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 $\mathscr{P}$ 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, so that $\mathbf{x}+t^*\mathbf{d}$ belongs to the boundary of $\mathscr{P}$. In [13], we only explored the idea of projecting the origin $\mathbf{0}_n$ along integer directions, focusing on dual polytopes $\mathscr{P}$ in `Column Generation` models. In this work, we address a more general projection (intersection) sub-problem, considering arbitrary interior points $\mathbf{x} \in \mathscr{P}$ and arbitrary non-integer directions $\mathbf{d} \in \mathbb{R}^n$, in more problems beyond `Column Generation`, *e.g.*, in robust optimization or Benders decomposition problems. By "upgrading" the separation sub-problem of the standard `Cutting-Planes` to the projection sub-problem, we designed a `Projective Cutting-Planes` algorithm to optimize over polytopes $\mathscr{P}$ with prohibitively-many constraints. At each iteration, the `Projective Cutting-Planes` selects a point $\mathbf{x}_{\text{new}}$ on the segment joining the points $\mathbf{x}$ and $\mathbf{x}+t^*\mathbf{d}$ determined at the previous iteration. Then, it projects $\mathbf{x}_{\text{new}}$ along a direction $\mathbf{d}_{\text{new}}$ that points towards the current optimal solution (of the current outer approximation of $\mathscr{P}$), so as to generate a new pierce point $\mathbf{x}_{\text{new}} + t^*_{\text{new}}\mathbf{d}_{\text{new}}$ and a new constraint of $\mathscr{P}$. By re-optimizing the linear program enriched with this new constraint, the algorithm finds a new current optimal 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 solution $\mathbf{x} + t^*\mathbf{d}$ (a primal bound) at each iteration. By generating a sequence of such feasible solutions that converge to the optimum $\texttt{opt}(\mathscr{P})$, 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 approach.

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 $\texttt{it}$ and outer approximation $\mathscr{P}_{\texttt{it}}$ of the feasible polytope $\mathscr{P}$, *i.e.*, a polytope $\mathscr{P}_{\texttt{it}}$ defined only by a subset of the constraints of $\mathscr{P}$, so that $\mathscr{P}_{\texttt{it}} \supseteq \mathscr{P}$. The standard `Cutting-Planes` can be seen as an outer method in the sense that it converges towards an optimal solution $\texttt{opt}(\mathscr{P})$ through a sequence of outer (infeasible) solutions; as such, for a maximization problem, it generates a convergent sequence of upper bounds $\texttt{optVal}(\mathscr{P}_1) \geq \texttt{optVal}(\mathscr{P}_2) \geq \texttt{optVal}(\mathscr{P}_3) \geq \cdots \geq \texttt{optVal}(\mathscr{P})$. The most canonical `Cutting-Planes` has no built-in mechanism that can generate (for any problem) a sequence of primal bounds (feasible solutions) that converge along the iterations. In contrast, an inner method constructs a convergent sequence of inner feasible solutions $\mathbf{x}_1,\ \mathbf{x}_2,\ \mathbf{x}_3, \ldots$ that converge towards $\texttt{opt}(\mathscr{P})$. 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 related to inner and outer methods or to the intersection sub-problem.

The proposed `Projective Cutting-Planes` is based on an iterative operation of projecting an interior point onto facets of $\mathscr{P}$, as illustrated in Figure 1. At each iteration $\texttt{it}$, an inner solution $\mathbf{x}_{\texttt{it}} \in \mathscr{P}$ is projected towards the direction $\mathbf{d}_{\texttt{it}}$ of the current outer optimal solution $\texttt{opt}(\mathscr{P}_{\texttt{it}-1})$, *i.e.*, setting $\mathbf{d}_{\texttt{it}} = \texttt{opt}(\mathscr{P}_{\texttt{it}-1}) - \mathbf{x}_{\texttt{it}}$. This is referred to as the projection (or intersection) sub-problem: determine $t^*_{\texttt{it}} = \max\{t : \mathbf{x}_{\texttt{it}} + t\mathbf{d}_{\texttt{it}} \in \mathscr{P}\}$. By solving this sub-problem, one determines a pierce (hit) point $\mathbf{x}_{\texttt{it}} + t^*_{\texttt{it}}\mathbf{d}_{\texttt{it}}$ and a new (first-hit) constraint of $\mathscr{P}$, which is added to the constraints of $\mathscr{P}_{\texttt{it}-1}$ to construct $\mathscr{P}_{\texttt{it}}$. At next iteration $\texttt{it}+1$, the `Projective Cutting-Planes` takes a new interior point $\mathbf{x}_{\texttt{it}+1}$ on the segment joining $\mathbf{x}_{\texttt{it}}$ and $\mathbf{x}_{\texttt{it}} + t^*_{\texttt{it}}\mathbf{d}_{\texttt{it}}$, and it projects $\mathbf{x}_{\texttt{it}+1}$ along $\mathbf{d}_{\texttt{it}+1} = \texttt{opt}(\mathscr{P}_{\texttt{it}}) - \mathbf{x}_{\texttt{it}+1}$.

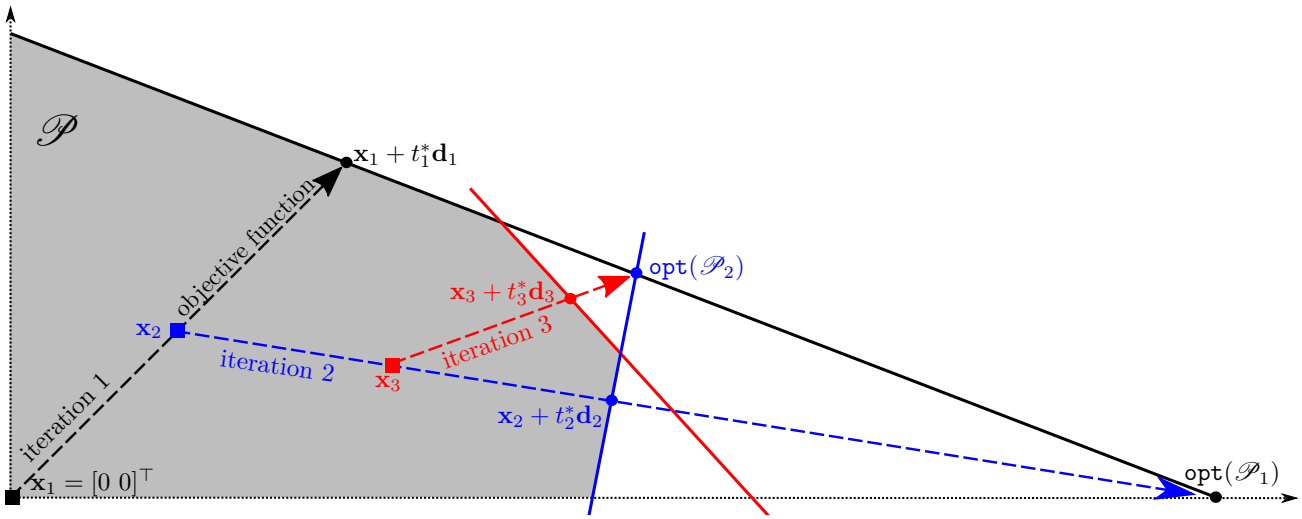

Figure 1: The first three iterations of the `Projective Cutting-Planes` on a linear program with 2 variables and 3 constraints. At the first iteration, the projection sub-problem projects $\mathbf{x}_1 = \mathbf{0} = [0\ 0]^\top$ along the objective function, as depicted by the black dashed arrow. At iteration $\mathtt{it} = 2$, the midpoint $\mathbf{x}_2$ of this black arrow is projected towards the direction of the outer optimal solution $\mathtt{opt}(\mathscr{P}_1)$ at iteration 1 (when $\mathscr{P}_1 \supset \mathscr{P}$ only contains the largest triangle). This generates a second facet (blue solid line) that is added to the facets of $\mathscr{P}_1$ to construct $\mathscr{P}_2$. 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 $\mathtt{opt}(\mathscr{P}_2)$.

To solve the intersection sub-problem $t^* = \max\{t : \mathbf{x} + t\mathbf{d} \in \mathscr{P}\}$, one also 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 $t \in \mathbb{R}_+$, *i.e.*, all points $\mathbf{x} + t\mathbf{d}$ with $t > t^*$ are separated. We have already studied a simplified version of the projection sub-problem where we could only project from $\mathbf{x} = \mathbf{0}_n$ in `Column Generation` [13] or in Benders decomposition models [14]. In the current work, we seek maximum generality in terms of projections: we will project arbitrary interior points $\mathbf{x} \in \mathscr{P}$ 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), in the sense that both methods produce 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 $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3, \ldots$ of the `Projective Cutting-Planes`. However, the central path in IPM consists of solutions which are interior with regards to the current outer approximation $\mathscr{P}_{\mathtt{it}} \supset \mathscr{P}$ at iteration `it`, but they do not necessarily belong all to $\mathscr{P}$.

By generalizing the separation sub-problem, the projection sub-problem is inherently more difficult. The projection subproblem may even seem to significantly more expensive computationally, but we will show this is not always the case. We will present four several 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 of the separation algorithm without increasing its 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 $c_a - (\mathbf{a}+\widehat{\mathbf{a}})^\top\mathbf{x}$ over a set of nominal constraints $(\mathbf{a}, c_a) \in \mathcal{A}_{\mathtt{nom}}$ and over all possible deviations $\widehat{\mathbf{a}}$ of the nominal coefficients $\mathbf{a}$. The intersection sub-problem

$\texttt{project}(\mathbf{x} \to \mathbf{d})$ reduces to minimizing a ratio $\dfrac{c_a - (\mathbf{a}+\widehat{\mathbf{a}})^\top \mathbf{x}}{(\mathbf{a}+\widehat{\mathbf{a}})^\top \mathbf{d}}$ over the same nominal constraints $\mathcal{A}_{\texttt{nom}}$ and over the same deviations $\widehat{\mathbf{a}}$ of $\mathbf{a}$. Both sub-problems can be solved by enumerating the nominal constraints $\mathcal{A}_{\texttt{nom}}$; for each $(\mathbf{a}, c_a) \in \mathcal{A}_{\texttt{nom}}$, the separation sub-problem tries to minimize the above difference while the intersection sub-problem tries to minimize the above ratio.

- A second technique is applicable to the (numerous) problems in which the constraints of $\mathscr{P}$ are defined as the feasible solutions of a second LP. This is the case for the Benders decomposition models from Section 3.2; their separation sub-problem requires solving 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-problem require the (same) asymptotic running time of solving an LP.
- The above technique can be generalized to the (numerous) problems in which the constraints of $\mathscr{P}$ represent the feasible solutions of an *Integer* LP (ILP). We will develop this idea in Section 4.1, where $\mathscr{P}$ is the dual polytope of a `Column Generation` model for graph coloring; the constraints of $\mathscr{P}$ are given by the primal columns, which are associated to the 0–1 stables of the considered graph. The projection sub-problem reduces to an integer linear-fractional program over the stable set polytope. Proposing a discrete version of the Charnes-Cooper transformation, we could translate this into a Disjunctive LP that can be solved with the same techniques as an ILP, *i.e.*, in both cases we apply a `Branch and Bound` in which the bounds are determined by solving continuous relaxations. We will argue in Section 4.1.3.1 that there is no in-depth reason why a Disjunctive LP should be harder in absolute terms than a similar-size ILP, especially when solving them 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, 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 difficult task of most Dynamic Programming schemes is to generate all states; once all states are generated, it is not difficult to return the state of minimum objective value (for any objective function form).

The remainder is organized as follows. Section 2 is devoted to a more detailed description of the proposed `Projective Cutting-Planes`. Section 3 illustrates the application of this algorithm on two problems in which $\mathscr{P}$ 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` to solve two `Column Generation` models (for graph coloring and *Multiple-Length Cutting-Stock*), where $\mathscr{P}$ 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 present more information on the experimental settings.

2 Algorithmic Description of the `Projective Cutting-Planes`

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

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

where $\mathcal{A}$ is a set of (unmanageably-many) constraints. We do not actually impose any condition on the size of $\mathcal{A}$, but we focus on sets $\mathcal{A}$ that are too large to be fully enumerated in practice. In fact, we will also be able to address a few 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 the dual LP obtained after relaxing an integer `Column Generation` model, the goal is actually to find the best *rounded-up* objective value, *i.e.*, "$\max \mathbf{b}^\top \mathbf{x}$" is 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 in the same manner.

We briefly recall the standard `Cutting-Planes` for solving the above LP. This method maintains at each iteration `it` an outer approximation $\mathscr{P}_{\texttt{it}}$ of $\mathscr{P}$ obtained by restricting the constraint set $\mathcal{A}$ to a subset $\mathcal{A}_{\texttt{it}}$, so that $\mathscr{P}_{\texttt{it}} \supset \mathscr{P}$. To (try to) separate the current outer optimal solution $\mathbf{x}^{\text{out}} = \texttt{opt}(\mathscr{P}_{\texttt{it}})$ of this polytope $\mathscr{P}_{\texttt{it}}$, the most standard `Cutting-Planes` solves the separation sub-problem $\min\limits_{(\mathbf{a},c_a)\in\mathcal{A}} c_a - \mathbf{a}^\top \mathbf{x}^{\text{out}}$. If the solution of this sub-problem is less than 0 for some $(\overline{\mathbf{a}}, \overline{c_a}) \in \mathcal{A}$, then $\mathbf{x}^{\text{out}}$ is infeasible. In this case, the `Cutting-Planes` method inserts $\overline{\mathbf{a}}^\top \mathbf{x} \leq \overline{c_a}$ into the current constraint set (*i.e.*, it performs $\mathcal{A}_{\texttt{it}+1} = \mathcal{A}_{\texttt{it}} \cup \{(\overline{\mathbf{a}}, \overline{c_a})\}$), so as to construct a new more refined outer approximation $\mathscr{P}_{\texttt{it}+1}$ and to separate $\mathbf{x}^{\text{out}} \notin \mathscr{P}_{\texttt{it}+1}$. The process is repeated by (re-)optimizing over $\mathscr{P}_{\texttt{it}+1}$ at the next iteration, until the current outer optimal solution $\mathbf{x}^{\text{out}}$ becomes optimal (non-separable).

In this work we propose a `Projective Cutting-Planes` method that replaces the above separation sub-problem with the the following one.

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

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

The initialization of the `Projective Cutting-Planes` consists of the following. At the very first iteration, it performs a projection along $\mathbf{d}_1 = \mathbf{b}$, so as to directly advance along the direction with the fastest rate of objective function improvement – although we do not forbid using a different problem-specific direction $\mathbf{d}_1$. An initial feasible (inner) solution $\mathbf{x}_1$ is always needed, because the algorithms from this paper are not designed for problems in which it is difficult to decide whether (2.1) is feasible or not. When possible, we will use $\mathbf{x}_1 = \mathbf{0}_n$ as in the `Column Generation` models from Section 4.[1] In certain cases, one can also provide a set of initial constraints $\mathcal{A}_0$, *e.g.*, to impose simple initial bounds on the variables like $\mathbf{x} \geq \mathbf{0}_n$.

By solving $\texttt{project}(\mathbf{x}_1 \to \mathbf{d}_1)$ at iteration $\texttt{it} = 1$, the `Projective Cutting-Planes` determines the pierce point $\mathbf{x}_1 + t_1^*\mathbf{d}_1$ and generates a first-hit constraint $(\mathbf{a}, c_a) \in \mathcal{A}$. By assigning $\mathcal{A}_1 = \mathcal{A}_0 \cup \{(\mathbf{a}, c_a)\}$, we construct the first outer approximation $\mathscr{P}_1$. Then, the `Projective Cutting-Planes` iterates the following steps for all $\texttt{it} \geq 2$:

1. Select an interior solution $\mathbf{x}_{\texttt{it}}$, usually by taking a point on the segment joining $\mathbf{x}_{\texttt{it}-1}$ and $\mathbf{x}_{\texttt{it}-1} + t^*_{\texttt{it}-1}\mathbf{d}_{\texttt{it}-1}$, *i.e.*, on the segment between the previous interior solution and the last pierce point.
2. Consider the direction $\mathbf{d}_{\texttt{it}}$ pointing from $\mathbf{x}_{\texttt{it}}$ towards the current optimal outer solution before iteration `it`, *i.e.*, set $\mathbf{d}_{\texttt{it}} = \texttt{opt}(\mathscr{P}_{\texttt{it}-1}) - \mathbf{x}_{\texttt{it}}$. Notice that the objective function value can not deteriorate by advancing along $\mathbf{x}_{\texttt{it}} \to \mathbf{d}_{\texttt{it}}$, because $\mathbf{x}_{\texttt{it}}$ is a simple feasible solution and $\mathbf{d}_{\texttt{it}}$ points to the optimal outer solution of $\mathscr{P}_{\texttt{it}-1} \supseteq \mathscr{P}$ such that we have $\texttt{optVal}(\mathscr{P}_{\texttt{it}-1}) \geq \texttt{optVal}(\mathscr{P}) \geq \mathbf{b}^\top \mathbf{x}_{\texttt{it}}$ for a maximization problem.
3. Solve the projection sub-problem $\texttt{project}(\mathbf{x}_{\texttt{it}} \to \mathbf{d}_{\texttt{it}})$ to determine the maximum step length $t^*_{\texttt{it}}$ associated to the feasible solution $\mathbf{x}_{\texttt{it}} + t^*_{\texttt{it}}\mathbf{d}_{\texttt{it}}$ (a pierce point) and to a first-hit constraint $(\overline{\mathbf{a}}, \overline{c_a}) \in \mathcal{A}$.
4. If $t^*_{\texttt{it}} \geq 1$, return $\texttt{opt}(\mathscr{P}_{\texttt{it}-1})$ as the optimal solution of the initial LP (2.1) over $\mathscr{P}$.

