Ray Projection for Optimizing Polytopes with Prohibitively Many Constraints in Set-Covering Column Generation

Daniel Porumbel*

Abstract

A recurrent task in mathematical programming consists of optimizing polytopes with prohibitively-many constraints, *e.g.*, the primal polytope in cutting-planes methods or the dual polytope in Column Generation (CG). We present a method to optimize such polytopes using the following *intersection sub-problem*: given ray $\mathbf{r} \in \mathbb{Z}^n$, find the maximum $t^* \geq 0$ such that $t^*\mathbf{r}$ is feasible. We interpret $\mathbf{0}_n \rightarrow \mathbf{r}$ as a direction of growth from the origin $\mathbf{0}_n$; $t^*\mathbf{r}$ is the intersection point between $\mathbf{0}_n \rightarrow \mathbf{r}$ and the polytope boundary. To ease the exposition, the first part of the paper is devoted to the general context of a (primal or dual) polytope with rational data such that: (i) the intersection sub-problem can be effectively solved in practice for input rays $\mathbf{r} \in \mathbb{Z}^n$ and (ii) $\mathbf{0}_n$ is feasible. By iterating over such rays $\mathbf{r}$, our method produces a sequence of feasible solutions $t^*\mathbf{r}$ that we prove to be finitely convergent. From Section 3 on, we focus on dual polytopes in CG formulations. Typically, if the CG (separation) sub-problem can be solved by Dynamic Programming (DP), so can be the *intersection sub-problem*. The proposed Integer Ray Method (IRM) only uses integer rays, and so, the intersection sub-problem can be solved using a DP scheme based on states indexed by integer ray values. We show that under such conditions, the *intersection sub-problem* can be even easier than the CG sub-problem, especially when no other integer data is available to index states in DP, *i.e.*, if the CG sub-problem input consists of fractional (or large-range) values. As such, the IRM can tackle *scaled* instances (with large-range weights) of capacitated problems that seem prohibitively hard for classical CG. This is confirmed by numerical experiments on various capacitated `Set-Covering` problems: `Capacitated Arc-Routing`, `Cutting-Stock` and other three versions of `Elastic Cutting-Stock` (*i.e.*, a problem class that includes `Variable Size Bin Packing`).

1 Introduction

The optimization of Linear Programs (LPs) with-prohibitively many constraints has a rich history in mathematical programming, *e.g.*, consider the primal LP in cutting-planes methods or the dual LP in Column Generation (CG). In both cases, the goal can be formulated using a (primal or dual) LP as follows

$$\max \left\{ \mathbf{b}^\top \mathbf{x} : \ A\mathbf{x} \leq \mathbf{c}, \ \mathbf{x} \geq \mathbf{0}_n \right\}, \tag{1.1}$$

where $\mathbf{x} = [x_1 \dots x_n]^\top$ are the decision variables, $\mathbf{b} \in \mathbb{R}^n$, $\mathbf{c} \in \mathbb{Q}_+^n$, and $A \in \mathbb{Q}^{m \times n}$. Although we never formally impose a condition of the form $m \gg n$, the paper is devoted to large-scale LPs: the constraint matrix A (polytope $\mathcal{P}$) has too many rows (facets) to be feasibly enumerated in practice.

A highly successful approach for optimizing large-scale LPs rely on *dual (outer) methods*: drive a dual (primal infeasible) solution towards an optimum solution of (1.1), by progressively removing infeasibility. For instance, in cutting-plane methods and *branch-and-cut*, one iteratively adds constraints that separate the current infeasible solution from $\mathcal{P}$. A similar (dual) process takes place in Column Generation (CG): dual constraints (primal columns) are iteratively generated to separate the current infeasible (dual) solution from the (dual) polytope $\mathcal{P}$. In both cases, the current solution at each iteration is the optimum of some "outer

*CEDRIC CS Lab, CNAM, 292 rue Saint-Martin, F-75141 Paris cedex 03, France
Contact: daniel.porumbel@cnam.fr

polytope" that contains $\mathcal{P}$. The process converges to an optimal $\mathcal{P}$ solution $\texttt{OPT}(\mathcal{P})$ through a sequence of such exterior (infeasible) solutions.

While this paper does use ideas from the literature of *dual methods* (*i.e.*, to get upper bounds using "outer polytopes" as above), the proposed IRM is rather designed as a *primal method*. Primal and dual methods for (1.1) are compared in more detail in Section 1.1.1, but for the moment it is enough to say that a *primal method* proceeds by converging a sequence of interior (feasible) solutions rather than exterior solutions.

In IRM, these interior (feasible) solutions are actually situated on some facets of the feasible polytope $\mathcal{P}$. They are generated using the *ray projection* technique: advance from $\mathbf{0}_n$ along ray direction $\mathbf{0}_n \rightarrow \mathbf{r}$ until intersecting the boundary of $\mathcal{P}$ on a first-hit facet. The intersection point is determined by solving the *intersection* (or IRM) sub-problem on input ray $\mathbf{r}$: find the *maximum feasible step length* t^* such that $t^*\mathbf{r} \in \mathcal{P}$.

Our ray approach offers a relatively-high flexibility in choosing the rays: any point in the space of $\mathcal{P}$ can be potentially used as a ray direction. This flexibility can be useful for simplifying the intersection sub-problems: it is generally easier to solve a (sub-)problem on some controllable input data $\mathbf{r}$ than on a rather unpredictable multiplier $\mathbf{x} = \text{OPT}(\mathcal{P}')$ determined by optimizing some outer polytope $\mathcal{P}' \supset P$. In our CG models, we only use *integer* rays $\mathbf{r}$, so as to render the intersection sub-problem tractable by $\mathbf{r}$-indexed Dynamic Programming (see Section 3.2). The fact that the rays are integer could also simplify intersection sub-problems in cutting-planes models (see an example in Section 1.1.1.3).

The very first ray is the objective function vector $\mathbf{b}$ (or a rounding of $\mathbf{b}$ when necessary). As such, IRM starts out by advancing on the direction with the fastest rate of objective improvement. Then, IRM iterates the following steps (see Figure 1):

(1) solve the intersection sub-problem for the current ray $\mathbf{r}$: determine the maximum t^* such that $t^*\mathbf{r}$ is feasible. This generates lower bound solution $\mathbf{x}_{\text{lb}} = t^* r$ and also a $\mathbf{x}_{\text{lb}}$-tight constraint (a "first hit" facet of $\mathcal{P}$).

(2) optimize the current outer polytope, *i.e.*, the polytope delimited by the first-hit facets discovered while solving intersection sub-problems at (1). This leads to an upper bound $\mathbf{x}_{\text{ub}}$, *i.e.*, an outer solution for $\mathcal{P}$.

(3) determine the next ray and repeat from (1). The next ray $\mathbf{r}_{\text{new}}$ is obtained by finding integer solutions in the proximity of the segment joining $\mathbf{x}_{\text{lb}}$ and $\mathbf{x}_{\text{ub}}$ (see Section 2.3), *i.e.*, in the proximity of some points of the form $\mathbf{r}+\beta(\mathbf{x}_{\text{ub}}-\mathbf{x}_{\text{lb}})$. Each new ray $\mathbf{r}_{\text{new}}$ can lead the IRM sub-problem to: (a) a better lower bound solution $t^*_{\text{new}}\mathbf{r}_{\text{new}}$, or (b) a new constraint that separates $\mathbf{x}_{\text{ub}}$ from $\mathcal{P}$, which later leads to an update of $\mathbf{x}_{\text{ub}}$ (Section 2.4).

When the IRM can no longer decrease the gap between $\mathbf{x}_{\text{lb}}$ and $\mathbf{x}_{\text{ub}}$, it applies discretization refining to increase (double) the coefficients of the next generated rays. The main idea is to look up more fine-grained rays, trying to get closer to $\mathbf{0}_n \rightarrow \texttt{OPT}(\mathcal{P})$.

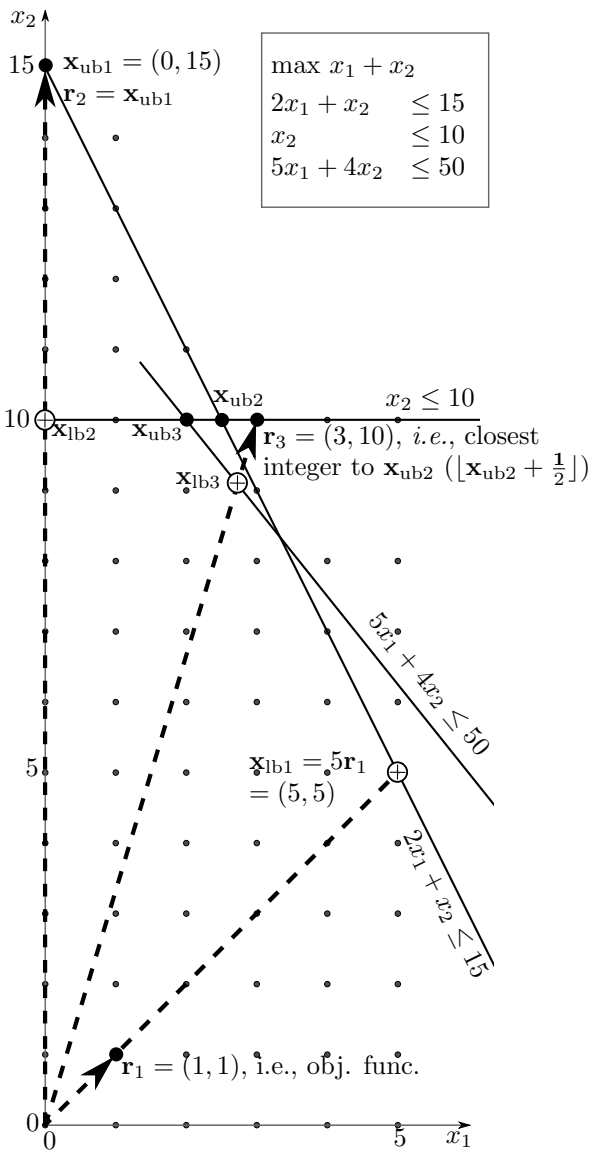


Figure 1: Three IRM steps at a glance. Some ray choices are voluntarily simplified to reduce clutter: (1) ray $\mathbf{r}_1$ leads to bounds $\mathbf{x}_{lb1}$ (*i.e.*, the hit point of $\mathbf{0}_n \rightarrow \mathbf{r}_1$) and $\mathbf{x}_{\text{ub1}}$ (*i.e.*, the optimum of outer polytope $\{\mathbf{x} \in \mathbb{R}^2_+ : \ 2x_1 + x_2 \leq 15\}$), (2) $\mathbf{r}_2$ leads to $\mathbf{x}_{\text{lb2}}$ and $\mathbf{x}_{\text{ub2}}$, (3) $\mathbf{r}_3$ to $\mathbf{x}_{\text{lb3}}$ and $\mathbf{x}_{\text{ub3}}$. At step (iv), IRM will use $\mathbf{r}_4 = \mathbf{x}_{\text{ub3}}$ and finish, see Appendix A for full numerical details.

1.1 Context and Related Ideas

We first place the proposed method in the general perspective of large-scale LPs (Section 1.1.1); then, we continue in Section 1.1.2 with related CG work.

1.1.1 General LPs with Prohibitively-Many Constraints

1.1.1.1 Primal and Dual Methods for Large-Scale LPs As hinted above, we consider two broad classes of optimization algorithms for large-scale LPs: dual (or outer) methods and primal (or inner) methods—see also [18], the introduction of [1], or [24]. While our goal is not to perform a detailed overview of such a vast topic,[1] we briefly place IRM in this context:

dual (outer) methods This is the most wide-spread approach, generally including: (i) most cutting-plane algorithms for solving primal Integer LPs (ILPs) and (ii) CG methods as interpreted from the perspective of the dual LP in CG formulations. They drive a dual-feasible solution towards primal feasibility.

primal (inner) methods Such methods were first studied in the 1960s for solving ILPs (*e.g.*, see references in [18, §1] or in [24, § 4.1]). They rely on repeating the following operation: start from the current integer feasible solution and find a cut on the simplex tableau such that the resulting simplex pivot leads to a new feasible solution of higher quality that is still integer. These cuts can be seen as a tool for driving the current primal feasible solution towards the optimum integer solution.

1.1.1.2 Ray Projections in Linear Programming We now focus on techniques closer to our central idea of *ray projection in large-scale LPs*. As hinted above, certain (generally) related ideas arise in primal algorithms for ILPs. Such algorithms are often described in terms of pivot operations in the Simplex tableau and on (Chvátal-Gomory) cuts. However, their dynamics can also be presented as a process that moves from one integer feasible solution to another by *integer augmentation*, *e.g.*, see Step 4 in the algorithm from [18, §3.1] or Step (2.b) in Algorithm 1.1 of [24]. In this latter example, the next *integer* feasible solution is determined by advancing an *integer* length *from the current solution along an integer augmenting direction*. These integer lengths are essential in an Integer LP context, because the goal is to produce integer solutions.

Certain cutting-plane methods do use interior points to provide a stable behavior for the convergent process. For instance, instead of solving the separation problem on the optimal solution $\mathbf{x} = \texttt{OPT}(\mathcal{P}')$ of the current outer dual polytope $P' \supset P$, one can replace $\mathbf{x}$ with a solution situated on the segment joining $\texttt{OPT}(\mathcal{P}')$ and an interior (feasible) solution of $\mathcal{P}$—see [5, p. 4] for more detailed discussions on these aspects.

Other forms of ray projections arise in the field of Submodular Function Minimization (SFM). One of the simplest approaches for optimizing the submodular polytope is the *Greedy* algorithm [21, Sec. 2.1]. Given a linear order of the variables, *Greedy* progresses towards the optimum by increasing one by one each variable to its maximum value. Each such maximum value can be seen as a "maximum step length" that one can advance in the direction of the variable until intersecting a polytope facet. Generalizing this idea, Schrijver's algorithm advances on more complex directions, by simultaneously increasing one variable while decreasing some other. In certain cases, one can exactly determine the maximum feasible step length on such directions [21, Lemma 2.5], but Schrijver's algorithm is making use of lower bounds for it. The problem of finding the intersection of a line with a sub-modular polytope (or polymatroid) is referred to as the "line search problem" [22] or as the "intersection problem" [10]. The intersection algorithm from [22] consists of

[1] More refined classifications can be found in the literature of Integer LPs, for instance in the introduction of [18]. As the abstract of this paper put it in 2002, "Dual fractional cutting plane algorithms, in which cutting planes are used to iteratively tighten a linear relaxation of an integer program, are well-known and form the basis of the highly successful branch-and-cut method. It is rather less well-known that various primal cutting plane algorithms were developed in the 1960s, for example by Young. In a primal algorithm, the main role of the cutting planes is to enable a feasible solution to the original problem to be improved. Research on these algorithms has been almost non-existent." The introduction of the more recent handbook [1] states: "Relaxation or dual methods, such as cutting plane algorithms, progressively remove infeasibility while maintaining optimality to the relaxed problem. Such algorithms have the disadvantage of possibly obtaining feasibility only when the algorithm terminates. Primal methods for integer programs, which move from a feasible solution to a better feasible solution, were studied in the 1960s but did not appear to be competitive with dual methods. However, recent development..."

solving a minimum ratio problem. Our intersection sub-problem (Definition 1, Section 2.2) is also formulated as a ratio minimization problem. The last section of [10] discusses certain perspectives on the applicability of related approaches to other polytopes with prohibitively many constraints, but there is no mention of CG.

Interior point algorithms for LPs also share some general ideas with the ray projection technique. They differ in the sense that each iteration of an interior point algorithm does not advance on a fixed direction until its hits (pierce) the boundary of the polytope; it rather stops on some strictly-interior point before changing the direction for the next iteration.

1.1.1.3 Possible Approaches for the Intersection Sub-problem The intersection sub-problem can be seen as a generalization of the more widespread separation problem. In some cases, the intersection sub-problem could be solved by generalizing existing separation techniques, *e.g.*, see the polarity technique described in Section 2.2.2. In addition, we will see that the intersection sub-problem reduces to finding a constraint that minimizes a cost/profit ratio. This approach for selecting the constraint also arises in the separation algorithms from the very recent CG work [2], as well as in certain Simplex flavors (in steepest-edge pivot rules). Furthermore, if all values of $\mathbf{c}$ are equal, the intersection sub-problem (cost/profit minimization) is equivalent to the separation sub-problem (minimizing a difference between cost and profit). More generally, the intersection sub-problem can always be solved using a repeated call to a separation oracle: solve the separation problem on solutions $t\mathbf{r}$ (with $t \geq 0$) and converge t to the sought t^* (see also Section 2.2.2).

The use of *integer-only* rays is very convenient for solving the intersection sub-problems in our CG models. However, generally speaking, cutting-planes models can also take profit from using integer-only data. For instance, the sub-problem of separating a solution (noted x^*) for the Symmetric Travelling Salesman Problem (STSP) can become easier when x^* is integer; as [18, § 3.2] put it, "it is much easier to identify violated subtour elimination inequalities when x^* is integral than when x^* is arbitrary. One merely has to compute connected components in a graph.". More generally, any flexibility in the input choice can be potentially useful for other (sub-)problems of exponential complexity for the worst-case input (see theoretical examples in Section 4.1.2.1). By slightly modifying the input, one could often avoid the worst-case complexity of a (sub-)problem of lower average complexity (or of good *smoothed complexity*[2]).

1.1.2 Column Generation (CG)

We first recall that CG can be seen as a cutting-plane algorithm (*e.g.*, Kelley's method) in the dual polytope $\mathcal{P}$. At each iteration, the standard CG method takes the optimum solution $\mathbf{x} = \texttt{OPT}(\mathcal{P}')$ of an outer dual polytope $\mathcal{P}' \supset \mathcal{P}$ and calls a separation oracle to solve the pricing sub-problem: find (and add) the most violated (primal column) dual constraint with respect to exterior solution $\mathbf{x}$. Important progress has been done in CG over the last decades by developing innovative methods of column management and stabilization. Such methods aim of providing a more stable behaviour to the evolution and convergence of the exterior solution $\mathbf{x}$—see, chronologically, the work in [3, 19, 27, 7, 9, 13]. Many CG stabilization ideas have a dual counterpart in the literature of cutting-plane acceleration. For instance, the separation sub-problem can lead to better constraints (cutting off larger parts of $\mathcal{P}' \setminus \mathcal{P}$) if the exterior solution $\mathbf{x}$ is chosen more carefully, as in the examples below (see also [5, §2] for more comprehensive discussions addressing both CG and cutting–planes). In most cases, one wants to minimize the number of constraints needed throughout the process.

However, IRM is not mainly devoted to minimizing the number of iterations, but to converging a sequence of feasible solutions by iterating over integer rays. The choice of the rays shares some similarities with the choice of the exterior solution $\mathbf{x}$ in stabilized CG. Instead of using $\mathbf{x} = \texttt{OPT}(\mathcal{P}')$, certain stabilized CG [7, 13] methods use some interior (more central) dual solutions. However, we are not aware of any CG methods that (try to) select *integer* dual solutions $\mathbf{x}$ as input for the sub-problem; typical choices are: the optimum of the current dual program (pure CG), the optimum of a stabilized dual program [7, Alg. 1.5], a well-centered ϵ-optimal solution [13, Def. 1], etc.

[2] An algorithm with good smoothed complexity performs well on almost all inputs in every small neighborhood of inputs (under slight perturbations of worst-case inputs).

To summarize, IRM differs from (stabilized) CG in several aspects. First, IRM automatically provides a sequence of built-in intermediate lower bounds $\mathbf{x}_{\text{lb}} = t^*\mathbf{r}$ that are different from the Lagrangean bounds (optionally) used in CG. Secondly, the IRM rays are not necessarily infeasible dual solutions as in CG, but simply integer directions in the dual space. Thirdly, the input $\mathbf{r}$ of the intersection sub-problem can be more easily controlled, and this can render the intersection sub-problem even easier than the separation one. By using integer rays $\mathbf{r}$, the (integer) intersection sub-problem can be tractable by ($\mathbf{r}$-indexed) Dynamic Programming (DP), while the CG separation sub-problem can be practically intractable on non-integer input multipliers $\mathbf{x} \notin \mathbb{Z}^n$.

Profit-indexed DP is often used in Fully Polynomial-Time Approximation Schemes (FPTASs), *e.g.*, the first knapsack FPTAS [14] scales and rounds the profits to apply *profit-indexed DP* afterwords. Such rounding was also used in certain CG approaches as a heuristic to accelerate the pricing [11, § 3]. However, the IRM does not really need rounding, as it directly chooses integer rays. It is the general idea of *profit-indexed (or* $\mathbf{r}$*-indexed) DP* that is fully exploited by IRM: ray $\mathbf{r}$ is interpreted as a vector of integer profits in all IRM sub-problems (see Section 3.2 and the examples in Sections 4.1 and 4.2).

To conclude, ray projection approaches seem partially overlooked in CG. Despite our best efforts, we did not find any other research thread that is more specifically relevant to the central IRM idea (ray projection) utilized in the context of dual LPs in CG.

1.2 Paper Organization

The paper is organized in two parts. We first present the IRM in Section 2 as a method for optimizing general LPs such as (1.1). This IRM description only relies on the following: (a) we are given a large-scale LP and (b) we are given a routine that solves intersection sub-problems for any input ray with integral components. Under such conditions, we will prove that the method converges in a finite number of steps (Section 2.4.3).

From Section 3 on, we introduce the CG interpretation of (1.1) and the second part of the paper is focuses on CG aspects. Section 4 provides solution methods for the intersection sub-problems of `Elastic Cutting-Stock` and `Capacitated Arc Routing`. Numerical experiments are provided in Section 5, followed by conclusions in Section 6.

2 Integer Ray Method

The constraint matrix A in (1.1) is considered too large to be fully recorded and manipulated by a practical algorithm. As such, it is more convenient to model it using a set $\overline{\mathcal{A}}$ of elements of the form $[c_a, \mathbf{a}]$ that correspond to constraints $\mathbf{a}^\top\mathbf{x} \leq c_a$; in (1.1), $[c_a, \mathbf{a}]$ is equivalent to $[c_i, A_i]$, where A_i is some row of A. Our method will often manipulate constraint sub-sets $\mathcal{A} \subset \overline{\mathcal{A}}$, assuming (but not formally imposing) $|\mathcal{A}| \ll |\overline{\mathcal{A}}|$, *e.g.*, $\mathcal{A}$ might represent the constraints discovered so far at some stage of the execution. We use the notation $\mathcal{P}_\mathcal{A}$ to refer to an outer polytope $\mathcal{P}_\mathcal{A} \supset \mathcal{P}$ that is constructed from $\mathcal{P}$ only keeping constraints $\mathcal{A} \subset \overline{\mathcal{A}}$.

Using these notations, we will describe the IRM on model (2.1) below, which is a translation of (1.1). Recall that we only consider two particularizing assumptions: (i) the intersection sub-problem can be solved in practice for any input ray $\mathbf{r} \in \mathbb{Z}^n$ (see Section 2.2); (ii) $\mathbf{0}_n$ is a feasible solution, which is equivalent to $c_a \geq 0, \forall [c_a, \mathbf{a}] \in \overline{\mathcal{A}}$ (to see this, only remark that $\mathbf{a}^\top\mathbf{0}_n$ is always 0).

$$\left.\begin{array}{l} \max \mathbf{b}^\top\mathbf{x} \\ \mathbf{a}^\top\mathbf{x} \leq c_a, \quad \forall [c_a, \mathbf{a}] \in \overline{\mathcal{A}} \\ \mathbf{x} \geq \mathbf{0}_n \end{array}\right\} \mathcal{P} \tag{2.1}$$

2.1 Main Building Blocks and Pseudocode

We introduce the IRM using 4 steps briefly described below. Each step is discussed in greater detail in a dedicated subsection indicated in parentheses (except Step 2 which is rather straightforward).

1. *Solve the IRM sub-problem to determine a lower bound solution (Section 2.2)* Advance on a given ray $\mathbf{r}$ until intersecting a first facet (constraint) of $\mathcal{P}$ at $t^*\mathbf{r} \in \mathcal{P}$. This leads to a lower bound solution $\mathbf{x}_{\text{lb}} = t^*\mathbf{r}$, and to a (first-hit) constraint $\mathbf{a}^\top\mathbf{x} \leq c_a$ that is $\mathbf{x}_{\text{lb}}$-tight (*i.e.*, $\mathbf{a}^\top\mathbf{x}_{\text{lb}} = c_a$). This first-hit constraint is added to the set $\mathcal{A}$ of currently available constraints ($\mathcal{A} \leftarrow \mathcal{A} \cup \{[c_a, \mathbf{a}]\}$);

2. *Calculate upper bound solution* $\mathbf{x}_{\text{ub}}$ *(no dedicated subsection)* This $\mathbf{x}_{\text{ub}}$ is simply determined by optimizing $\mathbf{b}^\top\mathbf{x}$ over $\mathbf{x} \in \mathcal{P}_\mathcal{A}$, where $\mathcal{P}_A \supset \mathcal{P}$ is the polytope obtained from $\mathcal{P}$ by only keeping constraints $\mathcal{A} \subset \overline{\mathcal{A}}$. We can write $\mathbf{x}_{\text{ub}} = \texttt{OPT}(\mathcal{P}_\mathcal{A})$, and so, $\mathbf{b}^\top\mathbf{x}_{\text{ub}}$ is an upper bound for (2.1). We will also consider the exceptional case in which this optimum is not a proper vertex, but an extreme (unbounded) ray of open polyhedron $\mathcal{P}$.

3. *Generate Next Rays (Section 2.3)* Given $\mathbf{r}, \mathbf{x}_{\text{lb}}$ and $\mathbf{x}_{\text{ub}}$, search for new rays among the closest integer points to $\mathbf{r}+\beta(\mathbf{x}_{\text{ub}}-\mathbf{x}_{\text{lb}})$, with $\beta > 0$. For a fixed β, the closest integer point is determined by applying a simple rounding on each of the n positions of $\mathbf{r} + \beta(\mathbf{x}_{\text{ub}} - \mathbf{x}_{\text{lb}})$. New potential better rays $\mathbf{r}_{\text{new}}$ are iteratively generated by gradually increasing β until one of them leads to some lower or upper bound update (using Step 4 below). In fact, if none of the proposed rays can update either bound, we consider that the current ray coefficients are too small (imprecise) to reduce the gap. In this case, IRM calls a "discretization refining" routine (see Section 2.3.3) that increases the ray coefficients: multiply $\mathbf{r}$ by $\lambda \in \mathbb{Z}_+$ (we used $\lambda = 2$) and divide t^* by λ; as such, $\mathbf{x}_{\text{lb}} = t^*\mathbf{r}$ and $\mathbf{x}_{\text{ub}}$ stay unchanged. This allows our ray generators to find new larger rays (*i.e.*, from updated $\mathbf{r}, \mathbf{x}_{\text{ub}}$ and $\mathbf{x}_{\text{ub}}$) with higher chances of improving the bounds (see below), at the cost of a potential slowdown of the intersection calculations.

