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

Daniel Porumbel*

Abstract

A recurrent task in mathematical programming requires optimizing polytopes with prohibitively-many constraints, *e.g.*, the primal polytope in cutting-plane methods or the dual polytope in Column Generation (CG). This paper is devoted to the *ray projection* technique for optimizing such polytopes: start from a feasible solution and advance on a given ray direction until intersecting a polytope facet. The resulting intersection point is determined by solving the *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 focus on dual polytopes associated to CG: if the CG (separation) sub-problem can be solved by Dynamic Programming (DP), so can be the *intersection sub-problem*. The convergence towards the CG optimum is realized through a sequence of intersection points $t^*\mathbf{r}$ (feasible dual solutions) determined from such rays $\mathbf{r}$. Our method only uses integer rays $\mathbf{r}$, so as to render the intersection sub-problem tractable by $\mathbf{r}$-indexed DP. We show that in 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 only consists of fractional (or large-range) values. As such, the proposed method 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 that include `Variable Size Bin Packing`). We also prove the theoretical convergence of the proposed method.

1 Introduction

The optimization over polytopes with-prohibitively many constraints has a rich history in mathematical programming. In cutting-plane methods and *branch-and-cut*, one iteratively adds constraints that separate the current infeasible solution from the feasible primal polytope. 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 feasible dual polytope $\mathcal{P}$. In both cases, at each iteration, the current infeasible solution is the optimum of some "outer polytope" that contains the initial feasible polytope. The process converges through a sequence of such exterior (infeasible) solutions by progressively removing infeasibility. We propose a method that converges through a sequence of interior (feasible) solutions instead of exterior solutions. These interior solutions are actually situated on some facets of the feasible polytope, *i.e.*, each interior solution is the intersection between a ray direction and the feasible polytope boundary. The method, hereafter called the `Integer Ray Method` (IRM), is computationally very well-suited to dual formulations in CG, because the above intersections can be more easily calculated.

Let us introduce in (1.1) below the main (dual) Linear Program (LP) that is optimized by the IRM.

$$\left.\begin{array}{ll} \max \mathbf{b}^\top \mathbf{y} & \\ \mathbf{a}^\top \mathbf{y} \leq c_a, & \forall [c_a, \mathbf{a}] \in \overline{A} \\ y_i \geq l_i, & i \in [1..n] \end{array}\right\} \mathcal{P} \tag{1.1}$$

*Univ Lille Nord de France, F-59000 Lille, France
UArtois, LGI2A, F-62400, Béthune, France
Contact: daniel.porumbel@univ-artois.fr, tel: +333.21.63.72.78, fax: +333.21.63.71.21

We interpret (1.1) as the *dual* LP of the fractional relaxation of a master Integer LP (ILP) for `Set-Covering` problems. Implicitly or explicitly, (1.1) arises in many `Set-Covering` problems (see general examples below) and we will discuss in great detail `Elastic Cutting-Stock` (Section 3.1) and `Capacitated Arc Routing` (Section 3.2). However, for now it is enough to introduce the general `Set-Covering` interpretation of (1.1).

- $\mathbf{y} = [y_1\ y_2 \dots y_n]^\top$ is a vector of dual (decision) variables and $\mathcal{P}$ is the dual polytope.
- each constraint $\mathbf{a}^\top\mathbf{y} \leq c_a$ from $\overline{A}$, also denoted $[c_a, \mathbf{a}]$ (by slightly abusing notation), corresponds to 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.
- $\mathbf{b} = [b_1\ b_2 \dots b_n]^\top \in \mathbb{Z}^n_+$ indicates the (dual) objective function. It arises from the covering demands in the master LP, *i.e.*, any $i \in [1..n]$ has to be covered (serviced, packed) b_i times;

A well-known example of a dual CG program fitting this interpretation is the dual LP of the Gilmore-Gomory model for `Cutting-Stock` [10]: any $[c_a, \mathbf{a}] \in \overline{A}$ represents an integer solution (pattern) of a knapsack sub-problem ($c_a = 1$ is a constant pattern cost), b_i indicate the covering demand for 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.