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

   - set $\mathcal{A}_{\texttt{it}} = \mathcal{A}_{\texttt{it}-1} \cup \{(\overline{\mathbf{a}}, \overline{c_a})\}$ to obtain a new enlarged constraint set, corresponding to a more refined outer approximation $\mathscr{P}_{\texttt{it}}$ that excludes $\texttt{opt}(\mathscr{P}_{\texttt{it}-1})$;
   - determine a new current outer optimal solution $\texttt{opt}(\mathscr{P}_{\texttt{it}})$;
   - if $\mathbf{x}_{\texttt{it}} + t^*_{\texttt{it}}\mathbf{d}_{\texttt{it}}$ and $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ are close enough, stop and return a (quasi-)optimal solution $\texttt{opt}(\mathscr{P}_{\texttt{it}})$. 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 bound.

[1] If $\mathbf{0}_n$ is infeasible, one has to find other methods to generate a feasible solution $\mathbf{x}_1$. For example, one can take a feasible solution in an LP whose feasible area is a subset of $\mathscr{P}$, as for the robust optimization problem from Section 3.1.2.

– repeat from Step 1 after updating $\texttt{it} \leftarrow \texttt{it} + 1$.

The above algorithm is finitely convergent because it implicitly solves a separation sub-problem on $\texttt{opt}(\mathscr{P}_{\texttt{it}-1})$ at each iteration `it`, generalizing the standard `Cutting-Planes`. As hinted at Step 4, if the intersection sub-problem returns $t^*_{\texttt{it}} < 1$ at iteration `it`, then the solution $\texttt{opt}(\mathscr{P}_{\texttt{it}-1})$ is certainly separated by the first-hit constraint $(\overline{\mathbf{a}}, \overline{c_a})$. In pure theory, in the worst case, this algorithm ends up enumerating all constraints of $\mathscr{P}$ and it then eventually returns $\texttt{opt}(\mathscr{P})$. 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.

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

We deliberately presented so far a rather generic `Projective Cutting-Planes`, allowing different problem-specific adaptations, as for a standard `Cutting-Planes`. We prefer such a generic methodology to a would-be perfect pseudo-code that should be closely followed in all imaginable implementations.

A key question for any implementation is the choice of the interior point $\mathbf{x}_{\texttt{it}}$ at (Step 1 of) each iteration `it`. One might attempt to define $\mathbf{x}_{\texttt{it}}$ as the best feasible solution found up to iteration `it` (of best objective value), which actually reduces to assigning $\mathbf{x}_{\texttt{it}} = \mathbf{x}_{\texttt{it}-1} + t^*_{\texttt{it}-1}\mathbf{d}_{\texttt{it}-1}$, *i.e.*, $\mathbf{x}_{\texttt{it}}$ is defined as the last pierce point. While this aggressive strategy does perform well in certain settings, it may also lead to poor results in the long run for other problems — partly because $\mathbf{x}_{\texttt{it}}$ can fluctuate too much from iteration to iteration (as we will see in Section 5.5). In practice, the best results have often been obtained with a formula of the form $\mathbf{x}_{\texttt{it}} = \mathbf{x}_{\texttt{it}-1} + \alpha t^*_{\texttt{it}-1}\mathbf{d}_{\texttt{it}-1}$, 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 to touch the boundary of the polytope before fully converging [7].

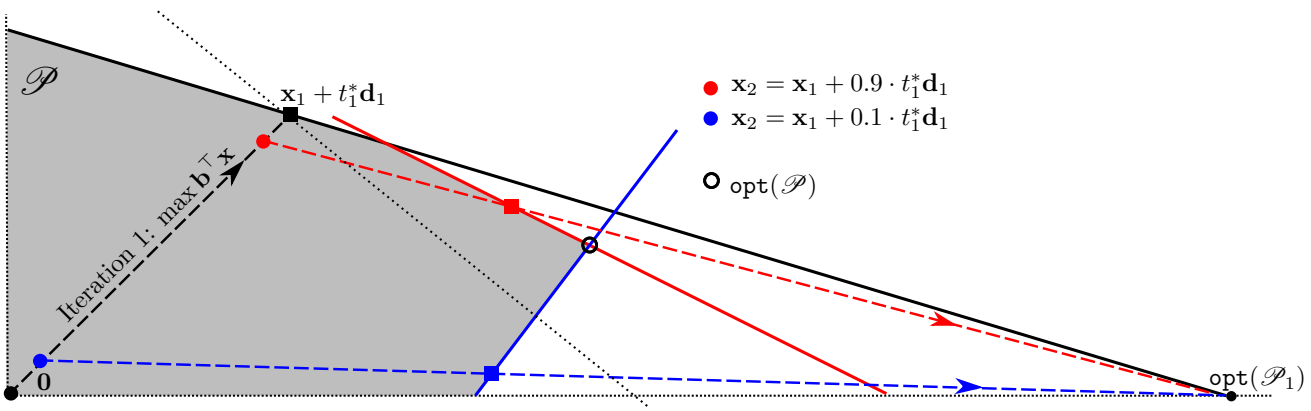

Figure 2: Intuitive illustration of two different choices of the interior point $\mathbf{x}_2$ at iteration 2. The red choice is more aggressive while the blue one is more cautious.

The only problem for which we do use the most aggressive choice $\mathbf{x}_{\texttt{it}} = \mathbf{x}_{\texttt{it}-1} + t^*_{\texttt{it}-1}\mathbf{d}_{\texttt{it}-1}$ is the graph coloring `Column Generation` model from Section 4.1. This choice has the advantage of improving the objective value $\mathbf{b}^\top\mathbf{x}_{\texttt{it}}$ at each new iteration `it`, because each projection $\mathbf{x}_{\texttt{it}} \rightarrow \mathbf{d}_{\texttt{it}}$ can only increase the objective value, as indicated at Step 2 above. This way, the lower bounds of the aggressive `Projective Cutting-Planes` are monotonically increasing (see Figures 4–5), eliminating the infamous "yo-yo" effect appearing very often in `Column Generation`. Graph coloring also differs from the other three problems explored in this paper in the sense that the above aggressive $\mathbf{x}_{\texttt{it}}$ choice does not lead to strong oscillations of the inner solutions $\mathbf{x}_{\texttt{it}}$ along the iterations `it` (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 $\mathbf{x}_1 + t^*_1\mathbf{d}_1$ by only advancing a little from $\mathbf{x}_2$ towards $\texttt{opt}(\mathscr{P}_1)$ — see how rapidly the red dashed arrow crosses the black doted line, *i.e.*, the level set of the last pierce point $\left\{\mathbf{x} \in \mathbb{R}^2_+ : \mathbf{b}^\top\mathbf{x} = \mathbf{b}^\top(\mathbf{x}_1 + t^*_1\mathbf{d}_1)\right\}$. 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 $\texttt{opt}(\mathscr{P}_1)$, 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 $\mathscr{P}_1$ (*i.e.*, of the largest triangle) than the solid red line.

2.2 Projection techniques for designing a fast intersection 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 more than two times slower than the separation iterations, the overall `Projective Cutting-Planes` would be too slow, *i.e.*, even by halving the number of iterations, it would remain slower than the standard `Cutting-Planes`. Since the projection sub-problem generalizes the separation one, it might seem difficult to design a projection algorithm that is no slower than the separation one. However, we will present several techniques that can bring us very close to this goal.

Let us first explain how the projection sub-problem $\texttt{project}(\mathbf{x} \to \mathbf{d})$ reduces to minimizing the following fractional program (for any feasible $\mathbf{x} \in \mathscr{P}$ or for any $\mathbf{d} \in \mathbb{R}^n$):

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

Given the value t^* that minimizes the above ratio, one can directly check that $t\mathbf{a}^\top \mathbf{d} \leq c_a - \mathbf{a}^\top \mathbf{x}$ 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 $\mathbf{a}^\top(\mathbf{x} + t\mathbf{d}) \leq \mathbf{a}^\top \mathbf{x}$ actually holds for all $t \in [0, \infty]$, simply because $\mathbf{a}^\top \mathbf{x} \leq c_a$ — which is true since $\mathbf{x} \in \mathscr{P}$. 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 ratio in (2.2.1). This also shows that it is enough to focus on the constraints $(\mathbf{a}, c_a) \in \mathcal{A}$ that satisfy $\mathbf{a}^\top \mathbf{d} > 0$ to solve the projection sub-problem.

We now discuss four techniques that use (2.2.1) to solve the projection sub-problem as rapidly as possible, with the aim of competing with the separation algorithms in terms of computational speed.

The first technique consists of extending the separation algorithm as applied on the model from Section 3.1, *without* simply calling the separation algorithm several times, which could be significantly too slow — as we will argue in the last paragraph of Section 3.1.3.1. In a nutshell, the constraints $\mathcal{A}$ of this problem are defined by extending an initial set of nominal constraints $\mathcal{A}_{\texttt{nom}}$ as follows: define a robust cut $(\mathbf{a} + \widehat{\mathbf{a}}, c_a)$ for each nominal constraint $(\mathbf{a}, c_a) \in \mathcal{A}_{\texttt{nom}}$ and for all deviations $\widehat{\mathbf{a}}$ of $\mathbf{a}$, *i.e.*, for all vectors $\widehat{\mathbf{a}} \in \mathbb{R}^n$ with at maximum Γ non-zero components such that $\widehat{a}_i \in \{-0.01 \cdot a_i, \ 0, \ 0.01 \cdot a_i\} \ \forall i \in [1..n]$. To solve the separation sub-problem on a given $\mathbf{x} \in \mathbb{R}^n$, one has to minimize $c_a - (\mathbf{a} + \widehat{\mathbf{a}})^\top \mathbf{x}$ over all $(\mathbf{a}, c_a) \in \mathcal{A}_{\texttt{nom}}$ and all deviations $\widehat{\mathbf{a}}$ of $\mathbf{a}$. The projection sub-problem (2.2.1) asks to minimize $\dfrac{c_a - (\mathbf{a} + \widehat{\mathbf{a}})^\top \mathbf{x}}{(\mathbf{a} + \widehat{\mathbf{a}})^\top \mathbf{d}}$ over the same $(\mathbf{a}, c_a)$ and the same $\widehat{\mathbf{a}}$ 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 algorithms, so that they can both require similar running times. More exactly, both sub-problems are essentially solved by iterating over all nominal constraints $\mathcal{A}_{\texttt{nom}}$, attempting at each element $(\mathbf{a}, c_a) \in \mathcal{A}_{\texttt{nom}}$ to decrease either the ratio $\dfrac{c_a - (\mathbf{a} + \widehat{\mathbf{a}})^\top \mathbf{x}}{(\mathbf{a} + \widehat{\mathbf{a}})^\top \mathbf{d}}$ or resp. the difference $c_a - (\mathbf{a} + \widehat{\mathbf{a}})^\top \mathbf{x}$.

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 classical LP using the Charnes–Cooper transformation [3]. This leads to an algorithm of the same complexity as the separation one, *i.e.*, 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. We will apply this technique to the `Column Generation` model for graph coloring in Section 4.1. Each constraint $(\mathbf{a}, c_a) = (\mathbf{a}, 1) \in \mathcal{A}$ corresponds 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. These stables are no-more-no-less than the integer solutions of the stable set polytope. 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 $\overline{a}_i \in \{0, \overline{\alpha}\}$, where $\overline{\alpha}$ 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 be harder in absolute terms than the ILP of the separation sub-problem, mainly because we solve them both using similar `Branch and Bound` methods in which the bounds are determined from continuous relaxations. To show this technique is really not limited to standard graph coloring, we will also provide brief experiments on a related problem with different constraints.

The fourth technique can be useful when the constraints $\mathcal{A}$ can be (implicitly or explicitly) enumerated by Dynamic Programming, as in many `Column Generation` models for combinatorial optimization problems. Typically, 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 enumerates 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 *Cutting-Stock* problem 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 to determine an underestimated feasible step length $t^h_{\mathtt{it}} \leq t^*_{\mathtt{it}}$ at certain iterations `it`, so that $\mathbf{x}_{\mathtt{it}} + t^h_{\mathtt{it}} \mathbf{d}_{\mathtt{it}}$ does not necessarily belongs to the boundary of $\mathscr{P}$ (*i.e.*, it is not necessary a pierce point). 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, one can prove that such $t^h_{\mathtt{it}}$ represents a feasible step length by simply calling the separation sub-problem on $\mathbf{x}_{\mathtt{it}} + t^h_{\mathtt{it}} \mathbf{d}_{\mathtt{it}}$, so that $t^h_{\mathtt{it}}$ can be seen as a heuristic step length. Given that the inner solutions $\mathbf{x}_{\mathtt{it}}$ are not usually chosen from the boundary points of $\mathscr{P}$ (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 $\mathscr{P}$ as a primal (master) polytope, obtaining an instance of the general LP (2.1) with unmanageably-many constraints. The first model is devoted to a robust optimization problem (Section 3.1) 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 many different methods have been proposed to define the set of acceptable coefficient deviations (*e.g.*, using linear or ellipsoid uncertainty sets). To avoid unessential complication, we here only focus on the robustness model from [6]; the reader is referred 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 the unfavorable direction.

3.1.1 The model with prohibitively-many constraints and their separation

Let us first consider a set $\mathcal{A}_{\mathtt{nom}}$ of nominal constraints that is small enough to be enumerated in practice, *i.e.*, there is not need of `Cutting-Planes` to solve the nominal version of the problem (with no robustness). We then associate to each $(\mathbf{a}, c_a) \in \mathcal{A}_{\mathtt{nom}}$ a prohibitively-large set $\mathtt{Dev}_\Gamma(\mathbf{a})$ of *deviation vectors* $\widehat{\mathbf{a}}$, *i.e.*, vectors $\widehat{\mathbf{a}} \in \mathbb{R}^n$ that have at maximum Γ non-zero components and that satisfy $\widehat{a}_i \in \{-\delta a_i,\ 0,\ \delta a_i\}\ \forall i \in [1..n]$, with $\delta = 0.01$ in practice. Each such deviation vector $\widehat{\mathbf{a}}$ yields a robust cut $(\mathbf{a} + \widehat{\mathbf{a}})^\top \mathbf{x} \leq c_a$, so that we can state $(\mathbf{a} + \widehat{\mathbf{a}}, c_a) \in \mathcal{A}$. In theory, each $\widehat{a}_i$ $(\forall i \in [1..n])$ 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 $\widehat{a}_i$ is either δa_i or $-\delta a_i$. One might find at

most $\binom{n}{\Gamma}2^{\Gamma}$ deviation vectors for each nominal constraint $(\mathbf{a}, c_a) \in \mathcal{A}_{\texttt{nom}}$, because there are $\binom{n}{\Gamma}$ ways to choose the non-zero components of $\widehat{\mathbf{a}}$ and each one of them can be either positive or negative, hence the 2^{Γ} factor.

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

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

Although this problem has a minimization objective unlike general LP (2.1), this has no impact on the steps of the (standard or new) `Cutting-Planes` method from Section 2. The only difference is that the feasible solutions (pierce points) determined by `Projective Cutting-Planes` represent upper (primal) bounds and not lower bounds. The last condition $a_i \in [\texttt{lb}_i, \texttt{ub}_i]$ in (3.1.1) represents initial constraints (in $\mathcal{A}_0$) that only impose simple bounds on the variables, most instances using $\texttt{lb}_i = 0 \ \forall i \in [1..n]$, *i.e.*, the variables are most often non-negative.

We will consider that the standard `Cutting-Planes` for the above (3.1.1) relies on the following separation sub-problem: given any solution $\mathbf{x} \in \mathbb{R}^n$, minimize $c_a - (\mathbf{a}+\widehat{\mathbf{a}})^{\top}\mathbf{x}$ over all $(\mathbf{a}, c_a) \in \mathcal{A}_{\texttt{nom}}$ and over all $\widehat{\mathbf{a}} \in \texttt{Dev}_{\Gamma}(\mathbf{a})$. For a fixed nominal constraint $(\mathbf{a}, c_a) \in \mathcal{A}_{\texttt{nom}}$, the strongest possible deviation $\widehat{\mathbf{a}}_{\mathbf{x}}^{\top}\mathbf{x}$ of $(\mathbf{a}, c_a)$ with respect to $\mathbf{x}$ is determined by solving $\widehat{\mathbf{a}}_{\mathbf{x}} = arg\ \max \left\{ \widehat{\mathbf{a}}^{\top}\mathbf{x} : \widehat{\mathbf{a}} \in \texttt{Dev}_{\Gamma}(\mathbf{a}) \right\}$. To determine this $\widehat{\mathbf{a}}_{\mathbf{x}}$, one needs to find the largest Γ absolute values in the terms of the sum $\mathbf{a}^{\top}\mathbf{x} = \sum_{i=1}^{n} a_i x_i$. The strongest possible deviation $\widehat{\mathbf{a}}_{\mathbf{x}}^{\top}\mathbf{x}$ is then 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 $|a_1x_1|$, $|a_2x_2|$, $\ldots|a_nx_n|$ as above) can be determined in $O(n)$ time. We use a straightforward partial-sorting algorithm essentially described as follows: iterate over each $i \in [1..n]$ and attempt at each iteration 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 $\Gamma \geq 10$.* □

The standard `Cutting-Planes` algorithm based on the above sub-problem is not identical to the one from [6], because their algorithm returns multiple robust cuts at each separation call. This idea might be very effective in practice both for the `Cutting-Planes` and for the `Projective Cutting-Planes`. However, for now, the goal of this study is to compare the projection and the separation sub-problem in a standard setting, and so, we prefer a canonical approach with a unique (robust) cut per iteration.