4. *Complete a major iteration: update* $\mathbf{x}_{\text{lb}}$ *or* $\mathbf{x}_{\text{ub}}$ *(Section 2.4)* The new rays $\mathbf{r}_{\text{new}}$ generated by Step 3 are iteratively provided as input to the intersection sub-problem. Each such $\mathbf{r}_{\text{new}}$ can either lead to a lower bound improvement (if the first-hit solution $t^*_{\text{new}}\mathbf{r}_{\text{new}} \in \mathcal{P}$ dominates $\mathbf{x}_{\text{lb}} = t^*\mathbf{r}$) or to an upper bound update (if the first-hit constraint separates $\mathbf{x}_{\text{ub}}$). As soon as $\mathbf{x}_{\text{lb}}$ or $\mathbf{x}_{\text{ub}}$ is updated, the ray generation from Step 3 is *restarted* with new input data (*i.e.*, with updated $\mathbf{r}, \mathbf{x}_{\text{ub}}$ or $\mathbf{x}_{\text{lb}}$). If neither bound can be improved as above, IRM continues generating new rays as described in Step 3; as such, Step 3 and Step 4 are actually iteratively intertwined until one of the new rays triggers an update of $\mathbf{x}_{\text{lb}}$ or $\mathbf{x}_{\text{ub}}$. In the worst case, Step 3 uses discretization refining to construct rays with increasingly larger coefficients. This can eventually make the new rays pass arbitrarily close to the segment joining $\mathbf{x}_{\text{lb}}$ and $\mathbf{x}_{\text{ub}}$, which is guaranteed to eventually trigger some bound update. Theorem 2 in Section 2.4 (Section 2.4.3) proves that IRM is finitely convergent.

Algorithm 1 above provides the general IRM pseudocode. The outer `repeat-until` loop (Lines 5-26) performs a *major IRM iteration*, intertwining the ray generation routines (Step 3 above, roughly corresponding to Lines 10-15) and the mechanism for updating $\mathbf{x}_{\text{lb}}$ or $\mathbf{x}_{\text{ub}}$ (Step 4 above, implemented in Lines 17-25). The inner `repeat-until` loop (Lines 9-19) performs a minor iteration, in which new rays are iteratively generated until one of the bounds can be updated.

Except `intersect-subprob` (see Section 2.2), all other routines are generic with respect to (2.1):

`optimize`$(\mathcal{P}_\mathcal{A}, \mathbf{b})$ stands for any LP algorithm that can return an optimal vertex (or extreme unbounded ray) $\mathbf{x}_{\text{ub}}$ that maximizes $\mathbf{b}^\top\mathbf{x}$ over $\mathbf{x} \in \mathcal{P}_\mathcal{A}$.

`nextRay` $(\mathbf{r}, \mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}, \mathbf{r}_{\mathbf{prv}})$ returns the next integer ray after $\mathbf{r}_{\text{prv}}$ in a sequence of new rays constructed from $\mathbf{r}$, $\mathbf{x}_{\text{lb}}$ and $\mathbf{x}_{\text{ub}}$. This construction takes place when we call `nextRay` with $\mathbf{r}_{\text{prv}} = \mathbf{0}_n$ (see Section 2.3).

2.2 The Intersection (IRM) Sub-problem

2.2.1 Formal Definition

Definition 1. *The intersection (sub)-problem is formally described as follows. Given given ray* $\mathbf{r} \in \mathbb{Z}^n$ *and a (possibly open) polyhedron* $\mathcal{P} \in \mathbb{R}^n$*, determine:*

Algorithm 1 Integer Ray Method for optimizing $\mathbf{b}^\top \mathbf{x}$ over large-scale polytope $\mathcal{P}$ in (2.1)

1: $\mathcal{A} \leftarrow \emptyset$ ▷ one can start with some initial constraints (*e.g.*, bounds on the variables)
2: $\mathbf{r} \leftarrow \mathbf{b}$ ▷ scale $\mathbf{b}$ down if all b_i are too large or fractional
3: $t^*, \mathbf{a}, c_a \leftarrow$`intersect-subprob`$(\mathbf{r})$
4: $\mathcal{A} \leftarrow \mathcal{A} \cup \{[c_a, \mathbf{a}]\}$
5: **repeat**
6: $\quad \mathbf{x}_{\text{lb}} \leftarrow t^* \mathbf{r}$ ▷ if $t^* = \infty$, return ∞
7: $\quad \mathbf{x}_{\text{ub}} \leftarrow$`optimize`$(\mathcal{P}_A, \mathbf{b})$ ▷ optimal solution (or extremal ray) of $\mathcal{P}_\mathcal{A}$ ($\mathcal{P}$ restricted to $\mathcal{A}$)
8: $\quad \mathbf{r}_{\text{prv}} \leftarrow \mathbf{0}_n$ ▷ $\mathbf{r}_{\text{prv}}$ stands for the previous new ray ($\mathbf{0}_n$ means none yet)
9: $\quad$ **repeat**
10: $\quad\quad \mathbf{r}_{\text{new}} \leftarrow$`nextRay`$(\mathbf{r}, \mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}, \mathbf{r}_{\text{prv}})$ ▷ return $\mathbf{0}_n$ if no more rays can be found after $\mathbf{r}_{\text{prv}}$
11: $\quad\quad$ **while** $(\mathbf{r}_{\text{new}} = \mathbf{0}_n)$ ▷ while no more rays:
12: $\quad\quad\quad \mathbf{r} \leftarrow \lambda \mathbf{r}$, $t^* \leftarrow \frac{t^*}{\lambda}$ ▷ i) discretization refining
13: $\quad\quad\quad \mathbf{r}_{\text{new}} \leftarrow$`nextRay` $(\mathbf{r}, \mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}, \mathbf{0}_n)$ ▷ ii) re-try ray generation
14: $\quad\quad$ **end while**
15: $\quad\quad \mathbf{r}_{\text{prv}} \leftarrow \mathbf{r}_{\text{new}}$
16: $\quad\quad t^*_{\text{new}}, \mathbf{a}, c_a \leftarrow$`intersect-subprob`$(\mathbf{r}_{\texttt{new}})$
17: $\quad\quad newLb \leftarrow \left(\mathbf{b}^\top (t^*_{\text{new}} \mathbf{r}_{\text{new}}) > \mathbf{b}^\top \mathbf{x}_{\text{lb}}\right)$ ▷ better lower bound if the expression is true
18: $\quad\quad newUb \leftarrow \left(\mathbf{a}^\top \mathbf{x}_{\text{ub}} > c_a\right)$ ▷ new constraint violated by current $\mathbf{x}_{\text{ub}}$
19: $\quad$ **until** ($newLb$ or $newUb$)
20: $\quad$ **if** ($newLb$)
21: $\quad\quad \mathbf{r} \leftarrow \mathbf{r}_{\text{new}},\ t^* \leftarrow t^*_{\text{new}}$ ▷ if $\mathbf{r}_{\text{new}} \gg \mathbf{r}$, scale $\mathbf{r}_{\text{new}}$ down to avoid implicit discretization refining
22: $\quad$ **end if**
23: $\quad$ **if** ($newUb$)
24: $\quad\quad \mathcal{A} \leftarrow \mathcal{A} \cup \{[c_a, \mathbf{a}]\}$ ▷ $\mathbf{x}_{\text{ub}}$ will be updated at the next loop at Line 7
25: $\quad$ **end if**
26: **until** $\left(\mathbf{b}^\top \mathbf{x}_{\text{ub}} - \mathbf{b}^\top \mathbf{x}_{\text{lb}} \leq \epsilon\right)$ ▷ Theorem 2 (§2.4.3) shows the convergence is finite for any $\epsilon > 0$

1. *the maximum step length*
$$t^* = \max \{t \in \mathbb{R}_+ \cup \{\infty\} :\ t\mathbf{r} \in \mathcal{P}\}.$$
A returned value of $t^* = \infty$ indicates a exceptional case: $\mathbf{r}$ is an unbounded ray in polyhedron $\mathcal{P}$, i.e., $t\mathbf{r} \in \mathcal{P}$ for any indefinitely large $t > 0$;

2. *any element $[c_a, \mathbf{a}]$ of $\overline{\mathcal{A}}$ associated to a "first-hit constraint", i.e., to a ($t^*\mathbf{r}$-tight) constraint $\mathbf{a}^\top \mathbf{x} \leq c_a$ verified with equality by $t^*\mathbf{r}$ (such that $\mathbf{a}^\top (t^*\mathbf{r}) = c_a$). If $t^* = \infty$, the returned constraint is technically $[0, \mathbf{0}_n]$.*

This definition assumes $\mathbf{0}_n \in \mathcal{P}$, but the case $\mathbf{0}_n \notin \mathcal{P}$ could also addressed by adding one more possible exceptional situation: if $\mathbf{0}_n \to \mathbf{r}$ does not even "touch" $\mathcal{P}$, the returned t^* could be $-\infty$.

2.2.2 Solutions methods and comparisons with the separation sub-problem

If $\mathcal{P}$ is a dual polytope in CG models, the intersection sub-problem can often be solved by Dynamic Programming, as described in greater detail in Section 3.2. In the remaining, we rather focus on the case of a primal polytope $\mathcal{P}$: we present two techniques that relate the intersection sub-problem to the more widespread separation sub-problem.

First, the intersection sub-problem could be solved by repeatedly calling a *separation oracle* as follows. One can start with a sufficiently large t_0 and solve the separation sub-problem on $t_0\mathbf{r}$ to obtain a $\mathcal{P}$ constraint that intersects $\mathbf{0}_n \to \mathbf{r}$ at some point $t_1\mathbf{r}$. Next, one applies the separation oracle on $t_1\mathbf{r}$ and obtains a point $t_2\mathbf{r}$ such that $t_2 < t_1$. By repeating this operation, one constructs an iterative sequence $t_0, t_1, t_2 \ldots$ that

finishes when the separation oracle concludes that $t_i\mathbf{r} \in \mathcal{P}$ for some $i \in \mathbb{Z}_+$. This approach can actually be applied in both CG and cutting-planes models; the idea is taken from [2, §3].

A second separation technique that can be generalized to intersection sub-problems relies on the idea of polarity, as described in [16, (7)-(9)]; consider the LP:

$$p^* = \max\{\mathbf{r}^\top \mathbf{a} : \ \mathbf{x}_v^\top \mathbf{a} \leq 1, \ \forall \mathbf{x}_v \in \mathcal{P}_V\},$$

where $\mathcal{P}_V$ is the set of vertices of $\mathcal{P}$. Observe that any feasible solution $\mathbf{a}$ of above LP corresponds to a constraint $\mathbf{a}^\top \mathbf{x} \leq 1$ in variables $\mathbf{x}$ verified by all $\mathbf{x} \in \mathcal{P}$. If a practical algorithm can optimize this LP, one could use the optimal solution $\mathbf{a}_{\max}$ to determine whether $\mathbf{r} \in \mathcal{P}$ or $\mathbf{r} \notin \mathcal{P}$. Indeed, if $p^* = \mathbf{r}^\top \mathbf{a}_{\max}$ is larger than 1, then $\mathbf{a}_{\max}^\top \mathbf{x} \leq 1$ is a valid $\mathcal{P}$ constraint that does separate $\mathbf{r}$. The same algorithm would allow one to solve the intersection sub-problem on input ray $\mathbf{r}$ by simply returning: (1) a maximum step length $t^* = \frac{1}{p^*}$, and (2) the first-hit constraint $[1, \mathbf{a}_{\max}]$. Indeed, these values do solve the intersection sub-problem because: (i) for any larger $t > \frac{1}{p^*}$, $t\mathbf{r}$ would violate by $\mathbf{a}_{\max}^\top \mathbf{x} \leq 1$, based on $\mathbf{a}_{\max}^\top (t\mathbf{r}) > \mathbf{a}_{\max}^\top \left(\frac{1}{p^*}\mathbf{r}\right) = 1$; and (ii) for any $t \leq \frac{1}{p^*}$, $t\mathbf{r} \in \mathcal{P}$. The last statement comes from the following. Consider any $\mathcal{P}$ constraint written in the form $\mathbf{a}^\top \mathbf{x} \leq 1$. The maximality of p^* ensures that $\mathbf{a}$ needs to verify $\mathbf{r}^\top \mathbf{a} \leq \mathbf{r}^\top \mathbf{a}_{\max} = p^*$, which is equivalent to $\mathbf{a}^\top \left(\frac{1}{p^*}\mathbf{r}\right) \leq 1$, or $\mathbf{a}^\top (t\mathbf{r}) \leq 1, \forall t \leq \frac{1}{p^*}$.

2.3 Generating the Next Integer Ray

The ray generation relies on a list *rLst* of new ray proposals that are provided *one by one* to Algorithm 1 at each call of `nextRay(`$\mathbf{r}, \mathbf{x}_{\rm lb}, \mathbf{x}_{\rm ub}, \mathbf{r}_{\rm prv}$`)`. The input of this routine consists of the current ray $\mathbf{r}$, the current bounds $\mathbf{x}_{\rm lb}$ and $\mathbf{x}_{\rm ub}$, as well as the last ray $\mathbf{r}_{\rm prv}$ returned by the previous `nextRay` call ($\mathbf{r}_{\rm prv} = \mathbf{0}_n$ if none). Each `nextRay` call returns the new ray situated right after $\mathbf{r}_{\rm prv}$ in *rLst*, or the first new ray in *rLst* if $\mathbf{r}_{\rm prv} = \mathbf{0}_n$. After trying all rays in *rLst*, if none of them leads to updating $\mathbf{x}_{\rm lb}$ or $\mathbf{x}_{\rm ub}$ (*i.e.*, *newLb* and *newUb* remain `false` in Lines 17-18), then `nextRay(`$\mathbf{r}, \mathbf{x}_{\rm lb}, \mathbf{x}_{\rm ub}, \mathbf{r}_{\rm prv}$`)` eventually returns $\mathbf{0}_n$. The condition in Line 11 thus leads to *discretization refining* (Section 2.3.3), *i.e.*, IRM generates next rays with larger coefficients, to try to update $\mathbf{x}_{\rm lb}$ or $\mathbf{x}_{\rm ub}$.

We distinguish a standard case and two exceptional cases:

proper bounds (the standard case): $\mathbf{x}_{\rm lb}$ and $\mathbf{x}_{\rm ub}$ are proper non-null solutions (Section 2.3.1);
unbounded upper bound (exceptional case 1): $\mathbf{x}_{\rm ub}$ is actually an unbounded ray in the (open) polyhedron $\mathcal{P}_A$ constructed by IRM up to the current iteration (Section 2.3.2.1);
null lower bound (exceptional case 2): all rays $\mathbf{r}$ tried so far have led to intersection point $\mathbf{x}_{\rm lb} = t^*\mathbf{r} = \mathbf{0}_n$ (Section 2.3.2.2).

These three cases cover all possible situations: if the exceptional cases are excluded, then the lower bound is non-null and the upper bound can not be unbounded, *i.e.*, we are in the standard case with proper bounds. The two exceptional cases will always be treated separately in the subsequent proofs and descriptions.

2.3.1 The standard case: $\mathbf{x}_{\rm lb}, \mathbf{x}_{\rm ub}$ are proper non-null solutions

This section is only devoted to non-exceptional cases: (i) $t^* \neq 0$ and $\mathbf{x}_{\rm lb} \neq \mathbf{0}_n$, and (ii) $\mathbf{x}_{\rm ub}$ is a proper solution and not an unbounded ray. The essential task is the construction of the sequence *rLst* and we describe it by picking out the key points in boldface.

Rationale To locate promising new rays, let us focus on the segment $[\mathbf{x}_{\rm lb}, \mathbf{x}_{\rm ub}] = \{\mathbf{x}^\alpha = \mathbf{x}_{\rm lb} + \alpha\Delta : \ \alpha \in [0, 1]\}$ (with $\Delta = \mathbf{x}_{\rm ub} - \mathbf{x}_{\rm lb}$) that joins $\mathbf{x}_{\rm lb}$ and $\mathbf{x}_{\rm ub}$. Also, consider a projection of $[\mathbf{x}_{\rm lb}, \mathbf{x}_{\rm ub}]$ onto the *t^*-scaled segment* $\{\mathbf{r}^\beta = \mathbf{r} + \beta\Delta : \ \beta \in [0, \beta_{\max}]\}$, where $\beta_{\max} = \frac{1}{t^*}$, see Figure 2 (p. 9) for a graphical representation; recall that $t^* = 0$ is treated separately. If this later segment contains an *integer* solution $\mathbf{r}^\beta$, a new ray $\mathbf{r}^\beta$ can surely lead to some update of $\mathbf{x}_{\rm lb}$ or $\mathbf{x}_{\rm ub}$. Indeed, such an integer new ray $\mathbf{r}^\beta = \mathbf{r} + \beta\Delta$ intersects the segment $[\mathbf{x}_{\rm lb}, \mathbf{x}_{\rm ub}]$ at some point $\mathbf{x}^\alpha = t^*(\mathbf{r} + \beta\Delta) = \mathbf{x}_{\rm lb} + t^*\beta\Delta$. By solving the intersection sub-problem on $\mathbf{r}^\beta$, one implicitly solves the intersection sub-problem on $\mathbf{x}^\alpha$. This also determines the feasibility status of $\mathbf{x}^\alpha$: (i)

if $\mathbf{x}^\alpha \in \mathcal{P}$, the lower bound can be improved because a call `intersect-subprob`$(\mathbf{r}^\beta)$ would lead (see Lines 16-17) to a feasible solution of higher quality than $\mathbf{x}_{\text{lb}}$; (ii) if $\mathbf{x}^\alpha \notin \mathcal{P}$, then the $\mathbf{x}_{\text{ub}}$ can be updated, because both $\mathbf{x}^\alpha$ and $\mathbf{x}_{\text{ub}}$ can be separated from $\mathcal{P}$. We will formally prove this in Section 2.4 (Proposition 2), but for the moment, it is enough to say that, intuitively, integer solutions closer to some $\mathbf{r} + \beta\Delta$ have higher chances of improving the optimality gap.

Locating the closest integer points to $\mathbf{r}^\beta = \mathbf{r} + \beta\Delta$ Given a fixed β_ℓ, the closest integer point to $\mathbf{r} + \beta_\ell\Delta$ is $\lfloor \mathbf{r}^{\beta_\ell} + \frac{1}{2}\mathbf{1}_n \rfloor$. To advance from a fixed β_ℓ to the next one, one starts with $\beta = \beta_\ell$ and increases β until $\lfloor \mathbf{r}^{\beta} + \frac{1}{2}\mathbf{1}_n \rfloor \neq \lfloor \mathbf{r}^{\beta_\ell} + \frac{1}{2}\mathbf{1}_n \rfloor$, *i.e.*, until the following holds $r_i^\beta + \frac{1}{2} = \lfloor r_i^{\beta_\ell} + \frac{1}{2} \rfloor \pm 1$ for some coordinate $i \in [1..n]$, where $\pm$ depends on the sign of Δ_i (use "+" if $\Delta_i > 0$ or "−" if $\Delta_i < 0$). Using this "± convention", we reformulate the last relation as: $\left(r_i^{\beta_\ell} + \frac{1}{2}\right) + \Delta_i(\beta - \beta_\ell) = \lfloor r_i^{\beta_\ell} + \frac{1}{2} \rfloor \pm 1$, or $\Delta_i(\beta - \beta_\ell) = \lfloor r_i^{\beta_\ell} + \frac{1}{2} \rfloor - \left(r_i^{\beta_\ell} + \frac{1}{2}\right) \pm 1$. As such, one could recursively generate a sequence $\beta_1, \beta_2, \beta_3, \ldots$ via $\beta_{\ell+1} = \beta_\ell + \min_{i \in [1..n]} \frac{\lfloor r_i^{\beta_\ell} + \frac{1}{2} \rfloor - \left(r_i^{\beta_\ell} + \frac{1}{2}\right) \pm 1}{\Delta_i}$. We now observe that $\mathbf{r}^{\beta_\ell}$ and $\mathbf{r}^{\beta_{\ell+1}}$ might only differ on a single coordinate i, *i.e.*, if only one $i \in [1..n]$ minimizes the above ratio. It can be more practical to construct a sequence with fewer rays, but in which consecutive rays differ on more coordinates. For this, instead of choosing a different coordinate i to determine each $\beta_{\ell+1}$, we propose fixing $i = \max\{|\Delta_j| : \ j \in [1..n]\}$ and use the same i at each step ℓ using the following formula:

$$\beta_{\ell+1} = \beta_\ell + \frac{\lfloor r_i^{\beta_\ell} + \frac{1}{2} \rfloor - \left(r_i^{\beta_\ell} + \frac{1}{2}\right) \pm 1}{\Delta_i}, \tag{2.2}$$

where "±" should be read using the above "± convention" (it is "+" if $\Delta_i > 0$, or "−" if $\Delta_i < 0$).

Algorithm 2 Construction of the new ray sequence $rLst$. The figure shows an illustration of the process: $\boxed{1}$, $\boxed{2}$, $\boxed{3}$, $\boxed{4}$ indicate the new rays and their final order in $rLst$. The smaller empty squares are instances of $\mathbf{r}^\beta$ and the arrows indicate the rounding operation $\lfloor \mathbf{r}^\beta + \frac{1}{2}\mathbf{1}_n \rfloor$ from Line 11.

Require: $\mathbf{r},\ \mathbf{x}_{\text{lb}} = t^*\mathbf{r},\ \mathbf{x}_{\text{ub}} = \mathbf{x}_{\text{lb}} + \Delta$
Ensure: Sequence (list) $rLst$ of integer rays
1: $\beta_{\max} = \frac{1}{t^*}$ ▷ $t^* = 0$ is treated separately below
2: $rLst \leftarrow \emptyset$, $\beta \leftarrow 0$, $\mathbf{r}^\beta \leftarrow \mathbf{r}$
3: $i \leftarrow \arg\max_{j \in [1..n]} |\Delta_j|$ ▷ max absolute value (random tie breaking)
4: **repeat**
5: **if** $\lfloor \mathbf{r}^\beta + \frac{1}{2} \rfloor \neq \mathbf{r}^\beta + \frac{1}{2}$
6: $\beta \leftarrow \beta + \frac{\lfloor r_i^\beta + \frac{1}{2} \rfloor - \left(r_i^\beta + \frac{1}{2}\right) \pm 1}{\Delta_i}$ ▷ recall the "± convention": ▷ "$\pm = +$" if $\Delta_i > 0$ and "$\pm = -$" if $\Delta_i < 0$
7: **else**
8: $\beta \leftarrow \beta + \frac{1}{|\Delta_i|}$
9: **end if**
10: $\mathbf{r}^\beta \leftarrow \mathbf{r} + \beta\Delta$
11: `append`$\left(rLst, \lfloor \mathbf{r}^\beta + \frac{1}{2}\mathbf{1}_n \rfloor\right)$ ▷ add new ray $\mathbf{r}_{\text{new}} = \lfloor \mathbf{r}^\beta + \frac{1}{2}\mathbf{1}_n \rfloor$
12: **until** $\beta > \beta_{\max}$
13: $rLst \leftarrow$ `lastToFront`$(rLst)$ ▷ better to try the last ray first

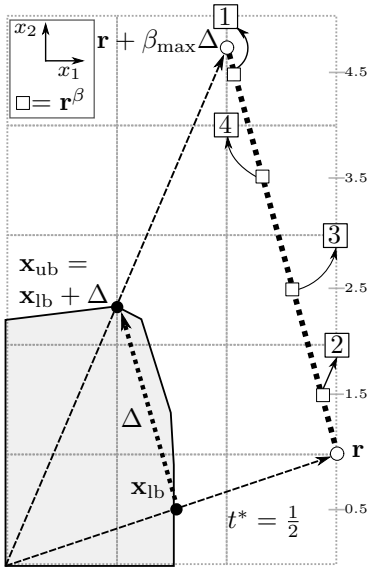


Figure 2: Small running example: Alg. 2 choses $i = 2$ in Line 3 and all $\mathbf{r}^\beta$ computed in Line 10 yield $\mathbf{r}_i^\beta \in \mathbb{Z} + \frac{1}{2}$, see the empty squares.

Algorithmic Construction of the New Ray Sequence Algorithm 2 completely describes the construction of the $rLst$ sequence. Above update formula (2.2) is employed in Line 6 with the β indices removed (any assigned (left) β stands for $\beta_{\ell+1}$, all others for β_ℓ); this is further simplified to $\beta \leftarrow \beta + \frac{1}{|\Delta_i|}$ when all involved values are integer (see Line 8). This $rLst$ sequence is construted after each update of $\mathbf{x}_{\text{lb}}$ or $\mathbf{x}_{\text{ub}}$ when IRM needs to generate new rays, *i.e.*, when the IRM calls `nextRay`$(\mathbf{r}, \mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}, \mathbf{0}_n)$ with a last argument

of $\mathbf{r}_{\text{prv}} = \mathbf{0}_n$ (see Line 8, Algorithm 1). In fact, $\mathbf{r}_{\text{prv}} = \mathbf{0}_n$ indicates that IRM requires the *very first* new ray for current $\mathbf{r}, \mathbf{x}_{\text{lb}}$ and $\mathbf{x}_{\text{ub}}$, triggering the execution of Algorithm 2. Afterwards, `nextRay` returns (one by one) the elements from $rLst$, *i.e.*, observe the assignment $\mathbf{r}_{\text{prv}} \leftarrow \mathbf{r}_{\text{new}}$ (Line 15). When all elements from $rLst$ are finished, `nextRay` returns $\mathbf{0}_n$.

Sequence Reshuffling The order of rays in $rLst$ has a strong influence on the general IRM evolution. A very practical enhancement of the initial order simply consists of moving the last element to the front (last line of Algorithm 2). The last discovered ray (close to the direction $\mathbf{0}_n \rightarrow \mathbf{x}_{\text{ub}}$) can be more useful for updating the upper bound (see Theorem 2 in Section 2.4.3 for more formal arguments), while the first rays are rather useful for updating $\mathbf{x}_{\text{ub}}$. The upper bounds are somehow more critical, because they can be *anywhere* in the search space. Reasonable lower bounds can be more easily located by the IRM from the very first iteration, see also point (3) of the general experimental conclusions (Section 5.3).

