\mathbf{x}_{it} = \mathbf{x}_{it-1} + \alpha t_{it-1}^* \mathbf{d}_{it-1}$ with $\alpha < 0.5$. Regarding the *Multiple-Length Cutting-Stock*, although we do not use $\mathbf{x}_{it} = \mathbf{x}_{it-1} + t_{it-1}^* \mathbf{d}_{it-1}$ in Section 5.4, this choice would still lead to reasonable results, as in the experiments referred by Footnote 10 (p. 39).

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

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

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

6 Conclusion and Prospects

We proposed a **Projective Cutting-Planes** method to optimize LPs over polytopes \mathcal{P} with unmanageably-many constraints. The key idea is to “upgrade” the widely-used separation sub-problem of the well-known **Cutting-Planes** to a more general projection sub-problem. Given an arbitrary inner (feasible) solution $\mathbf{x} \in \mathcal{P}$ and a direction $\mathbf{d} \in \mathbb{R}^n$, this sub-problem asks to determine the pierce (first-hit) point $\mathbf{x} + t^* \mathbf{d}$ encountered when advancing from \mathbf{x} along \mathbf{d} , *i.e.*, determine $t^* = \max\{t \geq 0 : \mathbf{x} + t\mathbf{d} \in \mathcal{P}\}$. The **Projective Cutting-Planes** generates a sequence of inner solutions \mathbf{x}_{it} and a sequence of outer solutions $\text{opt}(\mathcal{P}_{it})$ that both converge along the iterations it to the optimal solution $\text{opt}(\mathcal{P})$. Each inner solution \mathbf{x}_{it} is chosen as a point on the segment joining the previous inner solution \mathbf{x}_{it-1} and the last pierce point $\mathbf{x}_{it-1} + t_{it-1}^* \mathbf{d}_{it-1}$. Theoretical arguments and extensive experiments on four different problems highlight a number of advantages of the **Projective Cutting-Planes**:

- The feasible (interior) solutions $\mathbf{x}_{it} \in \mathcal{P}$ that converge to $\text{opt}(\mathcal{P})$ are generated using a generic *built-in mechanism* of the **Projective Cutting-Planes**. The standard **Cutting-Planes** does not have any built-in functionality to generate converging inner solutions for any problem. And although different ad-hoc techniques (*e.g.*, the Farley bound in **Column Generation**) can be used for certain problems to determine feasible inner solutions during the **Cutting-Planes**, these inner solutions usually remain a by-product of the algorithm, *i.e.*, they are *not* an important “driving force” guiding the **Cutting-Planes** evolution. In contrast, the inner solutions generated by the **Projective Cutting-Planes** are part of the main algorithmic engine and the **Projective Cutting-Planes** could not work without them.
- Except for the robust optimization experiments (Section 5.1), the new method could lead to important reductions of the total running time or of total the number of iterations. More exactly, in Section 5.2, both the number of iterations and the CPU time could be reduced in the best cases by factors of 3 or 4 (*e.g.*, see instances **b** and **d** in Table 3). A reduction of a factor of 2 can also be observed in Section 5.4 on *Multiple-Length Cutting-Stock* (see the last three rows in Table 6 and the last two rows in Table 8). For standard graph coloring (Section 5.3.1), the reduction of the number of iterations can also reach a factor of 4 (first instance in Table 4); we could even report a lower bound that has never been reported before in the literature of the (well studied) graph coloring problem — see last point of Remark 7 (p. 38).
- The separation sub-problem of the standard **Cutting-Planes** determines each new constraint using only one guide point, *i.e.*, the current optimal outer solution. The **Projective Cutting-Planes** generates new constraints by taking into account *a pair* of inner–outer solutions (two guide points). This, coupled with the fact that all considered projection sub-problems $\text{project}(\mathbf{x}_{it} \rightarrow \mathbf{d}_{it})$ satisfy $\mathbf{b}^\top \mathbf{d}_{it} > 0$, enables the **Projective Cutting-Planes** to more easily avoid many degeneracy issues (*i.e.*, iterations that keep the objective value constant) that can arise in **Cutting-Planes**. Although in our experiments these issues of the standard **Cutting-Planes** are only visible in Section 5.1 (Remark 6, p. 29), it is well-known that they do arise quite frequently in **Column Generation** as well.¹¹
- By defining \mathbf{x}_{it} as the best solution ever found (the last pierce point), one can prove that the lower bound $\mathbf{b}^\top \mathbf{x}_{it}$ becomes strictly increasing along the iterations it . This way, the lower bounds for graph coloring (Figure 4) or defective graph coloring (Figure 5) no longer show the infamous “yo-yo” effect that can be seen in most (if not all) existing **Column Generation** algorithms. The “yo-yo” effect could also be eliminated in certain *Multiple-Length Cutting-Stock* experiments, but only using a particular **Projective Cutting-Planes** variant that is not particularly effective in the long run (Footnote 10, p. 39).

There are also certain (inherent) deterrents to adopting the **Projective Cutting-Planes**.¹²

- Since the projection sub-problem is more general, it can be more difficult to design a projection algorithm than a separation one. The projection algorithm can also be more prone to numerical issues because a small precision error in computing the step length t^* can lead to an infeasible pierce point $\mathbf{x} + t^* \mathbf{d}$, which risks generating infeasibilities at the next iteration. For the standard separation, a small precision error can remain innocuous if the returned constraint does separate the current optimal outer solution. As such, more work may be needed to make the **Projective Cutting-Planes** reach its full potential.
- We do not (yet) have a fully comprehensive insight into why the **Projective Cutting-Planes** is more successful on some problems than on others. It remains rather difficult to explain why $\alpha < 0.5$ is often better than $\alpha = 1$ when defining the inner solution \mathbf{x}_{it} with a formula like $\mathbf{x}_{it} = \mathbf{x}_{it-1} + \alpha \cdot t_{it-1}^* \mathbf{d}_{it-1}$. However, we can advance the following arguments:

¹¹As [1, §4] put it, “Column generation processes are known to have a slow convergence and degeneracy problems”. Section 4.2.2 of [11] explains that “large instances are difficult to solve due to massive degeneracy” — see also the references from *loc. cit* for longer explanations of the mechanisms that lead to degeneracy issues.