3.1.2 `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). Considering the intersection algorithm as a black-box component, the implementation of most remaining components of the `Projective Cutting-Planes` are actually rather straightforward.

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

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 $(\mathbf{a}, c_a) \in \mathcal{A}_{\texttt{nom}}$, generate a constraint $\mathbf{a}^{\top}\mathbf{x} + 2\delta|\mathbf{a}|^{\top}\mathbf{x} \leq c_a$, where $|\mathbf{a}| = [|a_1| \ |a_2| \ \ldots|a_n|]^{\top}$. If the variables $\mathbf{x}$ are all non-negative (as in most

[2] This 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. In this case, by inserting v_{new}, the tree size might 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 an 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`.

instances), than any solution $\mathbf{x}$ that satisfies $\mathbf{a}^\top\mathbf{x} + 2\delta|\mathbf{a}|^\top\mathbf{x} \leq c_a \ \forall(\mathbf{a},\ c_a) \in \mathcal{A}_{\texttt{nom}}$ is feasible with regards to all robust cuts — because a robust cut uses a deviation vector $\widehat{\mathbf{a}}$ that satisfies $\widehat{\mathbf{a}} \leq \delta|\mathbf{a}|$, so that $(\mathbf{a}+\widehat{\mathbf{a}})^\top\mathbf{x} \leq \mathbf{a}^\top\mathbf{x} + 2\delta|\mathbf{a}|^\top\mathbf{x} \leq c_a$. 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 $\mathbf{d}_1 = \texttt{opt}(\mathscr{P}_0) - \mathbf{x}_1$, where $\mathscr{P}_0$ 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 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 $\dfrac{c_a - (\mathbf{a}+\widehat{\mathbf{a}})^\top\mathbf{x}}{(\mathbf{a}+\widehat{\mathbf{a}})^\top\mathbf{d}}$ over all nominal constraints $(\mathbf{a}, c_a) \in \mathcal{A}_{\texttt{nom}}$ and over all deviation vectors $\widehat{\mathbf{a}} \in \texttt{Dev}_\Gamma(\mathbf{a})$ such that $(\mathbf{a}+\widehat{\mathbf{a}})^\top\mathbf{d} > 0$. As for the separation sub-problem, the intersection algorithm iterates over all nominal constraints $\mathcal{A}_{\texttt{nom}}$, in an attempt to reduce the above ratio (the step length) at each $(\mathbf{a}, c_a) \in \mathcal{A}_{\texttt{nom}}$, *i.e.*, for each $(\mathbf{a}, c_a) \in \mathcal{A}_{\texttt{nom}}$ it is possible to find several increasingly stronger $\widehat{\mathbf{a}}$ that gradually decrease the step length (the above ratio). Let t_i^* denote the optimal step length obtained after considering the robust cuts associated to the first i constraints from $\mathcal{A}_{\texttt{nom}}$. It is clear that t_i^* can only decrease as i grows. Starting with $t_0 = 1$, the intersection 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 $\mathcal{A}_{\texttt{nom}}$.

2. Determine the strongest deviation vector $\widehat{\mathbf{a}}_{t\mathbf{d}}$ with respect to $\mathbf{x} + t\mathbf{d}$ by maximizing:

$$\widehat{\mathbf{a}}_{t\mathbf{d}} = arg\ \max\left\{\widehat{\mathbf{a}}^\top(\mathbf{x} + t\mathbf{d}) : \widehat{\mathbf{a}} \in \texttt{Dev}_\Gamma(\mathbf{a})\right\}. \tag{3.1.2}$$

   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 $(\mathbf{a} + \widehat{\mathbf{a}}_{t\mathbf{d}})^\top(\mathbf{x} + t\mathbf{d}) \leq c_a$, then $\mathbf{x} + t\mathbf{d}$ is feasible with regards to the first i constraints from $\mathcal{A}_{\texttt{nom}}$, because any deviation vector $\widehat{\mathbf{a}} \in \texttt{Dev}_\Gamma(\mathbf{a})$ satisfies $\widehat{\mathbf{a}}^\top(\mathbf{x} + t\mathbf{d}) \leq \widehat{\mathbf{a}}_{t\mathbf{d}}^\top(\mathbf{x} + t\mathbf{d})$. The algorithm can thus return $t_i^* = t$ in this case. Otherwise, the robust cut $(\mathbf{a} + \widehat{\mathbf{a}}_{t\mathbf{d}}, c_a)$ leads to a smaller feasible step length:

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

4. If $t' = 0$, then the overall intersection algorithm returns $t^* = 0$ without checking the remaining nominal constraints. Indeed, 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 $\widehat{\mathbf{a}}_{td}$ determined via (3.1.2) is not the strongest one with regards to $\mathbf{x} + t'\mathbf{d}$, because $\widehat{\mathbf{a}}_{td}$ generates the highest deviation in (3.1.2) with regards to a different point. But there might exist a different robust cut $(\mathbf{a} + \widehat{\mathbf{a}}_{t'\mathbf{d}}, c_a)$ for the same nominal constraint such that $\widehat{\mathbf{a}}_{t'\mathbf{d}}^\top(\mathbf{x} + t'\mathbf{d}) > \widehat{\mathbf{a}}_{t\mathbf{d}}^\top(\mathbf{x} + t'\mathbf{d})$. 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 $(\mathbf{a}, c_a) \in \mathcal{A}_{\texttt{nom}}$ one by one, the step length returned at the last constraint of $\mathcal{A}_{\texttt{nom}}$ provides the sought t^* value.

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

In theory, a nominal constraint can lead the above intersection algorithm to repeat the steps 2-5 many times for each value of 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 (for all i). For many nominal constraints $(\mathbf{a}, c_a) \in \mathcal{A}_{\texttt{nom}}$, 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 cases, 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. A exact separation algorithm could not stop earlier, because $c_a - (\mathbf{a} + \widehat{\mathbf{a}}_{\mathbf{x}})^\top \mathbf{x}$ can certainly decrease up to the last nominal constraint $(\mathbf{a}, c_a)$. As such, an intersection iteration can become even faster than the separation one in certain cases, *e.g.*, 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 in all, 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 intersection 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 in practice, a third or a fourth call might well be needed at many iterations. More generally, one of the goals of the paper is to explore (other) techniques that can make the intersection algorithms compete well in terms of speed with the separation algorithms.

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:

$$\min \left\{ \mathbf{b}^\top \mathbf{x} : \ \mathbf{B}\mathbf{x} + \mathbf{A}\mathbf{y} \geq \mathbf{c}, \ \mathbf{x} \in \mathbb{Z}^n_+, \ \mathbf{y} \geq \mathbf{0} \right\}. \tag{3.2.1}$$

Generally speaking, $\mathbf{x}$ can represent the main (design) decisions. 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 $\mathbf{x}$, under the constraint that these design decisions need 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. With these notations, we actually swapped the interpretation of $\mathbf{x}$ and $\mathbf{y}$ from many Benders decomposition papers [5], for the sake of notational consistency throughout the current paper

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). We do not (yet) propose a projection algorithm for this most general Benders model. However, the above Benders LP (3.2.1) with $\mathbf{f} = \mathbf{0}_n$ is still rather general, because it is often realistic to consider that $\mathbf{f} = \mathbf{0}_n$ can represent zero flow costs, *e.g.*, there is 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 $\min \left\{ \mathbf{0}^\top \mathbf{y} : \ \mathbf{A}\mathbf{y} \geq \mathbf{c} - \mathbf{B}\mathbf{x}, \ \mathbf{y} \geq \mathbf{0} \right\} = 0$. Writing the dual of this LP, any dual feasible solution $\mathbf{u}$ has to belong to $\mathcal{P} = \left\{ \mathbf{u} \geq \mathbf{0}_m : \mathbf{A}^\top \mathbf{u} \leq \mathbf{0}_n \right\}$, 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 the above condition, (3.2.1) can be equivalently written in the following Benders decomposition form:

$$\min \mathbf{b}^\top \mathbf{x} \tag{3.2.2a}$$

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

where

$$\mathcal{P} = \left\{ \mathbf{u} \geq \mathbf{0}_m : \mathbf{A}^\top \mathbf{u} \leq \mathbf{0}_n \right\} \tag{3.2.3}$$

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) and, although $\mathscr{P}$ is now a discrete set, the Benders' method applies the `Cutting-Planes` algorithm exactly as described in Section 2, iteratively refining a sequence of discrete sets $\mathscr{P}_1 \supsetneq \mathscr{P}_2 \supsetneq \mathscr{P}_2 \supsetneq \cdots \supset \mathscr{P}$. At each iteration `it`, we say $\mathscr{P}_{\mathtt{it}}$ 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 the fact that $\mathbf{x}$ is integer in (3.2.2c), so that $\mathtt{opt}(\mathscr{P}_{\mathtt{it}})$ needs to be determined using ILP solver instead of an LP solver. Given the current optimal solution $\mathbf{x} = \mathtt{opt}\,(\mathscr{P}_{\mathtt{it}})$, the separation algorithm solves the following LP to (try to) cut $\mathbf{x}$ off.

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

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 rays have an unbounded objective value and their quality is more difficult to compare — 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). The main consequence of this integrality is that each optimal solution $\mathtt{opt}(\mathscr{P}_{\mathtt{it}})$ has to be determined using an ILP solver, which becomes the most important computational bottleneck of the overall `Projective Cutting-Planes`. Indeed, we will see that the (projection or separation) sub-problem requires solving a standard LP which is substantially easier in computational speed terms than an ILP. This can actually be seen as a reason for adopting the `Projective Cutting-Planes`: the switch to the intersection sub-problem induces no relevant computational overhead.

We will also examine the linear relaxation of the Benders reformulation, replacing $\mathbf{x} \in \mathbb{Z}^n_+$ with $\mathbf{x} \in \mathbb{R}^n_+$. For this problem version, the only difference compared to the general large-scale LP (2.1) is that the objective function is minimized and not maximized. This does not essentially change our `Cutting-Planes` algorithms, for the same reasons mentioned in Section 3.1.1 for the robust minimization problem.

A key question concerns the choice of the interior point $\mathbf{x}_{\mathtt{it}}$ at each iteration $\mathtt{it} \geq 1$. 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 $\mathtt{it} > 1$, experiments suggest it is preferable to choose an interior point $\mathbf{x}_{\mathtt{it}}$ relatively far from the boundary — as for the robust optimization problem previously studied. As in Section 3.1.2, we apply the formula $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + \alpha t^*_{\mathtt{it}-1} \mathbf{d}_{\mathtt{it}-1}$, 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 $\mathbf{x}_{it} + t^*_{it}\mathbf{d}_{it}$ returned by the intersection 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 $\mathbf{x}_{it}+t^*_{it}\mathbf{d}_{it}$. At least when $\mathbf{x}$ represents decisions to install (transmission) facilities, there is generally no reason to forbids 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 `project`$(\mathbf{x} \to \mathbf{d})$ requires finding:

(i) the maximum step length $t^* \geq 0$ such that $\mathbf{x} + t^*\mathbf{d}$ respects all constraints (3.2.2b);

(ii) a vector $\mathbf{u} \in \mathcal{P}$ such that the associated constraint (3.2.2b) is satisfied with equality by the pierce point $\mathbf{x} + t^*\mathbf{d}$. We do not impose the condition that the returned $\mathbf{u}$ has 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 $-t^* (\mathbf{Bd})^\top \mathbf{u} \leq (\mathbf{Bx})^\top \mathbf{u} - \mathbf{c}^\top \mathbf{u} \; \forall \mathbf{u} \in \mathcal{P}$. 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 $-(\mathbf{Bd})^\top \mathbf{u} \leq 0$ would allow t^* to be arbitrarily large. As such, we hereafter only focus on the vectors $\mathbf{u} \in \mathcal{P}$ such that $-(\mathbf{Bd})^\top \mathbf{u} > 0$, and so, t^* can be determined by solving the following linear-fractional program:

$$t^* = \min \left\{ \frac{(\mathbf{Bx} - \mathbf{c})^\top \mathbf{u}}{-(\mathbf{Bd})^\top \mathbf{u}} : \; \mathbf{u} \in \mathcal{P}, \; -(\mathbf{Bd})^\top \mathbf{u} > 0 \right\} \tag{3.2.5}$$

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

$$t^* = \min \; (\mathbf{Bx} - \mathbf{c})^\top \overline{\mathbf{u}} \tag{3.2.6a}$$

$$\mathbf{A}^\top \overline{\mathbf{u}} \leq \mathbf{0}_n \tag{3.2.6b}$$

$$-(\mathbf{Bd})^\top \overline{\mathbf{u}} = 1 \tag{3.2.6c}$$

$$\overline{\mathbf{u}} \geq \mathbf{0}_m \tag{3.2.6d}$$

It is not difficult to check that the above change of variable $\mathbf{u} \to \overline{\mathbf{u}}$ 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 $\overline{\mathbf{u}}$ 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 The general Benders `Projective Cutting-Planes` applied on a network design problem

We introduced the `Projective Cutting-Planes` so far in a rather generic Benders model (3.2.2a)–(3.2.2c), because otherwise the understanding of the main projection ideas could have been impaired by 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 or telecommunication network $G = (V, E)$. 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 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 represent 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:

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

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

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

$$b_{\rm wd} x_{ij} - y_{ij} - y_{ji} \geq 0,\ \forall \{i,j\} \in E,\ i<j \tag{3.2.7d}$$

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

The principles behind 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} indicates 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 $\{i,j\}$ times the bandwidth $b_{\rm wd}$ 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) in decision variables $\mathbf{y}$ that can be dualized. 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 $b_{\rm wd} x_{ij}$ 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 $\sum_{i\in T} f_i u_i - \sum_{\{i,j\}\in E} b_{\rm wd} x_{ij} u_{ij} \leq 0$. 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):

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

$$\mathscr{P}\left\{ \begin{array}{ll} \displaystyle\sum_{i\in T} f_i u_i - \sum_{\{i,j\}\in E} b_{\rm wd} x_{ij} u_{ij} \leq 0\ \forall \mathbf{u} \in \mathcal{P} \text{ s. t. } \mathbf{1}^\top \mathbf{u} = 1 & (3.2.8b) \\ \mathbf{x} \in \mathbb{Z}^n_+, & (3.2.8c) \end{array} \right.$$

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.

$$\mathcal{P}\left\{ \begin{array}{ll} y_{ij}:\ -u_{ij} - u_i + u_j \leq 0\ \ \forall \{i,j\} \in E,\ i<j & (3.2.9a) \\ y_{ji}:\ -u_{ij} - u_j + u_i \leq 0\ \ \forall \{i,j\} \in E,\ i<j & (3.2.9b) \\ \mathbf{u} \geq \mathbf{0} & (3.2.9c) \end{array} \right.$$

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 a reasonable assumption in certain applications, e.g., if the transmission capacities correspond to acquiring bandwidth from a telecommunication carrier or to constructing (water or gas) pipes of arbitrary diameter. Furthermore, even if only the integer model is relevant, the linear relaxation could be needed by a* `Branch and Bound` *(B&B) 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 general large-scale LP (2.1). The `Projective Cutting-Planes` described in Section 3.2.2 for the general Benders reformulation can be directly translated to solve (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 for the robust optimization problem, confirming ideas used in interior point algorithms.

3.2.4.3 The intersection sub-problem

Consider an interior point $\mathbf{x} \in \mathbb{R}^n_+$ that satisfies all the 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 $(\mathbf{Bx})^\top \mathbf{u}$ 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 $(\mathbf{Bx})^\top \mathbf{u}$ 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 $(\mathbf{Bd})^\top \mathbf{u}$ 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:

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

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 standard LP. Accordingly, after writing $\overline{\mathbf{u}} = \dfrac{\mathbf{u}}{-\sum_{\{i,j\} \in E} b_{\text{wd}} d_{ij} u_{ij}}$, (3.2.10) becomes equivalent to:

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

$$-\overline{u}_{ij} - \overline{u}_i + \overline{u}_j \leq 0 \ \ \forall \{i,j\} \in E, \ i < j \tag{3.2.11b}$$

$$-\overline{u}_{ij} - \overline{u}_j + \overline{u}_i \leq 0 \ \ \forall \{i,j\} \in E, \ i < j \tag{3.2.11c}$$

$$-\sum_{\{i,j\} \in E} b_{\text{wd}} d_{ij} \overline{u}_{ij} = 1 \tag{3.2.11d}$$

$$\overline{\mathbf{u}} \geq \mathbf{0}, \tag{3.2.11e}$$

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 $\mathscr{P}$ 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 master 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:

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

The dual LP is:

$$\mathscr{P} \left\{ \begin{array}{rll} & \max \mathbf{b}^\top \mathbf{x} & \\ y_a: & \mathbf{a}^\top \mathbf{x} \leq c_a & \forall (\mathbf{a}, c_a) \in \mathcal{A} \\ & \mathbf{x} \geq \mathbf{0}_n & \end{array} \right. \tag{4.2}$$

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

The standard graph coloring belongs to a rather large class of coloring problems involving 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. In the most standard form, the graph coloring problem can be directly formulated as a set covering problem: find 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 $c_a = 1$ (each color counts once).

The `Column Generation` method optimizes (4.2) by solving at each iteration the separation sub-problem $\max_{(\mathbf{a},1)\in\mathcal{A}} \mathbf{a}^\top \mathbf{x} - 1$, where $\mathbf{x}$ is the current (outer) optimal solution $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ at iteration `it`. In standard graph coloring, the constraints $\mathcal{A}$ are constructed from the standard stables of G, so that the above separation sub-problem reduces to a 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, in the sense that it has no particularly skewed constraints. Accordingly, there seems to be little potential in analyzing, reformulating or reinterpreting the very simple coloring constraints using some ad-hoc techniques; 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 $b_i = 1 \ \forall i \in V = [1..n]$, *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 to vertices. Multiply-colored vertices could naturally arise in the 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 $\mathbf{b}^\top \mathbf{x}_1,\ \mathbf{b}^\top \mathbf{x}_2,\ \mathbf{b}^\top \mathbf{x}_3, \ldots$ of the `Projective Cutting-Planes` remain valid.