2.3.2 Exceptional cases: unbounded ray $\mathbf{x}_{\text{ub}}$ or null $\mathbf{x}_{\text{lb}} = \mathbf{0}_n$

2.3.2.1 Case "$\mathbf{x}_{\text{ub}}$ is an unbounded ray" First, observe that $\mathbf{b}^\top \mathbf{x}_{\text{ub}} > 0$, because no LP solver would return an extreme ray $\mathbf{0}_n \rightarrow \mathbf{x}_{\text{ub}}$ such that $\mathbf{b}^\top \mathbf{x}_{\text{ub}} \leq 0$ (*i.e.*, no better than $\mathbf{0}_n$). To build $rLst$, we first consider the sequence $\mathbf{r}^1_{\text{new}}$, $\mathbf{r}^2_{\text{new}}$, $\mathbf{r}^3_{\text{new}}$, ... with $\mathbf{r}^i_{\text{new}} = \lfloor i\mathbf{x}_{\text{ub}} \rfloor$ $(\forall i \in \mathbb{Z}_+)$. From this sequence, one needs to pay attention to keep in $rLst$ only the elements $\mathbf{r}^i_{\text{new}}$ such that $\mathbf{0}_n \rightarrow \mathbf{r}^i_{\text{new}}$ is a strictly positive direction (*i.e.*, $\mathbf{b}^\top \mathbf{r}^i_{\text{new}} > 0$). Proposition 1 shows that the sequence of rays inserted this way in $rLst$ is actually finite in IRM, simply because there is some sufficiently large i for which the integer ray direction $\mathbf{0}_n \rightarrow \mathbf{r}^i_{\text{new}}$ identifies with $\mathbf{0}_n \rightarrow \mathbf{x}_{\text{ub}}$. As such, the generation of the above sequence always terminates in one of the following:

(a) $\mathbf{0}_n \rightarrow \mathbf{r}^i_{\text{new}}$ is an unbounded (strictly positive) ray: the intersection sub-problem returns $t^* = \infty$ and IRM concludes that the optimum objective value is unbounded;

(b) the intersection sub-problem on $\mathbf{r}^i_{\text{new}}$ returns a constraint that does separate a part of the points in $\mathbf{0}_n \rightarrow \mathbf{x}_{\text{ub}}$. In this case, IRM ends up by updating $\mathbf{x}_{\text{ub}}$ and goes on to the next major iteration (Lines 5-26).

Proposition 1. *If $c_a \in \mathbb{Q}$ and $\mathbf{a} \in \mathbb{Q}^n$ for all $[c_a, \mathbf{a}] \in \overline{\mathcal{A}}$, there exists a sufficiently large integer i such that $\mathbf{0}_n \rightarrow \mathbf{r}^i_{new}$ and $\mathbf{0}_n \rightarrow \mathbf{x}_{\text{ub}}$ identify the same ray direction, where $\mathbf{r}^i_{new} = \lfloor i\mathbf{x}_{\text{ub}} \rfloor$. More technically, one can formalize a general notation $\mathbf{0}_n \rightarrow \mathbf{r} = \{t\mathbf{r} : t \geq 0\}$ and write the same conclusion as $\mathbf{0}_n \rightarrow \mathbf{r}^i_{new} = \mathbf{0}_n \rightarrow \mathbf{x}_{\text{ub}}$. If some $[c_a, \mathbf{a}] \in \overline{\mathcal{A}}$ contain non-rational data, IRM might not be finitely convergent due to this exceptional case (potential extremal rays $\mathbf{0}_n \rightarrow \mathbf{x}_{\text{ub}}$ with non-rational data).*

Proof. The existence of some $i \in \mathbb{Z}_+$ such that $\mathbf{0}_n \rightarrow \lfloor i\mathbf{x}_{\text{ub}} \rfloor = \mathbf{0}_n \rightarrow \mathbf{x}_{\text{ub}}$ is a direct consequence of the fact that the constraints of $\mathcal{P}$ are defined on rational data only, as mentioned in the description of (1.1). This i is simply the least common denominator of the n (rational) components of $\mathbf{x}_{\text{ub}}$.
For the second statement, it is enough to provide an example: IRM is not finitely convergent on $\mathcal{P} = \{\sqrt{2}x_1 - x_2 \leq 0,\ -\sqrt{2}x_1 + x_2 \leq 0\}$, because the unbounded ray $\mathbf{0}_n \rightarrow [1,\ \sqrt{2}]$ would lead to an infinite $rLst$ sequence built as above. This is the only proof in this paper that takes into account the fact that the constraint of $\mathcal{P}$ are defined on rational data. □

Finally, observe that this exceptional case can only arise during the first major IRM iterations (Lines 5-26 of Algorithm 1), as long as the optimum of $\mathcal{P}_{\mathcal{A}}$ is not a proper vertex but an unbounded (extreme) ray. As soon as the constraint set $\mathcal{A}$ becomes large enough to make $\mathbf{x}_{\text{ub}} = \text{OPT}(\mathcal{P}_{\mathcal{A}})$ become a proper solution, unbounded rays can no longer arise. This simply comes from the fact that the constraint set $\mathcal{A}$ can only increase; once the upper bound becomes bounded, it stays bounded indefinitely.

2.3.2.2 Case "null lower bound $\mathbf{x}_{\text{lb}} = \mathbf{0}_n$ and $t^* = 0$" We further consider that $\mathbf{x}_{\text{ub}}$ is not an extremal ray of $\mathcal{P}_{\mathcal{A}}$, but a proper solution (as long as this does not happen, one can repeat the routine from Section 2.3.2.1 until $\mathbf{x}_{\text{ub}}$ becomes a proper solution). We observe that the intersection sub-problem returns $t^* = 0$ only if the current main ray $\mathbf{r}$ verifies $t\mathbf{r} \neq \mathcal{P},\ \forall t > 0$ (see Definition 1). This can only happen if there

exists some $\mathbf{r}$-*blocking constraint* of the form $\mathbf{a}^\top\mathbf{x} \leq 0$ such that $\mathbf{a}^\top\mathbf{r} > 0$. One can say that such constraint "blocks" any advance on direction $\mathbf{0}_n \to \mathbf{r}$; such a ray $\mathbf{r}$ is called a *blocked ray*.

IRM handles this exceptional case as the one from Section 2.3.2.1. We first consider a sequence of rays $\mathbf{r}^1_{\text{new}}$, $\mathbf{r}^2_{\text{new}}$, $\mathbf{r}^3_{\text{new}}$, ... with $\mathbf{r}^i_{\text{new}} = \lfloor i\mathbf{x}_{\text{ub}} \rfloor$ $(\forall i \in \mathbb{Z}_+)$ and we insert in $rLst$ only strictly positive rays $\mathbf{0}_n \to \mathbf{r}^i_{\text{new}}$. The intersection sub-problem is solved iteratively on these rays until one of the following holds:

(i) The intersection sub-problem returns some $t^* > 0$. This would allow IRM to find some lower bound $\mathbf{x}_{\text{lb}} \neq \mathbf{0}_n$ and go on to the next major iteration;

(ii) The intersection sub-problem returns $t^* = 0$ and finds a first-hit constraint $\mathbf{a}^\top\mathbf{x} \leq 0$ that is violated by $\mathbf{x}_{\text{ub}}$.

Using the same Proposition 1 as in Section 2.3.2.1, we observe that the sequence $\mathbf{r}^1_{\text{new}}$, $\mathbf{r}^2_{\text{new}}$, $\mathbf{r}^3_{\text{new}}$, ... is finite. More exactly, there needs to be some index i for which $\mathbf{r}^i_{\text{new}} = \alpha_i\mathbf{x}_{\text{ub}}$ for some $\alpha_i > 0$. In such a case, the intersection sub-problem can either: (a) return $t^* > 0$, leading to point (i) above, or (b) find a first-hit constraint $[c_a, \mathbf{a}] \in \mathcal{A}$ such that $\mathbf{a}^\top\mathbf{r}^i_{\text{new}} > 0$ ($\mathbf{r}^i_{new}$-*blocking* constraint). Such a constraint does need to separate $\mathbf{x}_{\text{ub}}$, because $\mathbf{a}^\top(\mathbf{r}^i_{\text{new}}) > 0 \implies \mathbf{a}^\top(\alpha_i\mathbf{x}_{\text{ub}}) > 0 \overset{\alpha_i>0}{\implies} \mathbf{a}^\top\mathbf{x}_{\text{ub}} > 0$, leading to point (ii) above.

IRM can not continually repeat this exceptional case $\mathbf{x}_{\text{ub}} = \mathbf{0}_n$ and indefinitely only end up in above point (ii), because $\overline{\mathcal{A}}$ is finite (see also Theorem 2 in Section 2.4.3). Sooner or later, one of the following happens: either IRM concludes that $\mathbf{0}_n$ is the (2.1) optimum, or IRM finds some $\mathbf{x}_{\text{lb}} = t^*\mathbf{r}$ with $t^* > 0$ and $\mathbf{x}_{\text{lb}} \neq \mathbf{0}_n$, *i.e.*, point (i) above. Since such a positive lower bound can never be later replaced with some $\mathbf{x}_{\text{lb}} = t^*\mathbf{r} = \mathbf{0}_n$, the exceptional case can be considered definitely solved, *i.e.*, $\mathbf{x}_{\text{lb}}$ can never become $\mathbf{0}_n$ again.

2.3.3 Discretization Refining

We now address the following (unfortunate) situation: there is *no* ray in the list $rLst$ (as constructed by Algorithm 2, p. 9) that can generate any lower or upper bound update. In other words, there is *no* ray $\mathbf{r}_{\text{new}}$ in $rLst$ that leads the IRM sub-problem to an intersection point $t^*_{\text{new}}\mathbf{r}_{\text{new}}$ such that either (i) $\mathbf{b}^\top(t^*_{\text{new}}\mathbf{r}_{\text{new}}) > \mathbf{b}^\top\mathbf{x}_{\text{lb}}$ (lower bound update), or (ii) $t^*_{\text{new}}\mathbf{r}_{\text{new}}$ belongs to an as-yet-undiscovered facet of $\mathcal{P}$ that cuts $\mathbf{x}_{\text{ub}}$ off (upper bound update). This can happen because the ray coefficients need to be integer and this can make all potential ray directions too coarse—intuitively, imagine the "coarseness" of most rays if the polytope is completely included in the unit hypercube. Technically, a *necessary* condition to reach this situation is the following: $rLst$ contains no integer ray $\mathbf{r}_{\text{new}}$ such that $\mathbf{0}_n \to \mathbf{r}_{\text{new}}$ intersects $[\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}]$. We will later prove this in Proposition 2 (Section 2.4), but for now it is enough to say the following (see the proof of the proposition). If $\mathbf{0}_n \to \mathbf{r}_{\text{new}}$ is very close to $[\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}]$, then $\mathbf{r}_{\text{new}}$ has many chances to improve one of the bounds. If $\mathbf{0}_n \to \mathbf{r}_{\text{new}}$ is too distant from $[\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}]$, $\mathbf{r}_{\text{new}}$ has less chances to update either bound.

To deal with such issues, IRM uses a technique of discretization refining that leads to new larger rays, producing more fined-grained directions. In fact, an implicit form of discretization refining was already introduced in Section 2.3.2, where we used a sequence of increasingly larger rays to address the two exceptional cases. However, these cases are completely described in Section 2.3.2 and we can now only discuss the non-exceptional cases ($\mathbf{x}_{\text{lb}}$ and $\mathbf{x}_{\text{ub}}$ are proper solutions).

The technical conditions that make Algorithm 1 trigger discretization refining are the following. If new ray $\mathbf{r}_{\text{new}}$ fails in improving either bound, $newLb$ and $newUb$ remain `false` in Lines 17-18, and so, the inner `repeat-until` loop (Lines 9-19) continues trying more rays by calling `nextRay`. After finishing all rays in $rLst$, `nextRay` returns $\mathbf{0}_n$; this makes Algorithm 1 activate the condition $\mathbf{r} = \mathbf{0}_n$ in Line 11, triggering the *discretization refining* in Line 12.

The refining operation itself is rather straightforward: multiply $\mathbf{r}$ by $\lambda \in \mathbb{Z}_+$, multiply t^* by $\frac{1}{\lambda}$; observe $\mathbf{x}_{\text{lb}} = t^*\mathbf{r}$ remains constant. The discretization step parameter λ only needs to be integer (we used $\lambda = 2$), so as to keep $\mathbf{r}$ integer. The use of larger ray coefficients allows the next calls of the ray generator to construct directions $\mathbf{0}_n \to \mathbf{r}_{\text{new}}$ that are more refined, closer to $[\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}]$ (for more formal arguments on this, see also Theorem 1, Section 2.4.2). As discussed above, when $\mathbf{0}_n \to \mathbf{r}_{\text{new}}$ is closer to $[\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}]$, $\mathbf{r}_{\text{new}}$ has more chances to unlock the process.

The disadvantage of discretization refining is that it can naturally slowdown the sub-problem algorithms. In the context of our CG models, the proposed $\mathbf{r}$-indexed Dynamic Programming schemes (see Section 3.2.1) might easily need to generate and scan more states when $\mathbf{r}$ becomes larger.

2.4 Iteratively Updating $\mathbf{x}_{\text{lb}}$ and $\mathbf{x}_{\text{ub}}$ for Converging To Limit $\texttt{OPT}(\mathcal{P})$

We here analyze how IRM drives $\mathbf{x}_{\text{lb}}$ and $\mathbf{x}_{\text{ub}}$ towards an optimal solution $\texttt{OPT}(\mathcal{P})$ of (2.1). We start out with the best-case scenario (Section 2.4.1), we continue with a formal worst-case analysis (Section 2.4.2) and we put all pieces together to prove the theoretical convergence in Theorem 2 (Section 2.4.3).

2.4.1 Best-case Scenario: Early Update of $\mathbf{x}_{\text{lb}}$ or $\mathbf{x}_{\text{ub}}$

Proposition 2. *If $\mathbf{r}^\beta = \mathbf{r} + \beta\Delta$ is integer for some $\beta \in (0, \beta_{max}]$, then Algorithm 1 can surely update either $\mathbf{x}_{\text{lb}}$ or $\mathbf{x}_{\text{ub}}$ after calling `intersect-subprob`$(\mathbf{r}_{new})$ in Line 16 with $\mathbf{r}_{new} = \mathbf{r}^\beta$.*

Proof. Defining $\mathbf{x}^\alpha = t^*\mathbf{r}^\beta$, it is clear that $\mathbf{0}_n \rightarrow \mathbf{r}^\beta$ and $\mathbf{0}_n \rightarrow \mathbf{x}^\alpha$ identify the same ray direction. One can also write $\mathbf{x}^\alpha = t^*(\mathbf{r} + \beta\Delta) = \mathbf{x}_{\text{lb}} + t^*\beta(\mathbf{x}_{\text{ub}} - \mathbf{x}_{\text{lb}})$. This shows that $\mathbf{x}^\alpha \in [\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}]$ because $t^*\beta \leq 1$ (recall form Algorithm 2 that $\beta_{\max} = \frac{1}{t^*}$). More generally, we observe that the whole segment $[\mathbf{r}, \mathbf{r} + \beta_{\max}\Delta]$ can be multiplied by t^* and scaled into segment $[t^*\mathbf{r}, t^*\mathbf{r} + \Delta] = [\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{lb}} + \Delta]$, see Figure 2 in Section 2.3.1 for an intuitive representation of this scaling. We will use this property below, but let us now focus on the IRM sub-problem that returns t^*_β on $\mathbf{r}_{\text{new}} = \mathbf{r}^\beta$.

case $t^*_\beta \geq t^*$ leads IRM directly to new lower bound solution $t^*_\beta\mathbf{r}^\beta$ that strictly dominates $\mathbf{x}_{\text{lb}}$; technically, we write $\mathbf{b}^\top \left(t^*_\beta\mathbf{r}^\beta\right) > \mathbf{b}^\top\mathbf{x}_{\text{lb}}$. We show this by proving the following.

(1) $\mathbf{b}^\top \left(t^*_\beta\mathbf{r}^\beta\right) \geq \mathbf{b}^\top \left(t^*\mathbf{r}^\beta\right)$

(2) $\mathbf{b}^\top \left(t^*\mathbf{r}^\beta\right) > \mathbf{b}^\top\mathbf{x}_{\text{lb}}$

The first inequality is equivalent to $(t^*_\beta - t^*)\mathbf{b}^\top\mathbf{r}^\beta \geq 0$, which follows from $\mathbf{b}^\top\mathbf{r}^\beta > 0$ (this is true for any $\mathbf{r}^\beta \in [\mathbf{r}, \mathbf{r} + \beta_{\max}\Delta]$, based on $\beta^\top\mathbf{r}, \beta^\top(\mathbf{r} + \beta_{\max}\Delta) > 0$). Inequality (2) is equivalent to $\mathbf{b}^\top \left(t^*\mathbf{r}^\beta\right) > \mathbf{b}^\top \left(t^*\mathbf{r}\right)$ and to $\mathbf{b}^\top\mathbf{r}^\beta > \mathbf{b}^\top\mathbf{r}$ (recall we are in a non-exceptional case and $t^* > 0$). We still need to prove $\mathbf{b}^\top \left(\mathbf{r}^\beta - \mathbf{r}\right) > 0$, which is equivalent to $\mathbf{b}^\top \left(\beta\Delta\right) > 0$, or to $\mathbf{b}^\top\Delta > 0$ (based on $\beta > 0$). To prove this latter inequality, observe that $\mathbf{b}^\top\mathbf{x}_{\text{ub}} > \mathbf{b}^\top\mathbf{x}_{\text{lb}} \implies \mathbf{b}^\top \left(t^*\mathbf{r} + \Delta\right) > \mathbf{b}^\top \left(t^*\mathbf{r}\right)$, which leads to $\mathbf{b}^\top\Delta > 0$.

Finally, Algorithm 1 triggers $newLb =$ `true` in Line 17; $\mathbf{r}$ and t^* are updated in Line 21, which can directly update $\mathbf{x}_{\text{lb}} = t^*\mathbf{r}$.

case $t^*_\beta < t^*$ leads Algorithm 1 to update $\mathbf{x}_{\text{ub}}$ because the following operations are necessarily triggered:

1. Algorithm 1 calls in Line 16 the intersection sub-problem on $\mathbf{r}_{\text{new}} = \mathbf{r}^\beta$ and this yields a constraint $\mathbf{a}^\top\mathbf{x} \leq c_a$ satisfied with equality by $t^*_\beta\mathbf{r}^\beta$, *i.e.*, $\mathbf{a}^\top(t^*_\beta\mathbf{r}^\beta) = c_a$;
2. Algorithm 1 turns $newUb$ `true` in Line 18, because $\mathbf{a}^\top\mathbf{x}_{\text{ub}} > c_a$. This inequality is true, because otherwise the constraint $\mathbf{a}^\top\mathbf{x} \leq c_a$ would be verified by both $\mathbf{x}_{\text{lb}}$ and $\mathbf{x}_{\text{ub}}$. This would be equivalent to $\mathbf{a}^\top\mathbf{x} \leq c_a,\ \forall\mathbf{x} \in [\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}]$, which is impossible because $t^*\mathbf{r}^\beta \in [\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}]$ and $t^* > t^*_\beta \implies \mathbf{a}^\top(t^*\mathbf{r}^\beta) > \mathbf{a}^\top(t^*_\beta\mathbf{r}^\beta) = c_a$;
3. Algorithm 1 adds $[c_a, \mathbf{a}]$ to A in Line 24 (since $newUb$ is `true`), leading to an update of $\mathbf{x}_{\text{ub}}$ at the next $\texttt{optimize}(\mathcal{P}_A, \mathbf{b})$ call in Line 7. This does not necessarily mean that the bound value is strictly improved, but the initial infeasible $\mathbf{x}_{\text{ub}}$ is strictly separated by the new constraint. □

While the hypothesis condition of this property only arises in very fortunate exceptional situations, the main idea can be generalized to other rays $\mathbf{r}^\beta = \mathbf{r} + \beta\Delta$ that are "almost" integer. If the IRM generates an integer ray $\mathbf{r}_{\text{new}} = \lfloor\mathbf{r}^\beta + \frac{\mathbf{1}_n}{2}\rfloor$ very close to $\mathbf{r}^\beta$, the ray direction $\mathbf{0}_n \rightarrow \mathbf{r}_{\text{new}}$ is close to the segment $[\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}]$.

The main idea of the convergence process is the following: when $\mathbf{0}_n \rightarrow \mathbf{r}_{\text{new}}$ is closer to $[\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}]$, there are more chances to update $\mathbf{x}_{\text{lb}}$ or $\mathbf{x}_{\text{ub}}$ as above. We describe next how $\mathbf{0}_n \rightarrow \mathbf{r}_{\text{new}}$ can always become sufficiently close to $[\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}]$ by applying enough discretization refining.

2.4.2 Worst-Case Scenario: Later Update of $\mathbf{x}_{\text{ub}}$

We show that the proposed ray generation routine from Section 2.3 is able to eventually lead IRM to update any $\mathbf{x}_{\text{ub}} \notin \mathcal{P}$, *i.e.*, Algorithm 1 can never get locked in the inner `repeat-until` (Lines 9-19). First, recall that the exceptional cases are treated separately in Section 2.3.2: we first described the case of unbounded rays $\mathbf{x}_{\text{ub}}$ (Section 2.3.2.1), followed by the case $t^* = 0$ (Section 2.3.2.2).

We can here focus on the standard case: $t^* \neq 0$ and $\mathbf{x}_{\text{ub}}$ is a proper vertex of a polytope $\mathcal{P}_{\mathcal{A}}$. We show that the same causal events described in the second case ($t^* > t^*_\beta$) of above Proposition 2 can actually always be triggered, eventually surely leading to an update of any infeasible $\mathbf{x}_{\text{ub}}$. While larger ray coefficients might be needed in the worst case, this can always be achieved by discretization refining in finite time.

Theorem 1. *There exists a sufficiently small $t_\epsilon > 0$ such that if $\mathbf{x}_{\text{lb}} = t^*\mathbf{r}$ with $t^* < t_\epsilon$ (i.e., if $\mathbf{r}$ is large-enough), the standard ray generator from Algorithm 2 constructs at least a new ray leading to updating $\mathbf{x}_{\text{ub}} \notin \mathcal{P}$. Formally, for any input $\mathbf{r}$, $\mathbf{x}_{\text{lb}} = t^*\mathbf{r}$ and $\mathbf{x}_{\text{ub}} \notin \mathcal{P}$ such that $t^* < t_\epsilon$, Algorithm 2 (p. 9) finds at least a ray $\mathbf{r}_{new}$ that leads the IRM sub-problem to a constraint violated by infeasible $\mathbf{x}_{\text{ub}}$.*

Proof. Consider advancing along $\mathbf{0}_n \rightarrow \mathbf{x}_{\text{ub}}$ until first hitting a constraint $\mathbf{a}^\top\mathbf{x} \leq c_a$ (the dashed line in Figure 3) at some point $\mathbf{r}_{\text{ub}} = t_{\text{ub}}\mathbf{x}_{\text{ub}}$. This "first-hit" constraint needs to be satisfied with equality by $\mathbf{r}_{\text{ub}}$ (*i.e.*, $\mathbf{a}^\top\mathbf{r}_{\text{ub}} = c_a$). Using $\mathbf{x}_{\text{ub}} \notin \mathcal{P}$ and $\mathbf{r}_{\text{ub}} \in \mathcal{P}$, it is clear that $t_{\text{ub}} < 1$, which shows that $\mathbf{a}^\top\mathbf{x}_{\text{ub}} > c_a$ (based on $\frac{c_a}{\mathbf{a}^\top\mathbf{x}_{\text{ub}}} = t_{\text{ub}} < 1$), *i.e.*, such a constraint $\mathbf{a}^\top\mathbf{x} \leq c_a$ is $\mathbf{x}_{\text{ub}}$*–separating*.

We need to show that one of the rays $\mathbf{r}_{\text{new}}$ constructed by Algorithm 2 leads the IRM sub-problem to a $\mathbf{x}_{\text{ub}}$*-separating* constraint. We will first show there exists a small-enough box $B(\mathbf{x}_{\text{ub}}, \epsilon)$ with the following ϵ*-box property*: any $\mathbf{x}_{\text{ub}}{}' \in B(\mathbf{x}_{\text{ub}}, \epsilon)$, would lead the intersection sub-problem to a $\mathbf{x}_{\text{ub}}$-separating constraint. We need a formal ϵ-box definition; given $\overline{\mathbf{x}} \in \mathbb{R}^n$ and $\epsilon > 0$, the ϵ-box centered at $\overline{\mathbf{x}}$ is:

$$\begin{aligned} B(\overline{\mathbf{x}}, \epsilon) &= \{\mathbf{x} \in \mathbb{R}^n \ : \ (1-\epsilon)\overline{\mathbf{x}} \leq \mathbf{x} \leq (1+\epsilon)\overline{\mathbf{x}}\} \\ &= \{\overline{\mathbf{x}} + \mathbf{x}_\epsilon \in \mathbb{R}^n \ : \ |\mathbf{x}_\epsilon| \leq \epsilon\overline{\mathbf{x}}\} \end{aligned}$$

We consider any $\mathbf{x}_{\text{ub}}{}' = \mathbf{x}_{\text{ub}} + \mathbf{x}_\epsilon \in B(\mathbf{x}_{\text{ub}}, \epsilon)$. Let us take any constraint satisfied with equality by $\mathbf{r}_{\text{ub}} = t_{\text{ub}}\mathbf{x}_{\text{ub}}$, *i.e.*, such that $t_{\text{ub}} = \frac{c_a}{a^\top\mathbf{x}_{\text{ub}}}$ as described above. We observe that $\frac{c_a}{\mathbf{a}^\top\mathbf{x}_{\text{ub}}{}'} = \frac{c_a}{\mathbf{a}^\top(\mathbf{x}_{\text{ub}}+\mathbf{x}_\epsilon)}$ is arbitrarily close to $t_{\text{ub}} = \frac{c_a}{\mathbf{a}^\top\mathbf{x}_{\text{ub}}}$, because $\mathbf{a}^\top\mathbf{x}_\epsilon$ can be as close to 0 as one needs; indeed, since $|\mathbf{x}_\epsilon| \leq \epsilon\mathbf{x}_{\text{ub}}$, a small-enough ϵ can make the term $\mathbf{a}^\top\mathbf{x}_\epsilon$ "vanish" in comparison to $\mathbf{a}^\top\mathbf{x}_{\text{ub}}$. Furthermore, the intersection sub-problem applied on $\mathbf{x}_{\text{ub}}{}'$ can only return a constraint $[c'_a \ \mathbf{a}']$ such that $\frac{c'_a}{\mathbf{a}'^\top\mathbf{x}_{\text{ub}}{}'} \leq \frac{c_a}{\mathbf{a}^\top\mathbf{x}_{\text{ub}}{}'}$, which is arbitrarily close to $t_{\text{ub}} < 1$. We can thus deduce $\frac{c'_a}{\mathbf{a}'^\top\mathbf{x}_{\text{ub}}{}'} < 1$, equivalent to $\mathbf{a}'^\top\mathbf{x}_{\text{ub}} + \mathbf{a}'^\top\mathbf{x}_\epsilon > c'_a$. The second term can be arbitrarily small as discussed above, which leads to $\mathbf{a}'^\top\mathbf{x}_{\text{ub}} > c'_a$.