The classical CG method typically iterates the following steps: (i) optimize the current outer dual polytope $\mathcal{P}_A \supset \mathcal{P}$ (*i.e.,* the dual polytope with a restricted set of constraints $A \subset \overline{A}$) to construct the current outer solution $\mathbf{y}$, (ii) solve the CG (separation) sub-problem $\max_{[c_a,\mathbf{a}]\in\overline{A}} \mathbf{a}^\top\mathbf{y} - c_a$ to find a valid dual constraint $[c_a, \mathbf{a}] \in \overline{A}$ (primal column $\left[\begin{smallmatrix} c_a \\ \mathbf{a} \end{smallmatrix}\right]$) that separates $\mathbf{y}$ from $\mathcal{P}$ (report optimality when $\mathbf{y}$ can not be separated), and (iii) update the current outer polytope $\mathcal{P}_A$ by adding the violated constraint (*i.e.,* $A \leftarrow A \cup \{[c_a, \mathbf{a}]\}$) and repeat from (i). Important progress has been done in CG over the last decades by developing innovative methods of column management and stabilization: such techniques can significantly reduce the number of CG iterations, and so, the number of pricing (sub-problem) calls—see, chronologically, the work in [1, 18, 25, 5, 7, 12]. While the IRM uses implicitly certain CG techniques (to get upper bounds for (1.1)), the effectiveness of the IRM does not reside in a reduction of the number of sub-problems, but rather in the *ray projection* technique for locating solutions on the boundary of $\mathcal{P}$.

Indeed, the main IRM tool is the ray projection: start from the dual space origin (*e.g.*, $\mathbf{0}_n$) and advance along ray direction $\mathbf{0}_n \rightarrow \mathbf{r}$ until intersecting a $\mathcal{P}$ facet. To determine the resulting "intersection point", one needs to solve the *intersection* (or IRM) sub-problem, *i.e.*, determine the *maximum feasible step length* t^* such that $t^*\mathbf{r}$ is feasible. The fact that the rays $\mathbf{r}$ can be chosen (*i.e.*, $\mathbf{r}$ is always integer) is very useful for simplifying this intersection sub-problem. For a sub-problem algorithm, it is preferable to use some controllable input data $\mathbf{r}$ instead of a rather unpredictable dual vector $\mathbf{y}$ as in the sub-problem of (unstabilized) CG methods. IRM proceeds as follows (see Figure 1):

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

(2) optimize the outer polytope $\mathcal{P}_A \supset \mathcal{P}$ determined by the (dual) constraints $A \subset \overline{A}$ already generated at (1) to find an upper bound $\mathbf{y}_{\text{ub}}$ (*i.e.*, $\mathbf{b}^\top\mathbf{y}_{\text{ub}}$ is an upper bound for the (1.1) optimum);

(3) determine a new ray (see below) and repeat from (1).

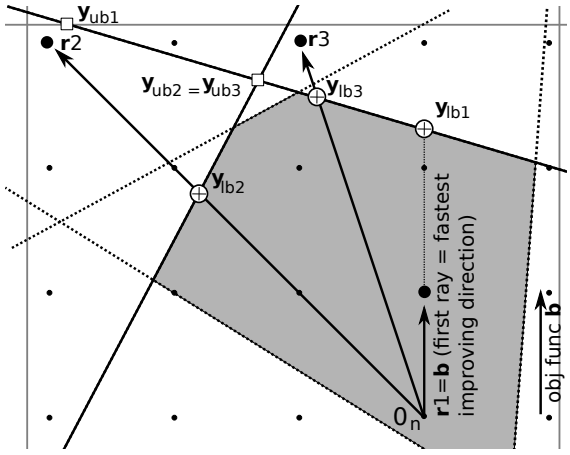

Figure 1: IRM at a glance: first ray $\mathbf{r}1$ leads to lower and upper bounds $\mathbf{y}_{lb1}$ and $\mathbf{y}_{\text{ub1}}$. Second ray $\mathbf{r}2$ leads to $\mathbf{y}_{\text{lb2}}$ and $\mathbf{y}_{\text{ub2}}$; $\mathbf{r}3$ to $\mathbf{y}_{\text{lb3}}$ and $\mathbf{y}_{\text{ub3}}$, etc. Except for $\mathbf{r}1$, the choices of the rays are depicted for indication only; they do not necessarily correspond to *real* IRM choices.