4.1.2 The `Projective Cutting-Planes` for Graph Coloring

Before presenting the intersection algorithm in Section 4.1.3, we notice that the main steps of the `Projective Cutting-Planes` from Section 2 can be directly applied on the dual `Column Generation` program (4.2), without needing many customizations. The choice of $\mathbf{x}_{\mathtt{it}}$ is given by $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}$ at each iteration $\mathtt{it} > 1$, so that $\mathbf{x}_{\mathtt{it}}$ becomes the best feasible solution found so far (the last pierce point); graph coloring is the only problem from this work where this choice leads to good results in the long run. Regarding the very first iteration, we take $\mathbf{x}_1 = \mathbf{0}_n$ because $\mathbf{0}_n \in \mathscr{P}$ and we construct $\mathbf{d}_1$ by assigning to each component $i \in V = [1..n]$ the value $\frac{1}{\mathtt{stab\text{-}size}(i)}$, where $\mathtt{stab\text{-}size}(i)$ is the size of the stable that contains i in a given initial feasible coloring (determined with heuristics as stated in Section 5.3.2, Footnote 8, p. 31). The use of an initial feasible coloring is useful for warm starting reasons and it brings several advantages:

- The heuristic solution provides an initial set of stables (constraints in the dual LP), so as to start from the very first iteration with a reasonable outer approximation $\mathscr{P}_1 \supsetneq \mathscr{P}$.
- The first outer approximation $\mathscr{P}_1$ obtained as above leads to a first upper bound $\mathbf{b}^\top \mathtt{opt}(\mathscr{P}_1)$ that is equal to the number of colors used by the heuristic coloring. Referring to this (quality) upper bound available since the very first iteration, one can better estimate the gap of the 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 $\mathbf{0}_n \to \mathbf{1}_n$, we would have obtained a very first pierce point $t^*_1\mathbf{1}_n = \frac{1}{\omega}\mathbf{1}_n$ that might correspond to multiple constraints associated to multiple cliques of maximum size ω. If one then takes $\mathbf{x}_2 = \mathbf{x}_1 + t^*_1\mathbf{1}_n = \frac{1}{\omega}\mathbf{1}_n$, the second projection can return $t^*_2 = 0$ because of a second clique of size ω. If this repeats a third or a fourth time, the process could stall for too many iterations, generating a form of degeneracy (like the one described later in Section 4.1.4.3).

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 far simpler projection sub-problem that ca be formulated (Section 4.1.4) as a pure LP. The disadvantage is that the new model is overly-constrained: it inserts artificial constraints in (4.2), because $\mathcal{P}$ does not contain only legitimate standard stables. However, each feasible inner solution $\mathbf{x}_{\mathtt{it}}$ 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 $\mathbf{b}^\top\mathbf{x}_{\mathtt{it}}$ calculated on the second (over-constrained) model can be very fast and can compete rather well with the bounds of 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 Intersection Sub-Problem with Standard Stables

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

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

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

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

This is an integer linear-fractional program that will be transformed into 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 $\overline{a}_i \in \{0, \overline{\alpha}\}\ \forall i \in [1..n]$. 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 $\min\left\{1 - \mathbf{x}^\top\mathbf{a} :\ \mathbf{a} \in \mathcal{P}_{0-1}\right\}$, where $\mathbf{x} \in \mathbb{R}^n_+$ is the current outer optimal 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 $\texttt{conv}(\mathcal{P}_{0-1})$ of the set $\mathcal{P}_{0-1}$ of standard stables is called the stable set polytope in the literature of stable set formulations.

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] as used in Section 3.2.3. Accordingly, let us consider a change of variables $\overline{\mathbf{a}} = \dfrac{\mathbf{a}}{\mathbf{d}^\top \mathbf{a}}$ and $\overline{\alpha} = \dfrac{1}{\mathbf{d}^\top \mathbf{a}}$; we will prove that (4.1.1.a)–(4.1.1.c) is completely equivalent to:

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

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

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

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

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

To prove the equivalence, let us first show that the change of variables $\mathbf{a} \rightarrow \overline{\mathbf{a}}, \overline{\alpha}$ 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 $\overline{a}_i = \dfrac{a_i}{\mathbf{d}^\top \mathbf{a}}$ $\forall i \in [1..n]$ and $\overline{\alpha} = \dfrac{1}{\mathbf{d}^\top \mathbf{a}} > 0$. Conversely, a feasible solution of (4.1.2.a)–(4.1.2.e) can be reversely mapped to $\mathbf{a} = \dfrac{1}{\overline{\alpha}} \cdot \overline{\mathbf{a}}$, which is a feasible solution of the initial program. To show this, first notice that the expression $\dfrac{1}{\overline{\alpha}} \cdot \overline{\mathbf{a}}$ is consistent in the sense that $\overline{\alpha} \neq 0$, because otherwise $\overline{\alpha} = 0$ would make $\overline{\mathbf{a}} = \mathbf{0}_n$ via (4.1.2.d), so that (4.1.2.c) would be certainly 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 $\overline{\alpha} - \mathbf{x}^\top \overline{\mathbf{a}} = \dfrac{\overline{\alpha}}{\mathbf{d}^\top \overline{\mathbf{a}}} - \dfrac{\mathbf{x}^\top \overline{\mathbf{a}}}{\mathbf{d}^\top \overline{\mathbf{a}}} = \dfrac{1}{\mathbf{d}^\top \mathbf{a}} - \dfrac{\mathbf{x}^\top \mathbf{a}}{\mathbf{d}^\top \mathbf{a}}$.

4.1.3.1 Solving our Disjunctive LP can be in theory as hard as solving 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 $\overline{a}_i \in \{0, \overline{\alpha}\}$. Both these constrains break the continuity in a similar manner and we find *no* deep fundamental 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 standard `Branch and Bound` algorithm that constructs a branching tree in which each node corresponds to a relaxation that lifts certain continuity-breaking constraints.

However, in practice, we solve both problems 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 be in theory 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 $\overline{\mathbf{a}}$ of the Disjunctive LP (*i.e.*, discrete in the sense that $\overline{a}_i \in \{0, \overline{\alpha}\}$ $i \in [1..n]$); it does not see $\overline{a}_i \in \{0, \overline{\alpha}\}$ somehow similar to an integrality constraint (similar to $\frac{\overline{a}_i}{\overline{\alpha}} \in \{0,1\}$). In fact, it does not even have the information that relaxing (lifting) a disjunctive constraint $\overline{a}_i \in \{0, \overline{\alpha}\}$ allows $\overline{a}_i$ to take a fractional value in the interval $[0, \overline{\alpha}]$. This also implies that `cplex` does not see a fractional $\overline{a}_i \in (0, \overline{\alpha})$ as a real value in the interval of two "discrete" bounds 0 and $\overline{\alpha}$, while for the ILP it can use elaborately-tuned branching rules based on evaluating the distance from each fractional variable to its bounds 0 and 1.

We did attempt to compensate the above advantage of the `cplex` 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 of size $k = 4$ at the root node of the branching tree before starting

[3] The 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 $\overline{a}_1 + \overline{a}_2 \leq 3.5\overline{\alpha}$ could be reduced to $\overline{a}_1 + \overline{a}_2 \leq 3\overline{\alpha}$.

to branch (on the disjunctive constraints). At this root node, all continuity-breaking 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 at Section 4.1.4.2, searching at each iteration to separate the current solution using a k-clique inequality $\sum_{i\in\mathscr{C}} a_i \leq 1$ (associated to a clique $\mathscr{C}$ of size k), equivalent to $\sum_{i\in\mathscr{C}} \overline{a}_i \leq \overline{\alpha}$ in the Charnes-Cooper reformulation. Whenever these 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 very 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 could have easily considered 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 each such constraint is only active only when $a_i = 1$. Applying the discrete Charnes–Cooper transformation, (4.1.3a) could be directly translated to (4.1.3b).

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

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

An important consequence of the above development concerns the potential extensions and generalizations of this discrete Charnes-Cooper transformation. If the sub-problem had any other linear constraints instead of $\overline{a}_i + \overline{a}_j \leq 1\ \forall\{i,j\} \in E$, 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 $\overline{a}_i \in \{0,\overline{\alpha}\}$, *i.e.*, a similar form of continuity breaking constraints. We focused on standard graph coloring only because it is a problem with no particularly skewed constraints and we want to avoid impairing the general understanding with the specific complexities of a particular problem.

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

We hereafter focus on a second coloring model constructed using a broader notion of reinforced relaxed stables (RR-stables) as specified by Definition 2 below. The constrains of (4.2) are given by $(\mathbf{a}, 1) \in \mathcal{A} \Longleftrightarrow \mathbf{a} \in \mathcal{P}$, where $\mathcal{P}$ is an auxiliary polytope of RR-stables, as defined by a set $\mathcal{R}$ of constraints formalized via (4.1.5) further below. This last coloring section 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;
- We discuss a special degenerate case (null step length) that can arise in the new model in Section 4.1.4.3. We will see we need to define $\mathbf{x}_{\mathtt{it}} = \alpha(\mathbf{x}_{i-1} + t^*_{i-1}\mathbf{d}_{i-1})$ with $\alpha = 0.9999$ to avoid such form of 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 $\mathtt{conv}(\mathcal{P}_{0-1})$. We construct $\mathcal{P} \supsetneq \mathtt{conv}(\mathcal{P}_{0-1})$ by*

adding reinforcing cuts to $\texttt{conv}(\mathcal{P}_{0-1})$ *in several stages. At first, we simply only consider the edge inequalities* $a_i + a_j \leq 1\ \forall\{i,j\} \in E$*, so that the initial outer approximation of* $\texttt{conv}(\mathcal{P}_{0-1})$ *only contains simple relaxed stables (solutions of the fractional stable set polytope). We then continue building the outer approximation by by adding inequalities like* $a_i + a_j + a_k \leq 1\ \forall\{i,j\},\{i,k\},\{j,k\} \in E$*, followed by 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 larger 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

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

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

where $\mathcal{P}$ from Definition 2 is formalized by formula 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.

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

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 $\overline{\mathbf{a}} = \dfrac{\mathbf{a}}{\mathbf{d}^\top \mathbf{a}}$ and $\overline{\alpha} = \dfrac{1}{\mathbf{d}^\top \mathbf{a}}$, one can show that (4.1.4) is completely equivalent to:

$$t^* = \min\ \overline{\alpha} - \mathbf{x}^\top \overline{\mathbf{a}} \tag{4.1.6a}$$

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

$$\mathbf{d}^\top \overline{\mathbf{a}} = 1 \tag{4.1.6c}$$

$$\overline{\mathbf{a}} \geq \mathbf{0}_n,\ \overline{\alpha} \geq 0 \tag{4.1.6d}$$

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) from Section 4.1.3. Following this proof point by point, the only difference is that the above new programs are pure LPs and we no longer have disjunctive or integrality constraints of the form $\overline{a}_i \in \{0, \overline{\alpha}\}$ or resp. $a_i \in \{0,1\}$. Even without these constraints, all feasible solutions of (4.1.6a)–(4.1.6d) still satisfy $\overline{\alpha} \neq 0$, because $\overline{\alpha} = 0$ would also impose $\overline{\mathbf{a}} = \mathbf{0}_n$, since (4.1.6b) contains all edge inequalities of the form $\overline{a}_i + \overline{a}_j \leq \overline{\alpha}\ \forall\{i,j\} \in E$. However, even if these edge inequalities did not exist, this Charnes-Cooper transformation is very general and it would still work for any constraints $\mathcal{R}$ defining $\mathcal{P}$.[4]

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 are slightly modified in the above LP formulation, *i.e.*, we use $\mathbf{e}^\top \overline{\mathbf{a}} \leq \overline{\alpha}$ 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 $\overline{\alpha}$, 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 this separation sub-problem and to accelerate the `cut generation` for (4.1.6a)-(4.1.6d); this is the goal of the next section.

[4]For any constraints (4.1.6b), any feasible solution $(\overline{\mathbf{a}}, \overline{\alpha})$ with $\overline{\alpha} = 0$ 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 $\mathbf{a} + z\overline{\mathbf{a}}$ of $\mathcal{P}$, *i.e.*, $\mathbf{a} + z\overline{\mathbf{a}}$ is feasible in (4.1.5) for all $z \geq 0$. To check this, notice that $\mathbf{e}^\top(\mathbf{a} + z\overline{\mathbf{a}}) \leq 1\ \forall(\mathbf{e},1) \in \mathcal{R}$ follows from $\mathbf{a} \in \mathcal{P}$ and $\mathbf{e}^\top \overline{\mathbf{a}} \leq 0\ \forall(\mathbf{e},1) \in \mathcal{R}$, which holds because $\overline{\alpha} = 0$ 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 $\mathbf{a} + z\overline{\mathbf{a}}$ in (4.1.4) converges to

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

4.1.4.2 The `cut generation` for the LP formulation 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 $\mathbf{d}$, *i.e.*, they remain the same for all iterations of the overall `Projective Cutting-Planes`. As such, after solving an intersection sub-problem `project`$(\mathbf{x}_{\texttt{it}} \rightarrow \mathbf{d}_{\texttt{it}})$ at some iteration `it` of the overall algorithm, one can keep all generated cuts (4.1.6b) and only update (4.1.6c) to move to the next iteration $\texttt{it}+1$.

The reinforcing cuts $\mathcal{R}$ are composed of six classes of inequalities (a)–(f) presented in greater detail in Appendix A.2.1. Their design has been inspired by reasearch in valid inequalities for the maximum stable problem [10], although there is a difference compared to the existing literature: we will need to solve too many (RR-)stable problems to afford using clique inequalities of arbitrary size, and so, we will only use cliques of bounded size k. In fact, the first four cut classes (a)–(d) are actually static and they 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, by repeatedly solving a separation sub-problem. This sub-problem reduces to finding the maximum of $\max_{(\mathbf{e},1)\in\mathcal{R}} \mathbf{e}^\top\overline{\mathbf{a}} - \overline{\alpha}$ and $\max_{(\mathbf{f},0)\in\mathcal{R}} \mathbf{f}^\top\overline{\mathbf{a}}$, for the current optimal solution $(\overline{\mathbf{a}},\overline{\alpha})$ at each `cut generation` iteration. The `cut generation` algorithm is described in greater detail in Appendix A.2.2.

The cut class (e) represents odd hole inequalities that can be separated in polynomial time by applying the Dijkstra algorithm on a bipartite graph with $2n+2$ vertices. The class (f) represents k-clique cuts of the form $\sum_{v\in\mathscr{C}} a_v \leq 1$, equivalent to $\sum_{v\in\mathscr{C}} \overline{a}_v \leq \overline{\alpha}$ in (4.1.6b), where $\mathscr{C}$ is a clique with k vertices. The separation of these cuts is computationally more expensive, because it requires solving a maximum weight clique (NP hard) problem with bounded size k. This problem is solved by a dedicated `Branch & Bound with Bounded Size` (BBBS) presented in Appendix A.2.3. This BBBS algorithm can become a major 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 ($k=\infty$) 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 allows 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 `Column Generation` (4.2) model with RR stables). Experiments suggest that by lowering k, the above maximum weight clique problem with bounded size k can be solved more rapidly, *i.e.*, BBBS becomes faster.[5] On the other hand, by lowering k, the outer approximation $\mathcal{P} \supsetneq \texttt{conv}(\mathcal{P}_{0-1})$ becomes coarser, generating more artificial RR stables that are not standard stables; this leads to more artificial constraints in the `Column Generation` model (4.2), so that the lower bound reported in the end becomes smaller.

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 $\left[\frac{1}{\omega}\ \frac{1}{\omega}\ \ldots\ \frac{1}{\omega}\right]^\top$ from $\mathcal{P}$. As such, the dual* ***`Column Generation`*** *model (4.2) does not necessarily contain a constraint of the form $\left[\frac{1}{\omega}\ \frac{1}{\omega}\ \ldots\ \frac{1}{\omega}\right]\mathbf{x} \leq 1$, and so, the dual objective function value does not necessarily satisfy $\mathbf{1}_n^\top\mathbf{x} \leq \omega$. Notice that $\left[\frac{1}{\omega}\ \frac{1}{\omega}\ \ldots\ \frac{1}{\omega}\right]$ 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 and the cuts (d) seem very useful for this. 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 the lower bound $\mathbf{b}^\top\mathbf{x}_{it}$ reported by the* ***`Projective Cutting-Planes`*** *at each iteration* ***`it`*** *remains perfectly valid.* □

[5] This 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).