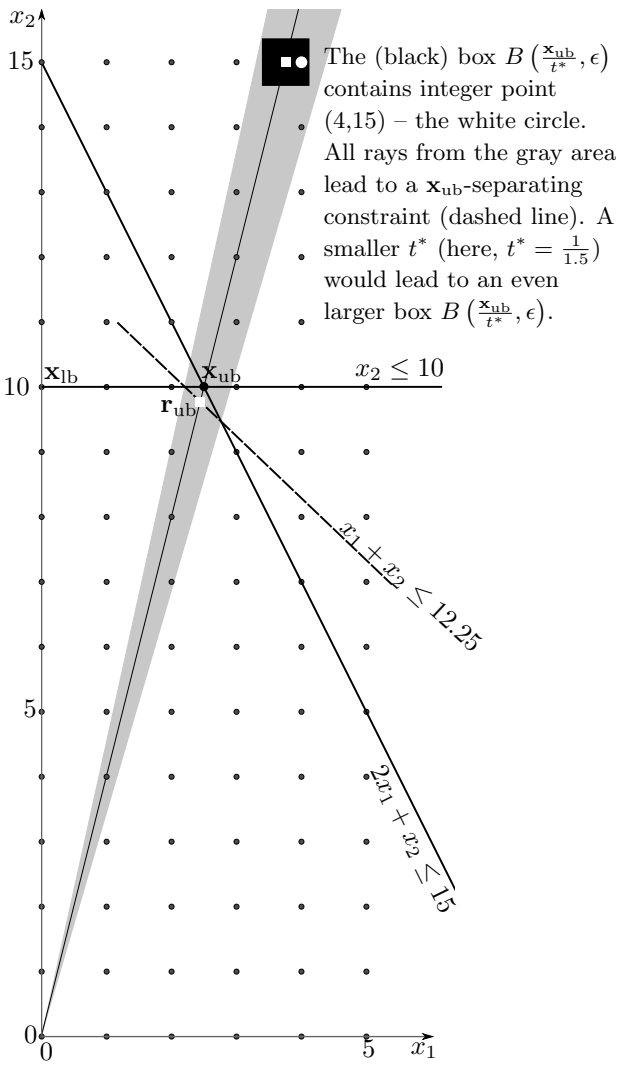


Figure 3: A modified version of Figure 1 in which the $\mathbf{x}_{\text{ub}}$-separating constraint (dashed line) is "hard to hit" after step 2, *i.e.*, using $t^* = 1$, the integer points on the segment $[\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}]$ are outside the gray area. However, a lower $t^* = \frac{1}{1.5}$ leads to integer points that do "hit" the dashed line.

Above paragraph proves the existence of a box $B(\mathbf{x}_{\text{ub}}, \epsilon)$ such that all rays (not necessarily integer) in $B(\mathbf{x}_{\text{ub}}, \epsilon)$ lead the IRM sub-problem to a $\mathbf{x}_{\text{ub}}$-separating constraint. We now show that the standard ray generation procedure from Section 2.3.1 does return *integer* rays $\mathbf{r}_{\text{new}}$ with the same property. Intuitively, all rays leading to $\mathbf{x}_{\text{ub}}$-separating constraints belong to the gray conic area in Figure 3. Formally, recall that Algorithm 2 (p. 9) generates each potential new ray $\mathbf{r}_{\text{new}}$ by rounding some solutions $\mathbf{r} + \beta\Delta$ generated by advancing on the line from $\mathbf{r}$ to $\mathbf{r} + \frac{1}{t^*}\Delta = \frac{\mathbf{x}_{\text{ub}}}{t^*}$. One can find a small enough t_ϵ such that, for any $t^* \leq t_\epsilon$, the box $B\left(\frac{\mathbf{x}_{\text{ub}}}{t^*}, \epsilon\right)$ becomes large enough to contain as many integer solutions as needed. This comes from the fact that each "edge" of this box has a length proportional to $\frac{1}{t^*}$—intuitively, imagine the size of the box in Figure 3 if t^* were 10 times smaller. A detailed investigation of Algorithm 2 shows the following: for any chosen $i \in [1..n]$ (in Line 3), Line 10 leads to some points $\mathbf{r} + \beta\Delta$ located (deep) inside the large-enough $B\left(\frac{\mathbf{x}_{\text{ub}}}{t^*}, \epsilon\right)$ and that remain inside the same box after rounding. Since the intersection problem returns the same "first-hit" constraint from $\mathbf{r}_{\text{new}} \in B\left(\frac{\mathbf{x}_{\text{ub}}}{t^*}, \epsilon\right)$ or $t^*\mathbf{r}_{\text{new}} \in B(\mathbf{x}_{\text{ub}}, \epsilon)$, this "first-hit" constraint has to be $\mathbf{x}_{\text{ub}}$-separating. □

2.4.3 Convergence of the Global Process

Theorem 2. *The Integer Ray Method, as specified in Sections 2.1-2.3, is finitely convergent.*

Proof. The exceptional cases are completely addressed in Section 2.3.2. We recall the argument. Based on Proposition 1, we proved that if $\mathbf{x}_{\text{ub}}$ is an unbounded ray or $\mathbf{x}_{\text{lb}} = \mathbf{0}_n$, then IRM constructs a finite sequence of rays such that one of them identifies with $\mathbf{0}_n \rightarrow \mathbf{x}_{\text{ub}}$. This shows that IRM constructs in finite time a $\mathbf{x}_{\text{ub}}$-separating constraint (*i.e.*, that separates $\mathbf{x}_{\text{ub}}$ from $\mathcal{P}$) if $\mathbf{x}_{\text{ub}}$ is not the optimal solution (or ray). By iteratively adding such constraints, IRM eventually ends up in one of the following: (i) stop and return an unbounded optimal value (in Section 2.3.2.1), (ii) stop and return 0 (in Section 2.3.2.2), or (iii) finish completely the exceptional cases ($\mathbf{x}_{\text{lb}}$ and $\mathbf{x}_{\text{ub}}$ become proper solutions) and continue with the standard case.

We can further focus on this standard case. Theorem 1 (p. 13) can be re-formulated as follows: if the discretization is refined enough (*i.e.*, $t^* < t_\epsilon$), the ray generation procedure (Algorithm 2) can find an integer ray that leads the IRM sub-problem to an $\mathbf{x}_{\text{ub}}$-separating constraint. This ensures that the inner `repeat-until` IRM loop (Lines 9-19, see Algorithm 1) can not be infinite, *i.e.*, after a finite number of iterations, the discretization refining (Line 12) would be executed enough times to reach $t^* < t_\epsilon$ and turn *newUb* `true` (*i.e.*, by finding a $\mathbf{x}_{\text{ub}}$-separating constraint as in Theorem 1).

However, the stopping condition of this inner `repeat-until` loop (*newLb* or *newUb*, see Line 19) can be triggered without upper bound update if *newLb* becomes `true` before *newUb*. Observe that *newLb* becomes `true` only when there is a strict improvement of $\mathbf{x}_{\text{lb}} = t^*\mathbf{r}$, leading to an update of $\mathbf{r}$ in Line 21. This strict improvement condition for updating $\mathbf{r}$ shows that the main pivot ray $\mathbf{r}$ can never take the same value twice. Considering a fixed t_ϵ, the number of integer rays $\mathbf{r}$ such that $t^*\mathbf{r} \in \mathcal{P}$ *and* $t^* \geq t_\epsilon$ is finite. This shows a state with $t^* \geq t_\epsilon$ can not hold indefinitely by only improving the lower bound, *i.e.*, by only exiting the inner `repeat-until` IRM loop with *newLb* = `true` and *newUb* = `false` in Line 19. In the worst case, IRM can always eventually reach a state with $t^* < t_\epsilon$, which can surely allow the discovery of $\mathbf{x}_{\text{ub}}$-separating constraints (based on Theorem 1). The queue reshuffling in the last line of Algorithm 2 makes the ray generator start with ray directions close to $\mathbf{0}_n \rightarrow \mathbf{x}_{\text{ub}}$; as such, the $\mathbf{x}_{\text{ub}}$-separating constraints can not be discovered after those that improve the lower bound.

This shows that, as long as $\mathbf{x}_{\text{ub}} \notin \mathcal{P}$, Algorithm 1 can surely find an $\mathbf{x}_{\text{ub}}$-separating constraint in finite time. Since the number of $\mathcal{P}$ constraints is finite, Algorithm 1 can generate in finite time all $\mathcal{P}$ constraints required for making $\mathbf{x}_{\text{ub}}$ equal to an optimum $\mathcal{P}$ vertex. Next, we prove that the lower bound $\mathbf{x}_{\text{lb}}$ also converges towards this $\mathcal{P}$ optimum, *i.e.*, after reaching a state with $\mathbf{x}_{\text{ub}} = \text{OPT}(\mathcal{P})$, $\mathbf{x}_{\text{lb}}$ can get arbitrarily close to $\mathbf{x}_{\text{ub}}$. This comes from the fact that Algorithm 1 can make t^* as small as desired in finite time (see above). The first ray $\mathbf{r}_{\text{new}}$ in the list *rLst* constructed by Algorithm 2 (consider the order after reshuffling) can make the ray line $\mathbf{0}_n \rightarrow \mathbf{r}_{\text{new}}$ pass at an any desired distance from $\mathbf{x}_{\text{ub}}$, and so, lead the IRM sub-problem to a lower bound arbitrarily close to $\mathbf{x}_{\text{ub}}$. The IRM stopping condition $\mathbf{b}^\top\mathbf{x}_{\text{ub}} - \mathbf{b}^\top\mathbf{x}_{\text{lb}} < \epsilon$ can surely be triggered in finite time for any fixed $\epsilon > 0$. □

This proves that IRM is finitely convergent. However, the stopping condition $\mathbf{b}^\top \mathbf{x}_{\text{ub}} - \mathbf{b}^\top \mathbf{x}_{\text{lb}} < \epsilon$ can be met even if $\mathbf{x}_{\text{lb}}$ and $\mathbf{x}_{\text{ub}}$ are not equal to an optimum $\mathcal{P}$ vertex, especially if ϵ is not very small. These solutions can become arbitrarily close, but $\mathbf{x}_{\text{lb}}$ is never determined as a vertex of $\mathcal{P}$.

3 The IRM in Dual LPs of CG Formulations

3.1 Dual Polytope Construction and CG Interpretation

The large-scale LP from Section 2 can be naturally interpreted as the *dual* LP of a fractional relaxation of a master CG model for `Set-Covering` problems. We will provide very specific problem examples in Section 4, but now it is enough to focus on the general dual interpretations (see also Appendix C for the master LP):

- $\mathbf{x} = [x_1\ x_2 \dots x_n]^\top$ is seen as a vector of dual (decision) variables and $\mathcal{P}$ becomes a dual polytope;
- each constraint $[c_a, \mathbf{a}] \in \overline{\mathcal{A}}$ (corresponding to $\mathbf{a}^\top \mathbf{x} \leq c_a$) is associated to a primal column $\left[\begin{smallmatrix} c_a \\ \mathbf{a} \end{smallmatrix}\right]$ and to a *configuration* (route, pattern, cluster) $\mathbf{a} = [a_1\ a_2 \dots a_n]$ with cost c_a, see examples below;
- $\mathbf{b} = [b_1\ b_2 \dots b_n]^\top \in \mathbb{Z}_+^n$ indicates the (dual) objective function. It arises from the covering demands in the master ILP, *i.e.*, any $i \in [1..n]$ has to be covered (serviced, packed) b_i times.

We introduce below a slightly generalized version of the LP from Section 2, simply obtained from (2.1) by replacing "$x_i \geq 0$" with "$x_i \geq l_i$", where l_i is some fixed non-positive value.

$$\left.\begin{array}{ll} \max \mathbf{b}^\top \mathbf{x} & \\ \mathbf{a}^\top \mathbf{x} \leq c_a, & \forall [c_a, \mathbf{a}] \in \overline{\mathcal{A}} \\ x_i \geq l_i, & i \in [1..n] \end{array}\right\} \mathcal{P} \tag{3.1}$$

A well-known example of a dual CG program fitting this interpretation is the dual LP of the Gilmore-Gomory model for `Cutting-Stock` [12]: any $[c_a, \mathbf{a}] \in \overline{\mathcal{A}}$ represents an integer solution (pattern) of a knapsack sub-problem: $c_a = 1$ is a constant pattern cost, b_i is the number of demanded copies of item i and $l_i = 0, \forall i \in [1..n]$. In vehicle or arc routing problems, $\mathbf{a}$ represents a feasible route in some graph, c_a is usually a route cost depending on certain traversed distances and $\mathbf{b}$ is traditionally $\mathbf{1}_n$ (each service is required only once). In location or p-median problems, $\mathbf{a}$ can represent a cluster of customers and c_a the cost of reaching them. Implicitly or explicitly, (3.1) arises in many other `Set-Covering` problems, but we will discuss in great detail `Elastic Cutting-Stock` (Section 4.1) and `Capacitated Arc Routing` (Section 4.2).

Finally, we introduce a few modifications to Algorithm 1, so as to handle two slightly particular CG features (points 1-2 below) and a generalization (point 3):

1. The stopping condition becomes $\lceil \mathbf{b}^\top \mathbf{x}_{\text{lb}} \rceil = \lceil \mathbf{b}^\top \mathbf{x}_{\text{ub}} \rceil$. Since (3.1) is actually a *fractional relaxation* of an *integer* LP, one is generally interested in the integer rounded value of the CG optimum.
2. Instead of starting with $\mathcal{A} \leftarrow \emptyset$ in Line 1, we could easily start with some simple initial constraints of the form $x_i \leq u_i$, where each u_i can be simply determined from some trivial configuration that only concerns element $i \in [1..n]$. We generated either a pattern filled only with copies of i in `Cutting-Stock`, or a route that only services edge i in `Arc-Routing`.
3. Instead of $\mathbf{x} \geq \mathbf{0}_n$, we used $x_i \geq l_i$: while the `Cutting-Stock` model uses $l_i = 0,\ \forall i \in [1..n]$, the `Arc-Routing` model uses $l_i = -\overline{c}_i$, where $\overline{c}_i$ is the length of edge i, see Section 4.2.2, model (4.8). As such, our implementation can even handle rays with negative components, although this would never actually be necessary in Section 2 (where we assumed $\mathbf{x} \geq \mathbf{0}_n$ to reduce clutter).

3.2 Solving the Intersection Sub-problem by Dynamic Programming in CG

We propose solving the intersection sub-problem from Definition 1 (p. 7) using a Dynamic Programming (DP) scheme relying on proposition below.

Proposition 3. *Given a dual polytope $\mathcal{P}$ defined by a constraint set $\overline{\mathcal{A}}$ such that $c_a \geq 0$ for any $[c_a, \mathbf{a}] \in \overline{\mathcal{A}}$, the min-max equivalence $t^* = t^\#$ holds for any $\mathbf{r} \in \mathbb{R}^n$:*

$$\begin{aligned} t^* = &\quad \max\{t \in \mathbb{R}_+ \cup \{\infty\} : \ t\mathbf{r} \in \mathcal{P}\} \\ t^\# = &\quad \min_{\substack{[c_a,\mathbf{a}]\in\overline{\mathcal{A}} \\ \mathbf{a}^\top\mathbf{r}\geq 0}} \frac{c_a}{\mathbf{a}^\top\mathbf{r}} \end{aligned}$$

Proof. We first observe that $t^\#$ can be ∞ only if all $[c_a, \mathbf{a}] \in \overline{\mathcal{A}}$ satisfy $\mathbf{a}^\top\mathbf{r} = 0$. Indeed, only such a situation would allow $\mathbf{a}^\top(t\mathbf{r}) \leq c_a$ to be true for any $[c_a, \mathbf{a}] \in \overline{\mathcal{A}}$ *and for any indefinitely large* t. In this case, $\mathbf{r}$ is an extremal ray in $\mathcal{P}$ and the intersection sub-problem (Definition 1) also returns $t^* = \infty$.
If $t^\# \neq \infty$, then any $t > t^\#$ would lead to $t\mathbf{r} \notin \mathcal{P}$: such $t\mathbf{r}$ would violate a constraint $a^\top\mathbf{y} \leq c_a$ that minimizes the above ratio. This simply follows from: $\mathbf{a}^\top(t\mathbf{r}) = t(\mathbf{a}^\top\mathbf{r}) > t^\#(\mathbf{a}^\top\mathbf{r}) = c_a$. Similarly, we observe that any $t \leq t^\#$ does lead to $t\mathbf{r} \in \mathcal{P}$. As such, the maximum step length t^* returned by Definition 1 is $t^* = t^\#$. □

3.2.1 A Generic Scheme of Profit-Indexed Dynamic Programming

We here present the general principles of a Dynamic Programming (DP) scheme for solving the intersection problem in CG models. The main idea is to calculate the above $t^\#$ by minimizing the ratio from Proposition 3 over all possible configurations in $\overline{\mathcal{A}}$. We interpret $\mathbf{r}$ as a *vector of profits* and $\mathbf{a}^\top\mathbf{r}$ becomes the total profit of configuration $\mathbf{a}$. As such, our DP solution methods aims at finding a configuration that *minimizes the cost/profit* ratio $\frac{c_a}{\mathbf{a}^\top\mathbf{r}}$.

We introduce a set $\mathcal{S}$ of *DP states*; a state $s \in \mathcal{S}$ corresponds to a number of s-configurations (or configurations in state s) that yield the same value on several *state indices*:

- *the integer profit* p_s: all $\mathbf{a} \in s$ have the same profit $p_s = \mathbf{a}^\top\mathbf{r}$;
- *the cost* c_s: all configurations $\mathbf{a}$ in state s have the same cost c_s. While c_s is always 1 in classical `Cutting-Stock`, it can be more complex in `Elastic Cutting-Stock` (Section 4.1.1) and `Arc-Routing` (Section 4.2). In certain versions, this cost is only indirectly used for indexing, *i.e.*, it is actually directly determined from other state indices, see Section 3.2.2 below;
- any other problem-specific state indices, *e.g.*, route start and end points in routing problems. We will have more to say about this in Section 4.2, but now it is enough to focus on above indices p_s and c_s.

In resource-constrained sub-problems (*e.g.*, like the `knapsack` problem), the most widely-used DP algorithms use the weights for state indexing and calculate a maximum profit using a recurrence relation between states. Using the fact that the profits $\mathbf{r}$ are integer, the main idea of our framework is to reverse the role of weights and profits (as in the `knapsack` FPTAS from [14]); we use profits for state indexing and the recurrence formula calculates *minimum weight values* $W(s)$ for all discovered $s \in \mathcal{S}$. This function $W : \mathcal{S} \to \mathbb{R}_+$ indicates the minimum total weight required to reach each state. After discovering all states $\mathcal{S}$, one identifies a state $s^* = (p_{s^*}, c_{s^*})$ that minimizes $\frac{c_{s^*}}{p_{s^*}}$, yielding the sought $t^\# = t^*$ value from Proposition 3.

We finish by pointing out the key idea behind the DP process that computes $\mathcal{S}$ and calculates $W(s)$, $\forall s \in \mathcal{S}$. We exemplify it on a very general recursion: $W(s) = \min_{i\in[1..n]} W\left(s_{\{-i\}}\right) + w_i$, where $s_{\{-i\}}$ is a state that can be obtained from s by removing a copy of i from any configuration $\mathbf{a} \in s$ (with $a_i > 0$). To compute $W(s), \forall s \in \mathcal{S}$, the DP process goes through the elements $i \in [1..n]$; for each $i \in [1..n]$, it scans the current $\mathcal{S}$: for each initialized state $s_{\{-i\}} \in \mathcal{S}$, it updates (or initializes for the first time) the state s by setting $W(s) = \min\left(W(s), W\left(s_{\{-i\}}\right) + w_i\right)$. Compared to $s_{\{-i\}}$, the new state s has one additional copy of i.[3]

[3]Care needs to be taken *not* to add element i more than b_i times by accumulating such additions. In (bounded) knapsack problems, this issue is simply circumvented by going exactly b_i times through element i in the DP process. In routing problems, b_i is usually 1 and this (well-known) phenomenon leads to a need of k-cycle elimination techniques—more details in Stage 1 of the `Arc-Routing` intersection algorithm from Appendix B.

3.2.2 DP Running Time and Comparisons with Weight-Indexed DP in CG

As for classical weight-indexed DP, the asymptotic running time of profit-indexed DP depends on the number of elements n and on the number of states $|\mathcal{S}|$. Let us compare this $|\mathcal{S}|$ to the number of states in an equivalent weight-indexed DP. In profit-indexed DP, $|\mathcal{S}|$ depends closely on the number of realizable *profit* values. In weight-indexed DP, the number of states depends closely on the number of realizable *weight* values.

Since IRM starts out with a very small $\mathbf{r} = \mathbf{b}$ (*i.e.*, $\mathbf{b} = \mathbf{1}_n$ in `Bin-Packing` or `Arc-Routing`), the profits (rays $\mathbf{r}$) are usually much smaller than the weights $\mathbf{w}$ of typical instances. Furthermore, even when this is not the case, IRM can scale down $\mathbf{b}$ in the first iteration—because $\mathbf{b}$ is a ray direction and not a (feasible or infeasible) dual solution. The profits $\mathbf{r}$ can only become larger than $\mathbf{w}$ in case of excessive discretization refining, because each refining step doubles the magnitude of $\mathbf{r}$ (see numerical discussions in Section 5.3).

A combinatorial explosion risk can arise (for both profit-indexed and weight-indexed DP) when there are too many cost values that generate different states. For instance, if the costs can take fractional or continuous values, it can even be impossible to put all feasible cost values in a table data structure and assign a state to each cell. IRM uses two techniques that (try to) tackle this issue. First, the costs c_s are not even always necessarily used for state indexing. For instance, `Elastic Cutting-Stock` defines the cost as a function of the pattern weight, see (4.1) in Section 4.1.1. In this case, the state cost c_s is actually determined from the *minimum* weight $W(s)$. Secondly, IRM can prune states with exceedingly high costs, *e.g.*, while there might exist countless routes covering an exaggeratedly long distance, they are not always associated to (useful) states. It is possible to detect and prune such configurations. We will have more to say about this in the `Arc-Routing` example (Section 4.2.3.3), but for the moment we only illustrate the main idea. Given state $s = (p_s, c_s)$, the best cost/profit ratio that can be reached from s is $\frac{c_s}{p_s + p_{\text{ub}}}$, where p_{ub} is an upper bound of the profit that can be realized through future transitions from s. State s can be pruned if $\frac{c_s}{p_s + p_{\text{ub}}} > t_{\text{ub}}$, where $t_{\text{ub}} \geq t^*$ is the best (minimum) cost/profit ratio reached so far.

4 Integer Ray Sub-problems for Cutting-Stock and Arc-Routing

We discuss below several cutting and routing problems (Section 4.1 and 4.2). In many cases, the (IRM) intersection sub-problem is not necessarily more difficult than the (CG) separation problem, at least when Dynamic Programming (DP) is used. It is rather for paper length reasons that we restricted now to cutting and routing problems, but we did envisage further work on other `Set-Covering` problems.

4.1 Cutting-Stock and Elastic Cutting-Stock

4.1.1 Model Formulation

The well-known Gilmore-Gomory model for `Cutting-Stock` is a direct particularization of (3.1): the cost of any pattern $\mathbf{a}$ is $c_a = 1$ and the feasibility of $\mathbf{a}$ only relies on knapsack condition $\mathbf{a}^\top \mathbf{w} \leq C$. In the elastic version of `Cutting-Stock`, feasible patterns are allowed to exceed a *base capacity* C by paying a penalty cost. The larger the excess over C, the higher the elastic pattern penalty. Let us give a natural bin-packing illustration: a knapsack (vehicle) constructed to hold $C = 15$ kilograms (tonnes) would be very often able to hold 16 kilograms (tonnes), but this would bear some additional cost (*e.g.*, for unwanted damage risk). We consider however a maximum extended capacity $C_{ext} > C$ that can *not* be feasibly exceeded, *e.g.*, we will use $C_{\text{ext}} = 2C$, considering that the base capacity can not be more than doubled by elasticity. Formally, the elastic cost of pattern $\mathbf{a} \in Z^n_+$ is:

$$c_a = f\left(\frac{\mathbf{a}^\top \mathbf{w}}{C}\right), \tag{4.1}$$

where f is a non-decreasing `Elastic` function $f : [0, \frac{C_{\text{ext}}}{C}] \to \mathbb{R}_+$ with a fixed value of 1 over $[0, 1]$.

While the elastic terminology is not widespread in the `Cutting-Stock` literature, this problem is not new. First, a stair-case function f can lead to `Variable-Size Bin Packing` (VSBP), as in example f_{VSBP} below. More generally, the new problem can be interpreted as `Residual Cutting Stock` in the typology of [28]

and reformulated as `Bin-Packing with Usable Leftovers`. In this residual interpretation, one considers that any leftover (unused material) can be returned to the stock, providing a benefit (cost reduction) for potential future leftover use. If one interprets C_{ext} as a total bin capacity, $C_{ext} - \mathbf{a}^\top \mathbf{w}$ becomes an usable leftover of pattern $\mathbf{a}$. The elastic cost $f(\frac{\mathbf{a}^\top \mathbf{w}}{C})$ can be reformulated as: $f(\frac{C_{\text{ext}}}{C}) - \left(f\left(\frac{C_{\text{ext}}}{C}\right) - f\left(\frac{\mathbf{a}^\top \mathbf{w}}{C}\right)\right) = c_{\text{fxd}} - f_{\text{bnf}}(C_{\text{ext}} - \mathbf{a}^\top \mathbf{w})$, *i.e.*, a fixed pattern cost c_{fxd} *minus* a benefit $f_{\text{bnf}}(C_{\text{ext}} - \mathbf{a}^\top \mathbf{w})$ obtained from usable leftover $C_{\text{ext}} - \mathbf{a}^\top \mathbf{w}$; the larger the leftover, the greater the benefit. An example of this interpretation arises in [26], where the pattern cost (see c_q^k in their Section 2) has a term (noted $-r_k L_q^k$) representing the value of returning to stock a length of L_q^k—this pattern cost could have been expressed using an elastic function f.