¹²We do *not* want to let the reader think that we consider the proposed algorithm to be close to revolutionary in any sense; we do not like papers with too much big talking.

1. In a successful **Projective Cutting-Planes** implementation, the feasible solutions \mathbf{x}_{it} generated along the iterations it are rather well-centered, *i.e.*, they do *not* exhibit a “bang-bang” behavior with strong oscillations (see also arguments in Section 5.5). In a loose sense, the inner solutions \mathbf{x}_{it} are reminiscent of an interior point algorithm in which the solutions follow a central path [8, § 3.3], while the outer solutions $\text{opt}(\mathcal{P}_{\text{it}})$ are reminiscent of the **Simplex** algorithm. Recall that the **Simplex** algorithm can sometimes exhibit a “bang-bang” behavior by moving along the edges from one extreme solution to another. We can argue that, by choosing $\mathbf{x}_{\text{it}} = \mathbf{x}_{\text{it}-1} + \alpha \cdot t_{\text{it}-1}^* \mathbf{d}_{\text{it}-1}$ with $\alpha < 0.5$, the **Projective Cutting-Planes** generates more well-centered paths, limiting the “bang-bang” effects.
2. The projection sub-problem of a successful **Projective Cutting-Planes** generates stronger constraints than the separation sub-problem. As described in Section 2.4.1 of [14], when $\mathbf{x} = \mathbf{0}_n$, the intersection subproblem $\text{project}(\mathbf{x} \rightarrow \mathbf{d})$ is equivalent to normalizing all constraints and then choosing one by separating $\mathbf{x} + \mathbf{d}$. The strength of two constraints can be compared without ambiguity only when the constraints are normalized, *i.e.*, when they have the same right-hand side value. Even if this paper uses $\mathbf{x} \neq \mathbf{0}_n$, each projection sub-problem could still generate stronger (normalized) constraints than the ones obtained by separation. For example, consider choosing between $2x_1 + 3x_2 \leq 1$ and $200x_1 + 300x_2 \leq 495$. When solving the separation sub-problem on $[1 \ 1]^\top$, the second constraint might seem more violated because $200 + 300 - 495 = 5 > 2 + 3 - 1 = 4$. But the (level sets of the) two constraints are parallel and the second constraint is considerably weaker, even redundant. It is not difficult to check that the projection sub-problem can never return this (redundant) second constraint, for any feasible \mathbf{x} and for any direction $\mathbf{d} \in \mathbb{R}^2$.

Last but not least, many existing **Cutting-Planes** algorithms could be “upgraded” to a **Projective Cutting-Planes**, provided that one can design a projection algorithm whose running time is similar to that of a separation algorithm. The **Projective Cutting-Planes** could thus be potentially useful to solve other LPs with prohibitively-many constraints, beyond the four problem examples addressed in this paper. This could help one overcome certain limitations of current practices used on canonical **Cutting-Planes**.

References

- [1] Hatem Ben Amor and José M. Valério de Carvalho. Cutting stock problems. In Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon, editors, *Column Generation*, volume 5, pages 131–161. Springer US, 2005. (ISBN 978-0-387-25485-2).
- [2] Jacques F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.
- [3] Abraham Charnes and William W. Cooper. Programming with linear fractional functionals. *Naval research logistics quarterly*, 9(3-4):181–186, 1962.
- [4] François Clautiaux, Cláudio Alves, and José M. Valério de Carvalho. A survey of dual-feasible and superadditive functions. *Annals of Operations Research*, 179(1):317–342, 2009.
- [5] Alysso M Costa. A survey on benders decomposition applied to fixed-charge network design problems. *Computers & operations research*, 32(6):1429–1450, 2005.
- [6] Matteo Fischetti and Michele Monaci. Cutting plane versus compact formulations for uncertain (integer) linear programs. *Mathematical Programming Computation*, 4(3):239–273, 2012.
- [7] Jacek Gondzio. Interior point methods 25 years later. *European Journal of Operational Research*, 218(3):587–601, 2012.
- [8] Jacek Gondzio, Pablo González-Brevis, and Pedro Munari. Large-scale optimization with the primal-dual column generation method. *Mathematical Programming Computation*, 8(1):47–82, 2016.

- [9] Stephan Held, William Cook, and Edward C Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4(4):363–381, 2012.
- [10] Adam N. Letchford, Fabrizio Rossi, and Stefano Smriglio. The stable set problem: Clique and nodal inequalities revisited. *European Journal of Operational Research*, Major revision invited (see <http://www.lancaster.ac.uk/staff/letchfoa/articles.htm>), 2018.
- [11] Marco E. Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.
- [12] Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174–190, 2011.
- [13] Daniel Porumbel. Ray projection for optimizing polytopes with prohibitively many constraints in set-covering column generation. *Mathematical Programming*, 155(1):147–197, 2016.
- [14] Daniel Porumbel. From the separation to the intersection subproblem for optimizing polytopes with prohibitively many constraints in a Benders decomposition context. *Accepted by Discrete Optimization*, in press 2018.
- [15] Daniel Porumbel and François Clautiaux. Constraint aggregation in column generation models for resource-constrained covering problems. *INFORMS Journal on Computing*, 29(1):170–184, 2017.
- [16] François Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86(3):565–594, 1999.

A Greater detail on three projection algorithms

A.1 Numerical difficulties when solving the Benders integer model

For both the separation and the projection sub-problem, the `Cutting-Planes` algorithm for the Benders reformulation (3.2.8a)–(3.2.8c) can encounter a number of numerical issues that are worthwhile investigating. The main one (Appendix A.1.1) regards the optimization of the relaxed master programs, to determine $\text{opt}(\mathcal{P}_{\text{it}})$ at each iteration `it`. The second one (Appendix A.1.2) concerns the projection algorithm.