4.1.4.3 A certain form of degeneracy: projecting a boundary point $\mathbf{x}_{\mathtt{it}}$ can lead to a null step length $t^*_{\mathtt{it}}$ in the model with RR stables

In the model with RR-stables, it is not efficient to define $\mathbf{x}_{\mathtt{it}}$ as the last pierce point because projecting boundary points is prone to a certain form of degeneracy. More exactly, the projection sub-problem might return $t^*_{\mathtt{it}} = 0$ in this case, because a boundary point can belong to multiple facets, so that a projection reduces to finding a new facet that touches $\mathbf{x}_{\mathtt{it}}$. This can 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, we first notice that $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}$ belongs to the first–hit constraint $\mathbf{a}^\top\mathbf{x} \leq 1$ returned by the projection sub-problem at iteration $\mathtt{it}-1$, so that $\mathbf{a}^\top\mathbf{x}_{\mathtt{it}} = 1$. Furthermore, the current optimal solution $\mathtt{opt}(\mathscr{P}_{\mathtt{it}-1})$ also belongs to the above first–hit facet, and so, by taking $\mathbf{d}_{\mathtt{it}} = \mathtt{opt}(\mathscr{P}_{\mathtt{it}-1}) - \mathbf{x}_{\mathtt{it}}$ as indicated by Step 2 from Section 2, we also obtain $\mathbf{a}^\top\mathbf{d}_{\mathtt{it}} = 0$. 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 $\widehat{\mathbf{a}} \in \mathcal{P}$ very close to $\mathbf{a}$. A part of these $\widehat{\mathbf{a}} \in \mathcal{P}$ can satisfy $\widehat{\mathbf{a}}^\top\mathbf{x}_{\mathtt{it}} = 1$, so that there are multiple first–hit constraints (in $\mathscr{P}$) that touch $\mathbf{x}_{\mathtt{it}}$ — recall that $\mathbf{a} = \frac{\bar{\mathbf{a}}}{\bar{\alpha}}$ was not determined by optimizing the LP (4.1.6a)–(4.1.6d) in the direction of $\mathbf{x}_{\mathtt{it}}$. As such, it often possible to find some $\widehat{\mathbf{a}} \in \mathcal{P}$ such that $\widehat{\mathbf{a}}^\top\mathbf{x}_{\mathtt{it}} = 1$ and $\widehat{\mathbf{a}}^\top\mathbf{d} = \epsilon > 0$, which leads to a step length of $t^*_{\mathtt{it}} = \dfrac{1 - \widehat{\mathbf{a}}^\top\mathbf{x}_{\mathtt{it}}}{\widehat{\mathbf{a}}^\top\mathbf{d}_{\mathtt{it}}} = \dfrac{0}{\epsilon} = 0$.

We thus propose to define $\mathbf{x}_{\mathtt{it}}$ as a strictly interior point using $\mathbf{x}_{\mathtt{it}} = \alpha(\mathbf{x}_{i-1} + t^*_{i-1}\mathbf{d}_{i-1})$ with $\alpha = 0.9999$. This way, the above RR stables $\widehat{\mathbf{a}}$ very close to $\mathbf{a}$ no longer cause any problem: they lead to $1 - \widehat{\mathbf{a}}^\top\mathbf{x}_{\mathtt{it}} > 0$ and to a small ("ϵ-sized") $\widehat{\mathbf{a}}^\top\mathbf{d}_{\mathtt{it}}$, so that $\frac{1-\widehat{\mathbf{a}}^\top\mathbf{x}_{\mathtt{it}}}{\widehat{\mathbf{a}}^\top\mathbf{d}_{\mathtt{it}}}$ 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 number of large standard-size pieces available in stock, the problem asks to cut these pieces into smaller pieces (items) to fulfill a given demand. The (pattern-oriented) *Cutting-Stock* formulation consists of a program with prohibitively-many variables (patterns). After the linear relaxation, the dual of this program takes the form of the generic `Column Generation` dual LP (4.2), recalled below for the reader's convenience.

$$\mathscr{P}\left\{\begin{array}{lll} & \max \mathbf{b}^\top\mathbf{x} & \\ y_a: & \mathbf{a}^\top\mathbf{x} \leq c_a, & \forall(\mathbf{a}, c_a) \in \mathcal{A} \\ & \mathbf{x} \geq \mathbf{0}_n & \end{array}\right. \tag{4.2.1}$$

The notations from (4.2.1) can be directly interpreted to address the *Cutting-Stock* problem. 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 cutting-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 standard-size pieces in stock. The vector $\mathbf{b} \in \mathbb{Z}^n_+$ indicates the demand of each of 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 length 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 outer optimal solution

$\mathbf{x} = \texttt{opt}(\mathscr{P}_{\texttt{it}})$ at iteration `it`. For *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 $\ell \in [1..W]$; the objective value of s_ℓ is given by a pattern $\mathbf{a}_\ell \in \mathbb{Z}^n_+$ of length ℓ that minimizes $c_\ell - \mathbf{a}_\ell^\top \mathbf{x}$, where c_ℓ is the pattern cost only depending on ℓ. The `Dynamic Programming` scheme generates transitions among these 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 for other problems explored in this work, a key observation is that defining $\mathbf{x}_{\texttt{it}}$ as the best solution discovered up to iteration `it` it is not efficient in the long run, partly because $\mathbf{x}_{\texttt{it}}$ could fluctuate too much along the iterations `it` (recall also Section 2.1). We will also later see (Section 4.2.3.2) that the *Cutting-Stock* problems have the particularity that the `Dynamic Programming` for the intersection sub-problem $\texttt{project}(\mathbf{x} \to \mathbf{d})$ can be faster when $\mathbf{x}$ is a "truncated" solution, *e.g.*, when x_i is a multiple of $\gamma = 0.2$ for each $i \in [1..n]$. However, for now, it is enough to first to describe the general steps of the overall `Projective Cutting-Planes` for *Multiple-Length Cutting-Stock*.

Based on above observations, we define $\mathbf{x}_{\texttt{it}}$ 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 $\mathbf{x}^{\text{bst}}$ denote the best truncated feasible solution generated up to the current iteration; this $\mathbf{x}^{\text{bst}}$ can be determined as follows: start with $\mathbf{x}^{\text{bst}} = \mathbf{0}_n$ at iteration $\texttt{it} = 1$, and replace $\mathbf{x}^{\text{bst}}$ with $\lfloor \mathbf{x}_{\texttt{it}} + t^*_{\texttt{it}} \mathbf{d}_{\texttt{it}} \rfloor$ at each iteration $\texttt{it} > 1$ where $\mathbf{b}^\top \lfloor \mathbf{x}_{\texttt{it}} + t^*_{\texttt{it}} \mathbf{d}_{\texttt{it}} \rfloor > \mathbf{b}^\top \mathbf{x}^{\text{bst}}$. The `Projective Cutting-Planes` choses the inner solution $\mathbf{x}_{\texttt{it}}$ at each iteration `it` as follows:

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

We did try to let $\mathbf{x}_{\texttt{it}}$ take the value of the best feasible solution (last pierce point) found up to iteration `it` using the formula $\mathbf{x}_{\texttt{it}} = \mathbf{x}_{\texttt{it}-1} + t^*_{\texttt{it}-1} \mathbf{d}_{\texttt{it}-1}$. This is a more "aggressive" choice that enables the `Projective Cutting-Planes` to start very well by strictly increasing the lower bound at each iteration `it`, *i.e.*, check that $\mathbf{b}^\top \mathbf{x}_{\texttt{it}} = \mathbf{b}^\top \left(\mathbf{x}_{\texttt{it}-1} + t^*_{\texttt{it}-1} \mathbf{d}_{\texttt{it}-1} \right) \geq \mathbf{b}^\top \mathbf{x}_{\texttt{it}-1}$ is surely satisfied because the objective function does not deteriorate by advancing along $\mathbf{x}_{\texttt{it}-1} \to \mathbf{d}_{\texttt{it}-1}$ (see arguments at Step 2 from Section 2). This proves that the lower bound $\mathbf{b}^\top \mathbf{x}_{\texttt{it}}$ is 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. 37) suggest that this aggressive choice leads to more iterations in the long run; furthermore, these iterations are also computationally more expensive because the above $\mathbf{x}_{\texttt{it}}$ is not truncated.

Regarding the iterations $\texttt{it} = 1$ and $\texttt{it} = 2$, 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 $\mathbf{0}_n \to \frac{1}{W}\mathbf{w}$ 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` 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 often use `Dynamic Programming` (DP) to solve the separation sub-problem. And, in many cases, if the separation sub-problem can be solved by `Dynamic Programming`, so can be the projection one.

Given a feasible $\mathbf{x} \in \mathscr{P}$ in (4.2.1) and a direction $\mathbf{d} \in \mathbb{R}^n$, the projection subproblem $\texttt{project}(\mathbf{x} \to \mathbf{d})$ asks to minimize (2.2.1), which is instantiated as follows for the *Multiple-Length Cutting-Stock* problem:

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

where the function $f : [0, W] \to \mathbb{R}_+$ maps each $\ell \in [0, W]$ to the cost of the cheapest (shortest) standard-size piece available in stock that can accommodate the length ℓ. The DP scheme from this section can work for any non-decreasing function f, *i.e.*, whenever we can use the natural assumption that shorter pieces are cheaper than longer pieces. This holds for quite many different functions f, including those that encode other *Cutting-Stock* variants, like variable size 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 $\mathcal{S}_\ell$ of DP states for every feasible length $\ell \in [0..W]$. Each state $\mathtt{s} \in \mathcal{S}_\ell$ is associated to all patterns $\mathbf{a} \in \mathcal{A}$ of length $\mathtt{s}_{\mathtt{len}} = \mathbf{w}^\top \mathbf{a} = \ell$, of cost $\mathtt{s}_c = f(\mathbf{w}^\top \mathbf{a}) - \mathbf{a}^\top \mathbf{x} = f(\ell) - \mathbf{a}^\top \mathbf{x}$ and of profit $\mathtt{s}_p = \mathbf{d}^\top \mathbf{a}$. Using this interpretation in terms of costs and profits, we can say that the objective of (4.2.2) asks to find a state $s \in \{\mathcal{S}_\ell : \ \ell \in [0..W]\}$ that minimizes the cost/profit ratio $\mathtt{obj}(\mathtt{s}) = \dfrac{\mathtt{s}_c}{\mathtt{s}_p}$, *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 $\mathtt{s}_c = f(\mathbf{x}^\top \mathbf{a}) - \mathbf{a}^\top \mathbf{x}$ is always non-negative because $\mathbf{x} \in \mathscr{P}$.

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. Each such DP iteration generates transitions from the current states to update other (or produce new) states. A state transition $\mathtt{s} \to \mathtt{s}'$ associated to an item i leads to a state $\mathtt{s}'$ such that:

- $\mathtt{s}'_{\mathtt{len}} = \mathtt{s}_{\mathtt{len}} + w_i$, *i.e.*, the length simply increases by adding a new item;
- $\mathtt{s}'_p = \mathtt{s}_p + d_i$, *i.e.*, we add the profit of item i;
- $\mathtt{s}'_c = \mathtt{s}_c + f(\mathtt{s}'_{\mathtt{len}}) - f(\mathtt{s}_{\mathtt{len}}) - x_i$, *i.e.*, the term $f(\mathtt{s}'_{\mathtt{len}}) - f(\mathtt{s}_{\mathtt{len}})$ updates the cost of the 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

1. **for** $\ell = W - w_i$ **to** 0: ▷ actually scan a linked list of existing pattern lengths
2. **for each** $\mathtt{s} \in \mathcal{S}_\ell$: ▷ for each state with length ℓ
3. initialize state $\mathtt{s}'$ with $\mathtt{s}'_{\mathtt{len}} = \ell + w_i$
4. calculate $\mathtt{s}'_p$, $\mathtt{s}'_c$ with above formulae
5. **if** $\mathtt{s}'$ is not dominated by an existing state in $\mathcal{S}_{\ell+w_i}$ (see Section 4.2.3.2) **then**
   - $\mathcal{S}_{\ell+w_i} \leftarrow \mathcal{S}_{\ell+w_i} \cup \{\mathtt{s}'\}$
   - record the transition $\mathtt{s} \to \mathtt{s}'$ (to reconstruct an optimal pattern in the end)

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

We hereafter focus on how to reduce the number of states that need to be recorded in each $\mathcal{S}_\ell$, 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 $\mathcal{S}_\ell$. For this, we consider two states $\mathtt{s}^*, \mathtt{s} \in \mathcal{S}_\ell$ such that $\mathtt{s}^*_c = \mathtt{s}_c$ and $\mathtt{s}^*_p > \mathtt{s}_p$ and we can check that $\mathtt{s}$ is dominated and can be ignored. Indeed, any transition(s) applied on such states would lead to the same cost $\mathtt{s}^*_c + \Delta_c = \mathtt{s}_c + \Delta_c > 0$ and to profits $\mathtt{s}^*_p + \Delta_p > \mathtt{s}_p + \Delta_p$ and it is easy to check that $\dfrac{\mathtt{s}^*_c + \Delta_c}{\mathtt{s}^*_p + \Delta_p} < \dfrac{\mathtt{s}_c + \Delta_c}{\mathtt{s}_p + \Delta_p}$ — we are only interested in positive profit states (positive denominators) because of the condition $\mathbf{d}^\top \mathbf{a} > 0$ from (4.2.2).

But now consider a state $\mathtt{s} \in \mathcal{S}_\ell$ such that $\mathtt{s}^*_c < \mathtt{s}_c$ and $\mathtt{s}^*_p \geq \mathtt{s}_p$, *i.e.*, of higher cost and no better profit than $\mathtt{s}^*_c$. Such a state is also dominated by $\mathtt{s}^*$, because it can only lead via transitions to $\dfrac{\mathtt{s}^*_c + \Delta_c}{\mathtt{s}^*_p + \Delta_p} < \dfrac{\mathtt{s}_c + \Delta_c}{\mathtt{s}_p + \Delta_p}$. As such, if a state $\mathtt{s} \in \mathcal{S}_\ell$ has a higher cost than an existing state $\mathtt{s}^* \in \mathcal{S}_\ell$ (*i.e.*, $\mathtt{s}^*_c < \mathtt{s}_c$), then it can only be non-dominated if it has a better profit as well, *i.e.*, only if $\mathtt{s}^*_p < \mathtt{s}_p$. 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 $\mathcal{S}_\ell$ can thus be ordered using a relation of the following 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 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 recored 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 final number of states is reduced. Accordingly, by using only truncated solutions $\mathbf{x}_{\mathtt{it}}$ in which both the pattern costs $f(\ell)$ ($\forall \ell \in [0, W]$) and the components of $\mathbf{x}_{\mathtt{it}}$ are 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 very 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 (4.2.3.a)–(4.2.3.b). Such a data structure could enable Algorithm 1 to perform the following two tasks as rapidly as possible: (i) to iterate over all elements of $\mathcal{S}_\ell$ at Step 2 and (ii) to insert 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 list for the above task (i) is relatively straightforward. It is most difficult to achieve a very fast insertion for the above task (ii), because one should* ***not*** *scan the whole list to check if the new state is dominated. Furthermore, the insertion of a new non-dominated state can result in the removal of other existing states that become dominated. To perform such operations as rapidly as possible, we propose in Appendix A.3.1 to use a data structure that relies on a self-balancing binary tree as the main core engine.* □

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 $\dfrac{w_i}{1 + \mathbf{x}^{\text{bst}}_i}$. Precisely, Algorithm 1 is executed for each of the items $[1..n]$ considered in this order. In a loose selor dense, this amounts to considering that it is better to start with longer items that did not contribute too much to the best truncated inner solution $\mathbf{x}^{\text{bst}}$ ever found.

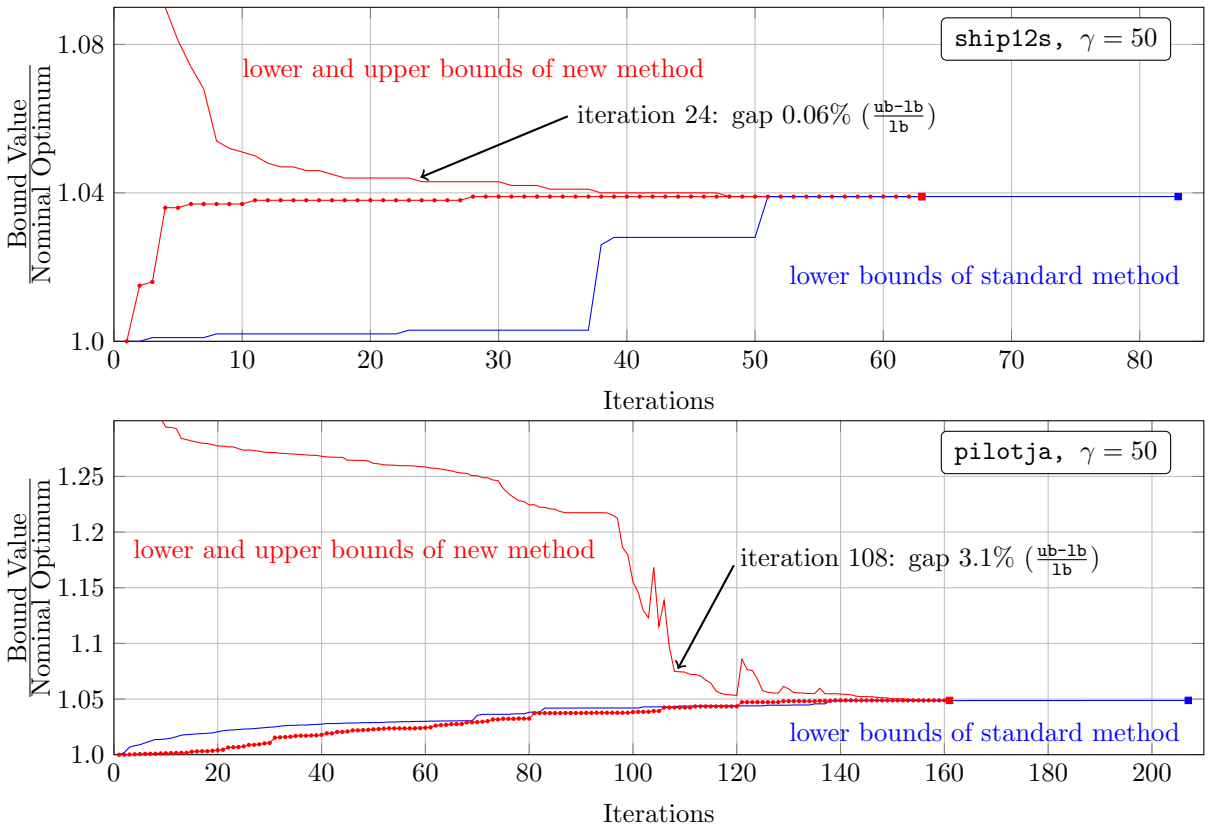

Figure 3: The progress over the iterations of the lower and upper bounds reported by the `Projective Cutting-Planes` (in red), compared to those of the standard `Cutting-Planes` (lower bounds only, in blue).