However, let us focus on the elastic interpretation from function (4.1); we use $C_{\texttt{ext}} = 2C$. Any non-decreasing function $f : \mathbb{R} \to \mathbb{R}$ can actually be used, after restriction to $[0, \frac{C_{\text{ext}}}{C}]$ and truncation to 1 over $[0, 1]$. Paradigmatic examples of elastic functions include:

- $f_{\text{CS}}(x) = \infty$ for $x \in (1, \infty)$: classical `Cutting-Stock`, any excess is prohibited by an infinite penalty;
- $f_k(x) = x^k$ for $x \in (1, \infty)$: the penalty is polynomial with respect to the excess. We will use $k = 2$ and $k = 3$ in practical instances, but larger k values will also be discussed.
- f_{VSBP}: this represents any staircase function that is constant over intervals $\left(\frac{C_{\ell-1}}{C}, \frac{C_\ell}{C}\right]$, with $\ell \in [1..m]$, where $C = C_0 < C_1 < C_2 < \cdots < C_m$ represent $m + 1$ different bin sizes, each with its own cost. This function actually encodes `Variable Size Bin Packing` [28, § 7.13] with $m + 1$ bin sizes.

The main `Set-Covering` CG dual model (3.1) becomes:

$$\left.\begin{array}{ll} \max \mathbf{b}^\top \mathbf{x} & \\ \mathbf{a}^\top \mathbf{x} \leq c_a = f\left(\frac{\mathbf{w}^\top \mathbf{a}}{C}\right), & \forall [c_a, \mathbf{a}] \in \overline{\mathcal{A}} \\ x_i \geq 0 & i \in [1..n] \end{array}\right\} \mathcal{P} \ , \tag{4.2}$$

where $\overline{\mathcal{A}}$ is formally $\left\{[c_a, \mathbf{a}] \in [1, \infty) \times \mathbb{Z}_+^n \ : \ \mathbf{a}^\top \mathbf{w} \leq C_{\text{ext}} = 2C, \ c_a = f\left(\frac{\mathbf{a}^\top \mathbf{w}}{C}\right)\right\}$.

To fully evaluate the interest in the IRM sub-problem with integer rays (Section 4.1.3), let us first present the CG sub-problem of (4.2) and discuss its difficulty for certain functions f (Section 4.1.2). If C and $\mathbf{w}$ consists of bounded integers, the CG sub-problem is tractable. However, if the data is fractional, the CG sub-problem can be as hard as finding some (random) element in an exponential-size set for certain functions f—see examples in §4.1.2.1. Since IRM uses only integer rays $\mathbf{r}$, it can always overcome such computational difficulties by using $\mathbf{r}$-indexed Dynamic Programming in the IRM sub-problem.

4.1.2 The CG Sub-problem

The CG sub-problem can be written:

$$\text{SUB}_{CG}(\mathbf{x}) = \max_{\substack{\mathbf{a} \in \mathbb{Z}_+^n \\ \mathbf{w}^\top \mathbf{a} \leq C_{\text{ext}}}} \mathbf{x}^\top \mathbf{a} - f\left(\frac{\mathbf{w}^\top \mathbf{a}}{C}\right) \tag{4.3}$$

We call this (sub-)problem f-`Elastic Knapsack Problem`: maximize the difference between the profit resulting from benefits $\mathbf{x}$ and the elastic pattern cost determined by elastic function f.

A straightforward CG method for f-`Elastic Cutting Stock` can solve the pricing of (4.2) using an extension of standard knapsack Dynamic Programming (DP). If C is not prohibitively large, the standard knapsack DP can be extended as follows. Instead of generating a state for each realizable total weight $\mathbf{w}^\top \mathbf{a}$ in $[1..C]$, we simply replace this interval with $[1..C_{\text{ext}}] = [1..2C]$. As such, we generate a state for each realizable weight $\mathbf{w}^\top \mathbf{a} \in [1..2C]$ and we evaluate it by the difference between the maximum profit (of a pattern with weight $\mathbf{w}^\top \mathbf{a}$) and the cost $f\left(\frac{\mathbf{w}^\top \mathbf{a}}{C}\right)$; the maximum of this difference is returned in the end.

If $\mathbf{w}$ is larger than $\mathbf{r}$, the above $\mathbf{w}$-indexed DP can often require substantially more states than the IRM $\mathbf{r}$-indexed DP. We will describe this $\mathbf{r}$-indexed DP in Section 4.1.3, but for now it is enough to say that it

generates an $\mathbf{r}$-indexed state for each realizable profit value $\mathbf{r}^\top\mathbf{a}$. As long as the values of $\mathbf{r}$ are close to $\mathbf{1}$ (*e.g.*, in `Bin-Packing`, IRM starts out with $\mathbf{r} = \mathbf{b} = \mathbf{1}_n$), the number of $\mathbf{r}$-indexed states can often stay in the order of tens or hundreds. We observed that the number of of realizable total weight values $\mathbf{w}^\top\mathbf{a}$ is generally much larger, because most instances are defined using weights of hundreds or thousands. As long as $\mathbf{r}$ does not become too large by discretization refining, the $\mathbf{r}$-indexed DP is naturally much faster than $\mathbf{w}$-indexed DP. However, if C is not prohibitively large, the classical $\mathbf{w}$-indexed DP remains a reasonable pricing method *for any function* f. It is (much) more problematic (or impossible) to generalize other knapsack algorithms (*e.g.*, `Minknap`) for more complex functions f.

4.1.2.1 Examples of CG `Elastic` Sub-problems Prohibitively Hard for Fractional Data The main advantage of IRM is that it only uses integer input data for its sub-problem. Since this does not happen in CG, the CG sub-problem could be prohibitively hard for certain multipliers $\mathbf{x}$. We provide below some examples that both consider $\mathbf{x} = \mathbf{w}$ (profits equal to the weight). However, this is only an illustration of more general phenomena: if the multipliers $\mathbf{x}$ are unpredictable, worst-case scenarios of CG sub-problems can more often arise during a CG process.

Consider the following elastic function $f_{\mathrm{r}}(\alpha) : [0, 2] \to \mathbb{R}$:

$$f_{\mathrm{r}}(\alpha) = \begin{cases} 1 & \text{if } \alpha \geq 1 \\ (\alpha - 1)C + 1 - \epsilon \cdot [\alpha = \alpha_{\mathrm{rnd}}] & \text{if } \alpha > 0 \end{cases},$$

where ϵ is a sufficiently small real, α_{rnd} is a randomly chosen value and "$[\alpha = \alpha_{\mathrm{rnd}}]$" makes reference to the Iverson bracket operator (*i.e.*, it takes value 1 if $\alpha = \alpha_{\mathrm{rnd}}$ and 0 otherwise). We evaluate $\mathrm{SUB}_{CG}(\mathbf{x})$=$\mathrm{SUB}_{CG}(\mathbf{w})$ on $\mathbf{a} \in \mathbb{Z}^n_+$ on two cases: (i) $\mathbf{w}^\top\mathbf{a} \leq C$ and (ii) $\mathbf{w}^\top\mathbf{a} = \alpha C > C$. In case (i), the evaluation of $\mathrm{SUB}_{CG}(\mathbf{x})$ on pattern $\mathbf{a}$ yields: $\mathbf{w}^\top\mathbf{a} - f_{\mathrm{r}}\left(\frac{\mathbf{w}^\top\mathbf{a}}{C}\right) = \mathbf{w}^\top\mathbf{a} - 1 \leq C - 1$. In case (ii), $\mathrm{SUB}_{CG}(\mathbf{x})$ is evaluated at $\mathbf{w}^\top\mathbf{a} - f_{\mathrm{r}}(\alpha) = \alpha C - \big((\alpha - 1)C + 1 - \epsilon \cdot [\alpha = \alpha_{\mathrm{rnd}}]\big) = C - 1 + \epsilon \cdot [\alpha = \alpha_{\mathrm{rnd}}]$. A maximum objective value $C - 1 + \epsilon$ can be reached at $\mathbf{w}^\top\mathbf{a} = \alpha_{\mathrm{rnd}}C$. If C and $\mathbf{w}$ consist of fractional (or large-range) data, the number of realizable total weights can easily explode, *i.e.*, the resulting CG sub-problem might actually require finding some (random) element α_{rnd} in an exponential-size subset of interval $[1, 2]$.

The function f_{r} can simply be generalized by replacing the above term "$\epsilon \cdot [\alpha = \alpha_{\mathrm{rnd}}]$" with "$g(\alpha)$", using any function $g : [1, 2] \to [0, \epsilon)$. This ϵ bound is needed to keep f_{r} non-decreasing, *i.e.*, ϵ needs to be lower than the difference between the closest two values $\alpha_1, \alpha_2 \in [1, 2]$ such that $\alpha_1 C$ and $\alpha_2 C$ do represent *realizable total weights* of feasible patterns.

When C is a bounded integer, a weight-indexed DP approach could solve any of the above problems by generating at maximum C states that do cover all realizable total weights. However, the same problem can become prohibitively-hard when C and $\mathbf{w}$ are unbounded or fractional, *i.e.*, the number of realizable total weights can become exponential. In such cases, the last example above can not be solved in reasonable time, unless one can optimize in reasonable time any function g over an exponential sub-set of $[1, 2]$. We discuss next (Section 4.1.3) that the IRM sub-problem associated to functions above is always "vulnerable" to profit-indexed DP if the profits (rays) are bounded integer (regardless of C).

4.1.3 The IRM Approach: the Intersection Sub-problem via Dynamic Programming

By particularizing Proposition 3 from Section 3.2, the `Elastic Cutting-Stock` IRM sub-problem is defined as follows. The input data is: a base capacity $C \in \mathbb{R}$, (possibly fractional) weights $\mathbf{w} \in \mathbb{R}^n$, demands $\mathbf{b} \in \mathbb{Z}^n_+$, non-decreasing function $f : [0, 2] \to \mathbb{R}$ (with $f(x) = 1, \forall x \in [0, 1]$), and ray $\mathbf{r} \in \mathbb{Z}^n_+$. The goal is to find a feasible pattern $\mathbf{a} \in \mathbb{Z}^n_+$ (with $\mathbf{a}^\top\mathbf{r} > 0$) such that:

$$\frac{f\left(\frac{\mathbf{w}^\top\mathbf{a}}{C}\right)}{\mathbf{r}^\top\mathbf{a}} \text{ is minimized}$$

This is a cost/profit ratio minimization objective: $c_a = f\left(\frac{\mathbf{w}^\top\mathbf{a}}{C}\right)$ is a cost and $\mathbf{r}^\top\mathbf{a}$ is a (realizable) integer profit. We can particularize the generic Dynamic Programming (DP) ideas from Section 3.2.1 to the

setting of `Elastic Cutting Stock`. The state associated to profit p only needs to record (see below) the minimum total weight $W(p) = \mathbf{w}^\top \mathbf{a}_p^*$ required to realize a profit $p = \mathbf{r}^\top \mathbf{a}_p^*$. It is clear that it is not necessary to explicitly record the cost $f(\frac{W(p)}{C})$. Given some other pattern $\mathbf{a}_p \neq \mathbf{a}_p^*$ such that $\mathbf{r}^\top \mathbf{a}_p = \mathbf{r}^\top \mathbf{a}_p^* = p$ and $\mathbf{w}^\top \mathbf{a}_p > W(p)$, the cost of $\mathbf{a}_p$ is dominated by that of $\mathbf{a}_p^*$, *i.e.*, $f(\frac{\mathbf{w}^\top \mathbf{a}_p}{C}) \geq f(\frac{W(p)}{C})$, owing to the fact that f is non-decreasing.

The DP pseudo-code is provided in Algorithm 3. To fully ensure its technical correctness, we need to more closely investigate how it discovers and updates the states. We observe that the state of pattern $\mathbf{a}_p^*$ is only determined by its profit $p = \mathbf{r}^\top \mathbf{a}_p^*$ and that we ignore other patterns $\mathbf{a}_p$ such that $\mathbf{r}^\top \mathbf{a}_p = p$ and $\mathbf{w}^\top \mathbf{a}_p > \mathbf{w}^\top \mathbf{a}_p^* = W(p)$. We need to verify that the states generated from $\mathbf{a}_p^*$ do dominate all states that could have been generated from $\mathbf{a}_p$. In other words, we need to check whether the state update condition from Line 12 is correct: can the new state replace the old one only because the new weight $newW$ is lower? This is true, because the updated state accepts all the transitions that the old state could have generated from the current point on (even if the older state was discovered at some lower value of j, this does not influence on the transitions that can be generated from the current point on).

Algorithm 3 Intersection Sub-problem for `Elastic Cutting-Stock`

Require: ray $\mathbf{r} \in \mathbb{Z}^n$, capacity $C \in \mathbb{R}$, weights $\mathbf{w} \in \mathbb{R}^n$, demands $\mathbf{b} \in Z_+^n$ and function f
Ensure: minimum cost/profit t^* ratio

1: $t_{\text{ub}} \leftarrow \infty$ ▷ an upper bound that will converge to t^*
2: `states`$\leftarrow \{0\}$ ▷ a linked list of realizable profits p
3: $W(0) \leftarrow 0$ ▷ a linked list records the minimum weight $W(p), \forall p \in$ `states`
4: **for** i **in** $[1..n]$ ▷ scan each new article
5: **for** j **in** $[1..b_i]$ ▷ at maximum b_i times (no need for more).
6: **for** p **in** `states` ▷ scan only the current states (not updated by the current loop itself)
7: **if** $p + r_i \notin$ `states`
8: `states` $\leftarrow$ `states` $\cup \{p + r_i\}$ ▷ $p + r_i$ is a new realizable profit
9: $W(p + r_i) \leftarrow \infty$ ▷ $W(p + r_i)$ will be updated below
10: **end if**
11: $newW = W(p) + w_i$
12: **if** $newW < 2C$ **and** $newW < W(p + r_i)$ ▷ state update
13: $W(p + r_i) \leftarrow newW$ ▷ the lost state is not completely deleted
14: $cost = f(\frac{newW}{C})$ ▷ to keep track of precedence relations.
15: $t_{\text{ub}} \leftarrow \min(t_{\text{ub}}, \frac{cost}{p+r_i})$
16: **end if**
17: **end for**
18: **end for**
19: **end for**
20: **return** t_{ub} ▷ An associated pattern $\mathbf{a}$ can be built from precedence relations between states

4.2 Capacitated Arc-Routing

We now turn to a more complex problem whose *intersection sub-problem* does not apparently seem well-suited to the generic profit-indexed DP scheme from Section 3.2.1. The configurations of the `Capacitated Arc Routing` Problem (CARP) represent paths in a graph; their costs do not depend on weights, but on distances. More problem-specific concepts (*e.g.*, deadheading) do need to be taken into account. We first start out with the classical CARP CG model (Section 4.2.1). We then continue with a reformulated model (Section 4.2.2) that allows profit-indexed DP to naturally tackle the intersection sub-problem. We finish by providing the main DP scheme in Section 4.2.3; the full pseudo-code is completely specified in Appendix B.

4.2.1 CARP Formulation and Classical CG Model

Consider a graph $G = (V, E)$; each edge $e_i = \{u, v\} \in E$ has a *traversal cost* (length) $\overline{c}_i = \overline{c}(u, v)$. The set $E_R \subseteq E$ (with $|E_R| = n$) represents the edges that require service: each $e_i = \{u, v\} \in E_R$ requires a service that consumes a weight (demands a supply) of $w_i = w(u, v) \in \mathbb{R}_+$. We also consider a (vehicle) capacity C that indicates the maximum total weight (supply delivered) of a feasible route. The value of C belongs to $O(n)$ in existing instances, but it can also be unbounded (large-range) in the scaled instances from Section 5.2.1. A complete *route* is a path in G that starts and ends at a special vertex $v_0 \in V$ (the depot); a route that does not end in v_0 is *open*. The cost $\overline{c}_a$ of a route is the sum of the lengths of the edges traversed by the route. The CARP requires finding a *minimum cost* set of routes that service b_i times each $e_i \in E_R$ (practical instances use $b_i = 1$). We use a slight notation abuse to lighten the text; $\mathbf{w}$, $\overline{\mathbf{c}}$ and $\mathbf{b}$ represent: (i) column vectors with elements indexed by $i \in [1..n]$, *and* (2) functions on E_R, *i.e.*, $w_i = w(u, v)$, $b_i = b(u, v)$ and $\overline{c}_i = \overline{c}(u, v)$ $\forall e_i = \{u, v\} \in E_R$.

A route that traverses edge $e_i \in E_R$ can either service e_i or *deadhead* e_i (traverse e_i without service). Deadheaded edges only matter in calculating route (traversal) costs, but have no impact on route weights (supply delivered). We consider route $\mathbf{a}$ as a vector with $n = |E_R|$ positions: a_i indicates how many times edge $e_i \in E_R$ is serviced. The total weight (load, supply) of $\mathbf{a}$ is $\mathbf{w}^\top \mathbf{a}$. While vector $\mathbf{a}$ does not encode the non-serviced edges, its traversal cost $\overline{c}_a$ is determined as the cost of a *shortest* $\mathbf{a}$*-route*, *i.e.*, the cost of a *minimum cost* route servicing a_i times each $e_i \in E_R$. Indeed, there is no use in considering different routes that service a_i times each $e_i \in E_R$, but travel longer distances. The classical CARP dual CG model is:

$$\begin{array}{lll} & \max \mathbf{b}^\top \overline{\mathbf{x}} & \\ \mathbf{a}^\top \overline{\mathbf{x}} \leq \overline{c}_a, & \forall [c_a, \mathbf{a}] \in \overline{\mathcal{A}} & \\ \overline{y}_i \geq 0, & i \in [1..n] & \end{array}, \tag{4.4}$$

where $\overline{\mathcal{A}} = \left\{ [c_a, \mathbf{a}] \in \mathbb{R}_+ \times \mathbb{Z}_+^n : \ \mathbf{a}^\top \mathbf{w} \leq C, \ \overline{c}_a = \text{total cost of a shortest } \mathbf{a}\text{-route} \right\}$. The main constraint in above model can also be written:

$$\mathbf{a}^\top \overline{\mathbf{x}} \leq \mathbf{a}^\top \overline{\mathbf{c}} + \mathbf{a}_{\mathrm{D}}^\top \overline{\mathbf{c}} \tag{4.5}$$

where $\mathbf{a}_D$ is a vector indicating for each $e_i \in E$ the number of times e_i is deadheaded by a shortest $\mathbf{a}$-route (one can chose any $\mathbf{a}$-route).

4.2.2 Integer Ray Formulation

The above model does not naturally allow one to fully exploit ray integrality properties for solving the intersection sub-problem. Any DP scheme would be complicated by the fact that the right-hand cost $\mathbf{a}^\top \overline{c} + \mathbf{a}_D^\top \overline{c}$ could be very large in (4.5), *i.e.*, using this cost for state indexing could lead to prohibitively-many states. We propose an alternative formulation in which the (right-hand) cost is restricted to the dead-heading cost. Using only dead-heading costs reduces the number of states for two reasons. First, the dead-heading cost $\mathbf{a}_D^\top \overline{c}$ is naturally much smaller (even 0) than the total cost $\mathbf{a}^\top \overline{c} + \mathbf{a}_D^\top \overline{c}$. Secondly, states with smaller dead-heading cost are discovered more rapidly (see below) and they can be later used to prune high-deadheading states. The new formulation only requires a simple variable substitution in (4.4):

$$\overline{\mathbf{x}} = \overline{\mathbf{c}} + \mathbf{x}$$

The objective function $\mathbf{b}^\top \overline{\mathbf{x}}$ simply becomes $\mathbf{b}^\top (\overline{\mathbf{c}} + \mathbf{x})$; $x_i \geq 0$ becomes $x_i \geq -\overline{c}_i, \ \forall i \in [1..n]$. The main constraint (4.5) reduces to:

$$\begin{array}{rl} \mathbf{a}^\top (\overline{\mathbf{c}} + \mathbf{x}) & \leq \mathbf{a}^\top \overline{\mathbf{c}} + \mathbf{a}_{\mathrm{D}}^\top \overline{\mathbf{c}}, \text{ or} \\ \mathbf{a}^\top \mathbf{x} & \leq \mathbf{a}_{\mathrm{D}}^\top \overline{\mathbf{c}} \end{array} \tag{4.6}$$

This constraint is further processed as follows. We consider a route is an alternating sequence of *service segments* and *segment transfers*. A *service segment* $v_{\mathrm{st}} \rightarrowtail v_{\mathrm{end}}$ consists of a continual sequence of serviced edges that start at $v_{\mathrm{st}} \in V$ and finish at $v_{\mathrm{end}} \in V$. Given two consecutive service segments $v_{\mathrm{st}}^1 \rightarrowtail v_{\mathrm{end}}^1$ and $v_{\mathrm{st}}^2 \rightarrowtail v_{\mathrm{end}}^2$ (with $v_{\mathrm{end}}^1 \neq v_{\mathrm{st}}^2$) in a route, v_{end}^1 and v_{st}^2 need to be linked by a *segment transfer*, *i.e.*, by a

shortest cost path joining v^1_{end} and v^2_{st} (no service is provided during this transfer). For any potential *segment transfer* $\{u, v\} \in V \times V$, let us note $c(u, v)$ the length of the shortest path between u and v. Given route $\mathbf{a} \in \mathbb{Z}^n_+$, let us define by $T(\mathbf{a})$ the set of segment transfers of a fixed shortest $\mathbf{a}$-route. The dead-heading cost $\mathbf{a}^\top_{\text{D}}\overline{\mathbf{c}}$ becomes $\sum_{\{u,v\}\in T(a)} c(u, v)$, *i.e.*, (4.6) can be written as:

$$\mathbf{a}^\top \mathbf{x} \leq c_a = \sum_{\{u,v\}\in T(\mathbf{a})} c(u, v) \tag{4.7}$$

The IRM model can now be written below with the new $\mathbf{x}$ variables. It fits (3.1) completely; the only term not arising in (3.1) is $\mathbf{b}^\top\overline{\mathbf{c}}$, which is merely a constant added to the objective function.

$$\left.\begin{array}{ll} \multicolumn{2}{c}{\mathbf{b}^\top\overline{\mathbf{c}} + \max \mathbf{b}^\top\mathbf{x}} \\ \mathbf{a}^\top\mathbf{x} \leq c_a, & \forall[c_a, \mathbf{a}] \in \overline{\mathcal{A}} \\ x_i \geq -\overline{c}_i, & i \in [1..n] \end{array}\right\} \mathcal{P}, \tag{4.8}$$

where $\overline{\mathcal{A}} = \left\{[c_a, \mathbf{a}] \in \mathbb{R}_+ \times \mathbb{Z}^n : \ \mathbf{a}^\top w \leq C, c_a = c_a = \sum_{\{u,v\}\in T(\mathbf{a})} c(u, v)\right\}$. This set $\overline{\mathcal{A}}$ is basically defined as in the original model (4.4); the only difference arises in the cost definition: the new $\overline{\mathcal{A}}$ replaces classical costs $\overline{c}_a$ with dead-heading costs c_a—computed via (4.7) above. We observe that $\mathbf{x}$ can be negative, but $\mathbf{0}_n \in \mathcal{P}$.

4.2.3 IRM Sub-problem Algorithm

4.2.3.1 IRM sub-problem definition for model (4.8)**.** Generally speaking, we proceed as in Section 4.1.3, *i.e.*, we particularize Proposition 3 from Section 3.2. Given integer ray $\mathbf{r} \in \mathbb{Z}^n$, the goal is to find a feasible route $\mathbf{a} \in \mathbb{Z}^n_+$ (with $\mathbf{a}^\top\mathbf{r} > 0$) that minimizes:

$$\frac{c_a}{\mathbf{r}^\top\mathbf{a}} = \frac{\sum_{\{u,v\}\in T(\mathbf{a})} c(u, v)}{\mathbf{r}^\top\mathbf{a}},$$

where $T(\mathbf{a})$ is defined as in (4.7).

This definition has a straightforward cost/profit ratio interpretation: $c_a \geq 0$ is the deadheading cost of any shortest $\mathbf{a}$-route and $\mathbf{r}^\top\mathbf{a}$ is the profit of such a route. We give an example. A route that service a_i times each $e_i \in E_R$ leads to $\frac{c_a}{\mathbf{r}^\top\mathbf{a}} = 0$. If all edges E_R could be serviced by disjoint routes with no deadheading, the IRM sub-problem would return $t^* = 0$ for any input $\mathbf{r}$. This would lead to an optimal solution $\mathbf{x} = \mathbf{0}_n$ in (4.8); its optimum value would simply be given by the constant $\mathbf{b}^\top\overline{\mathbf{c}}$, *i.e.*, the minimum traversal cost that is anyway necessary to service all required edges.

4.2.3.2 Data Structures and State Indexing Following the DP framework from Section 3.2, we will use a set of states $\mathcal{S}$ such that each $s \in \mathcal{S}$ is defined by the following *state indices*:

- p_s is the profit $\mathbf{a}^\top\mathbf{r}$ of any configuration (route) $\mathbf{a}$ in state s;
- c_s is the deadheading cost $\sum_{\{u,v\}\in T(a)} c(u, v)$ of any shortest $\mathbf{a}$-route in state s;
- v_s is the end vertex for any route $\mathbf{a}$ in state s: if $v_s \neq v_0$, the route $\mathbf{a}$ is open, *i.e.*, it does not (yet) return back to the depot.

As in Section 3.2, we introduce a function W such that $W(s) = W(p_s, c_s, v_s) \leq C$ represent the minimum weight (load) of a configuration in state $s = (p_s, c_s, v_s)$, *i.e.*, the minimum load (supply delivered) by an open route ending at v_s with a profit of p_s and a (deadheading) cost of c_s. After generating all relevant states, the DP process returns the lowest cost/profit ratio reached by some state (p^*, c^*, v_0).