A.1.1 Numerical problems when solving the integer master problem

The ILP solver for determining $\text{opt}(\mathcal{P}_{\text{it}})$ at each iteration `it` can be particularly prone to numerical or precision problems, especially if \mathcal{P}_{it} contains too many constraints (3.2.8b) with exceedingly large coefficients – as determined by the sub-problem algorithm. For the `cp1ex` LP solver, many values (of variables or slacks) can be zero in theory and slightly larger than zero in practice; multiplying such “ ϵ -sized” values with extremely large coefficients can generate noising terms and numerical precision problems.

Recall that the separation sub-problem performs a normalization of these coefficients by imposing $\mathbf{1}^\top \mathbf{u} = 1$ in (3.2.4). Regarding the projection sub-problem, we mentioned at point (ii) from Section 3.2.3 that the returned \mathbf{u} does not need to be normalized. This is perfectly fine in theory, but if the optimal solution of the LP (3.2.11a)–(3.2.11e) used by the sub-problem has some exorbitant coefficients (see reasons in Appendix A.1.2 below), the projection algorithm can return a constraint (3.2.8b) with exceedingly large coefficients. To avoid such drawbacks, we apply the following principles when solving the projection sub-problem:

- If (3.2.11a)–(3.2.11e) has multiple optimal solutions, it is better to take one with reasonable coefficients; as such, when solving (3.2.11a)–(3.2.11e), the projection algorithm breaks ties by minimizing $\mathbf{1}^\top \bar{\mathbf{u}}$.
- Before inserting a constraint (3.2.8b) into the master problem, it is better to normalize it; for this, we multiply \mathbf{u} by a scalar such that the largest coefficient u_{ij} of a term $u_{ij}x_{ij}$ in (3.2.8b) becomes equal to 10.

Despite above efforts, the master ILP solver (for both the standard or the new **Cutting-Planes**) might take too long to optimize certain relaxed master ILPs associated to (3.2.8a)–(3.2.8c), *i.e.*, it can be too difficult to determine $\text{opt}(\mathcal{P}_{\text{it}})$ at certain (rare) iterations it . Although such problems are not frequent, they could completely block the overall algorithm for a prohibitively long time; accordingly, the *integer* Benders model can become very difficult to solve in such cases, almost impossible.

The key for overcoming this drawback comes from the fact that it is not really essential to determine the optimal solution $\text{opt}(\mathcal{P}_{\text{it}})$ at each iteration it (as described soon). Accordingly, we enforce a limit of $\frac{n}{120} + 1$ seconds on the running time of the ILP solver; as soon as this limit is exceeded, the **Cutting-Planes** continues with the best sub-optimal solution of \mathcal{P}_{it} found by the ILP solver in the given time, which is different from $\text{opt}(\mathcal{P}_{\text{it}})$. This does not change the correctness of the overall algorithm, because this sub-optimal solution could be separated anyway, at the next call to the sub-problem algorithm. If this is not the case, we allow 500 more times to the ILP solver and we let it try again to determine $\text{opt}(\mathcal{P}_{\text{it}})$. If this second try fails, we consider the instance can not be solved. This technique is described in greater detail in Appendix C.2 of [14].

A.1.2 Numerical problems when solving the projection sub-problem

The LP (3.2.11a)–(3.2.11e) used to solve the projection subproblem (p. 15) is also prone to numerical problems, mainly because the decision variables $\bar{\mathbf{u}}$ can be unbounded. In fact, the only constraint that can limit the magnitude of $\bar{\mathbf{u}}$ is $-\sum_{\{i,j\} \in E} b_{\text{wd}} d_{ij} \bar{u}_{ij} = 1$ from (3.2.11d); but since the terms d_{ij} can be positive, negative or zero, it is possible to satisfy this constraint by assigning some extremely high values to certain \bar{u}_{ij} variables. Furthermore, certain factors $b_{\text{wd}} x_{ij}$ in the sum $\sum_{\{i,j\} \in E} b_{\text{wd}} x_{ij} \bar{u}_{ij}$ from the objective function (3.2.11a) can be zero in theory and slightly different from zero in practice (at least when using **Cplex**); multiplying such non-zero factors $b_{\text{wd}} x_{ij}$ with an extremely large \bar{u}_{ij} can lead to non-zero artificial (noising) terms in the objective function.

To reduce such phenomena, we impose a limit of 100 on the maximum value the variables $\bar{\mathbf{u}}$ can take in (3.2.11a)–(3.2.11e). In fact, any practical algorithm for (3.2.11a)–(3.2.11e) has to impose such a limit in practice because the memory is finite and the variables $\bar{\mathbf{u}}$ can not be effectively unbounded. This leads to restricting the feasible set of (3.2.11a)–(3.2.11e); in theory, the resulting restricted LP might not minimize t^* as much as possible, and so, it might return an overestimated t^* so that $\mathbf{x} + t^* \mathbf{d}$ could be potentially infeasible. However, one can certify that the projection sub-problem is correctly solved by checking that the optimal solution satisfies $\bar{u}_{ij} < 100 \forall \{i, j\} \in E$. This is very often confirmed because the optimal $\bar{\mathbf{u}}$ hardly ever contains values larger than 0.5 in practice. When there is however some $\bar{u}_{ij} = 100$, this is most certainly due to the numerical issues above, *i.e.*, $b_{\text{wd}} x_{ij}$ is slightly different from zero in practice although it should be zero in theory. Furthermore, the fact that an intersection point $\mathbf{x}_{\text{it}} + t^* \mathbf{d}_{\text{it}}$ might be infeasible at some iteration it does not change the correctness of the solution returned by the **Projective Cutting-Planes** in the end (for the reasons provided two paragraphs above).