Compared to CG, IRM offers two key advantages: (i) the input $\mathbf{r}$ of the IRM (intersection) sub-problem can be easily controlled, which can make this sub-problem even easier than the CG (separation) sub-problem, (ii) a sequence of lower bounds solutions $\mathbf{y}_{\text{lb}} = t^*\mathbf{r}$ is automatically provided throughout the IRM search (Lagrangean bounds in CG are optional). The first point (i) is very useful, for instance, in capacitated `Set-Covering` sub-problems with fractional weights (supply demands). Such (sub-)problems can be very difficult in the general case, but simply tractable by dynamic programming when $\mathbf{r}$ is integer. Over all our tests on `Set-Covering` problems with large-range (fractional) input, IRM is *never* less effective than classical CG.

Regarding the lower bounds $\mathbf{b}^\top\mathbf{y}_{\text{lb}} = \mathbf{b}^\top(t^*\mathbf{r})$, they can even be nearly optimal *from the first iterations*. For example, certain `Bin-Packing` instances (*e.g.*, the triplets) have an optimal dual solution of the form $\mathbf{y}^* = t^*\mathbf{b}$ ($\mathbf{b}$ is $\mathbf{1}_n$ in `Bin-Packing`) that can be discovered at the very first iteration, when $\mathbf{r} = \mathbf{b}$. Using this first ray choice, the IRM starts out by advancing along the direction with the highest rate of objective function increase. The next rays are determined by exploiting the fact that the segment joining $\mathbf{y}_{\text{lb}}$ and $\mathbf{y}_{\text{ub}}$ can contain more high-quality feasible solutions (see Section 2.3). As such, IRM tries to generate the next integer ray $\mathbf{r}_{\text{new}}$ in the proximity of some points of the form $\mathbf{r} + \beta(\mathbf{y}_{\text{ub}} - \mathbf{y}_{\text{lb}})$. Each new ray $\mathbf{r}_{\text{new}}$ can lead the IRM sub-problem to: (i) a better lower bound solution $t^*_{\text{new}}\mathbf{r}_{\text{new}}$, or (ii) a new constraint that separates $\mathbf{y}_{\text{ub}}$ from $\mathcal{P}$, which allows IRM to improve the upper bound (Section 2.4). When the IRM can no longer decrease the gap between $\mathbf{y}_{\text{lb}}$ and $\mathbf{y}_{\text{ub}}$, it increases (doubles) the ray coefficients (discretization refining). This can later lead to a reduction of the gap between the bounds (Section 2.5), at the cost of slowing down the IRM sub-problem algorithm.

1.1 Context and Related Ideas

The main reason for only using *integer* rays $\mathbf{r}$ is that this renders the intersection sub-problem tractable by Dynamic Programming (DP). This technique is often used in Fully Polynomial-Time Approximation Schemes (FPTASs), *e.g.*, the first knapsack FPTAS [13] 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 [9, § 3]. However, the IRM does not really need rounding, as it directly chooses integer rays. It is the general idea of *profit-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 2.2 and the examples in Sections 3.1 and 3.2). More generally, this controllability of the sub-problem input $\mathbf{r}$ can render tractable more sub-problems that would otherwise be prohibitively hard (see theoretical examples in Section 3.1.2.1). The worst-case running time could be avoided when the sub-problem has a very low average complexity (or good *smoothed complexity*[1]).

To compare IRM with CG, we recall that classical CG can be seen in the dual space as an iterative cutting-plane algorithm, *i.e.*, the Kelley's method. At each iteration, this method takes the optimum solution $\mathbf{y}$ of an outer dual polytope and calls a separation oracle to find (and add) the most violated constraint with respect to $\mathbf{y}$. In fact, this optimum solution can be replaced by some interior (more central) dual solutions in certain stabilized CG methods [5, 12]. IRM differs from such approaches in the following aspects. First, the constraint returned by the IRM sub-problem can always be different from the one returned by the CG sub-problem. Let us just give a short example for $n = 1$: given constraints $2y_1 \leq 1$, $1005y_1 \leq 1000$ and input $\mathbf{r} = 1$ (or $\mathbf{y} = 1$ for CG), the intersection sub-problem returns $2y_1 \leq 1$ (with $t^* = 0.5$), while a separation oracle would rather return $1005y_1 \leq 1000$ (this is usually considered the most violated constraint). Secondly, IRM provides intermediate lower bounds as a built-in feature, while CG only gives the option of using Lagrangean lower bounds (whose calculation is not always straightforward). Thirdly, the IRM rays are not necessarily infeasible dual solutions as in CG, but simply integer directions in the dual space. Furthermore, we are not aware of any CG methods that (try to) select *integer* dual solutions 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 [5, Alg. 1.5], a well-centered ϵ-optimal solution [12, Def. 1], etc.