4.2.3.3 The DP recursion and comparisons with CG pricing We provide below the main recursion formula for W. We use the following slight abuse of notation c: (i) the usual one-index notation c_a refers to the total cost of configuration $\mathbf{a}$, (ii) as a function, $c(u,v)$ is the shortest path between $u, v \in V$, and (iii) when no index is used, c indicates the cost of a state (p, c, v).

$$W(p,c,v) = \min \begin{cases} W\left(p, c - c\left(u,v\right), u\right) & ,\forall u,v \in V \\ w(u,v) + W\left(p - r\left(u,v\right), c, u\right) & ,\forall \{u,v\} \in E_R \text{ not serviced at } \left(p - r\left(u,v\right), c, u\right) \end{cases} \tag{4.9}$$

This recurrence is exploited by Algorithm 4 (Appendix B, p. 36) that provide the full pseudocode. Its running time does not directly depend on the fractionality or the range of $\mathbf{w}$ and C, because these values have no direct influence on the number of DP states (see below).

To our knowledge, all existing CG pricing algorithms *do explicitly use* states or iterations indexed by feasible values of total weight in $[1..C]$, leading to a pricing complexity of at least $O(C(|E|+|V|\log|V|))$ [17]. Other variants can have a complexity of $O(C|V_R|^2)$ (V_R represents the vertices with at least an incident required edge) or more often $O(C|E_R|^2)$, as argued in [17, § 3.1]—see also the more recent pricing algorithm discussed in [4, § 7.1]. However, the term C is always present; the instances commonly tackled by CG (see Section 5.2.1) have an average capacity of about one hundred. On the other hand, state-of-the-art CG pricing algorithms can be very fast in terms of the complexity factors $|V|$, $|V_R|$ or $|E_R|$.

The number of states in our DP scheme is at most $|V|p_{\text{cnt}}c_{\text{cnt}}$, where p_{cnt} and c_{cnt} are reasonably-large integers in many practical instances:

- p_{cnt} is the number (count) of realizable profit values; p_{cnt} is naturally (much) smaller than the number of realizable profit values in a knapsack problem with profits $\mathbf{r}$, weights $\mathbf{w}$ and capacity C.
- c_{cnt} is the maximum number of states that can be discovered for any v and p. For some v and p, there needs to exist c_{cnt} cost values $c_1 < c_2 < \ldots c_{\max}$ such that $W(p,c_1,v) > W(p,c_2,v) > \ldots W(p,c_{\max},v)$. Indeed, the only case in which a state $(p, c_{\ell+1}, v)$ with $c_{\ell+1} > c_\ell$ is not dominated by (p, c_ℓ, v) is when it has a lower weight $W(p, c_{\ell+1}, v) < W(p, c_\ell, v)$. To limit $c_{\max}$, certain exceedingly high-cost states can be pruned (dominated) by low-cost states. For instance, a state (c, p, v) can be pruned if $\frac{c}{p+pMaxRes} \geq t_{\text{ub}}$, where $pMaxRes$ is a maximum residual profit that can be obtained from state (c, p, v), and t_{ub} is the minimum cost/profit ratio reached so far (see Stage 4, Algorithm 4, Appendix B).

5 `Elastic Cutting Stock` and `Capacitated Arc Routing` Experiments

This section performs an IRM evaluation[4] on four `Elastic Cutting Stock` variants and on `Capacitated Arc Routing`. For all experiments, we will consider two types of instances: (i) the original standard-size instances and (ii) *scaled* instances that we define as follows. Given a standard instance with capacity C and weights w_i ($\forall i \in [1..n]$), the *scaled* instance is constructed using a *scaled capacity* of $1000C$ and *scaled weights* $1000w_i - \rho_i$, where ρ_i is a (small) random "noise" adjustment (*i.e.*, we used $\rho_i = i \bmod 10$). This noise only renders the scaled instance more realistic; without it, the scaling operation could be easily reversed by dividing all weights by their greatest common divisor. Both instance versions use integer data, but the scaled weights can be seen as fractional weights recorded with a precision of 3 decimals. Preliminary experiments show that a higher precision (scale factors larger than 1000) could slightly skew the results, but not enough to upset our main conclusions.

Practical instances usually have the same IP optimum in both scaled or unscaled versions. In standard `Cutting Stock` and `Arc Routing`, the feasibility of a configuration (pattern, route) is not influenced by scaling: (i) any valid configuration remains valid after scaling ($\sum_{i=1}^{n} a_i w_i \leq C \implies \sum_{i=1}^{n} a_i(w_i - \rho_i) \leq$

[4]The implementation used for this evaluation is publicly available on-line at `http://www.lgi2a.univ-artois.fr/~porumbel/irm/`. The main IRM algorithm consists of a core of problem-independent IRM routines. The integration of a (new) problem requires providing two functions (*i.e.*, `loadData` and `irmSub-problem`) to be linked to the IRM core (the file `subprob.h` describes the exact parameters of these functions). It is worthwhile noticing that it is much easier to manipulate $\frac{1}{t^*}$ than t^* in the code.

$1000C$), and (ii) any configuration exceeding C is turned into a configuration exceeding $1000C$ (observe that $\sum_{i=1}^{n} a_i w_i \geq C+1 \implies \sum_{i=1}^{n} a_i(1000w_i - \rho_i) \geq 1000C + 1000 - \sum_{i=1}^{n} a_i\rho_i > 1000C$). The last inequality is only valid if $1000 > \sum_{i=1}^{n} a_i\rho_i$ (for any configuration $\mathbf{a}$), but this is surely valid if $100 > \sum_{i=1}^{n} a_i$ (recall $\rho_i < 10$, $\forall i \in [1..n]$), which is typically true because practical configurations have less than 10-20 items (or serviced edges).

The *computing effort* is typically indicated as an expression "`iters (discr)/time`$^{\texttt{-fails}}$" where `iters` is the number of iterations (in parentheses, the number `discr` of discretization refining calls), `time` is the CPU time in seconds and `fails` is an (optional) number of failures, *i.e.*, instances *not* solved within the allowed time (if any). If these failures represents more than 25% of a benchmark set, we simply report "tm. out". All times are reported on a HP ProBook 4720 laptop clocked at 2.27GHz (Intel Core i3), using `gnu g++` with code optimization option `-O3` on Linux Ubuntu, kernel version 2.6 (Cplex version 12.3).

5.1 Cutting-Stock and Elastic Cutting-Stock

5.1.1 Problem Classes, Knapsack Algorithms and Benchmark Sets

The IRM is compared with classical CG on a set of 13 benchmark sets and 4 `Elastic` problem variants (discussed in Section 4.1). For a fair comparison, we did our best to use the fastest knapsack algorithms for CG pricing, *i.e.*, the following *three* approaches have been considered:

`Minknap` This is one of the most competitive knapsack algorithms from the literature. The evaluation on the extensive benchmark sets from [23] indicates that it "has an overall stable performance, as it is able to solve all instances within reasonable time. It is the fastest code for uncorrelated and weakly correlated instances". Such weak correlations between profits and weights can be natural in CG sub-problems. Even for rather strongly correlated instances, `Minknap` is one of the best three algorithms from the literature when the weights have wide ranges [23, Table 6-8]. We adapted and used the code available at `www.diku.dk/~pisinger/minknap.c`.

`Cplex` This is the IP solver provided by `Cplex`. The CG results on classical `Cutting-Stock` are very close to those that would have been reported by the `Cutting-Stock` CG implementation from the C++ examples of `Cplex Optimization Studio` (version 12.3).

`Std-DP` Standard Dynamic Programming with weight-indexed states (implemented with linked lists).

We summarize below the applicability of above approaches on our `Elastic` problem variants; we identify a `Elastic` variant by the elastic penalty function, using the notations from Section 4.1.1.

$f_{\mathbf{CS}}$ This `Elastic` variant is the pure `Cutting-Stock`; the pricing sub-problem is the knapsack problem that can be solved by all above knapsack algorithms;

$f_{\mathbf{VSBP}-m}$ In `Variable-Sized Bin Packing`, one can simply run any of the above knapsack algorithms $m+1$ times, once for each capacity in $C = C_0, C_1, C_2, \ldots C_m$. We used $m = 10$, $C_\ell = C + \ell\frac{C}{10}$ $\forall \ell \in [1..m]$, and so, the maximum bin size is $C_m = 2C$;

$f_2(x) = x^2, \forall x > 1$ The CG sub-problem for this `Elastic` variant can be written as a quadratic IP, using (4.3). We could solve it with `Std-DP` and with the `Cplex` solver for Quadratic IP. `Minknap` could not be adapted to this pricing sub-problem, because it uses too many specialized pure knapsack features.

$f_3(x) = x^3, \forall x > 1$ The sub-problem becomes a Cubic IP that could only be solved by `Std-DP`. The `Cplex` solver does not address Cubic IPs. We are not aware of other conventional (and practical) algorithms for this sub-problem. The situation is similar to that of other `Elastic` power functions (*e.g.*, x^4, x^5) or polynomials. We are far from claiming that such sub-problems can never be solved with other techniques, but the goal of this paper is not to delve into the depths of polynomial integer programming.

Given an instance of pure `Cutting-Stock`, any f-`Elastic` variant can simply be constructed by a applying function f: the cost of a pattern $\mathbf{a}$ becomes $f(\frac{\mathbf{a}^\top \mathbf{w}}{C})$. The 13 `Cutting-Stock` benchmark sets are described

in Appendix D (see Table 5 for references and instance characteristics). Most instances have at most $n = 100$ items and the capacity C can be 100, 10.000, 30.000 and rarely 100.000 (only for `Hard` instances, the most difficult ones). The capacity of scaled instances thus ranges from 10^5 to 10^8.

5.1.2 Numerical Comparison IRM-CG: Standard and Scaled Instances

Tables 1 and 2 (on next two pages) present the results for scaled instances (observe suffix "$_{\text{scl}}$" in the instance name) and respectively for unscaled instances. Both tables use the following format: the first two columns indicate the instance set (the `Elastic` function in Column 1 and the benchmark set in Column 2), Column 3 reports the average IRM *computing effort* (under the format described by the third paragraph of Section 5) to reach a gap of 10% between the lower and the upper bound (*i.e.*, $\mathbf{b}^\top \mathbf{x}_{\text{lb}} \geq \frac{9}{10}\mathbf{b}^\top \mathbf{x}_{\text{ub}}$), Column 4 provides the average IRM *computing effort* for complete convergence (we stop when $\lceil \mathbf{b}^\top \mathbf{x}_{\text{lb}} \rceil = \lceil \mathbf{b}^\top \mathbf{x}_{\text{ub}} \rceil$). The last three columns indicate the CG *computing effort*, *i.e.*, there is one column for each of the three CG pricing approaches discussed above (Section 5.1.1).

We present the main conclusions of these tables, picking out the key claims along the way in boldface.

The IRM is a very reasonable approach for standard instances and it is generally faster than CG on *scaled* instances. Confirming theoretical arguments, the 1000-fold weight increase (the scaling) has a limited impact on IRM time running time (*i.e.*, a CPU time increase in $[-10\%, 30\%]$). We compare IRM with the above three CG algorithms:

(i) The results of CG[`Minknap`] are on a par with those of IRM, but it can only be applied to pure `Cutting-Stock` and `Variable-Sized Bin Packing` (see more detailed `Minknap` comparisons below).

(ii) CG[`Cplex`] seems quite slow and it can never compete with IRM.

(iii) CG[`Std-DP`] is the only CG approach that can be used on all `Elastic` versions. It requires a similar number of iterations as IRM, but the 1000-fold weight increase can make the **w**-indexed DP in CG[`Std-DP`] 10 times slower than the **r**-indexed DP in IRM. Confirming theoretical arguments from Section 3.2.2 and 4.1.2, the state space is often smaller in **r**-indexed DP than in **w**-indexed DP. Indeed, `Std-DP` indexes states using weight values that are usually (substantially) larger than the values of the rays **r** used for indexing DP states in IRM. The very first value of **r** is **b** (*e.g.*, $\mathbf{1}_n$ for `Bin-Packing`), and, even after a few discretization refining operations (usually less than 5, see columns "Gap 10%" in Table 2), the ray values **r** can still be generally smaller than values of the (unscaled) weights **w**. This is why IRM can also scale $\mathbf{r} = \mathbf{b}$ down if **b** contain some very large values (see Line 2 in Algorithm 1).

On pure `Cutting Stock`, IRM competes well with CG[`Minknap`]. The fact that IRM can solve most `Hard` and `vb` instances in less than 15 seconds (on a mainstream PC) can be generally seen as acceptable, even compared to some stabilized CG methods. On `Variable-Sized Bin Packing`, IRM is generally faster than CG[`Minknap`], because the latter one requires solving 11 sub-problems at each CG iteration. In terms of iterations, IRM stays in the same order of magnitude compared to all CG versions. However, the number of CG iterations can be very different among the three CG versions. For example, on `m35`, all dual constraints have the form $x_i + x_j \leq 1$ $(i, j \in [1..n])$; since the values of **x** are often in $\{0, 0.5, 1\}$, numerous columns have the same reduced cost (-0.5), leading to very different ways of breaking ties at each iteration.

The only important IRM disadvantage is a certain lack of robustness. It can fail in solving 100% of all instances in certain sets. This is due to some *early* excessive discretization refining that slows down the **r**-indexed DP. At the first IRM iterations, the upper bound can be far from the optimum, and so, it can lead the ray generation (Algorithm 2, Section 2.3) to ineffective new rays, which can further trigger too rapidly some (unfortunate) discretization refining. We consider that such issues could be addressed by further IRM studies, *e.g.*, by the use of other ray generation methods, by applying IRM stabilization techniques, by reversing the discretization refining when possible, etc.

Elast. Ver.	Instance Set	IRM *Computing Effort* (`iters (discr)/time`$^{\texttt{-fails}}$)		CG *Computing Effort* (`iters/time`$^{\texttt{-fails}}$)		
		Gap 10%	Full convergence	`Minknap`	`Cplex`	`Std. DP`
$f(x) = x^3$	$\texttt{vb10}_{\text{scl}}$	5(0.0) / 0.01	21(1.6) / 0.05	—	—	tm. out
	$\texttt{vb20}_{\text{scl}}$	9(0.0) / 0.03	41(1.8) / 1.2^{-1}	—	—	tm. out
	$\texttt{vb50-c1}_{\text{scl}}$	56(0.0) / 0.2	162(4.8) / 1.4	—	—	tm. out
	$\texttt{vb50-c2}_{\text{scl}}$	27(0.0) / 0.3	160(3.5) / 26.2^{-2}	—	—	tm. out
	$\texttt{vb50-c3}_{\text{scl}}$	12(0.0) / 0.3	85(2.6) / 20.9^{-4}	—	—	tm. out
	$\texttt{vb50-c4}_{\text{scl}}$	30(0.0) / 0.1	171(3.4) / 17.2^{-2}	—	—	tm. out
	$\texttt{vb50-c5}_{\text{scl}}$	13(0.0) / 0.1	115(3.3) / 38.0^{-5}	—	—	tm. out
	$\texttt{vb50-b100}_{\text{scl}}$	30(0.0) / 0.1	tm. out	—	—	tm. out
	$\texttt{m01}_{\text{scl}}$	154(2.2) / 0.4	272(5.5) / 1.1	—	—	tm. out
	$\texttt{m20}_{\text{scl}}$	158(1.6) / 0.5	249(6.0) / 0.8	—	—	291 / 62.4^{-2}
	$\texttt{m35}_{\text{scl}}$	97(2.3) / 0.3	161(6.2) / 0.4	—	—	246 / 19.4
	$\texttt{Hard}_{\text{scl}}$	149(2.1) / 1.1	578(6.0) / 16.2^{-1}	—	—	tm. out
	$\texttt{Triplets}_{\text{scl}}$	43(1.0) / 0.1	76(1.4) / 0.3	—	—	tm. out
$f(x) = x^2$	$\texttt{vb10}_{\text{scl}}$	5(0.0) / 0.01	23(2.1) / 0.01	—	23 / 1.4	tm. out
	$\texttt{vb20}_{\text{scl}}$	9(0.0) / 0.03	41(2.1) / 1.0^{-1}	—	tm. out	tm. out
	$\texttt{vb50-c1}_{\text{scl}}$	51(0.0) / 0.2	155(4.0) / 1.6	—	138 / 26.9	tm. out
	$\texttt{vb50-c2}_{\text{scl}}$	26(0.0) / 0.2	154(3.6) / 22.8^{-4}	—	tm. out	tm. out
	$\texttt{vb50-c3}_{\text{scl}}$	12(0.0) / 0.3	86(2.8) / 24.4^{-2}	—	tm. out	tm. out
	$\texttt{vb50-c4}_{\text{scl}}$	29(0.0) / 0.1	159(3.4) / 16.3^{-3}	—	tm. out	tm. out
	$\texttt{vb50-c5}_{\text{scl}}$	13(0.0) / 0.1	tm. out	—	tm. out	tm. out
	$\texttt{vb50-b100}_{\text{scl}}$	29(0.0) / 0.1	tm. out	—	tm. out	tm. out
	$\texttt{m01}_{\text{scl}}$	138(2.3) / 0.4	265(5.6) / 1.1	—	210/46.4^{-8}	tm. out
	$\texttt{m20}_{\text{scl}}$	144(2.1) / 0.4	224(5.8) / 0.8	—	202/35.2^{-3}	292 / 63.2^{-1}
	$\texttt{m35}_{\text{scl}}$	88(2.5) / 0.2	150(6.1) / 0.5	—	208/16.3	290 / 22.6
	$\texttt{Hard}_{\text{scl}}$	151(2.1) / 1.0	517(6.1) / 14.9^{-1}	—	tm. out	tm. out
	$\texttt{Triplets}_{\text{scl}}$	42(1.0) / 0.1	71(1.0) / 0.2	—	tm. out	tm. out
Variable Sized `Cut. Stock`	$\texttt{vb10}_{\text{scl}}$	5(0.0) / 0.01	21(2.2) / 1.0	$32_{\times 11}$/ 1.0	$32_{\times 11}$/ 60.0^{3}	tm. out
	$\texttt{vb20}_{\text{scl}}$	8(0.0) / 0.04	37(2.4) / 4.0	$55_{\times 11}$/ 4.9	tm. out	tm. out
	$\texttt{vb50-c1}_{\text{scl}}$	33(0.0) / 0.1	139(4.1) / 23.0^{-4}	$184_{\times 11}$/ 34.3^{-1}	tm. out	tm. out
	$\texttt{vb50-c2}_{\text{scl}}$	23(0.0) / 0.3	97(3.4) / 17.7^{-5}	$131_{\times 11}$/ 25.2	tm. out	tm. out
	$\texttt{vb50-c3}_{\text{scl}}$	11(0.0) / 0.7	67(3.0) / 20.4^{-1}	$97_{\times 11}$/ 17.0	tm. out	tm. out
	$\texttt{vb50-c4}_{\text{scl}}$	25(0.0) / 0.0	110(3.4) / 19.9^{-5}	$134_{\times 11}$/ 24.3	tm. out	tm. out
	$\texttt{vb50-c5}_{\text{scl}}$	13(0.0) / 0.1	82(3.4) / 27.4^{-3}	$101_{\times 11}$/ 15.6	tm. out	tm. out
	$\texttt{vb50-b100}_{\text{scl}}$	24(0.0) / 0.1	tm. out	$133_{\times 11}$/ 46.8	tm. out	tm. out
	$\texttt{m01}_{\text{scl}}$	70(1.9) / 0.2	236(5.1) / 1.0^{-2}	$149_{\times 11}$/ 0.5	tm. out	tm. out
	$\texttt{m20}_{\text{scl}}$	94(2.4) / 0.3	201(5.6) / 1.2^{-2}	$176_{\times 11}$/ 0.6	tm. out	tm. out
	$\texttt{m35}_{\text{scl}}$	98(2.6) / 0.3	231(6.2) / 1.4	$206_{\times 11}$/ 0.8	tm. out	tm. out
	$\texttt{Hard}_{\text{scl}}$	165(2.0) / 1.4	445(5.3) / 8.2^{-1}	$715_{\times 11}$/ 37.8	tm. out	tm. out
	$\texttt{Triplets}_{\text{scl}}$	44(1.0) / 0.2	73(1.0) / 0.5	$398_{\times 11}$/ 13.6	tm. out	tm. out
$f_{\text{cs}}(x) = 1$, pure `Cut. Stock`	$\texttt{vb10}_{\text{scl}}$	5(0.0) / 0.01	17(0.9) / 0.03	14 / 0.03	15 / 1.0	15 / 1.1
	$\texttt{vb20}_{\text{scl}}$	10(0.0) / 0.02	41(2.0) / 0.8	51 / 0.3	tm. out	tm. out
	$\texttt{vb50-c1}_{\text{scl}}$	55(0.0) / 0.1	116(2.0) / 0.4	105 / 0.3	104 / 13.2	tm. out
	$\texttt{vb50-c2}_{\text{scl}}$	28(0.0) / 0.1	176(3.5) / 7.0^{-1}	201 / 3.2	tm. out	tm. out
	$\texttt{vb50-c3}_{\text{scl}}$	12(0.0) / 0.1	93(2.7) / 12.3^{-1}	124 / 1.7	tm. out	tm. out
	$\texttt{vb50-c4}_{\text{scl}}$	30(0.0) / 0.0	183(3.3) / 7.4	181 / 2.3	tm. out	tm. out
	$\texttt{vb50-c5}_{\text{scl}}$	12(0.0) / 0.0	132(3.6) / 19.8^{-2}	133 / 1.8	132 / 70.4^{-1}	tm. out
	$\texttt{vb50-b100}_{\text{scl}}$	30(0.0) / 0.0	199(4.8) / 7.5^{-2}	181 / 4.1	tm. out	tm. out
	$\texttt{m01}_{\text{scl}}$	109(0.9) / 0.3	307(3.5) / 1.2	123 / 0.4	129 / 7.9	132 / 4.8
	$\texttt{m20}_{\text{scl}}$	98(0.9) / 0.2	303(1.9) / 1.0^{-1}	198 / 0.5	141 / 6.9	247 / 1.1
	$\texttt{m35}_{\text{scl}}$	199(0.2) / 0.5	199(0.4) / 0.6	165 / 0.4	89 / 1.8	199 / 0.6
	$\texttt{Hard}_{\text{scl}}$	254(2.0) / 1.9	872(6.1) / 14.9^{-1}	716 / 13.5	tm. out	tm. out
	$\texttt{Triplets}_{\text{scl}}$	317(1.9) / 2.5	1001(2.7) / 15.8	546 / 5.2	tm. out	551 / 23.7

Table 1: Results for `Elastic Cutting Stock` *scaled* (large-range) instances with a time limit of 100 seconds.

[-f] *f is the number of failures, instances not solved in 100 seconds. If f represent* $\geq 25\%$ *of instances, we mark "tm. out".*

Elast. Ver.	Instance Set	IRM *Computing Effort* (`iters (discr)/time`$^{\texttt{-fails}}$)		CG *Computing Effort* (`iters/time`$^{\texttt{-fails}}$)		
		Gap 10%	Full convergence	`Minknap`	`Cplex`	`Std. DP`
$f(x) = x^3$	`vb10`	5(0.0) / 0.01	21(1.7) / 0.04	—	—	20 / 18.7
	`vb20`	9(0.0) / 0.03	43(2.3) / 2.4	—	—	tm. out
	`vb50-c1`	56(0.0) / 0.2	165(4.7) / 1.5	—	—	tm. out
	`vb50-c2`	27(0.0) / 0.3	165(3.6) / 29.0^{-2}	—	—	tm. out
	`vb50-c3`	12(0.0) / 0.3	84(2.7) / 22.3^{-2}	—	—	tm. out
	`vb50-c4`	30(0.0) / 0.1	170(3.3) / 19.7^{-2}	—	—	tm. out
	`vb50-c5`	13(0.0) / 0.1	116(3.3) / 34.3^{-5}	—	—	tm. out
	`vb50-b100`	30(0.0) / 0.1	tm. out	—	—	tm. out
	`m01`	159(2.1) / 0.4	277(5.2) / 0.8	—	—	199 / 3.7
	`m20`	172(1.7) / 0.4	258(5.4) / 0.6	—	—	289 / 3.5
	`m35`	97(2.2) / 0.2	154(5.9) / 0.3	—	—	244 / 2.3
	`Hard`	148(2.2) / 0.7	568(6.0) / 19.3^{-1}	—	—	tm. out
	`Triplets`	43(1.0) / 0.1	76(1.4) / 0.2	—	—	tm. out
$f(x) = x^2$	`vb10`	5(0.0) / 0.01	24(2.1) / 0.04	—	23 / 1.3	22 / 19.1
	`vb20`	9(0.0) / 0.03	41(2.2) / 4.2	—	56 / 14.2^{-1}	tm. out
	`vb50-c1`	51(0.0) / 0.2	156(4.0) / 1.6	—	140 / 28.9	tm. out
	`vb50-c2`	26(0.0) / 0.2	153(3.5) / 23.9^{-3}	—	tm. out	tm. out
	`vb50-c3`	12(0.0) / 0.3	83(2.8) / 20.6^{-2}	—	129 / 43.7	tm. out
	`vb50-c4`	29(0.0) / 0.1	160(3.6) / 18.2^{-2}	—	tm. out	tm. out
	`vb50-c5`	13(0.0) / 0.1	109(3.2) / 34.6^{-4}	—	tm. out	tm. out
	`vb50-b100`	29(0.0) / 0.1	175(3.5) / 33.7^{-5}	—	tm. out	tm. out
	`m01`	141(2.2) / 0.4	255(5.7) / 1.1	—	204 / 13.6	218 / 4.0
	`m20`	143(2.1) / 0.4	223(5.9) / 0.8	—	190 / 11.6	292 / 3.6
	`m35`	88(2.5) / 0.3	150(6.1) / 0.5	—	199 / 7.7	289 / 2.7
	`Hard`	153(2.2) / 1.1	526(6.3) / 17.5^{-1}	—	tm. out	tm. out
	`Triplets`	42(1.0) / 0.1	71(1.0) / 0.1	—	tm. out	tm. out
Variable Sized `Cut. Stock`	`vb10`	5(0.0) / 0.01	21(2.2) / 0.7	$32_{\times 11}$/ 0.9	$32_{\times 11}$/ 60.20^{-2}	tm. out
	`vb20`	8(0.0) / 0.04	37(2.5) / 2.6	$55_{\times 11}$/ 3.5	tm. out	tm. out
	`vb50-c1`	33(0.0) / 0.1	136(4.0) / 15.3^{-4}	$185_{\times 11}$/ 27.6^{-1}	tm. out	tm. out
	`vb50-c2`	23(0.0) / 0.4	tm. out	$132_{\times 11}$/ 16.9	tm. out	tm. out
	`vb50-c3`	11(0.0) / 0.7	66(2.9) / 15.4^{-2}	$98_{\times 11}$/ 8.0	tm. out	tm. out
	`vb50-c4`	25(0.0) / 0.1	111(3.5) / 20.5^{-5}	$134_{\times 11}$/ 19.5	tm. out	tm. out
	`vb50-c5`	13(0.0) / 0.1	82(3.4) / 25.5^{-3}	$101_{\times 11}$/ 9.8	tm. out	tm. out
	`vb50-b100`	24(0.0) / 0.1	tm. out	$132_{\times 11}$/ 37.2	tm. out	tm. out
	`m01`	70(1.8) / 0.2	232(5.1) / 0.9^{-6}	$148_{\times 11}$/ 0.7	$140_{\times 11}$/ 73.35	$138_{\times 11}$/ 57.6
	`m20`	94(2.4) / 0.3	197(5.6) / 1.1^{-2}	$175_{\times 11}$/ 0.9	tm. out	$145_{\times 11}$/ 42.7
	`m35`	99(2.6) / 0.3	230(6.2) / 1.3^{-3}	$206_{\times 11}$/ 1.0	tm. out	$157_{\times 11}$/ 36.0
	`Hard`	165(2.0) / 1.4	442(5.3) / 8.0^{-1}	$715_{\times 11}$/ 37.3	tm. out	tm. out
	`Triplets`	44(1.0) / 0.2	73(1.0) / 0.5	$398_{\times 11}$/ 13.7	tm. out	tm. out
$f_{\text{CS}}(x) = 1$, pure `Cut. Stock`	`vb10`	5(0.0) / 0.01	17(0.9) / 0.03	14 / 0.03	16 / 1.12	15 / 0.6
	`vb20`	10(0.0) / 0.02	42(1.9) / 0.6	35 / 0.1	49 / 11.06^{-1}	51 / 61.5
	`vb50-c1`	55(0.0) / 0.1	115(2.0) / 0.3	105 / 0.4	101 / 12.3	tm. out
	`vb50-c2`	28(0.0) / 0.1	169(3.5) / 5.7^{-1}	154 / 1.3	tm. out	tm. out
	`vb50-c3`	12(0.0) / 0.1	91(2.8) / 12.2	82 / 0.5	124 / 32.3	tm. out
	`vb50-c4`	30(0.0) / 0.1	186(3.4) / 6.6	153 / 1.2	tm. out	tm. out
	`vb50-c5`	12(0.0) / 0.0	131(3.5) / 12.8^{-3}	101 / 0.8	130 / 61.3	tm. out
	`vb50-b100`	30(0.0) / 0.1	199(4.8) / 6.6^{-2}	171 / 3.2	tm. out	tm. out
	`m01`	109(0.9) / 0.3	300(3.5) / 1.1	122 / 0.4	128 / 6.7	130 / 0.9
	`m20`	97(0.9) / 0.3	302(1.9) / 1.1^{-1}	199 / 0.5	141 / 6.7	248 / 0.9
	`m35`	199(0.2) / 0.6	199(0.4) / 0.6	166 / 0.4	89 / 2.0	199 / 0.6
	`Hard`	254(2.0) / 1.9	872(6.1) / 14.8^{-1}	454 / 4.5	tm. out	tm. out
	`Triplets`	317(1.9) / 2.5	1001(2.7) / 15.8	546 / 4.9	tm. out	551 / 23.7

Table 2: Results for `Elastic Cutting Stock` *unscaled* (standard) instances with a time limit of 100 seconds.