A.2 Graph Coloring Projection: Reinforced Relaxed Stables by Cut Generation

A.2.1 Cut families used to reinforce relaxed stables

The **Projective Cutting-Planes** from Section 4.1.4 uses a graph coloring **Column Generation** model in which each constraint $\mathbf{a}^\top \mathbf{x} \leq 1$ from (4.2) is defined by an RR-stable $\mathbf{a} \in \mathcal{P}$, *i.e.*, by an (extreme) solution \mathbf{a} of the auxiliary polytope \mathcal{P} from Definition 2 (p. 20). This auxiliary polytope \mathcal{P} is an outer approximation of the stable set polytope, *i.e.*, of the convex closure of the standard stables. We construct \mathcal{P} by reinforcing with cuts the description of the relaxed stables (*i.e.*, of the vectors respecting the edge inequalities), hence the name reinforced relaxed stables (RR-stables). More exactly, \mathcal{P} is defined by six classes of (reinforcing) cuts of the form $\mathbf{e}^\top \mathbf{a} \leq 1 \forall (\mathbf{e}, 1) \in \mathcal{R}$ or $\mathbf{f}^\top \mathbf{a} \leq 0 \forall (\mathbf{f}, 0) \in \mathcal{R}$, as indicated in (4.1.5). We present below these six cuts (a)–(f) *without* applying the Charnes–Cooper transformation to translate them to a form like (4.1.6b). In fact, the cuts (a)–(d) are statically added when calling the first intersection sub-problem (and they are re-used for all the next sub-problems), while the cuts (e)–(f) are dynamically added one by one using the **cut generation** algorithm from Section 4.1.4.2.

- (a) The first cut class simply comes from the edge inequalities defining the standard 0–1 stables, *i.e.*, at this stage, we only impose $a_u + a_v \leq 1 \forall \{u, v\} \in E$, obtaining the description of the relaxed stables, *i.e.*, the fractional stable polytope.
- (b) Generalizing the above idea, we use cuts (b) to generate a number of clique inequalities of the form $\mathbf{a}(\mathcal{C}) = \sum_{v \in \mathcal{C}} a_v \leq 1$, using only cliques \mathcal{C} of maximum size $\min(5, k)$. These cliques are enumerated one by one using a backtracking approach; the role of the parameter 5 is to keep the number of such cliques within reasonable limits. To avoid combinatorial explosions, the backtracking algorithm uses the rule that any vertex has to be ignored after appearing in 20 inequalities, *i.e.*, after 20 apparitions, the vertex is discarded from generating future cliques. In other words, we use the most standard backtracking algorithm to enumerate all cliques of maximum size $\min(5, k)$, with the exception that we discard any vertex after it appears in 20 cliques.
- (c) To generate cuts (c), we actually construct a collection of clique inequalities that “cover” V . They are generated by iterating over the vertices $V = [1..n]$, using a method that is reminiscent of Algorithm 1 from [10] or of [12, § 2.2.2]. At the first iteration $i = 1$, this method simply selects a clique \mathcal{C} of a given maximum size k' (see below) that contains the vertex $i = 1$ and imposes the clique inequality $\mathbf{a}(\mathcal{C}) \leq 1$. We now introduce a set $V' = V \setminus \mathcal{C}$ that will evolve along the iterations; all subsequent cliques will be determined by maximizing the number of elements from V' . At the second iteration, we move to the next element $i \in V'$ to determine a new clique $\mathcal{C} \ni \{i\}$ of maximum size k' and impose $\mathbf{a}(\mathcal{C}) \leq 1$. After performing $V' \leftarrow V' \setminus \mathcal{C}$, we move to the next iteration and repeat. At each iteration, we search for a clique \mathcal{C} of bounded size k' with a maximum of elements from V' , *i.e.*, we apply the **Branch & Bound with Bounded Size** (BBBS) from Appendix A.2.3 with very large weights for all $v' \in V'$ and with small weights to all $v \in V \setminus V'$. The value of k' is given by the minimum clique size for which this BBBS algorithm can solve within *at most 0.01 seconds* the standard maximum clique of bounded size (with weights $\mathbf{1}_n$) on G . Experiments suggest that such cuts (c) can even accelerate the **cut generation** by a factor of ten on the Leighton graphs (1e450_25c, 1e450_25c, etc.), especially when k' is much larger than the value of k used at point (f).
- (d) A cut of this class can be associated to any $u, v, w \in V$ such that $\{u, v\} \in E$, $\{u, w\} \notin E$ and $\{v, w\} \notin E$. Using notation $N_v = \{v' \in V : \{v, v'\} \in E\}$, a maximum standard 0–1 stable \mathbf{a}^{std} satisfies the following:

$$a_u^{\text{std}} + a_v^{\text{std}} \leq a_w^{\text{std}} + \mathbf{a}^{\text{std}}(N_w - N_u \cap N_v),$$

because if the maximum stable \mathbf{a}^{std} contains u or v (exclusively), then it also has to contain either w or a neighbor of w . This neighbor of w can only belong to $N_w - N_u \cap N_v$ because it can not be connected to both u and v (since one of u or v belongs to the stable). This idea has also been generalized to the case of triangles $\{\mu, u, v\} \subset V$ not connected to a vertex $w \in V$. We obtain the following cuts:

$$\begin{aligned} a_u + a_v &\leq a_w + \mathbf{a}(N_w - N_u \cap N_v) && \forall \{u, v\} \in E, \{u, w\} \notin E, \{v, w\} \notin E \\ a_\mu + a_u + a_v &\leq a_w + \mathbf{a}(N_w - N_\mu \cap N_u \cap N_v) && \forall \{\mu, u\}, \{\mu, v\}, \{u, v\} \in E, \{\mu, w\} \notin E, \{u, w\} \notin E, \{v, w\} \notin E \end{aligned}$$

We decided to insert such cuts only when they have less than 10 non-zero coefficients; experiments suggest they are the most effective when they have 3 or 4 non-zero coefficients. For instance, when $N_w - N_u \cap N_v = \emptyset$, the first cut simplifies to $a_u + a_v \leq a_w$. Such a cut would eliminate $[1/\omega \ 1/\omega \ 1/\omega \dots 1/\omega]$ from \mathcal{P} , where ω is the maximum clique size of G . Based on this, Remark 4 (p. 22) shows that the optimum of the proposed **Column Generation** model with RR stables can be larger than ω .