We now focus on techniques closer to our central idea of *ray projection in polytopes with exponentially*

[1] 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).

many constraints. Certain (generally) related ideas can be found in Submodular Function Minimization (SFM). One of the simplest approaches for optimizing the submodular polytope is the *Greedy* algorithm [19, 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 [19, 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" [20] or as the "intersection problem" [8]. The intersection algorithm from [20] consists of solving a minimum ratio problem. Our intersection sub-problem (Definition 1, Section 2.2.1) is also formulated as a ratio minimization problem. The last section of [8] discusses certain perspectives on the applicability of related approaches to other polytopes with prohibitively many constraints, but there is no mention of CG.

Besides this line intersection work in SFM, such ray projection approaches seem partially overlooked in other fields dealing with polytopes with exponentially-many constraints—including CG. Despite our best efforts[2], 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 Practical Impact and Paper Organization

IRM is fully described in Section 2 and examples of intersection algorithms are provided in Section 3. To justify the practical interest, please let us present in advance some experimental conclusions:

- if the input is fractional (large-range), the IRM can solve certain instances that could be prohibitively hard to conventional CG, *e.g.*, see the IRM-CG experimental comparison on large-range (scaled) versions of f_2-`Elastic` and f_3-`Elastic` `Cutting-Stock` (Section 4.1), or also the large-range `Capacitated Arc Routing` results (Section 4.2);
- the running time of IRM stays in the same order of magnitude for integer (standard-size) and fractional (large-range) input data—compare Tables 1 and 2 (Section 4.1), or Tables 3 and 4 (Section 4.2);
- For roughly half of the tested instances (of either problem), a gap of 10% between the lower and the upper bounds $\mathbf{b}^\top \mathbf{y}_{\text{lb}}$ and $\mathbf{b}^\top \mathbf{y}_{\text{ub}}$ can be reached after about 20% of the total IRM convergence time (either for integer or fractional data);

The paper is organized as follows. Section 2 is devoted to a general presentation of the IRM. In Section 3, we provide solution methods for the intersection sub-problems of `Elastic Cutting-Stock` and `Capacitated Arc Routing`. Numerical experiments are provided in Section 4, followed by conclusions in Section 5.

2 Integer Ray Method

2.1 Main Building Blocks and Pseudocode

IRM consists of 5 steps briefly described below. Each step is discussed in greater detail in a dedicated subsection indicated in parentheses (except for Stage 2 which is rather straightforward).

1. *Solve the IRM sub-problem (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 generates a lower bound solution $\mathbf{y}_{\text{lb}} = t^*\mathbf{r}$ and a (first-hit) constraint $\mathbf{a}^\top y \leq c_a$ that is $\mathbf{y}_{\text{lb}}$-tight (*i.e.*, $\mathbf{a}^\top \mathbf{y}_{\text{lb}} = c_a$). This first-hit constraint is added to the set A of currently available constraints ($A \leftarrow A \cup \{[c_a, \mathbf{a}]\}$);

[2]Including personal communications with five researchers whose help will be mentioned in the acknowledgements of the final version.

2. *Calculate upper bound solution* $\mathbf{y}_{\rm ub}$ *(no dedicated subsection)* This $\mathbf{y}_{\rm ub}$ is simply determined by optimizing $\mathbf{b}^\top\mathbf{y}$ over $\mathbf{y} \in \mathcal{P}_A$, where $\mathcal{P}_A$ is the polytope obtained from $\mathcal{P}$ by only keeping constraints $A \subset \overline{A}$; as such, $\mathbf{b}^\top\mathbf{y}_{\rm ub}$ is an upper bound for the optimum of $\mathbf{b}^\top\mathbf{y}$ over $\mathbf{y} \in \mathcal{P}$ (the IRM/CG optimum). No fancy LP method is needed.