[-f] *f is the number of failures, instances not solved in 100 seconds. If f represent* $\geq 25\%$ *of instances, we mark "tm. out".*

5.1.3 Comparisons with Lagrangian lower bounds

Above results show that IRM can reach in one second a lower bound within 90% of the upper bound *for all* `Elastic` variants (see Columns "Gap 10%" in both Tables 1 and 2). We now perform a more detailed comparison of these lower bounds with the Lagrangean bounds in CG. A well-known Lagrangean bound in the `Cutting-Stock` literature is the Farley bound—see [6, § 2.2], [25, § 3.2] or [19, § 2.1] for interesting descriptions or Appendix C.

Figure 4 below compares the evolution of the IRM and the CG (Farley) lower bounds on several classical and elastic `Cutting-Stock` instances. While the performance of the two bounds seem globally similar, a closer investigation of the first iterations reveals the following IRM advantage: CG might require hundreds of iterations to reach the bound reported by IRM in the very beginning. This phenomenon is most visible on the first (`Triplets`) instance: the optimal dual solution $\mathbf{x} = \frac{\mathbf{1}_n}{3}$ is reported at the very first IRM iteration (when $\mathbf{r} = \mathbf{b} = \mathbf{1}_n$ and $t^* = \frac{1}{3}$), while CG actually needs to finish the search to reach a similar performance. The `Bin-Packing` instances `m01`, `m20` reveals the same phenomenon at a closer look. However, on the rest of the instances, the lower bounding performances are globally in the same order of magnitude.

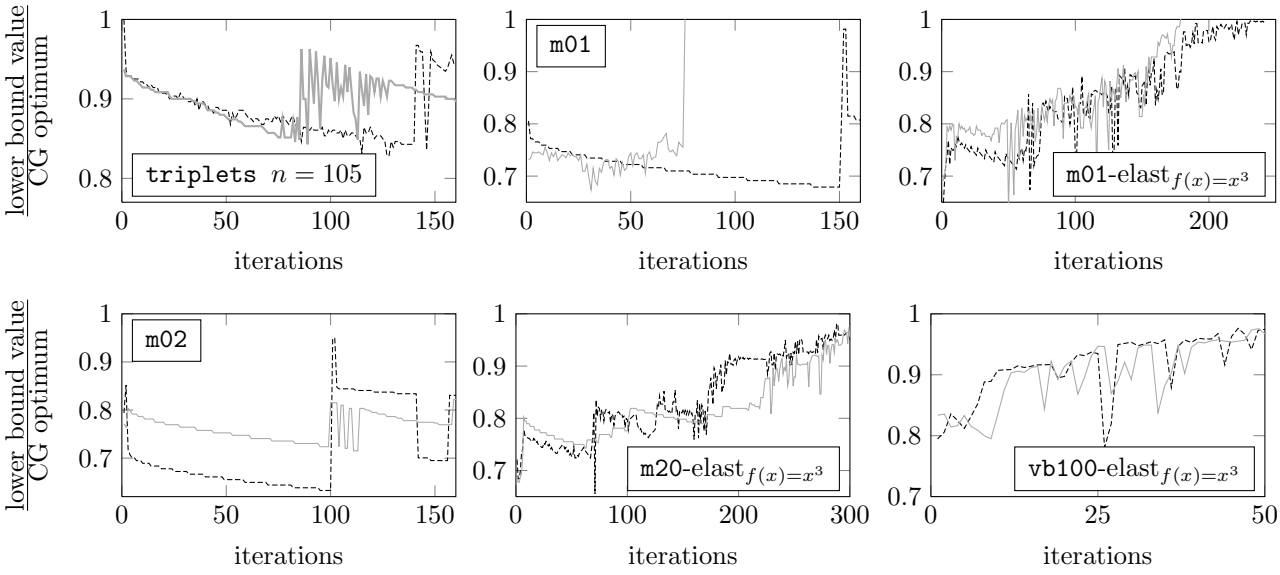


Figure 4: Comparison between the IRM lower bounds (dotted black lines) and the Farley CG lower bounds (solid gray lines) on various unscaled `Cutting-Stock` instances, using both classical and f_3–elastic versions.

5.2 Capacited Arc Routing

5.2.1 Benchmark Sets

We focus on four `Capacitated Arc Routing` (CARP) benchmark sets that are publicly available on-line (see `www.uv.es/~belengue/carp.html`): `gdb`, `kshs`, `val` (also called `bccm`) and `egl`. All these instances are defined using integer data; the capacity C ranges from from 5 to 305 (with very small C only for `gdb` instances). Using the scaling operation from the first paragraph of Section 5, we generate for each such instance a scaled (large-range) version with 1000-fold increased weights and capacities. The CARP instances vary substantially in size: n ranges from 11 (`gdb19`) to 190 (`egl-s*`) and $|V|$ ranges from 7 (`gdb14` and `gdb15`) to 140 (`egl-s*`). Existing CG work [17, Table 6] show, broadly speaking, that: (i) `gdb` and `kshs` instances can be solved within a few seconds, (ii) the `val` instances require 10-20 times more running time and (iii) the `egl` instances requires 100-300 more time then `gdb` or `kshs`. Based on this, we impose the

following time limits to IRM: 100 seconds to `gdb` and `kshs` (rarely reached in practice), 500 seconds to `val`, and 5000 seconds to `egl`.

The number of vehicles is not taken into account in our IRM model (4.8), p. 22. It would be possible to add a fleet size primal constraint and a corresponding dual variable in the sub-problem algorithm, but we do not think this is essential for evaluating the IRM. To focus on the essential IRM aspects, please accept the use of an unlimited fleet. This relaxes the original (primal) problem, and so, the optimum of our model can be smaller that the optimum of other models that do enforce a fleet size constraint. Benchmark sets `val` and `egl` provide several instance classes associated to the same graph: they are referred to, in increasing order of their fleet size, as A, B, C, and (or) D. In each case, we only report results for the instance class with the largest fleet size (C or D), so as to (try to) reduce the impact of relaxing this fleet constraint.

5.2.2 Numerical Results and Comparison with Existing CG Results

The CARP results are provided on the next two pages, first for scaled instances (Table 3, see suffix "$_{\text{scl}}$" for "scaled") and then for unscaled instances (Table 4). Both tables use the following format: the first three columns provide instance information (instance name, $n = |E_R|$ and $|V|$), Column 4 reports the *computing effort* (under the format described by the third paragraph of Section 5) required to reach an optimality gap of 10%, Column 5 provides the *computing effort* needed to fully converge, Columns 6-7 indicate the lower and upper bounds returned by the IRM in the end. In the case of Table 4, there are some additional columns that provide: (i) two CG lower bounds "$[17]_{\text{bst}}$" (best CG optimum in columns E and NE in Tables 1-4) and "$[20]_{\text{bst}}$" (best CG optimum in columns ER and NER in Tables 1-4) in the next-to-last column and (ii) the IP optimum when available (or an upper bound marked with a star) in the last column (see `logistik.bwl.uni-mainz.de/benchmarks.php`).

We comment below these CARP results, by picking out the key claims along the way in boldface.

IRM can tackle *scaled* instances with 1000-fold larger capacities then standard-size instances used so far in CG CARP work. The 1000-fold capacity increase might easily lead to serious slowdowns in other DP pricing routines in classical CARP CG. This comes from the fact that all elements of $[0..1000C] \times V_R$ can generate a state in the DP CG pricing schemes we are aware of (see Section 4.2.3.3).

IRM provides reasonable lower bounds before fully converging. Indeed, IRM could be used to produce lower bounds even if the full convergence (either for CG or IRM) takes too long, *i.e.*, on instances of very large size in both $|V|$ and C. For example, in less than 30 minutes, IRM reached an optimality gap of less than 10% on `egl-e1-C`$_{\text{scl}}$ (with $n = 51$, $|V| = 77$ and $C = 160000$!). To compare these lower bounds of large-scale instances (Table 3) to their (yet-unknown) IP optimum, one can refer to the IP optimum of standard-size instances (last columns in Table 4). Under reasonable route length conditions (see the second paragraph of Section 5), the set of valid routes does not change by scaling.

On standard-size `gdb` and `kshs` instances, the running time of IRM is generally comparable to that of other CG methods from the literature, *i.e.*, about one second or less—see also the NE-S (non-elementary sparse) column in [17, Table 6]. However, as the graph size increases (*i.e.*, `val` and `egl` instances have $|V| > 30$), the IRM running time increases more rapidly than the typical running time of classical CG. This is rather due to the intersection sub-problem: as discussed at the end of Section 4.2.3.3, the CG pricing routines are usually faster in terms of complexity factors $|V|$ or $|E|$.

The number of IRM iterations is similar in large-range and standard-size instances, but the large-range version requires 2-10 more CPU time. Such CPU time increases come from a sub-problem algorithm slowdown, which is rather due to the noise adjustment introduced by scaling than to the 1000-fold weight increase itself. Indeed, by introducing this noise, certain equivalent routes in the unscaled version can be discriminated in the scaled version, *i.e.*, they no longer have exactly the same weight. This implicitly causes more state updates in the IRM sub-problem DP. In fact, Algorithm 4 (Appendix B) can waste time "rediscovering" states that only differ by small noise weight differences. Each rediscovery of a state s can trigger the rediscovery of numerous other states that are generated from s.

Instance Name	n	$\|V\|$	IRM *Computing Effort* (`iters (discr)/time`) Gap 10%	Full converg.	Final IRM bounds *lb*	*ub*
gdb1$_{scl}$	22	12	76(0)/0.6	133(5)/2.5	284	284
gdb2$_{scl}$	26	12	73(0)/1.0	119(2)/1.5	313	313
gdb3$_{scl}$	22	12	41(0)/0.2	134(4)/1.7	250	250
gdb4$_{scl}$	19	11	61(0)/0.3	85(3)/0.6	270	270
gdb5$_{scl}$	26	13	84(0)/1.1	126(4)/2.9	359	359
gdb6$_{scl}$	22	12	55(0)/0.3	103(3)/1.0	282	282
gdb7$_{scl}$	22	12	523(0)/1.1	538(1)/1.7	291	291
gdb8$_{scl}$	46	27	163(1)/57	time out	317	328
gdb9$_{scl}$	51	27	147(0)/74	time out	274	298
gdb10$_{scl}$	25	12	17(0)/0.1	49(2)/0.3	254	254
gdb11$_{scl}$	45	22	33(0)/1.2	149(3)/28.0	364	364
gdb12$_{scl}$	23	13	63(0)/0.7	101(8)/1.5	445	445
gdb13$_{scl}$	28	10	58(0)/0.7	146(6)/4.0	526	526
gdb14$_{scl}$	21	7	18(0)/0.1	68(6)/0.4	99	99
gdb15$_{scl}$	21	7	13(0)/0.1	40(0)/0.4	57	57
gdb16$_{scl}$	28	8	30(0)/0.3	89(3)/0.8	122	122
gdb17$_{scl}$	28	8	18(0)/0.2	88(7)/1.6	86	86
gdb18$_{scl}$	36	9	30(0)/0.7	72(0)/3.0	159	159
gdb19$_{scl}$	11	8	10(0)/0.05	14(0)/0.09	53	53
gdb20$_{scl}$	22	11	25(1)/0.2	71(3)/2.8	114	114
gdb21$_{scl}$	33	11	28(0)/0.3	136(3)/1.9	152	152
gdb22$_{scl}$	44	11	39(0)/0.5	152(4)/3.4	197	197
gdb23$_{scl}$	55	11	79(0)/1.5	261(7)/6.8	233	233
kshs1$_{scl}$	15	8	52(0)/2.1	103(10)/6.6	13553	13553
kshs2$_{scl}$	15	10	58(0)/1.6	101(10)/5.4	8681	8681
kshs3$_{scl}$	15	6	13(0)/0.2	63(7)/0.8	8498	8498
kshs4$_{scl}$	15	8	17(0)/0.4	47(7)/1.5	11297	11297
kshs5$_{scl}$	15	8	46(0)/0.8	113(10)/2.9	10358	10358
kshs6$_{scl}$	15	9	15(0)/0.3	104(8)/38.1	9213	9213
val1c$_{scl}$	39	24	93(0)/23	204(6)/152	225	225
val2c$_{scl}$	34	24	100(1)/42	159(7)/231	453	453
val3c$_{scl}$	35	24	141(1)/36	188(5)/143	131	131
val4d$_{scl}$	69	41		time out	414	1621
val5d$_{scl}$	65	34		time out	443	848
val6c$_{scl}$	50	31	145(0)/97	time out	295	296
val7c$_{scl}$	66	40		time out	264	414
val8c$_{scl}$	63	30		time out	426	557
val9d$_{scl}$	92	50		time out	298	790
val10d$_{scl}$	97	50		time out	426	1480
egl-e1-C$_{scl}$	51	77	98(2)/1755	time out	5043	5300
egl-e2-C$_{scl}$	72	77		time out	7345	8193
egl-e3-C$_{scl}$	87	77		time out	8046	10276
egl-e4-C$_{scl}$	98	77		time out	7972	13322
egl-s1-C$_{scl}$	75	140		time out	7324	8252
egl-s2-C$_{scl}$	147	140		time out	6006	67850
egl-s3-C$_{scl}$	159	140		time out	3380	78561
egl-s4-C$_{scl}$	190	140		time out	4187	92326

Table 3: IRM results for scaled (large-scale capacity) CARP instances. The values *lb* and *ub* represent final lower and upper bounds at the end of the search (they are equal unless IRM times out).

Instance Name	n	$\|V\|$	IRM *Computing Effort* (iters (discr)/time) Gap 10%	Full Conv	Final IRM bounds *lb*	*ub*	OPT_{CG}[a] $[17]_{\text{bst}}$, $[20]_{\text{bst}}$	OPT_{IP} (best IP*)
gdb1	22	12	71(0)/0.4	125(4)/1.7	284	284	285, 288	316
gdb2	26	12	51(0)/0.2	95(3)/0.6	313	313	314, 318	339
gdb3	22	12	61(0)/0.2	146(5)/1.3	250	250	250, 254	275
gdb4	19	11	66(0)/0.5	90(3)/0.7	270	270	272, 272	287
gdb5	26	13	91(0)/0.6	137(5)/1.8	359	359	359, 364	377
gdb6	22	12	85(0)/0.5	134(3)/1.2	282	282	284, 287	298
gdb7	22	12	221(0)/0.3	261(3)/1.4	291	291	293, 293	325
gdb8	46	27	188(0)/60	time out	320	327	330, 331	348
gdb9	51	27	204(0)/98	time out	271	297	294, 294	303
gdb10	25	12	16(0)/0.1	50(0)/0.2	254	254	254, 256	275
gdb11	45	22	32(0)/0.6	126(2)/6.2	364	364	364, 368	395
gdb12	23	13	63(0)/0.4	101(8)/1.0	445	445	444, 445	458
gdb13	28	10	51(0)/0.2	178(4)/3.0	526	526	525, 526	536
gdb14	21	7	17(0)/0.04	61(3)/0.2	99	99	98, 100	100
gdb15	21	7	12(0)/0.03	39(0)/0.1	57	57	56, 58	58
gdb16	28	8	29(0)/0.1	75(5)/0.3	122	122	122, 122	127
gdb17	28	8	20(0)/0.1	79(3)/0.5	86	86	85, 87	91
gdb18	36	9	36(0)/0.4	60(0)/0.7	159	159	159, 164	164
gdb19	11	8	10(0)/0.03	14(0)/0.07	53	53	55, 55	55
gdb20	22	11	23(0)/0.1	85(3)/3.4	114	114	114, 114	121
gdb21	33	11	25(0)/0.1	149(5)/1.0	152	152	151, 152	156
gdb22	44	11	39(0)/0.2	168(8)/1.7	197	197	196, 197	200
gdb23	55	11	76(0)/0.7	268(7)/2.8	233	233	233, 233	233
kshs1	15	8	52(0)/0.4	103(10)/2.7	13553	13553	13553, 13876	14661
kshs2	15	10	58(0)/0.5	101(10)/2.7	8681	8681	8723, 8929	9863
kshs3	15	6	12(0)/0.03	63(7)/0.1	8498	8498	8654, 8614	9320
kshs4	15	8	17(0)/0.1	47(7)/0.5	11297	11297	11498, 11498	11498
kshs5	15	8	46(0)/0.1	113(10)/0.7	10358	10358	10370, 10370	10957
kshs6	15	9	15(0)/0.05	104(8)/24.2	9213	9213	9232, 9345	10197
val1c	39	24	93(0)/19	193(6)/205	225	225	239[b] 235	245
val2c	34	24	91(1)/30	140(7)/169	453	453	457[b], 457	457
val3c	35	24	104(1)/28	146(5)/97	131	131	131[b], 131	138
val4d	69	41		time out	418	1164	499[b], 500	530
val5d	65	34		time out	442	819	546[b], 547	575
val6c	50	31	158(0)/86	257(5)/459	296	296	298[b], 299	317
val7c	66	40		time out	265	394	304[b], —	334
val8c	63	30		time out	431	551	502[b], —	521
val9d	92	50		time out	299	612	357[b], —	388
val10d	97	50		time out	434	1340	486[b], —	525
egl-e1-C	51	77	115(1)/1382	time out	5110	5275	5472, 5473	5595
egl-e2-C	72	77		time out	7199	8138	8187, 8188	8335
egl-e3-C	87	77		time out	8725	10035	10086, 10086	10292*
egl-e4-C	98	77		time out	7621	13093	11416, 11411	11562*
egl-s1-C	75	140	196(2)/4897	time out	7498	8169	8423, 8247	8518
egl-s2-C	147	140		time out	7086	65255	16262, 16262	16425
egl-s3-C	159	140		time out	3380	78561	17014, 17014	17188
egl-s4-C	190	140		time out	4187	92326	20197, 20144	20481*

Table 4: IRM results for standard CARP. The CG optima from next-to-last columns indicate the best value of all CG bounds from [17] and [20]. Each CG model defines its feasible routes in slightly different manners.[a]

[a]Some pricing schemes allow non-elementary (NE) routes and some not, others apply 2-cycle eliminations, different domination criteria or various strengthening methods. Our IRM sub-problem (Appendix B) allows non-elementary routes, eliminates 2-cycles and forbids the repetitive use of the same service segment. An important particularity of our model is the use of an unlimited fleet size (relaxed primal constraint). This makes the IRM bound of (slightly) lower quality than the CG bound based on pricing without NE routes, but one can make no absolute comparison with regards to CG with NE routes.

[b]For these 10 instances, we subtracted the following offset from the values reported in [17]: 74, 71, 24, 122, 143, 107, 103, 136, 127, and respectively 209—see the "offset" column in the corresponding tables at `logistik.bwl.uni-mainz.de/benchmarks.php`.

5.3 General Experimental Conclusions

Let us insist on a few IRM advantages that show up across several numerical tests and problems. Generally speaking, notice we tried to address problems with rather diverse features, *e.g.*, there is no restriction on the column costs (they have large variations in `CARP`) or on the covering demands $\mathbf{b}$ (they vary in `Cutting-Stock`).

(1) On large-range (scaled) instances, IRM is never less effective than CG, especially when the 1000-fold weight increase (scaling) renders the separation sub-problem prohibitively-hard for CG. Indeed, the IRM was able to solve certain large-range instances for which conventional CG could take excessively long time, *e.g.*, see the IRM-CG experimental comparison on large-range versions of f_2-`Elastic` and f_3-`Elastic Cutting-Stock` (Section 5.1), or also the large-range `Capacitated Arc Routing` results (Section 5.2). Generally speaking, the IRM running time stays in the same order of magnitude for standard-size and large-range input—compare Tables 1 and 2 (Section 5.1), or Tables 3 and 4 (Section 5.2).

(2) On standard-size instances, the results of IRM are on a par with those of CG. For instance, the IRM solves certain pure `Cutting Stock` instances within a similar time as CG with `Minknap` pricing. One should be aware that `Minknap` is actually one of the most competitive knapsack algorithms from a very vast literature. IRM solves the intersection sub-problems using a rather ad-hoc implementation of the $\mathbf{r}$-indexed (profit-indexed) DP. In terms of the number of iterations, the comparison between CG and IRM can be described as balanced (both in `Cutting-Stock` and `Arc-Routing`).

(3) IRM offers intermediate lower bounds $\mathbf{b}^\top \mathbf{x}_{\text{lb}}$ as a built-in component, while CG only offers an option (not systematically used) to determine Lagrangean intermediate lower bounds. Coupled with upper bounding techniques, the IRM can generally reach an optimality gap of 10% using about 25% of its total convergence time for roughly half of the tested instances (of either problem, see columns "10% Gap" in Tables 2 and 4). At best, the IRM lower bound $\mathbf{b}^\top \mathbf{x}_{\text{lb}} = \mathbf{b}^\top (t^* \mathbf{r})$ can even be nearly optimal *from the first iterations*. Indeed, certain `Bin-Packing` instances (*e.g.*, the triplets) have an optimal dual solution of the form $\mathbf{x}^* = t^* \mathbf{b}$ ($\mathbf{b}$ is $\mathbf{1}_n$ in `Bin-Packing`) that is "hit" at the very first iteration with $\mathbf{r} = \mathbf{b}$;

Finally, let us emphasize the main reasons lying behind the success of IRM over CG on large-range instances. The most important IRM performances are rather due to the rapidity of the sub-problem algorithm than to the number of iterations—which usually stays in the same order of magnitude for both methods. This rapidity comes from the fact that IRM can more easily provide very small $\mathbf{r}$ values as input to the sub-problem, speeding-up the $\mathbf{r}$-indexed DP (Section 3.2). Indeed, if $\mathbf{r}$ is smaller than $\mathbf{w}$, the number of states in $\mathbf{r}$-indexed DP is naturally smaller than in $\mathbf{w}$-indexed DP. For instance, when $\mathbf{r} = \mathbf{b} = \mathbf{1}_n$ in `Bin-Packing`, the sub-problem becomes equivalent to solving a knapsack problem with equal profits, which is very easy (regardless of the weight range). As long as $\mathbf{r}$ is generally smaller than $\mathbf{x}$, IRM can usually solve its sub-problems more rapidly. Also, IRM does not generally need more than 5 discretization refining operations throughout the search (see Tables 1-4), which is enough to keep $\mathbf{r}$ at relatively low values at most times, *i.e.*, $\mathbf{r}$ contains values that are usually no larger than $2^5 = 32$, less than typical weight values (that can easily exceed 1000 in many cases, see Table 5, p. 38 for the `Cutting-Stock` instances).