- (e) These cuts are classical odd-cycle (or odd-hole) inequalities, dynamically added by solving a separation sub-problem on each current optimal solution $\bar{\mathbf{a}}$ of (4.1.6a)–(4.1.6d). First, notice that a (simple) odd cycle H yields a cut $\sum_{v \in H} a_h \leq \frac{|H|-1}{2}$ because a stable with $\frac{|H|+1}{2}$ vertices of H would have selected two consecutive vertices of the cycle. To separate such a cut, it is enough to re-write it in the form $1 \leq \sum_{v \in H} (1 - 2a_v)$, equivalent to $1 \leq \sum_{\{u, v\} \in EC(H)} (1 - a_u - a_v)$, where $EC(H)$ represents the $|H|$ edges of the cycle inside H . The separation sub-problem can be solved by finding the shortest *odd cycle* in G

considering edge weights $1 - a_u - a_v \forall \{u, v\} \in E$ — these weights are always non-negative because of above cuts (a). This shortest odd cycle can be found by applying Dijkstra’s algorithm on an augmented graph with: (i) a source linked to all vertices V , (ii) all vertices V without any edges between them, (iii) a set V' of copies of V linked to V via edges $\{u, v'\} \in V \times V'$ of weight $1 - a_u - a_v$ for any $\{u, v\} \in E$ (i.e., v' is a copy of v), and (iv) a target vertex linked to all vertices V' .

- (f) The last cut class consists of k -clique inequalities $\mathbf{a}(\mathcal{C}) \leq 1$ associated to cliques \mathcal{C} with at maximum k elements, where k is a parameter that defines the model — it always has to be indicated in the numerical results as in (Column 9 of) Table 5. Separating these cuts reduces to solving a maximum weight clique problem with bounded size k ; the weights \mathbf{a} are given by the optimal solution at the current **cut generation** iteration. For large values of k , (the iterative call to) this problem can become the main computational bottleneck of the overall **Cutting-Planes**. This is why we present in Appendix A.2.3 a specific **Branch & Bound with Bounded Size** (BBBS) algorithm dedicated to this clique problem with bounded size.

A.2.2 Accelerating the Cut Generation using Stabilization

We recall that the Charnes–Cooper LP (re-)formulation (4.1.6a)–(4.1.6d) of the projection sub-problem from Section 4.1.4.1 is solved by **cut generation**. This **cut generation** algorithm was first described in Section 4.1.4.2; at each iteration, it attempts to separate (the current optimal solution using) the above (a)–(f) cuts. More exactly, the constraints (a)–(d) are statically inserted in (4.1.6a)–(4.1.6d) at the very first iteration; only cuts (e)–(f) have to be iteratively separated.

A positive distinguishing characteristic of (4.1.6a)–(4.1.6d) is that all constraints (4.1.6b) generated at a given iteration it can be kept throughout all subsequent iterations of the overall **Cutting-Planes**, because they do not depend on the current \mathbf{x}_{it} or \mathbf{d}_{it} . In (4.1.6a)–(4.1.6d), only the constraints (4.1.6c) are specific to the current iteration it because they depend on \mathbf{d}_{it} . The cuts (a)–(f) that produce constraints (4.1.6b) never change along the **Projective Cutting-Planes** iterations.

All cuts (e) can be separated quite rapidly by applying Dijkstra’s algorithm once. Most of the computing effort is spent on repeatedly separating the constraints (f) by solving a maximum weight clique problem. Besides designing in Appendix A.2.3 a dedicated **Branch & Bound with Bounded Size** (BBBS) algorithm for this (bounded size) maximum clique problem, we also propose the following two ideas to further accelerate the **cut generation**:

1. We use a simple-but-effective solution smoothing technique: instead of calling the separation algorithm on the current optimal solution, we call it on the midpoint between the current optimal solution and the previous optimal solution. If the current optimal solution can not be separated this way, we have to call the separation algorithm again, this second time on the current optimal solution.
2. We propose a (meta-)heuristic algorithm for the sub-problem before calling the BBBS algorithm. This heuristic executes $5 \cdot n$ iterations of a Tabu Search algorithm.¹³ We always start the **cut generation** in a heuristic mode, trying to solve all maximum weight cliques with this heuristic. But once the heuristic fails, the **cut generation** algorithm switches to an exact mode (running only BBBS) for 15 iterations. After each 15 iterations, it tries again to solve the problem heuristically. Unless this repeated heuristic call is successful, the **cut generation** remains in the exact mode for another 15 iterations.

A.2.3 Maximum Weight Clique Branch-and-Bound with Bounded Size (BBBS)

The maximum weight clique with bounded size is a rather general graph-theoretic problem that could be modeled and solved with many different methods. However, perhaps rather surprisingly at first glance, we did

¹³This Tabu Search algorithm encodes each candidate solution as a bit string of length n with exactly k ones. The objective function is the sum of the edge weights induced by the vertices selected by the bit string. Each two vertices $u, v \in V$ are associated to an edge weight, either $\frac{1}{2}(a_u + a_v)$ if $\{u, v\} \in E$ or a prohibitively-small negative weight when $\{u, v\} \notin E$. A Tabu Search iteration selects the best non-Tabu vertex swap, the one maximizing the objective function. We use incremental data structures to perform a fast streamlined calculation of the objective function variation associated to each vertex swap. After deselecting a vertex, it becomes Tabu for $10 + \mathbf{random}(5)$ iterations, where $\mathbf{random}(5)$ returns a uniformly random integer value in $\{0, 1, 2, 3, 4, 5\}$. For $k = \infty$, we used the multi-neighborhood Tabu search (www.info.univ-angers.fr/~hao/cliquote.html) due to Q. Wu, J.K. Hao and F. Glover.

not find any dedicated off-the-shelf software to solve it as rapidly as necessarily. We thus have to introduce a new **Branch & Bound with Bounded Size** (BBBS) algorithm devoted to this problem. This BBBS was mainly used to separate the cuts (f) from Appendix A.2.1, as needed by the **cut generation** algorithm from Appendix A.2.2 above. At the same time, BBBS can directly solve the complementary problem, *i.e.*, the (bounded-size) maximum weight stable. We thus also used BBBS to solve the separation sub-problem of the classical **Column Generation** algorithm with standard stables (for the results reported in Section 5.3.2).