5 Numerical Evaluations

Sections 5.1 and 5.2 provide numerical results for the (minimization) problems explored in Section 3, in which $\mathscr{P}$ is defined as a primal polytope. Section 5.3 and 5.4 are devoted to the problems from Section 4 in which $\mathscr{P}$ is the dual polytope in a `Column Generation` model with a maximization (dual) objective. We will finish with a brief Section 5.5 to (try to) explain the reasons why the `Projective Cutting-Planes` can be more successful on certain problems than on others, also investigating why choosing $\mathbf{x}_{\texttt{it}} = \mathbf{x}_{\texttt{it}-1} + \alpha t^*_{\texttt{it}-1}\mathbf{d}_{\texttt{it}-1}$ 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/graphs/projcutplanes/`; see the beginning of Appendix B for more technical compilation and machine information.

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 provide 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 the problem has a minimization objective so that the feasible solutions $\mathbf{x}_{\texttt{it}}$ determined by the `Projective Cutting-Planes` generate upper bounds $\mathbf{b}^\top \mathbf{x}_{\texttt{it}}$ along the iterations `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 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 total computational effort needed to fully converge, we notice that the upper bounds generated by the `Projective Cutting-Planes` can have a high-quality, in some cases even since the beginning of the search. 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 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 substantially many more iterations than normally expected — see the figures in bold 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 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* $\texttt{opt}(\mathscr{P}_{it-1})$ *represent together a more "meaningful measure" for selecting a new constraint, avoiding iterations that keep the objective value constant (also recall* $\mathbf{b}^\top \mathbf{d}_{it} > 0$*).*

To summarize, the proposed intersection algorithm enables the `Projective Cutting-Planes` to benefit from two advantages at a 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 `Projective Cutting-Planes` leads to a really spectacular acceleration in terms of iterations or CPU time to fully converge on this robust optimization problem. However, we will see 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 experiments 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 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 $[0, \texttt{dem}_{\texttt{max}}]$. We will present statistical results over 10 runs,[7] reporting the average, the standard deviation and the minimum value of two main performance indicators: 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 $\texttt{dem}_{\texttt{max}}$ in Column 5. Column 6 reports either the optimum of the linear relaxation (LP OPT in Table 2) or

[6] It is also true that the starting solution $\mathbf{x}_1$ constructed as indicated in Section 3.1.2 might already have an objective value within 101% of the nominal optimum for three instances (`ganges`, `ship08l` and `ship12l`). This is not surprising given that the optimal robust solution for these instances is at most 1.0053 times the nominal optimum.

[7] By 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 search, as in the experimental section of [14].

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

Table 1: Comparison between the `Projective Cutting-Planes` and the standard `Cutting-Planes` on 21 instances for robust optimization from [6]. 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 at the first iteration `it` when the gap between the upper bound $\mathbf{b}^\top \mathbf{x}_{\mathtt{it}}$ and the lower bound $\mathtt{optVal}(\mathscr{P}_{\mathtt{it}})$ is below 1%, *i.e.*, either $0 < \mathtt{optVal}(\mathscr{P}_{\mathtt{it}}) \leq \mathbf{b}^\top \mathbf{x}_{\mathtt{it}} \leq 1.01\mathtt{optVal}(\mathscr{P}_{\mathtt{it}})$ or $\mathtt{optVal}(\mathscr{P}_{\mathtt{it}}) \leq \mathbf{b}^\top \mathbf{x}_{\mathtt{it}} \leq 0.99\mathtt{optVal}(\mathscr{P}_{\mathtt{it}}) < 0$.

Instance						Projective Cutting-Planes				Standard Cutting-Planes		
Graph class	ID	$\|E\|$	$\|V\|$	$\mathtt{dem}_{\max}$	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	4.4%	35 (4.9) 31	0.09 (0.01) 0.07	5.5%
b	1	105	60	10	245.67	265	73.8 (2.7) 72	0.2 (0.006) 0.2	6.1%	131 (11.8) 115	0.4 (0.04) 0.3	8.7%
c	1	110	50	10	204.33	220	56.5 (1.5) 56	0.2 (0.004) 0.2	4.9%	78.5 (16) 68	0.2 (0.05) 0.2	5.8%
d	1	120	60	10	299.33	317	67.5 (3) 63	0.2 (0.01) 0.2	4.3%	104 (4.3) 101	0.4 (0.02) 0.4	6.1%
e	1	120	30	10	67.333	77	35.4 (0.8) 35	0.1 (0.006) 0.1	4.2%	39.5 (5.5) 34	0.1 (0.02) 0.1	5.5%
f	1	600	90	10	281.33	306	150 (35.3) 121	6.2 (1.6) 4.8	2.5%	251 (28.8) 199	8.1 (1.1) 6.3	3.5%
g	1	1000	100	10	284.33	312	147 (52.1) 107	14.3 (6.1) 9.7	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	5.3%	24 (0) 24	0.04 (<0.001) 0.04	6.2%
	2	70	25	10	64.667	74	19 (0) 19	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	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	5.1%	37.6 (1.2) 37	0.07 (0.003) 0.07	6.3%
	1	90	35	10	124.67	137	43.4 (1.2) 43	0.1 (0.003) 0.1	5.8%	78.5 (7.5) 71	0.2 (0.02) 0.2	7.4%
rnd-100	1	600	90	100	2918.7	2946	153 (20.7) 129	6.2 (1) 5	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	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	2%	360 (23) 306	28.8 (2.3) 23.7	4.2%
	2	1000	110	100	3080.3	3112	196 (72.8) 136	19.3 (8.2) 12.6	2.3%	389 (38.2) 313	30.6 (3.7) 22.2	3.6%
	1	1500	130	100	3922	3956	382 (182) 193	89.6 (45.9) 41.1	3%	466 (20.8) 430	74.8 (4.7) 65.9	2.9%
	2	1500	130	100	4033.7	4073	282 (77.4) 190	65 (19.8) 40.1	2.2%	536 (42.5) 446	90.4 (7.8) 74.4	4.2%
	1	2000	150	100	4638	4684	259 (106) 166	103 (50.7) 59.1	1%	744 (65.8) 638	218 (18.8) 193	5.4%
rnd-300	1	2000	150	300	13314	13365	302 (187) 164	122 (92.6) 58.1	1.8%	744 (87.9) 626	215 (29.1) 175	4.9%
	2	2000	150	300	13358	13404	295 (172) 168	115 (76.6) 57.5	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` resp. 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.

Instance						Projective Cutting-Planes				Standard Cutting-Planes			
Graph class	ID	$\|E\|$	$\|V\|$	$\texttt{dem}_{\max}$	IP OPT	Iterations	Time total [secs]	Time solve master		Iterations	Time total [secs]	Time solve master	
						avg (std) min	avg (std) min			avg (std) min	avg (std) min		
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%	
b	1	105	60	10	260	824 (206) 464	1073 (636) 379	99.5%		2987 (375) 2427	4129 (819) 2971	99.8%	
c	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%	
d	1	120	60	10	313	336 (53.4) 251	321 (103) 156	99.2%		1315 (184) 1049	2367 (469) 1837	99.8%	
e	1	120	30	10	74	1336 (138) 1020	4907 (1640) 2086	99.8%		2250 (450) 1292	6703 (2857) 2450	99.9%	
rnd-10	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. The new method can roughly require 4 times less iterations (row `d`), 3 times less iterations (row `b`) or 2 times less iterations (antepenultimate row).
Comparing to [14, Table 2], one notices it is not usually possible to reduce the number of iterations by factors of 3 or 4 by simply stabilizing or improving 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 do not reduce the number of iterations of the standard `Cutting-Planes` above by more than half.
The notation $^{-k}$ in Columns 7 or 14 indicates that k runs out of 10 failed due to numerical difficulties, as described in Appendix A.1.

the optimum of the original integer model (IP OPT in 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 $\texttt{opt}(\mathscr{P}_1)$, $\texttt{opt}(\mathscr{P}_2)$, $\texttt{opt}(\mathscr{P}_3)$, etc.

Table 2 contains an additional Column 7 (Best IP Sol) that reports the best integer upper bound found by the `Projective Cutting-Planes` along all runs on the LP relaxation. This reveals an advantage of the new method: it generates 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 integer solution from Column 7, *i.e.*, the average integrality gap is 5.5%. If we only consider the last 9 instances with $\texttt{dem}_{\texttt{max}} \geq 100$, the average integrality gap is below 1%. These instances are so large that we are skeptical their integer optimum can 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. Since the `Projective Cutting-Planes` reports a root node gap below 1% for these 9 largest instances, this new method seems particularly promising for integration in a `Branch and Bound`, 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 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 $\frac{1}{3}$ less iterations in average. This shows there is no need for a statistical test to confirm there is a real 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 intersection sub-problem is slightly computationally more expensive than the separation one.

Regarding the original integer Benders problem, Table 3 confirms that both the separation and the intersection sub-problem can be solved significantly faster than the ILP master problems. The columns "Time solve master" indicate that 97% of the total running time is spent on solving these ILP master problems, which clearly become the major computational bottleneck of either the new or the standard `Cutting-Planes`. This confirms theoretical expectations, since an ILP is significantly more difficult than a pure LP — recall both sub-problems reduce to solving a pure LP.

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 solving two instances with a 100% success rate. The running time is not perfectly proportional to the number of iterations, because the strucure 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 working with two different coloring models.

- In Section 5.3.1, we will compare the `Projective Cutting-Planes` and the classical `Column Generation` on the original graph coloring model with standard stables. The projection sub-problem will be solved as described in Section 4.1.3.
- In Section 5.3.2 we will apply the `Projective Cutting-Planes` on a second coloring model that replaces the standard stables with the broader notion of reinforced relaxed (RR) stables (recall Definition 2, page 19). We will compare this `Projective Cutting-Planes` variant with the classical `Column Generation` working on the model with standard stables. The projection sub-problem is solved as described in Section 4.1.4.

We here use 15 coloring instances from the second DIMACS implementation challenge that have been widely-used for benchmarking coloring algorithms since the 1990s. 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

The steps of the `Projective Cutting-Planes` have been presented in Section 4.1.2. The intersection sub-problem is solved using the Disjunctive LP (4.1.2.a)–(4.1.2.e) generated by applying the discrete Charnes–Cooper reformulation as described in Section 4.1.3. Recall that the separation sub-problem of the standard `Column Generation` reduces to optimizing an ILP over the set of standard stables $\mathcal{P}_{0-1}$ (*i.e.*, the maximum stable problem). We prefer to solve both the ILP and the Disjunctive LP with similar mathematical programming tools (based on `Branch and Bound`), as provided in the `cplex` software package. If we used two (very) different methods for these two sub-problems, we would have skewed the results in the favor of the most refined of the two methods.

To have a clear view of the optimality gap for each reported lower bound, we will also refer to an upper bound value determined from a heuristic coloring.[8] Based on this heuristic coloring, we construct an initial direction $\mathbf{d}_1$ as indicated in Section 4.1.2 and we also generate a set of initial constraints $\mathcal{A}_0 \subsetneq \mathcal{A}$ of the form $\mathbf{a}^\top \mathbf{x} \leq 1$, where $\mathbf{a} \in \mathbb{Z}_+^n$ is an incidence vector of any stable from the heuristic coloring. While the `Projective Cutting-Planes` starts out by solving `project`$(\mathbf{0}_n \rightarrow \mathbf{d}_1)$, the standard `Column Generation` starts out by separating $\mathbf{d}_1$ at the first iteration, so as to ensure similar starting conditions for the two methods.

Given the above $\mathbf{d}_1$, by instantiating (4.1.1.a)–(4.1.1.c), the first intersection sub-problem can be written in the form $t_1^* = \min\left\{\dfrac{1}{\mathbf{d}_1^\top \mathbf{a}} :\ \mathbf{a} \in \mathcal{P}_{0-1},\ \mathbf{d}_1^\top \mathbf{a} > 0\right\}$, where recall $\mathcal{P}_{0-1}$ is actually the set of standard stables. 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:

$$\frac{\mathbf{1}_n^\top \mathbf{d}_1}{\max\left\{\mathbf{d}_1^\top \mathbf{a} :\ \mathbf{a} \in \mathcal{P}_{0-1}\right\}} \tag{5.3.1}$$

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 outer optimal solution `opt`$(\mathscr{P}_{\texttt{it}})$ 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 $\dfrac{\mathbf{1}_n^\top \mathbf{x}}{1 - \min\{1 - \mathbf{x}^\top \mathbf{a} :\ \mathbf{a} \in \mathcal{P}_{0-1}\}}$; for $\mathbf{x} = \mathbf{d}_1$, this reduces to $\dfrac{\mathbf{1}_n^\top \mathbf{d}_1}{\max\{\mathbf{d}_1^\top \mathbf{a} :\ \mathbf{a} \in \mathcal{P}_{0-1}\}}$. 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 of 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 exhibits the infamous "yo-yo" effect. This effect comes from the strong oscillations of the current optimal solution `opt`$(\mathscr{P}_{\texttt{it}})$ along the iterations `it` of the standard `Column Generation` (also referred to as the "bang-bang" behaviour). By stabilizing the `Column Generation`, one can reduce such effects, but we are not aware of any other work in `Column Generation` in which the "yo-yo" effect has been completely eliminated.

The `Projective Cutting-Planes` variant from this section does eliminate this "yo-yo" effect completely, because $\mathbf{x}_{\texttt{it}} = \mathbf{x}_{\texttt{it}-1} + t^*_{\texttt{it}-1} \mathbf{d}_{\texttt{it}-1}$ is the best feasible solution discovered up to iteration `it`. It is certainly better than $\mathbf{x}_{\texttt{it}-1}$ since the objective value can not decrease by advancing along from $\mathbf{x}_{\texttt{it}-1} \rightarrow \mathbf{d}_{\texttt{it}-1}$, *i.e.*, a projection can only improve the objective value, based on the arguments provided at Step 2 from Section 2.

Table 4 reports three lower bounds determined by the classical `Column Generation` (Columns 2-4) and by the `Projective Cutting-Planes` (last three columns). These three bounds respectively correspond to the beginning (Columns 2 and 6), to a midpoint (Columns 3 and 7) and to the to end (Columns 4 and 8) of the solution process. In fact, we tried to make the columns 2–6, 3–7, and resp. 4–8 correspond to equal rounded-up bound values; this explains why Column 3 might actually report more than 100 iterations, *i.e.*, the `Column Generation` might needs hundreds of iterations to reach the bound value reported by the `Projective Cutting-Planes` in a dozen of iterations. For each bound, we indicate the number of iterations

[8] Such proper colorings have been determined in our previous work on heuristic algorithms for graph coloring. 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.

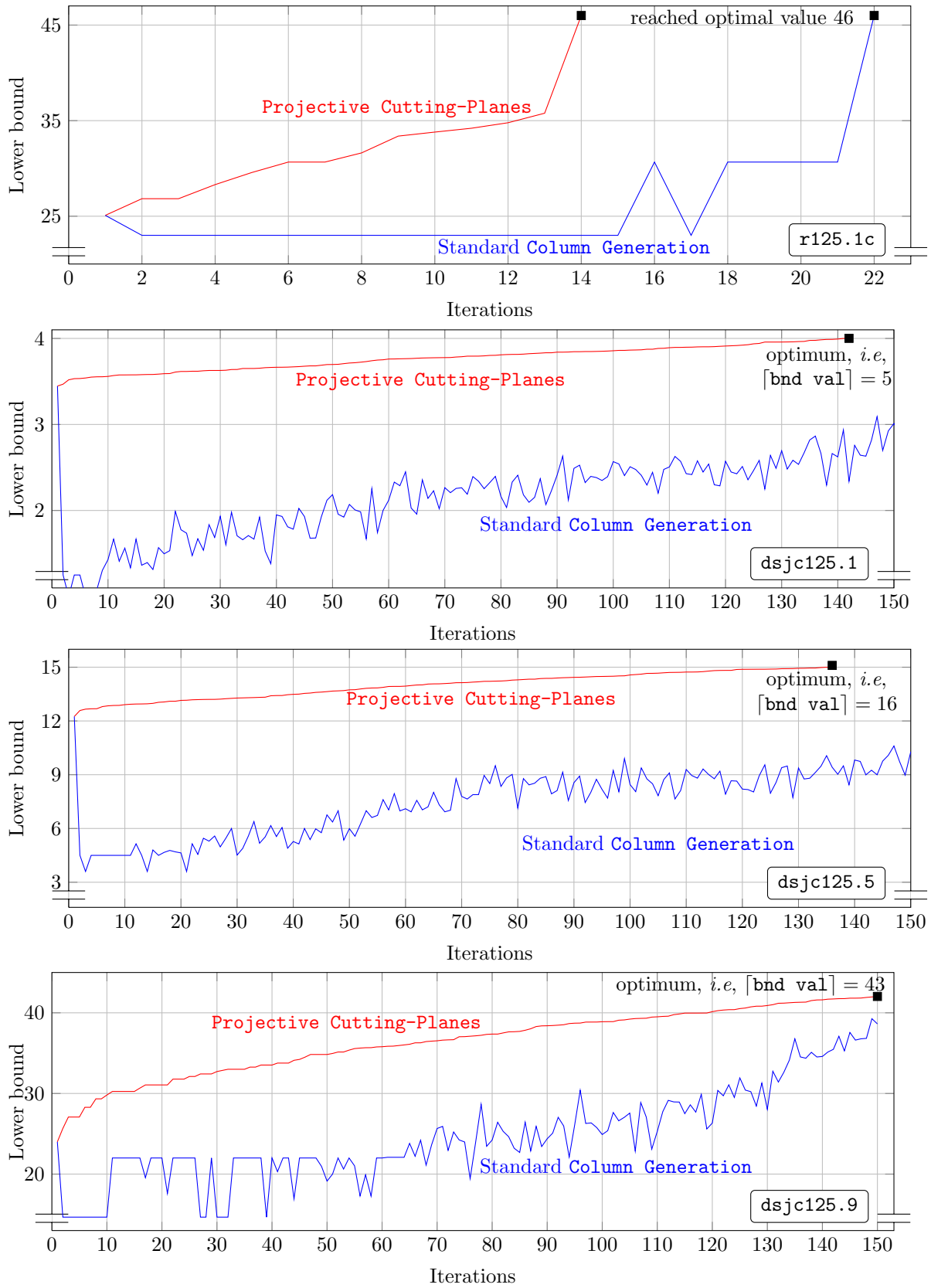

Figure 4: The running profile (lower bounds) of the `Projective Cutting-Planes` and of the classical `Column Generation` on 4 standard coloring instances. The new method does *not* exhibit the infamous "yo-yo" effect of the `Column Generation` bounds.