6 Conclusions

We proposed a ray projection approach for optimizing (primal or dual) LPs in which the feasible area is a polytope $\mathcal{P}$ with prohibitively many constraints. Such LPs are commonly optimized by cutting-planes (if $\mathcal{P}$ is a primal) or CG (if $\mathcal{P}$ is a dual). In these methods, one uses a separation sub-problem to gradually refine an outer polytope $\mathcal{P}_\mathcal{A} \supset \mathcal{P}$, so as to iteratively converge a sequence of dual bounds $\text{OPT}(\mathcal{P}_\mathcal{A})$. Our Integer Ray Method (IRM) relies on an *intersection sub-problem* that generalizes the separation sub-problem: given a (feasible or infeasible) solution $\mathbf{r}$, find the maximum t^* such that $t^* \mathbf{r} \in \mathcal{P}$. By iterating over such rays $\mathbf{r}$, the IRM evolves as a *primal method* (see Section 1.1.1.1), *i.e.*, it relies on a sequence of built-in primal bounds $\mathbf{x}_{\text{lb}} = t^* \mathbf{r}$ that converges to the optimum.

The IRM was tested on various `Set-Covering` dual LPs that are typically solved by CG. The conclusions of the experimental tests (Section 5.3) confirm the practical interest in the approach, especially on large-range instances on which the classical CG sub-problem seems prohibitively-hard.

Last but not least, the application of the IRM to a new LP would only require providing two routines: (i) a routine for the intersection sub-problem and (ii) a routine indicating the number of variables n, along with a feasible interval for each variable. This applies both to the theoretical design of Algorithm 1 and to the implementation (publicly available on-line, see Footnote 4, p. 23). We hope this paper can shed useful light for solving other primal LPs such as (1.1) or dual LPs such as (3.1). A direct continuation of this work consists of applying IRM in a primal context, by implementing the above two routines for a primal LP.

Acknowledgements We thank François Clautiaux, Cédric Joncour, Ibrahima Diarrassouba, Enrico Malaguti, Adam Letchford, Marco Lübbecke for answering various questions (by mail or face to face); this was very useful in clarifying Section 1.1. We thank two anonymous referees for their comments that helped on two aspects: (i) the context of the method in the literature, and (ii) the insight analysis of the convergence proof.

References

[1] K. Aardal, G. Nemhauser, and R. Weismantel. Discrete Optimization, volume 12 of Handbooks in Operations Research and Management Science. Elsevier, 2005.

[2] L. Alfandari, J. Sadki, A. Plateau, and A. Nagih. Hybrid column generation for large-size covering integer programs: Application to transportation planning. Computers & OR, 40(8):1938–1946, 2013.

[3] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance. Branch-and-price: Column generation for solving huge integer programs. Operations research, 46(3):316–329, 1998.

[4] E. Bartolini, J.-F. Cordeau, and G. Laporte. Improved lower bounds and exact algorithm for the capacitated arc routing problem. Mathematical Programming, 137(1-2):409–452, 2013.

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

[6] H. Ben Amor and J. M. V. de Carvalho. Cutting stock problems. In G. Desaulniers, J. Desrosiers, and M. Solomon, editors, Column Generation, pages 131–161. Springer US, 2005.

[7] O. Briant, C. Lemaréchal, P. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck. Comparison of bundle and classical column generation. Mathematical Programming, 113(2):299–344, 2008.

[8] F. Clautiaux, C. Alves, and J. Carvalho. A survey of dual-feasible and superadditive functions. Annals of Operations Research, 179(1):317–342, 2009.

[9] F. Clautiaux, C. Alves, J. M. V. de Carvalho, and J. Rietz. New stabilization procedures for the cutting stock problem. INFORMS Journal on Computing, 23(4):530–545, 2011.

[10] J. Fonlupt and A. Skoda. Strongly polynomial algorithm for the intersection of a line with a polymatroid. In W. Cook, L. Lovász, and J. Vygen, editors, Research Trends in Combinatorial Optimization, pages 69–85. Springer Berlin Heidelberg, 2009.

[11] R. Fukasawa, H. Longo, J. Lysgaard, M. P. de Aragão, M. Reis, E. Uchoa, and R. F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Mathematical programming, 106(3):491–511, 2006.

[12] P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem. Operations Research, 9:849–859, 1961.

[13] J. Gondzio, P. González-Brevis, and P. Munari. New developments in the primal–dual column generation technique. European Journal of Operational Research, 224(1):41 – 51, 2013.

[14] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. Journal of the ACM, 22(4):463–468, 1975.

[15] S. Irnich and D. Villeneuve. The shortest-path problem with resource constraints and k-cycle elimination for k 3. INFORMS Journal on Computing, 18(3):391–406, 2006.

[16] K. Kaparis and A. N. Letchford. Separation algorithms for 0-1 knapsack polytopes. Mathematical programming, 124(1-2):69–91, 2010.

[17] A. Letchford and A. Oukil. Exploiting sparsity in pricing routines for the capacitated arc routing problem. Computers & Operations Research, 36:2320–2327, 2009.

[18] A. N. Letchford and A. Lodi. Primal cutting plane algorithms revisited. Math. Meth. of OR, 56(1):67–81, 2002.

[19] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. Operations Research, 53(6):1007–1023, 2005.

[20] R. Martinelli, D. Pecin, M. Poggi, and H. Longo. Column generation bounds for the capacitated arc routing problem. In XLII SBPO, 2010.

[21] S. T. McCormick. Submodular function minimization. In G. N. K. Aardal and R. Weismantel, editors, Discrete Optimization, volume 12 of Handbooks in Operations Research and Management Science, pages 321 – 391. Elsevier, 2005.

[22] K. Nagano. A strongly polynomial algorithm for line search in submodular polyhedra. Discrete Optimization, 4(34):349 – 359, 2007.

[23] D. Pisinger. Where are the hard knapsack problems? Computers & Operations Research, 32(9):2271 – 2284, 2005.

[24] B. Spille and R. Weismantel. Primal integer programming. In Discrete Optimization [1], pages 245–276.

[25] F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. Mathematical Programming, 86(3):565–594, 1999.

[26] F. Vanderbeck. A nested decomposition approach to a three-stage, two-dimensional cutting-stock problem. Management Science, 47(6):864–879, June 2001.

[27] F. Vanderbeck. Implementing mixed integer column generation. In G. Desaulniers, J. Desrosiers, and M. Solomon, editors, Column Generation, pages 331–358. Springer, 2005.

[28] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. European Journal of Operational Research, 183(3):1109 – 1130, 2007.

A A Small LP Optimized by Iterating over Rays

We here illustrate the convergence process the LP from Figure 1, copied here to ease the reading. Importantly, the ray choices from this example do not accurately represent the real IRM choices: the ray generator is voluntarily simplified, as we only want to illustrate the general idea of iterating over rays rather then the details of our ray generator.

Step 1 IRM starts out with $\mathbf{r}_1 = \mathbf{b} = [1\ 1]$. The intersection sub-problem leads to:

- $t^* = 5$;
- $c_a = 15$ and $\mathbf{a} = [2\ 1]$, *i.e.*, $2x_1 + x_2 \leq 15$;
- $\mathbf{x}_{\text{lb}} = [5, 5]$;
- $\mathbf{x}_{\text{ub}} = [0\ 15]$, *i.e.*, the optimal solution over the polytope delimited by $\mathbf{x} \geq \mathbf{0}_n$ and above "first-hit" constraint $2x_1 + x_2 \leq 15$.

Step 2 As hinted above, the ray generator from this example is voluntary simplified[a] and we consider that the next ray is $\mathbf{r}_{\text{new}} = \mathbf{x}_{\text{ub}} = [0\ 15]$. After solving the intersection sub-problem, we obtain:

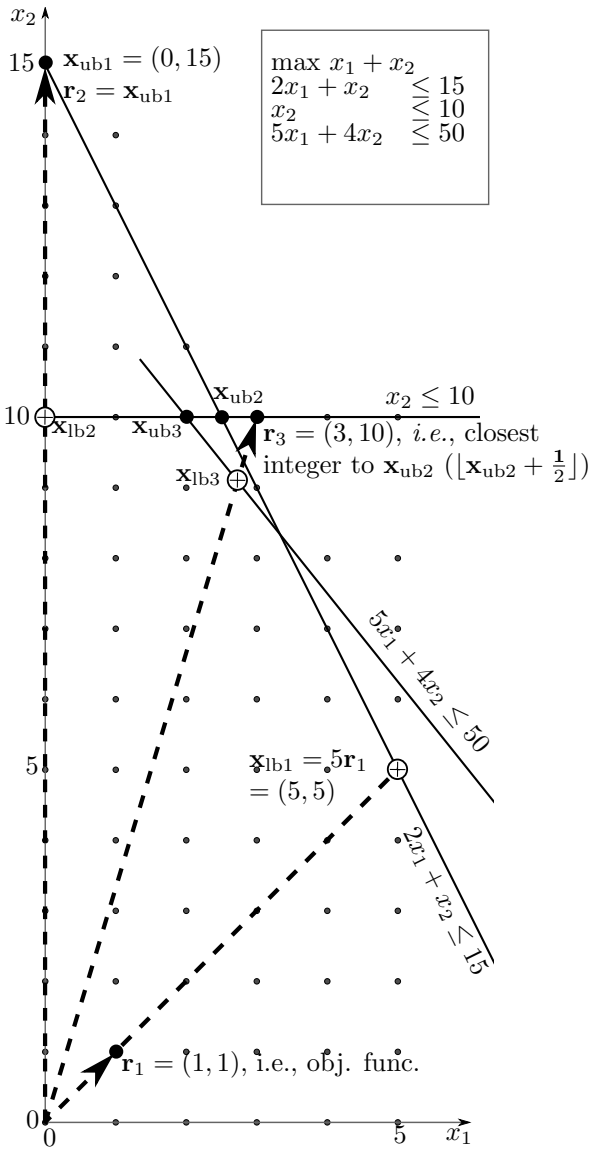
- $t^* = 10/15$;
- $c_a = 10$ and $\mathbf{a} = [0\ 1]$, *i.e.*, $x_2 \leq 10$;
- $\mathbf{x}_{\text{lb}} = [0, 10]$,
- $\mathbf{x}_{\text{ub}} = [2.5, 10]$, *i.e.*, the above first-hit constraint separates the old $\mathbf{x}_{\text{ub}} = [0\ 15]$, thus updating $\mathbf{x}_{\text{ub}} = [2.5, 10]$. In the figure, we observe that $\mathbf{x}_{\text{ub1}}$ advances towards $\mathbf{x}_{\text{ub2}}$.

Step 3 IRM could take $\mathbf{r}_{\text{new}} = [3\ 10]$, using a rounding of the form $\lfloor \mathbf{x}_{\text{ub}} + \frac{1}{2}\mathbf{1}_n \rfloor$. The intersection sub-problem leads to:

- $t^* = 50/55 = 10/11$;
- $c_a = 50$ and $\mathbf{a} = [5\ 4]$, *i.e.*, $5x_1 + 4x_2 \leq 50$;
- $\mathbf{x}_{\text{lb}} = [\frac{30}{11}, \frac{100}{11}]$;
- $\mathbf{x}_{\text{ub}} = [2\ 10]$.

Observe that $\mathbf{b}^\top \mathbf{x}_{\text{lb}} = \frac{130}{11} \approx 11.78$ and that $\mathbf{b}^\top \mathbf{x}_{\text{ub}} = 12$, *i.e.*, the gap is very small.

Step 4 IRM could consider $\mathbf{r} = \mathbf{x}_{\text{ub}} = [2\ 10]$, solve the intersection sub-problem to find $\mathbf{x}_{\text{lb}} = \mathbf{x}_{\text{ub}} = \text{OPT}(\mathcal{P}) = [2\ 10]$.



[a] The ray generator from Algorithm 2 (Section 2.3.1) would search for new rays by rounding points of the form $\mathbf{r} + \beta(\mathbf{x}_{\text{ub}} - \mathbf{x}_{\text{lb}})$, with $\beta < \frac{1}{t^*}$. Technically, Algorithm 2 tries to locate new rays on the segment joining $\mathbf{r} = [1\ 1]$ and $\mathbf{r} + \frac{1}{5}([0\ 15] - [5\ 5])$, *i.e.*, it advances from $[1\ 1]$ in the direction of $[-5\ 10]$. This could yield rays $[1\ 2]$ and $[0\ 2]$.

B Algorithm for the CARP Intersection Sub-problem

The states from Section 4.2.3 are recorded as follows. For a given $v \in V$, a list data structure `states`$[v]$ keeps all states (p_s, c_s, v) sorted in a lexicographic order, *i.e.*, first ordered by profit and then by cost. In fact, for each integer profit $p > 0$, one can access all states (p, c_s, v) as if they were recorded in a separate sublist

corresponding to a fixed profit p. Since all existing instances use integer edge lengths, we can consider that the deadheading traversal costs c_s are also integer (they are sums of edge lengths). This is not a theoretical IRM requirement, but it is useful for reducing the number of states in practice.

Algorithm 4 IRM Sub-problem Algorithm for CARP

Require: ray $r \in \mathbb{Z}^n$, $G = (V, E)$, required edges E_R, cost $c(u, v)$ $\forall u, v \in V$, weights w, and capacity C
Ensure: minimum cost/profit ratio t and an associated route a that yields this optimum t value

1: $\texttt{states}[v_0] \leftarrow \{(0, 0, v_0)\}$ ▷ null state: no move, 0 profit, 0 dead-heading cost
2: **do**
Stage 1: Build service segments by chaining serviced edges $\{u, v\}$: increase profits, keep costs fixed
3: **repeat**
4: **for** $(u \in V,\ (p, c, u) \in \texttt{states}[u])$ **do** ▷ For each segment ending in u,
5: **for** $\{u, v\} \in E_R$ **do** ▷ augment service with $\{u, v\}$
6: $newW \leftarrow W(p, c, u) + w(u, v)$
7: **if** $newW \leq C$ **then**
8: $oldW \leftarrow W(p + r(u, v), c, v)$ ▷ W returns ∞ if $(p + r(u, v), c, v) \notin \texttt{states}[v]$
9: $\texttt{states}[v] \leftarrow \texttt{states}[v] \cup \{(p + r(u, v), c, v)\}$
10: $W(p + r(u, v), c, v) \leftarrow \min(newW, oldW)$ ▷ State update
11: **end if**
12: **end for**
13: **end for**
14: **until** {*no state update in the last iteration*} ▷ no W improvement in Line 10 ($newW \geq oldW$)
Stage 2: Add segment transfers from u to v: keep profits fixed, increase costs
15: **for** $u, v \in V$(with $u \neq v$) **do**
16: **for** $(p, c, u) \in \texttt{states}[u]$ **do**
17: $newW \leftarrow W(p, c, u)$ ▷ total weight (load) uninfluenced by deadheading
18: $oldW \leftarrow W(p, c + c(u, v), v)$ ▷ W returns ∞ if $(p, c + c(u, v), v) \notin \texttt{states}[v]$
19: $\texttt{states}[v] \leftarrow \texttt{states}[v] \cup \{(p, c + (u, v), v)\}$
20: $W(p, c + c(u, v), v) \leftarrow \min(newW, oldW)$ ▷ State Update
21: **end for**
22: **end for**
Stage 3: Check for better (lower) cost/profit ratios
23: **for** $(p, c, v_0) \in \texttt{states}[v_0]$ **do** ▷ complete routes ending in v_0
24: $t_{\text{ub}} = \min(t_{\text{ub}}, \frac{c}{p})$
25: **end for**
Stage 4:Prune: (A) old states, not generated from last Stage 1 or 2, or (B) states with large costs
26: **for** $(v \in V,\ (p, c, v) \in \texttt{states}[v])$ **do**
27: $wRes = C - W(p, c, v)$
28: $pMaxRes = \texttt{pMaxResidual}(wRes)$ ▷ profit bound using capacity $wRes$
29: **if** $\left(W(p, c, v)$ not updated in Line 10 or 20 OR $\frac{c}{p + pMaxRes} \geq t_{\text{ub}}\right)$ **then**
30: $\texttt{states}[v] \leftarrow \texttt{states}[v] - \{(p, c, v)\}$
31: **end if**
32: **end for**
33: **while** $\texttt{states} \neq \emptyset$
34: **return** t_{ub} ▷ An associated route is reconstructed by following precedence relations between states

The (rather self-explanatory) pseudocode is provided in Algorithm 4 and commented upon in the subsequent paragraphs. We emphasize the following key stages:

Stage 1: build continuous service segments Any required edge $\{u, v\} \in E_R$ can be used to augment (continue) a service segment ending in $u \in V$. Each ***repeat-until*** *iteration* of Stage 1 (Lines 3-14)

considers all such segments (in Line 4) and tries to augment them to higher weights (increased load) by servicing any edge $\{u, v\} \in E_R$ (in Line 5). When all such service increases lead to higher weights exceeding C, the Stage 1 ends by breaking the ***repeat-until*** *loop* with the condition in Line 14. Also, at each new update of $s \in$`states`$[v]$, we record the state $(p, c, u) \in$`states`$[u]$ and the edge (*i.e.*, $\{u, v\}$) that led to s. We thus build a network of precedence relations between states that is later useful to: (i) reconstruct an actual optimal route $\mathbf{a}$ in the end, (ii) eliminate 2-cycles (U-turns).[5]

Stage 2: continue with transfer segments For any $u, v \in V$ $(u \neq v)$, a single transfer $u \to v$ may be required to link a service segment ending in u to a service segment starting in v. This transfer can only increase the dead-heading cost, but not the weight (load). By chaining Stage 1 and 2, one can generate any sequence of service and dead-heading segments. The null service (no move) is added in Line 1 to allow Stage 2 to discover routes that actually directly start by dead-heading.

Stage 3: optimum update Check for better cost/profit ratios on complete routes (ending at v_0) in Lines 23–25.

Stage 4: prune useless states Remove as many states as possible so as to break the main `do-while` loop (Lines 2-33) as soon as possible by triggering the condition in Line 33 (*i.e.*, no state left). Given $v \in V$, a state $(p, c, v) \in$`states`$[v]$ can be removed in one of the following cases:

(A) (p, c, v) was not updated by the last call of Stage 1 or Stage 2. This indicates that state (p, c, v) has already generated other states in Stage 1 or 2 and any future use becomes redundant. However, the value $W(p, c, v)$ is recorded because it can still be useful: if state (p, c, v) is re-discovered later, there is no use to re-insert it—unless the rediscovered weight is smaller than $W(p, c, v)$.

(B) the cost c is so high that the state (p, c, v) can only lead to ratios cost/profit thare are guaranteed to stay above the current upper bound t_{ub}. For instance, if $t_{\text{ub}} = 0$ and $c > 0$, all states can be removed and the algorithm finishes very rapidly—at the second call of Line 33, after removing already-existing states using (A).

The only external routine is `pMaxResidual(`$wRes$`)` in Stage 4, Line 28. It returns the maximum profit realized by a route ending at the depot using a capacity of no more then $wRes$ (for any starting point). We used the following upper bound for this profit: ignore the dead-heading costs (the vehicle can directly "jump" from one vertex to another when necessary) and obtain a classical knapsack problem with capacity $wRes$, weights $\mathbf{w}$ and profits $\mathbf{r}$. This upper bound can be calculated once (*e.g.*, via DP) for all realizable values $wRes \in [0..C]$ in a pre-processing stage.

C Intermediate Lagrangean Bounds in Column Generation

To (try to) make the paper self-contained, let us discuss in greater detail the Lagrangean CG lower bounds mentioned throughout the paper. The numerical `Cutting-Stock` experiments from Section 5.1.3 rely on the Farley bound, which is one of the specializations of the Lagrangean CG bound. While such CG bounds are a very general CG tool, their calculation can be different from problem to problem. To ease the exposition, we recall the main `Set-Covering` CG model from Section 3 and we present below both the primal (left) and the dual (right) programs:

$$\mathbf{x}: \begin{array}{ll} \min \sum c_a \lambda_a & \\ \sum a_i \lambda_a \geq b_i, & \forall i \in [1..n] \\ \lambda_a \in \mathbb{R}_+ & \forall [c_a\ \mathbf{a}]^\top \in \overline{\mathcal{A}} \end{array} \qquad \lambda: \begin{array}{ll} \max \mathbf{b}^\top \mathbf{x} & \\ \mathbf{a}^\top \mathbf{x} \leq c_a, & \forall [c_a, \mathbf{a}] \in \overline{\mathcal{A}}, \\ x_i \geq 0, & i \in [1..n] \end{array}$$

where all sums are carried out over all primal columns $[c_a, \mathbf{a}] \in \overline{\mathcal{A}}$.

We describe the Lagrangean bound on the primal (left) program, by reformulating ideas from work such as [6, § 2.2], [25, §. 3.2], [19, §. 2.1] or [7, § 1.2]. Each iteration of the CG process yields some dual values $\mathbf{x}$

[5] However, k-cycles with $k > 2$ are not detected, but this would require more computational effort; the situation is analogous to that of CG pricing methods for CARP [17, §3.1] and VRP [15]. By going more steps back on precedence relations, we could avoid longer cycles, but this study is mainly focused on IRM and not on cycle elimination.

that we use as Lagrangean multipliers *to relax* the primal set-covering constraints $\sum a_i\lambda_a \geq b_i$. More exactly, for any such fixed $\mathbf{x}$, the Lagrangean bound $\mathcal{L}_\mathbf{x}$ is simply obtained from the primal by introducing penalty x_i for violating $\sum a_i\lambda_a \geq b_i$. Technically, the primal becomes: $\mathcal{L}_\mathbf{x} = \min_\lambda \left(\sum c_a\lambda_a + \mathbf{x}^\top(\mathbf{b} - \sum \mathbf{a}\lambda_\mathbf{a})\right)$, where all sums are carried out over all primal columns $[c_a, \mathbf{a}] \in \overline{\mathcal{A}}$. After classical algebraic reformulations, we obtain the following relation between the CG optimum OPT_{CG} and $\mathcal{L}_\mathbf{x}$.

$$\mathrm{OPT}_{CG} \geq \mathcal{L}_\mathbf{x} = \min_{\lambda\geq\mathbf{0}} \left(\sum (c_a - \mathbf{a}^\top\mathbf{x})\lambda_a\right) + \mathbf{b}^\top\mathbf{x} \geq \min_{\lambda\geq\mathbf{0}} \left(m_{\mathrm{rdc}} \sum \lambda_a\right) + \mathbf{b}^\top\mathbf{x}, \tag{C.1}$$

where $m_{\mathrm{rdc}} = \min_{[c_a,\mathbf{a}]\in\overline{\mathcal{A}}} c_a - \mathbf{a}^\top\mathbf{x} \leq 0$ is the minimum reduced cost at the current iteration. Assuming that a primal upper bound u_b is known, one could add $\sum\lambda_a \leq u_b$ to the primal; as such, $\sum\lambda_a \leq u_b$ also holds in (C.1), which can that evolve to:

$$\mathrm{OPT}_{CG} \geq \mathbf{b}^\top\mathbf{x} + u_b \cdot m_{\mathrm{rdc}}, \tag{C.2}$$

We see that one requires an upper bound value u_b. This can be obtained, for instance, by classical primal heuristics. However, for pure `Cutting-Stock`, the fact that c_a is always 1 leads to $\sum\lambda_a = \sum c_a\lambda_a \leq \mathrm{OPT}_{CG}$, *i.e.*, one can use $u_b = \mathrm{OPT}_{CG}$, which would transform (C.2) into $\mathrm{OPT}_{CG} \geq \mathbf{b}^\top\mathbf{x} + \mathrm{OPT}_{CG} \cdot m_{\mathrm{rdc}}$. The Farley lower bound $\mathcal{L}^F_\mathbf{x}$ follows immediately:

$$\mathcal{L}^F_\mathbf{x} = \frac{\mathbf{b}^\top\mathbf{x}}{1 - m_{\mathrm{rdc}}} \leq \mathrm{OPT}_{CG}.$$

Since all our `Elastic Cutting-Stock` problems verify $c_a \geq 1, \forall[c_a, \mathbf{a}] \in \overline{\mathcal{A}}$, we could still use $u_b = \mathrm{OPT}_{CG}$, simply because $\sum\lambda_a \leq \sum c_a\lambda_a \leq \mathrm{OPT}_{CG}$; in this case, the above bound $\mathcal{L}^F_\mathbf{x}$ is still correct.

In any case, to exploit (C.2), one needs a primal upper bound u_b on $\sum\lambda_a$. Generally speaking, to obtain high-quality upper bounds for other problems, one might need more specific modelling work.

D `Cutting-Stock` Instances: Characteristics and References

Name	n	C	avg. **b** span	avg. **w** span	Description
`vb10`	10	10000	$[10, 100]$	$[1, \frac{1}{2}C]$	20 random instances [25]: `CSTR10b50c[1-5]*` files[a]
`vb20`	20	10000	$[10, 100]$	$[1, \frac{1}{2}C]$	25 random instances [25]: `CSTR20b50c[1-5]*` files[a]
`vb50-c1`	50	10000	$[50, 100]$	$[1, \frac{3}{4}C]$	20 random instances [25]: `CSTR50b50c1*` files[a]
`vb50-c2`	50	10000	$[50, 100]$	$[1, \frac{1}{2}C]$	20 random instances [25]: `CSTR50b50c2*` files[a]
`vb50-c3`	50	10000	$[50, 100]$	$[1, \frac{1}{4}C]$	20 random instances [25]: `CSTR50b50c3*` files[a]
`vb50-c4`	50	10000	$[50, 100]$	$[\frac{1}{10}C, \frac{1}{2}C]$	20 random instances [25]: `CSTR50b50c4*` files[a]
`vb50-c5`	50	10000	$[50, 100]$	$[\frac{1}{10}C, \frac{1}{4}C]$	20 random instances [25]: `CSTR50b50c5*` files[a]
`vb50-b100`	50	10000	$[1, 210]$	$[\frac{1}{10}C, \frac{1}{2}C]$	20 random instances [25]: `CSTR50b100c4*` files[a]
`m01`	100	100	1	$[1, C]$	1000 random `bin-packing` instances [8];
`m20`	100	100	1	$[\frac{20}{100}C, C]$	1000 random `bin-packing` instances [8];
`m35`	100	100	1	$[\frac{35}{100}C, C]$	1000 random `bin-packing` instances [8];
`Hard`	≈ 200	100000	$1 - 3$	$[\frac{20}{100}C, \frac{35}{100}C]$	Bin-packing instances, known to be more difficult
`Triplets`	15,45,75 ... 285	30000	1	$[\frac{C}{3} - \frac{n-1}{2}, \frac{C}{3} + \frac{n-1}{2}]$	The smallest 10 triplets; the optimal solution consists of $\frac{n}{3}$ bins, all filled with exactly 3 items.

Table 5: Original `Cutting-Stock` instance files. The columns "avg. **b** span" and "avg. **w** span" indicate the general interval of the demand value, and, respectively, item weights.

[a] In the archive `http://www.math.u-bordeaux1.fr/~fvanderb/data/randomCSPinstances.tar.Z`