The main algorithmic engine

The BBBS algorithm relies on a fairly straightforward Branch & Bound (B&B) routine that successively adds vertices to existing cliques to construct increasingly larger cliques (B&B nodes); it generates a branching tree in a deep-first manner. The number of generated B&B nodes (and the total running time) depends substantially on the quality of the lower and upper bounds used for branch pruning. The lower bound is simply given by the best clique ever generated, *i.e.*, there is a unique global lower bound for the whole branching tree at each moment. The upper bound is determined at each B&B node and it is more critical for reducing the running time; we will present below a dedicated algorithm for it.

All cliques are constructed (to generate B&B nodes) by adding vertices to existing cliques following an initial order v_1, v_2, \dots, v_n such that $w_1 \geq w_2 \geq w_3 \geq \dots \geq w_n$, *i.e.*, the vertices are initially sorted by decreasing weight. As such, the very first B&B node is simply the clique $\{v_1\}$. Since the B&B tree is constructed in a deep-first-search manner, the second generated B&B node is $\{v_1, v_i\}$ where $i = \min \{i : \{v_i, v_1\} \in E\}$ and the third node is $\{v_1, v_i, v_j\}$ where $j = \min \{j : \{v_j, v_1\} \in E, \{v_j, v_i\} \in E\}$, assuming $k \geq 3$.

The global lower bound

The best clique ever constructed provides the global lower bound. In addition, recall (point 2 from Appendix A.2.2) that one can first try to solve the bounded-size maximum weight clique problem using a (meta-)heuristic prior to launching BBBS; this can provide a second lower bound. Preliminary experiments suggest that trying other better or faster (meta-)heuristics do not usually lead to an impressive acceleration of the BBBS, and so, we hereafter focus on the upper bound.

The upper bound of each node

The running time of BBBS seems more sensitive to the quality of the upper bound. Let us first present the most basic upper bound to be generalized next. Consider the current B&B node corresponding to a constructed clique \mathcal{C} of k' elements with $k' < k$ (otherwise the node is a leaf). The remaining as-yet-unconsidered vertices constitute a list (u_1, u_2, u_3, \dots) sorted by decreasing weight — because this is the original sorting order of all vertices. After eliminating from (u_1, u_2, u_3, \dots) all vertices that are not connected to all $v \in \mathcal{C}$, one obtains a reduced list $L_{\mathcal{C}}$ of vertices linked to all vertices from \mathcal{C} . The simplest upper bound is then given by the sum of the weights of the first $k - k'$ vertices in $L_{\mathcal{C}}$ (plus the weight of \mathcal{C}).

We now present a higher-quality upper bound that can improve BBBS up to reducing the running time by a factor of seven. The pseudo-code below presents the algorithm for calculating this bound and it relies on the following ideas. First, recall that the above basic upper bound simply sums up the weights of the first $k - k'$ elements in $L_{\mathcal{C}}$. Going beyond this idea, the new algorithm investigates in greater detail which of these vertices should really contribute to the upper bound value. A vertex u of $L_{\mathcal{C}}$ can *not* contribute to the upper bound value, if there is a preceding $v \in L_{\mathcal{C}}$ (of higher weight) such that $\{v, u\} \notin E$ that has already contributed to the bound value. We say that v shadows u ; in such case, the pseudo-code below breaks the loop processing u using the first **continue** statement. However, this idea can not be applied twice: if there is a second vertex u' such that $\{v, u'\} \notin E$, v can not shadow both u and u' because selecting u and u' can be better than selecting v (assuming $\{u, u'\} \in E$). This explains why the pseudo-code below first inserts u into a list L of vertices that can shadow other vertices (Line 15), but then removes any $v \in L$ at Line 6 if v shadows u , *i.e.*, v can shadow only one vertex at most. However, v could still shadow some $u' \in V$, but only if $\{v, u, u'\}$ is a stable and such cases are detected using a second list L' .

```

1:  $\text{ub} \leftarrow \sum_{v \in \mathcal{C}} \text{weight}(v)$ ,  $\text{addedVtx} = 0$ 
2:  $L \leftarrow \emptyset$  ▷ vertices  $v$  that shadow other vertices

```

```

3:  $L' \leftarrow \emptyset$  ▷ non-edges  $\{v, u\} \notin E$  such that  $v$  shadows  $u$ 
4: for all  $u \in L_{\mathcal{C}}$  do ▷ scan  $L_{\mathcal{C}}$  by descending order of weight
5:   if  $\exists v \in L$  such that  $\{v, u\} \notin E$  then ▷  $v$  shadows  $u$  and no clique contains both  $v$  and  $u$ 
6:      $L \leftarrow L \setminus \{v\}$  ▷  $v$  can not shadow more vertices using the test at Line 5
7:      $L' \leftarrow L' \cup \{(v, u)\}$  ▷ but it can shadow some  $u'$  if  $\{v, u, u'\}$  is a stable at Line 10
8:     continue ▷ we do nothing else with  $u$ , because  $u$  is shadowed by  $v$ 
9:   end if
10:  if  $\exists (v, u') \in L'$  such that  $\{v, u, u'\}$  is a stable then ▷  $v$  shadows both  $u$  and  $u'$ 
11:     $L' \leftarrow L' \setminus \{(v, u')\}$  ▷  $v$  can shadow at maximum two vertices
12:    continue ▷ we do nothing else with  $u$ , as  $u$  is shadowed by  $v$  and  $u'$ 
13:  end if
14:   $\text{ub} \leftarrow \text{ub} + \text{weight}(u)$ ,  $\text{addedVtx} \leftarrow \text{addedVtx} + 1$ 
15:   $L \leftarrow L \cup \{u\}$  ▷  $u$  can shadow subsequent vertices in  $L_{\mathcal{C}}$ 
16:  if ( $\text{addedVtx} == k - |\mathcal{C}|$ ) then
17:    break
18:  end if
19: end for
20: return  $\text{ub}$ 

```

Finally, to make the BBBS reach its full potential, one could still apply a number of further engineering and implementation optimizations (as for many applied algorithms). For instance, experiments suggest that the BBBS can be faster if we limit the size of the lists L and L' to $\min(10, \frac{2}{3}k)$. In addition, we decided not to use the above improved upper bound when k is exceptionally large (greater than half the average degree of G).