Instance	Classical `Column Generation` beginning `iter:lb/tm`	mid iter `iter:lb/tm`	last iter `iter:lb/tm`	clique cut sz. k	`Projective Cutting-Planes` beginning `iter:lb/tm`	mid iter `iter:ca lb/tm`	last iter `iter: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 $t^*_{\mathtt{it}} < 10^{-6}$ at some iteration `it`, we switch to a new formula to determine future inner solutions: $\mathbf{x}_{\mathtt{it}} = 0.99\,(\mathbf{x}_{\mathtt{it}-1} + t *_{\mathtt{it}-1} \mathbf{d}_{\mathtt{it}-1})$. We thus avoid the degeneracy-like issues from Section 4.1.4.3.

iter, the bound value `lb` and the CPU time in seconds `tm`. A digit in Column 5 indicates that we used k-clique inequalities to accelerate the intersection sub-problem algorithm (see Section 4.1.3.1). We excluded from this report graphs like `le450_25c`, `le450_25d`, `le450_15c`, `le450_15d`, `dsjc500.1` that require a prohibitively-large computing time — not only to our sub-problem algorithms, see also Remark 7.

Two interesting conclusions can be drawn from Table 4. First, for half of the instances, the classical `Column Generation` might need hundreds of iterations to reach the lower bounds produced by `Projective Cutting-Planes` in less than 20 iterations (compare Columns 2 and 6 labelled "beginning"), mostly because 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 (`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 `cplex` solver for the separation ILP compared to the speed of the solver for the projection Disjunctive LP. By changing this solver, one could easily obtain different results.

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 does depend substantially on the running time of this sub-problem algorithm. Regarding 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 a `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 $\overline{a}_i \in \{0, \overline{\alpha}\}$ from "$\overline{a}_i \leq 0$ `or` $\overline{a}_i \geq \overline{\alpha}$" to "$\overline{a}_i = 0$ `or` $\overline{a}_i = \overline{\alpha}$" — 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 be fundamentally slower in absolute terms than any `Branch and Bound` algorithm for the associated ILP.

5.3.1.1 Using the same 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 specific to graph coloring can well apply to other inequalities, transformming the ILP of the separation sub-problem into a Disjunctive LP, so as to solve the projection sub-problem. 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 provide a very 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.

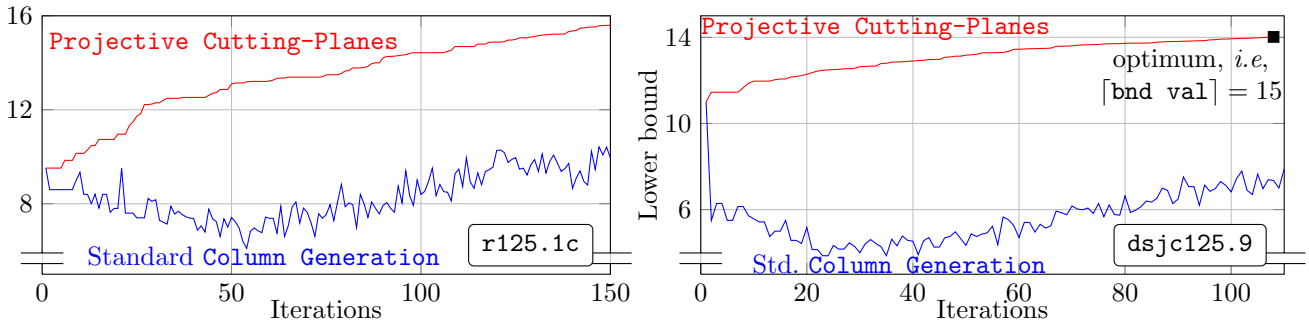

Figure 5: The running profile (lower bounds) of the `Projective Cutting-Planes` and of the standard `Column Generation` on two instances of the defective coloring problem.

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. More exactly, the constraints $\mathcal{A}$ are defined by the extreme solutions of the polytope $\mathcal{P}$ from Definition 2. 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 thus a lower bound for the original model. We will now 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 intersection 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), to construct the polytope $\mathcal{P}$ described by (4.1.5), reformulated into (4.1.6b) by the Charnes-Cooper transformation in Section 4.1.4.1. Confirming theoretical arguments from Section 4.1.4.2, a higher k generates stronger lower bounds at the cost 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 in 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 cliques for the reinforcing cuts (f) from Section 4.1.4.2. For the standard `Column Generation`, the maximum stable size (Column 5 in Table 5) provides the input value of the size needed by the BBBS algorithm. If we had used some elaborately-tuned state-of-the-art software to solve the above maximum stable problem, we would have skewed the results in the favor of the standard `Column Generation`.

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 provide 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 discussed 2 paragraphs above. 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 presents 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 standard stables. 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`.

Density	Instance	n	Upper bound	Optimal	Standard `Column Generation` with standard stables			`Projective Cutting-Planes` with RR stables				one last iter with std. stab.
				max stable	lb/tm	lb/tm	lb/tm	clq size k	lb/tm	lb/tm	lb/tm	$\lceil$lb$\rceil$/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
								4	2.95/0.53	3.01/1.22	4/15.51	5/21.06
0.17	`le450_25c`	450	25	47	2.5/5.03	5.2/195	6.87/2535	5	9.11/2.34	10.01/41	**25**/1242	optim ended
								25	9.11/2.28	10.01/43	**25**/1412	optim ended
0.17	`le450_25d`	450	25	43	2.77/5.26	5.45/409	6.15/2440	5	10.21/3.28	13.01/224	**25**/1364	optim ended
								25	10.21/3.14	13.01/210	**25**/1393	optim ended
0.24	`p_hat300-1`	300	19	39	2.11/3.4	7.36/206	10.19/5418	3	4.54/1.58	6.01/86	8/292	9/8479
								8	6.01/33	7.01/160	8/267	9/3212
0.50	`dsjc125.5`	125	18	35	4.5/2.25	10.01/2.44	15.06/4.88	7	7.91/0.66	8.01/0.89	9/8.11	14/76
								10	8.12/2.50	8.95/13	10/13.3	11/62
0.90	`dsjc125.9`	125	44	4	24/1.54	40.03/1.60	42.26/1.61	10	16.25/2.68	24/52	25/259	26/265
								∞	24/7.75	29.06/747	31.01/5497	32/5812
0.10	`dsjc250.1`	250	9	70	2.01/2.28	3.05/270	3.70/9180	3	3.5/0.35	3.75/10.9	3.81/18.2	5*/ 1076
								4	3.5/0.46	3.7/6.1	3.75/9.8	5*/91
0.50	`dsjc250.5`	250	28	12	5.6/4.93	15.04/10.34	25.01/225	6	8.24/7.03	9.01/13.36	10.001/119	21/5921
								∞	9.98/83	10.01/86	12/391	13/2492
0.90	`dsjc250.9`	250	73	5	34.3/10.4	60.01/11.18	70.09/11.6	10	10/3.17	10/3.17	10/3.17	50/586
								∞	time out (>10000)			–
0.10	`dsjc500.1`	500	12	122	2.25/5.14	2.50/66	3.06/3376	3	3.55/4.19	4.01/272	5/340	6*/5514
								5	4.08/79	4.50/295	5/682	6*/5857
0.03	`r125.1`	125	5	49	2.31/0.02	**5**/0.2	**5**/0.2	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
0.97	`r125.1c`	125	46	7	23/2.66	34.5/2.67	**46**/2.67	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
0.50	`r125.5`	125	36	5	20.57/0.09	25.2/0.22	**36**/0.25	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
0.17	`le450_15c`	450	15	49	3.07/6.23	4.22/145	5.32/1172	3	8.34/6.74	10.01/308	**15**/1751	optim ended
								15	9.19/14.52	13.01/210	**15**/3331	optim ended
0.17	`le450_15d`	450	15	49	3.07/6.33	4.23/145	5.33/1172	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. 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 the last step length. 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.

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 can achieve 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,*[9] *the lower bounds of the* **Projective Cutting-Planes** *are very general. They could be determined in the same manner for a (dual) objective function* $\mathbf{b} \neq \mathbf{1}_n$, *as in a multi-coloring problem (see Section 4.1.1) for which 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 a feasible solution of such quality has been found by exploiting only the standard* **Column Generation** *model* (4.2). *As for the four graphs above, the bound value in itself has been already reported but only using external methods (based on a reduced induced subgraph in [9]). We can even describe this feasible solution of* (4.2)*: it assigns* $0.9999 + 0.00000101$ *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 risking any numerical problem, we can prove this solution is feasible because there is no stable of size 100 that contains one of the vertices 4, 30, 47, 361, or 475 (these vertices form a clique). Indeed,* **cplex** *showed in less than 2 hours that this stable is upper bounded by 99; the solution is 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 about 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 projection, after about 36 hours (with* **cplex** *using up to 20 threads and 40GB of RAM on a stronger, multi-core CPU),* **cplex** *reported a lower bound of* 0.0088 *on the last step length, i.e.,* $t^*_{last} > 0.0088$. *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, the solver 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 been reported on this graph.*

5.4 Multiple-Length Cutting-Stock

This section is devoted to a *Multiple-Length Cutting-Stock* variant with (at least) two types of standard-size pieces available in stock: one of length W and cost 1, and the other 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).

Table 6 compares the `Projective Cutting-Planes` (as described in Section 4.2.2) to the standard `Column Generation`. Column 1 represents the instance, Column 2 present the optimal value, Columns 3–6 provide the results of the new method, and Columns 7–10 report 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 `ub`$\leq 1.2 \cdot$`lb`) and then the total computing effort needed to fully converge.

Table 6 suggests 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

[9] One can use a meta-heuristic to find an upper bound and any algorithm to determine the maximum clique size. Since the upper bound is often equal to the maximum clique size for such graphs, this directly 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 standard Projective Cutting-Planes compared to the classical Column Generation on *Multiple-Length Cutting-Stock*. The Projective Cutting-Planes needs 40% less iterations on almost a quarter 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.

to the fact that the lower bounds of the Projective Cutting-Planes are generally better than the (Lagrangian) lower bounds of the standard Column Generation calculated via (A.3.2).[10] 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 classical 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 confirmed by statistical results over 10 runs. For both the new and the standard method, Table 7 reports the average, the standard

[10]For the evolution of these lower bounds along the iterations, we refer the reader to the results available on-line at cedric.cnam.fr/~porumbed/projcutplanes/cutstock/. These bounds are also be compared to those of a so-called "aggressive" Projective Cutting-Planes that determines $\mathbf{x}_{\mathtt{it}}$ as the best feasible solution found up to iteration it; this Projective Cutting-Planes version eliminates the yo-yo effect (as in Figures 4–5) but it is slower in the long run.

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 of Projective Cutting-Planes and standard Column Generation with regards to the number of iterations needed to fully converge on *Multiple-Length Cutting-Stock*, using ten runs per instance.

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.

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 $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ by randomly breaking ties in case of equality (at each iteration it). The maximum number of iterations of the Projective Cutting-Planes is usually less 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, *i.e.*, they often represent less than 5% of the average value. Other preliminary experiments confirm that similar trends show up across all instances from each class, *e.g.*, the instances m01-1, m01-2, m01-3 lead to similar results.

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. One notices that the Projective Cutting-Planes requires twice less iterations than the standard Column Generation for 4 instances out of 10. For another 3 instances, the Projective Cutting-Planes requires 40% less iterations.

5.5 The oscillations of the inner solutions and the "bang-bang" effects

It could be interesting to gain a more insight into why an "aggressive" definition of $\mathbf{x}_{\texttt{it}}$ like $\mathbf{x}_{\texttt{it}} = \mathbf{x}_{\texttt{it}-1} + t^*_{\texttt{it}-1}\mathbf{d}_{\texttt{it}-1}$ leads to very poor results on certain problems and to reasonable results on others. A possible explanation is related to the oscillations of the inner solutions $\mathbf{x}_{\texttt{it}}$ along the iterations it. The above

aggressive $\mathbf{x}_{\mathtt{it}}$ definition generates stronger oscillations (a "bang-bang" effects) for the first two problems (the robust optimization problem and the Benders reformulation model) than for the last two (graph coloring and *Multiple-Length Cutting-Stock*).

We provide below the values of the first 15 components of $\mathbf{x}_{\mathtt{it}+1} = \mathbf{x}_{\mathtt{it}} + t^*_{\mathtt{it}}\mathbf{d}_{\mathtt{it}}$ for $\mathtt{it} \in \{1,\ 11,\ 21,\ 31,\ 41\}$, as generated by a `Projective Cutting-Planes` using the above aggressive $\mathbf{x}_{\mathtt{it}}$ definition. For each problem, we selected the very first instance from the main table of results, *i.e.*, from 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 $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}$ is appropriate for graph coloring, while the best settings for the first two problems take a form $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + \alpha t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}$ with $\alpha < 0.5$. Regarding the *Multiple-Length Cutting-Stock*, although we do not use $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}$ in Section 5.4, this choice would still lead to reasonable results, see the experiments referred by Footnote 10 (p. 37).

```
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 cuting 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` methods to optimize polytopes $\mathscr{P}$ 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. This new sub-problem can be stated as follows: given an arbitrary inner (feasible) solution $\mathbf{x} \in \mathscr{P}$ and a direction $\mathbf{d} \in \mathbb{R}^n$, determine the pierce (first-hit) point $\mathbf{x} + t^*\mathbf{d}$ encountered when advancing from $\mathbf{x}$ along $\mathbf{d}$, *i.e.*, determine $t^* = \max\{t \geq 0 :\ \mathbf{x} + t\mathbf{d} \in \mathscr{P}\}$. The resulting `Projective Cutting-Planes` generates a sequence of inner solutions $\mathbf{x}_{\mathtt{it}}$ and a sequence of outer solutions $\mathtt{opt}(\mathscr{P}_{\mathtt{it}})$ that both converge to the optimal solution $\mathtt{opt}(\mathscr{P})$. The inner solution $\mathbf{x}_{\mathtt{it}}$ at each iteration `it` is usually chosen as a point on the segment joining the previous inner solution $\mathbf{x}_{\mathtt{it}-1}$ and the last pierce point $\mathbf{x}_{\mathtt{it}-1} + t^*_{\mathtt{it}-1}\mathbf{d}_{\mathtt{it}-1}$. Theoretical arguments and extensive experiments presented along the paper highlight a number of advantages of the `Projective Cutting-Planes`, for different problems:

– It has a built-in mechanism to generate feasible solutions $\mathbf{x}_{\mathtt{it}} \in \mathscr{P}$ that converge along the iterations `it` to an optimal solution $\mathtt{opt}(\mathscr{P})$. The most standard `Cutting-Planes` does not have a general mechanism to generate such converging inner solutions for any problem. And although one can use for certain problems different ad-hoc techniques (*e.g.*, the Farley bound in `Column Generation`) to determine feasible inner solutions during the standard `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, so that the `Projective Cutting-Planes` could not work without them.

– Except for the robust optimization experiments (Section 5.1), the new method could reduce the total running time and the number of iterations by important factors in the best cases. More exactly, in Section 5.2, both the number of iterations and the CPU time could be reduced 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 have never been reported before in the literature of the (well studied) graph coloring problem (Remark 7, p. 35).

– The standard `Cutting-Planes` determines each new constraint by taking only the current optimal solution as the unique reference point. The `Projective Cutting-Planes` generates new constraints guided by *a pair* of inner–outer solutions. Using only projections $\mathbf{x} \to \mathbf{d}$ that satisfy $\mathbf{b}^\top \mathbf{d}_{\mathtt{it}} > 0$, the `Projective Cutting-Planes` can naturally avoid the infamous degeneracy issues (*i.e.*, iterations the keep the objective value constant) of certain `Cutting-Planes` algorithms. Although in our experiments these issues are most visible only in the standard `Cutting-Planes` from Section 5.1 (see Remark 6, p. 26), it is well-known that they can easily arise in `Column Generation` as well.[11]

– By defining $\mathbf{x}_{\mathtt{it}}$ as the best solution (pierce point) ever found, one can prove that the lower bound $\mathbf{b}^\top \mathbf{x}_{\mathtt{it}}$ 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 could arise in most (if not all) existing `Column Generation` algorithms.

There are also certain (inherent) deterrents to adopting the `Projective Cutting-Planes`:

– 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 $\mathbf{x}_{\mathtt{it}}$ with a formula like $\mathbf{x}_{\mathtt{it}} = \mathbf{x}_{\mathtt{it}-1} + \alpha \cdot t^*_{\mathtt{it}-1} \mathbf{d}_{\mathtt{it}-1}$. However, we can advance the following arguments:

  1. In a successful `Projective Cutting-Planes` implementation, the feasible solutions $\mathbf{x}_{\mathtt{it}}$ constructed along the iterations `it` are rather well-centered and do *not* exhibit a "bang-bang" behaviour characterized by strong oscillations (see also arguments in Section 5.5). In a loose sense, the inner solutions $\mathbf{x}_{\mathtt{it}}$ are reminiscent of an interior point algorithm in which the solutions follow a central path [8, § 3.3], while the outer solutions $\mathtt{opt}(\mathscr{P}_{\mathtt{it}})$ are reminiscent of the `Simplex` algorithm. Recall that the `Simplex` algorithm can sometimes exhibit a "bang–bang" behaviour when advancing from one extreme solution to another. We can argue that, by linking the inner and the outer solutions, the `Projective Cutting-Planes` generates more well-centered paths, reducing the "bang-bang" effects.
  2. The projection sub-problem can generate stronger (normalized) constraints than the separation one. As described in Section 2.4.1 of [14], when $\mathbf{x} = \mathbf{0}_n$, the intersection subproblem $\mathtt{project}(\mathbf{x} \to \mathbf{d})$ is equivalent to normalizing all constrains and then choosing one by separating $\mathbf{x} + \mathbf{d}$. It is indeed most meaningful to compare two constraints when they are normalized, *i.e.*, when they have the same right-hand side value. Even when generalizing to $\mathbf{x} \neq \mathbf{0}_n$ as in the current paper, the

[11] As [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.

projection sub-problem could often still lead to one of the the most violated normalized constraints, stronger than the constraints 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 an intersection algorithm whose running time is similar to that of a separation algorithm. The `Projective Cutting-Planes` could thus be potentially useful to solve other problems with prohibitively-many constraints, beyond the four problem examples addressed in this paper. This could help one overcome certain limitations of current practices 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] Alysson 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 The numerical difficulties arising in the Benders reformulation 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 $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ 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 $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ at each iteration `it` can be particularly prone to numerical or precision problems especially if $\mathscr{P}_{\texttt{it}}$ contains too many constraints (3.2.8b) with disproportionately small or disproportionately large coefficients. The coefficients $\mathbf{u}$ of the variables $\mathbf{x}$, as determined by the sub-problem algorithm, can certainly be problematic. 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 this, we propose the following:

- If (3.2.11a)–(3.2.11e) has multiple optimal solutions, it is better to take one with reasonable coefficients; as such, to solve (3.2.11a)–(3.2.11e), the intersection algorithm breaks ties by minimizing $\mathbf{1}^\top \overline{\mathbf{u}}$.
- Before inserting a generated 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 and the new `Cutting-Planes` can sometimes require too much computation time for certain relaxed master ILPs associated to (3.2.8a)–(3.2.8c), *i.e.*, it can be too difficult to determine $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ at certain (rare) iterations `it`. Although such problems are not frequent, they could completely block the overall algorithm for a prohibitively long time, which can actually make the *integer* Benders model very difficult to solve.

The key to overcome this drawback cames from the fact that it is not really essential to determine the optimal solution $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ at each iteration `it`, as described next. Accordingly, we enforce a limit ($\frac{n}{120} + 1$ seconds) on the running time of the ILP solver; if this limit is exceeded, the `Cutting-Planes` continues with the best sub-optimal solution of $\mathscr{P}_{\texttt{it}}$ found by the ILP solver in the given time, which is different from $\texttt{opt}(\mathscr{P}_{\texttt{it}})$. 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 $\texttt{opt}(\mathscr{P}_{\texttt{it}})$. 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 $\overline{\mathbf{u}}$ can be unbounded. In fact, the only constraint that can limit the magnitude of $\overline{\mathbf{u}}$ is $-\sum_{\{i,j\}\in E} b_{\text{wd}} d_{ij} \overline{u}_{ij} = 1$ 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 $\overline{u}_{ij}$ variables. Furthermore, certain factors $b_{\text{wd}} x_{ij}$ in the sum $\sum_{\{i,j\}\in E} b_{\text{wd}} x_{ij} \overline{u}_{ij}$ 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 $b_{\rm wd}x_{ij}$ with an extremely large $\overline{u}_{ij}$ can lead to non-zero artificial (noising) terms in the objective function.

To reduce the above effects, we impose a limit of 100 on the maximum value the variables $\overline{\mathbf{u}}$ can take in (3.2.11a)–(3.2.11e). In fact, any imaginable algorithm for (3.2.11a)–(3.2.11e) has to impose such a limit in practice because the memory is nevertheless finite. This leads to restricting the feasible set of (3.2.11a)–(3.2.11e), and, in theory, the resulting restricted model 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 in theory. However, one can certify that the projection sub-problem is correctly solved by verifying that the optimal solution satisfies $\overline{u}_{ij} < 100\ \forall\{i,j\} \in E$. This correctness is very often confirmed, because the optimal $\overline{\mathbf{u}}$ hardly ever contains values larger than 0.5 in practice. When there is however some $\overline{u}_{ij} = 100$, this is most certainly due to the numerical issues above, *i.e.*, $b_{\rm wd}x_{ij}$ is slightly different from zero in practice although it should be zero in theory. Furthermore, the fact that an intersection point $\mathbf{x}_{\mathtt{it}}+t^*\mathbf{d}_{\mathtt{it}}$ 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` for the graph coloring model with RR stables (Section 4.1.4) optimizes a `Column Generation` model in which the constraints $\mathbf{a}^\top\mathbf{x} \leq 1$ are defined by cut-reinforced relaxed stables (RR stables) $\mathbf{a} \in \mathcal{P}$, *i.e.*, by the (extreme) solutions of the auxiliary polytope $\mathcal{P}$ from Definition 2 (p. 19). This auxiliary polytope $\mathcal{P}$ is an outer approximation of the stable set polytope (*i.e.*, of the convex closure of the standard stables). It is defined by six cut classes 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 cut classes (a)–(f) without translating them to a form like (4.1.6b) using the Charnes–Cooper transformation. 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 `cut generation` (as already hinted in Section 4.1.4.2).

(a) The first cut class simply comes from the edge inequalities defining the standard 0–1 stables, *i.e.*, impose $a_u + a_v \leq 1\ \forall\{u,v\} \in V$ to construct the fractional stable polytope.

(b) Generalizing the above idea, the class cut (b) generates one by one a number of clique inequalities of the form $\mathbf{a}(\mathscr{C}) = \sum_{v\in\mathscr{C}} a_v \leq 1$, for cliques $\mathscr{C}$ of maximum size $\min(5,k)$. The role of the parameter 5 is to keep the number of such cliques to small values so as to simplify their enumeration. These cliques are enumerated using a very straightforward backtracking algorithm, but any vertex is ignored if it already appeared in 20 inequalities, *i.e.*, after 20 apparitions, the vertex is discarded to avoid combinatorial explosions.

(c) The cuts (c) construct a collection of clique inequalities that "cover" V. They are iteratively generated by going through 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 $\mathscr{C}$ of a given maximum size k' (see below) that contains the vertex $i = 1$ and imposes the clique inequality $\mathbf{a}(\mathscr{C}) \leq 1$. We now introduce a set $V' = V\backslash\mathscr{C}$ that will evolve along the iterations; all subsequent cliques will be determined by maximizing the number of elements from V'. We then move to the next element $i \in V'$ to determine a new clique $\mathscr{C} \ni \{i\}$ of maximum size k' and impose $\mathbf{a}(\mathscr{C}) \leq 1$. After performing $V' \leftarrow V'\backslash\mathscr{C}$, we move to the next iteration and repeat. At each iteration, we search for a clique $\mathscr{C}$ 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 $v \in V \setminus V'$. We assign to k' the minimum clique size for which this BBBS algorithm can solve within a CPU time of less than 0.01s 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 for the Leighton graphs (`le450_25c`, `le450_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 $\mathbf{a}^{\rm std}$ satisfies the following:

$$a^{\rm std}_u + a^{\rm std}_v \leq a^{\rm std}_w + \mathbf{a}^{\rm std}(N_w - N_u \cap N_v),$$

because if the maximum stable $\mathbf{a}^{\text{std}}$ 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:

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

However, we decided to insert such cuts only when they have less than 10 non-zero coefficients, because 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 the RR stable $[1/\omega \;\; 1/\omega \;\; 1/\omega \ldots 1/\omega]$, where ω is the maximum clique size of G. Based on this, Remark 4 (p. 21) 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 as described next. First, notice than a (simple) odd cycle H yields a cut $\mathbf{a}(H) \leq \frac{|H|-1}{2}$, because a stable with $\frac{|H|+1}{2}$ vertices of H would have select 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 searching for the shortest odd cycle in a graph with edge weights $1 - a_u - a_v \; \forall \{u,v\} \in E$, which 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 $\mathbf{a}(\mathscr{C}) \leq 1$ generated by cliques $\mathscr{C}$ with at maximum k elements, where k is a parameter that defines the model — it 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, repeatedly solving this problem can represent the most important computational bottleneck of the overall general `Cutting-Planes`. It is worth investigating in Appendix A.2.3 a specific `Branch & Bound with Bounded Size` (BBBS) algorithm for this problem.

A.2.2 Accelerating the Cut Generation using Stabilization

Recall that the Charnes–Cooper LP formulation (4.1.6a)–(4.1.6d) of the projection sub-problem from Section 4.1.4.1 is solved by `cut generation`, repeatedly separating the above (a)–(f) cuts as first described in Section 4.1.4.2. More exactly, the constraints (a)–(d) are statically inserted in (4.1.6a)–(4.1.6d) at the very first `Cutting-Planes` iteration. A positive distinguishing characteristic of this (4.1.6a)–(4.1.6d) formulation is that the cuts (4.1.6b) generated at each iteration `it` of the overall `Cutting-Planes` can be kept throughout all subsequent iterations, because they not depend on the current $\mathbf{x}_{\texttt{it}}$ or $\mathbf{d}_{\texttt{it}}$.

All cuts (e) can be separated quite rapidly by applying the 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 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 this fails separating the current optimal solution, we have to call the separation algorithm again, this second time on the current optimal solution.

2. We propose a heuristic algorithm for the maximum weight clique problem before applying the BBBS algorithm. This heuristic algorithm executes $5 \cdot n$ iterations of a Tabu Search algorithm.[12] We always start the `cut generation` in a heuristic mode, trying to solve all maximum weighted 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 bounded size maximum weight clique is a rather general graph–theoretic problem that could be modelled and solved with many different methods. However, perhaps surprisingly at first glance, we did not find any dedicated off-the-shelf software to solve it as rapidly as we would have liked. 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, this BBBS can directly solve the complementary problem, *i.e.*, the maximum weight stable with bounded size. We thus also used BBBS to solve the separation sub-problem of the classical `Column Generation` algorithm with standard stables in Section 5.3.2.

The BBBS algorithm relies on a fairly straightforward Branch & Bound (B&B) routine that constructs increasingly larger cliques (B&B nodes) by successively adding vertices to existing cliques, generating a branching tree. The number of generated B&B nodes (and the total running time) depends substantially on the quality of the lower and upper bound determined at each node of the branching tree. While the lower bound can be simply given by the best clique ever generated, we will see the upper bound is more important and we will present 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 $w_1 \geq w_2 \geq w_3 \geq \cdots \geq w_n$, *i.e.*, the vertices are initially sorted by decreasing weight. As such, the very first B&B node is simply the clique $\{v_1\}$. The B&B tree is constructed in a deep-first-search manner. For example, 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 lower bound at each B&B node is simply determined by the best clique ever constructed. Also, recall (point 2 from Appendix A.2.2) that one can first try to solve the maximum weighted clique using a heuristic prior to launching BBBS; this can provide a second lower bound. Preliminary experiments suggest that trying other better or faster heuristics do not usually lead to an impressive acceleration of the BBBS.

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

We now present a higher-quality upper bound that could significantly improve BBBS, up to reducing the running time by a factor of seven. The algorithm for calculating it is outlined by the pseudo-code below; the underlying ideas are described next. Recall that the above basic upper bound simply sums up the weights of the first $k-k'$ elements in $L_{\mathscr{C}}$. 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 $L_{\mathscr{C}}$ can *not* contribute to the upper bound value, if there is a preceding $v \in L_{\mathscr{C}}$ (of higher weight) that already contributed to the bound and $\{v, u\} \notin E$. We say that v shadows u; notice how the pseudo-code below breaks the loop processing u because of the **continue** statements. 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

[12] This Tabu Search algorithm encodes the candidate solutions as bit strings 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+`random(5)` iterations, where `random(5)` returns a uniformly random integer value in $\{0, 1, 2, 3, 4, 5\}$. For $k = \infty$, we used the we used the multi-neighborhood Tabu search (`www.info.univ-angers.fr/~hao/clique.html`) due to Q. Wu, J.K. Hao and F. Glover.

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: $\texttt{ub} \leftarrow \sum_{v \in \mathscr{C}} \texttt{weight}(v)$, $\texttt{addedVtx} = 0$
2: $L \leftarrow \emptyset$ ▷ vertices v that shadow other vertices
3: $L' \leftarrow \emptyset$ ▷ pairs (v, u) such that v shadows u
4: **for all** $u \in L_{\mathscr{C}}$ **do** ▷ scan $L_{\mathscr{C}}$ by descending order of weight
5: **if** $\exists v \in L$ such that $\{v, u\} \notin E$ **then** ▷ v shadows u, *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: $\texttt{ub} \leftarrow \texttt{ub} + \texttt{weight}(u)$, $\texttt{addedVtx} \leftarrow \texttt{addedVtx} + 1$
15: $L \leftarrow L \cup \{u\}$ ▷ u can shadow subsequent vertices in $L_{\mathscr{C}}$
16: **if** $(\texttt{addedVtx} == k - |\mathscr{C}|)$ **then**
17: **break**
18: **end if**
19: **end for**
20: **return** 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 $\min(10, \frac{2}{3}k)$. Also, 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 originally used in *Cutting-Stock* are essentially more general: a fast data structure for handling Pareto frontiers and Lagrangian bounds

This section examines two concepts that were originally needed by the *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 discusses 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) that the Dynamic Programming (DP) scheme has to manipulate a list of of states whose cost and profits satisfy the relation (4.2.3.a)–(4.2.3.b) that we can recall as follows. We are given a list of $|I|$ cost/profit pairs $c_i/p_i \ \forall i \in I$ that constitute a Pareto frontier in the sense that they respect:

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

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

As stated in Remark 5, one of the most computationally expensive operations (of Algorithm 1, p. 24) is the insertion of a new pair (at Step 5), because 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 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).

For each new pair c^+/p^+, one has to decide as rapidly as possible if c^+/p^+ should be inserted into the list or if c^+/p^+ is dominated; scanning the whole list of existing pairs for this task could be too expensive computationally. To avoid this, we propose to record the list I in a self-balancing binary tree following 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. The insertion operator of this binary tree needs 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 if c^+/p^+ should be inserted before or after c^*/p^* in the binary tree. Any implementation of the self-balancing tree thus has to provide a means to determine c^*/p^* — at least, it is certainly possible to (temporary) insert c^+/p^+ in the tree and return the element before c^+/p^+.

Once c^*/p^* is determined, we propose an insertion routine that performs the following. First, if $p^* \geq p^+$, then the new pair c^+/p^+ is directly rejected (recall that $p^* < p^+$ has to hold if $c^* < c^+$ and c^*/p^* is non-dominated). Otherwise, if $p^* < p^+$, then c^+/p^+ has to be inserted in the tree, and so, other recorded pairs can 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 insertion routine enumerates one by one all next recorded pairs c^-/p^- ordered 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 sections for both graph coloring and *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 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

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

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

In *Multiple-Length Cutting-Stock*, the column costs are no longer unitary and the above bound can evolve to $\dfrac{\mathbf{b}^\top \mathbf{x}}{1 - \frac{1}{c_{\min}} m_{rdc}(\mathbf{x})}$, where $c_{\min}$ is the minimum cost of a feasible 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 based on 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, $\dfrac{\mathbf{b}^\top \mathbf{x}}{1 - m_{rdc}(\mathbf{x})}$ evolves to:

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

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 $\frac{0.9}{1 - \frac{1}{0.6} 0.1} = 1.08$ which is *not* a valid lower bound, since the optimum for this instance is 1 (cut both items from a standard-size piece of length 1).

Recall (last paragraph of Section 4.2.2) that the first two iterations of the `Projective Cutting-Planes` for *Multiple-Length Cutting-Stock* solve $\mathtt{project}(\mathbf{0}_n \to \frac{1}{W}\mathbf{w})$ and $\mathtt{project}(\mathbf{0}_n \to \mathbf{b})$, which leads to 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 $\mathbf{x} = \frac{1}{W}\mathbf{w}$ which is a feasible solution with $m_{rdc}(\frac{1}{W}\mathbf{w}) \geq 0$. 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 $\frac{1}{W}\mathbf{w}$ and $\mathbf{b}$.

B Complementary experimental information

We recall the C++ source code is publicly available on-line for all four considered problems at cedric.cnam.fr/~porumbed/graphs/projcutplanes/. 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. 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 The robust optimization problem

Most instances used in 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/graphs/projcutplanes/robust/.

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, ..., g. The instances rnd-10 corresponds 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 exact references on their (industrial) origins. For each set, the number 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 below.

Name	n	capacity W	avg. demand $\mathbf{b}$	avg. $\mathbf{w}$ span	description
`m01`	100	100	1	$[1, 100]$	1000 random instances;
`m20`	100	100	1	$[20, 100]$	1000 random instances;
`m35`	100	100	1	$[35, 100]$	1000 random 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* instances