A.3 Two concepts used in *Cutting-Stock* that are essentially more general: a fast data structure for handling Pareto frontiers and Lagrangian bounds

This section explores two notions that were originally needed by our (*Multiple-Length*) *Cutting-Stock* algorithms, but that are essentially more general and even not necessarily related to *Cutting-Stock*. First, Appendix A.3.1 presents a fast data structure to manipulate a Pareto frontier. Then, Appendix A.3.2 examines the Lagrangian bounds of the standard **Column Generation**, placing a special emphasis on the case in which $c_a = 1 \forall (\mathbf{a}, c_a) \in \mathcal{A}$ does not hold.

A.3.1 A fast data structure to record a Pareto frontier

Recall from Section 4.2.3.2 (Remark 5, p. 26) that the **Dynamic Programming** (DP) scheme has to manipulate a list of states whose cost and profits constitute a Pareto frontier, satisfying the relation (4.2.3.a)–(4.2.3.b). More exactly, the considered states correspond a list of $|I|$ cost/profit pairs $c_i/p_i \forall i \in I$ that satisfy the following (Pareto dominance) relations:

$$c_1 < c_2 < c_3 \cdots < c_{|I|}$$

$$p_1 < p_2 < p_3 \cdots < p_{|I|}$$

As stated in Remark 5, one of the most computationally intensive tasks (of Algorithm 1, p. 25) is the insertion of a new pair (at Step 5); one should not scan the whole list I to check whether the new pair is dominated or not. We propose to record this list I in a *self-balancing binary tree* because this data structure is designed to manipulate ordered lists and it can perform a lookup, an insertion and a removal in logarithmic time (with respect to the number of pairs already recorded in I).

Given a new pair c^+/p^+ , one has to decide as rapidly as possible if c^+/p^+ should be inserted into the list I or if c^+/p^+ is dominated; as hinted above, this task can not be efficiently solved by simply scanning one by one all elements in I . To avoid this (computationally-taxing) enumeration, we record I in a self-balancing binary tree that uses the order 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 . Regardless of the implementation, any self-balancing binary tree has to be able to compare c^+

to the pair c^*/p^* with the highest cost no larger than c^+ , *i.e.*, such that $c^* = \max\{c_i : c_i \leq c^+, i \in I\}$. Without comparing to c^*/p^* , it is certainly impossible to decide whether c^+/p^+ should be inserted before or after c^*/p^* in the binary tree. Thus, any implementation of the self-balancing tree has to provide a means to determine c^*/p^* . In the worst case, it is certainly possible to (temporary) insert c^+/p^+ in the tree and return the element before c^+/p^+ ; this operation relies on the insertion operator and other constant-time manipulations, and so, it takes logarithmic time.

Once c^*/p^* is determined, we apply an insertion routine that performs the following. First, if $p^* \geq p^+$, then the new pair c^+/p^+ is directly rejected because it is dominated by definition. Otherwise, if $p^* < p^+$, then c^+/p^+ has to be inserted in the tree, and so, other recorded pairs may become dominated and need to be removed. For instance, if $c^* = c^+$ and $p^* < p^+$, then c^*/p^* is immediately removed from the tree. Furthermore, the proposed 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 $c^\#/p^\#$ is ordered after c^*/p^* in the tree.

A.3.2 The Lagrangian Bounds of the Standard Column Generation

In the numerical experiments for both graph coloring and (*Multiple-Length*) *Cutting-Stock*, we compared the lower bounds of the **Projective Cutting-Planes** to the Lagrangian lower bounds of the standard **Column Generation**. When all columns have equal unitary costs (*i.e.*, $c_a = 1 \forall (\mathbf{a}, c_a) \in \mathcal{A}$ as in graph coloring), we simply used the Farley Lagrangian lower bound

$$\frac{\mathbf{b}^\top \mathbf{x}}{1 - m_{rdc}(\mathbf{x})}, \quad (\text{A.3.1})$$

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

In *Multiple-Length Cutting-Stock*, the column costs are no longer unitary and the above bound can evolve to $\frac{\mathbf{b}^\top \mathbf{x}}{1 - \frac{1}{c_{\min}} m_{rdc}(\mathbf{x})}$, where c_{\min} is the minimum cost of a feasible (cutting-)pattern. To show this, one can extend any of the interesting proofs from [1, § 2.2], [16, § 3.2] or [11, § 2.1]. However, it is more convenient for us to prove it using our formalism from our previous work [13, Appendix C]. Using (C.2) from *loc. cit.*, the lower bound can be written as $\mathbf{b}^\top \mathbf{x} + u_b \cdot m_{rdc}(\mathbf{x})$, where u_b is any valid upper bound of $\sum_{(\mathbf{a}, c_a) \in \mathcal{A}} y_a$, *i.e.*, if one imposes $\sum_{(\mathbf{a}, c_a) \in \mathcal{A}} y_a \leq u_b$ in the primal (4.1), the optimum of (4.1) has to remain the same. When $c_a = 1 \forall (\mathbf{a}, c_a) \in \mathcal{A}$, one can say that this optimum of (4.1) is itself an upper bound of $\sum_{(\mathbf{a}, c_a) \in \mathcal{A}} y_a = \sum_{(\mathbf{a}, c_a) \in \mathcal{A}} c_a y_a$, which leads to the above Farley bound. When $c_a = 1 \forall (\mathbf{a}, c_a) \in \mathcal{A}$ does not hold, one can only infer $\sum_{(\mathbf{a}, c_a) \in \mathcal{A}} y_a \leq \frac{1}{c_{\min}} \cdot \sum_{(\mathbf{a}, c_a) \in \mathcal{A}} c_a y_a$. Thus, one notices that in all formulae after (C.2) from [13, Appendix C] the term m_{rdc} has to be multiplied by $\frac{1}{c_{\min}}$, and so, $\frac{\mathbf{b}^\top \mathbf{x}}{1 - m_{rdc}(\mathbf{x})}$ evolves to:

$$\frac{\mathbf{b}^\top \mathbf{x}}{1 - \frac{1}{c_{\min}} m_{rdc}(\mathbf{x})}. \quad (\text{A.3.2})$$

The above formula yields a valid lower bound only when $m_{rdc}(\mathbf{x}) \leq 0$, *i.e.*, by closely investigating the Lagrangian relaxation proof from [13, Appendix C], we notice it uses the fact that \mathbf{x} is the optimal dual solution of a reduced master program. As such, the above (A.3.2) is not always necessarily a lower bound for any arbitrary $\mathbf{x} \in \mathbb{R}_+^n$ with $m_{rdc}(\mathbf{x}) > 0$. 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 (A.3.2) yields $\frac{0.9}{1 - \frac{0.1}{0.6}} = 1.08$ 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).

Recall (last paragraph of Section 4.2.2, p. 23) that the first two iterations of the **Projective Cutting-Planes** for *Multiple-Length Cutting-Stock* solve the projection sub-problems $\text{project}(\mathbf{0}_n \rightarrow \frac{1}{W} \mathbf{w})$

and $\text{project}(\mathbf{0}_n \rightarrow \mathbf{b})$, generating two initial lower bounds. As described above, the standard **Column Generation** could not generate a lower bound by applying (A.3.2) on an arbitrary $\mathbf{x} \in \mathbb{R}_+^n$, including on $\mathbf{x} = \frac{1}{\bar{w}} \mathbf{w}$ which is a feasible solution with $m_{rdc}(\frac{1}{\bar{w}} \mathbf{w}) \geq 0$. However, to generate useful initial columns and to provide similar starting conditions for the **Column Generation** and the **Projective Cutting-Planes**, we also start the standard **Column Generation** by first solving the separation sub-problem on $\frac{1}{\bar{w}} \mathbf{w}$ and \mathbf{b} .

B Complementary experimental information

We recall that the C++ source code is publicly *available on-line* for all four considered problems at <http://cedric.cnam.fr/~porumbed/projcutplanes/> — there are 13279 lines all together as of August 2018. We compiled these C++ files with gcc using the code optimization option `-O3`; we used the Cplex 12.6 library for C++ to solve all (integer) linear programs (concert technology). All reported results have been generated by these programs on a mainstream Linux computer using a i7-5500U CPU with 16GB of RAM; unless otherwise stated, all programs use a single thread.

B.1 Robust optimization instances

Most instances from Section 5.1 (Table 1) have between $n = 1000$ and $n = 5000$ variables and a number of constraints between 500 and 3000. They are all taken from Table 1 of [6] and we refer the reader to this table 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.

B.2 The network design problem formulated using the Benders decomposition

As mentioned in Section 5.2, the instances for the network design problem from Section 3.2 are either taken from [14] or generated for the first time in this work (seven instances). More exactly, the seven instances generated now for the first time are `a`, `b`, `c`, `d`, `e`, `f`, `g`. The (set of) instances `rnd-10` from Tables 2 and 3 correspond to the `random-10-bnd3` instances from Table 2 of [14]. The instances `rnd-100` and `rnd-300` resp. correspond to `random-100-bnd3` and `random-300-bnd3` from Table 4 of [14]. Their main characteristics are described in the first five columns of Table 2 or Table 3.

B.3 Graph coloring

All coloring instances were generated during the second DIMACS implementation challenge, and they are publicly available at <http://cedric.cnam.fr/~porumbed/graphs/>, along with their descriptions and characteristics. They can have different structural properties. The instances `le450.X` are called Leighton graphs and their chromatic number is `X`; they all have a clique of size `X`. The instances `dsjcX.Y` represent random graphs with `X` vertices generated using the classical Erdős–Rényi model. The instances `r.X` are called random geometrical graphs; they are generated by picking `X` points uniformly at random in a square and by inserting an edge between all pairs of vertices situated within a certain distance; a suffix “c” indicated the graph was complemented.

B.4 Multiple-Length Cutting-Stock

The test bed used in this paper consists of 10 instance sets and we refer the reader to Table 1 of [15] for (bibliographic) references on their (industrial) origins. For each set, the number (ID) of each individual instance is indicated by a suffix, *e.g.*, we use the formulation `m01-1`, `m01-2`, `m01-3` to refer to the first, second, resp. third instance from the set `m01`. The main characteristics of each instance set are provided in Table 9 next.

Instance set	n	capacity W	avg. demand \mathbf{b}	avg. \mathbf{w} span	description
m01	100	100	1	$[1, 100]$	1000 random bin-packing instances
m20	100	100	1	$[20, 100]$	1000 random bin-packing instances
m35	100	100	1	$[35, 100]$	1000 random bin-packing instances
vb50c1	50	10000	$[50, 100]$	$[1, \frac{3}{4}W]$	20 random instances
vb50c2	50	10000	$[50, 100]$	$[1, \frac{1}{2}W]$	20 random instances
vb50c3	50	10000	$[50, 100]$	$[1, \frac{1}{4}W]$	20 random instances
vb50c4	50	10000	$[50, 100]$	$[\frac{1}{10}W, \frac{1}{2}W]$	20 random instances
vb50c5	50	10000	$[50, 100]$	$[\frac{1}{10}W, \frac{1}{4}W]$	20 random instances
wäscher	57-239	10000	1	$[1, \frac{1}{2}W]$	17 hard bin-packing instances
hard-sch	≈ 200	100000	$[1, 3]$	$[\frac{20}{100}W, \frac{35}{100}W]$	the ten hardest scholl instance sets.

Table 9: Characteristics of the *Cutting-Stock* instance set, also used for *Multiple-Length Cutting-Stock*.