3. *Next Ray Generation (Section 2.3)* Given $\mathbf{r}, \mathbf{y}_{\rm lb}$ and $\mathbf{y}_{\rm ub} = \mathbf{y}_{\rm lb} + \Delta$, search for new rays among the closest integer points to $\mathbf{r} + \beta\Delta$, with $\beta > 0$. For a fixed β, the closest integer point is determined by applying a simple rounding on each of the n elements of $\mathbf{r} + \beta\Delta$. New potential better rays $\mathbf{r}_{\rm new}$ are iteratively generated by gradually increasing β until a stopping condition is met. This step is interwined with Step 4, as described below.

4. *Improving lower or upper bounds (Section 2.4)* To try to improve $\mathbf{y}_{\rm lb}$ or $\mathbf{y}_{\rm ub}$, IRM iteratively solves the intersection sub-problem on the new rays $\mathbf{r}_{\rm new}$ generated by Step 3. For each such $\mathbf{r}_{\rm new}$, this can lead to lower bound improvement (if the first-hit solution $t^*_{\rm new}\mathbf{r}_{\rm new} \in \mathcal{P}$ dominates $\mathbf{y}_{\rm lb} = t^*\mathbf{r}$) or to upper bound improvement (if the first-hit constraint separates $\mathbf{y}_{\rm ub}$). As soon as either of the bounds is thus improved, the ray generation from Step 3 is *restarted* with new input data (*i.e.*, with $\mathbf{r}, \mathbf{y}_{\rm ub}$ and $\mathbf{y}_{\rm ub}$ updated accordingly to the new bound values). If neither bound can be improved as above, IRM continues generating new rays by only increasing β in Step 3; as such, Step 3 and Step 4 are actually iteratively interwined until either bound is improved.

5. *Discretization Refining (Section 2.5)* If all rays that could be generated by Step 3 fail to improve either bound in Step 4, we consider that the current ray coefficients are too small (imprecise) to reduce the gap. This step aims at refining the ray coefficients: multiply $\mathbf{r}$ by $\lambda \in \mathbb{Z}_+$ (we used $\lambda = 2$) and divide t^* by λ; $\mathbf{y}_{\rm lb} = t^*\mathbf{r}$ and $\mathbf{y}_{\rm ub}$ stay unchanged. The next rays generated by Step 3 will have larger coefficients (their construction is based on a doubled $\mathbf{r}$), and so, some of them will be closer to intersecting the segment $[\mathbf{y}_{\rm lb}, \mathbf{y}_{\rm ub}]$ joining $\mathbf{y}_{\rm lb}$ and $\mathbf{y}_{\rm ub}$. After enough repetitions of Stages 3-5, one can make the new rays pass as close as necessary to $[\mathbf{y}_{\rm lb}, \mathbf{y}_{\rm ub}]$, which is guaranteed to lead to some bound improvement, as discussed later. We theoretically prove an eventual finite convergence in Section 2.5.

Algorithm 1 provides the general IRM pseudocode. All notations and external routines are discussed below. Except routines `initialConstraints` and `IRM-subprob`, all other components are generic with respect to our main `Set-Covering` CG model (1.1). The above stages are implemented as follows: Stage 1 is implicitly used at each `IRM-subprob` call; Stage 2 arises at each `optimize` call and update of A (Lines 7 and 23-25); Stage 3 is roughly represented by the `nextRay` calls; Stage 4 corresponds to Lines 17-25 and Stage 5 to Lines 11-12. We now list below all external routines used by Algorithm 1; the first two are problem-dependent, while the last two are general.

`initialConstraints()` simply provides some initial *lower (and optionally upper)* bounds l_i (and u_i) for variables $y_i, \forall i \in [1..n]$. In `Cutting-Stock` we used $l_i = 0$ and in `Arc-Routing` we used $l_i = -\overline{c}_i$, where $\overline{c}_i$ is the traversal cost of edge i, see Section 3.2.2, model (3.8).

`IRM-subprob()` solves the IRM sub-problem, this is generally described below in Section 2.2 and completely specified in Section 3.1 (on four `Cutting-Stock` variants) and 3.2 (on `Capacitated Arc Routing`);

`optimize`$(\mathcal{P}_A, \mathbf{b})$ is a standard LP algorithm returning the optimum of $\mathbf{b}^\top\mathbf{y}$ over $\mathbf{y} \in \mathcal{P}_A$;

`nextRay` $(\mathbf{r}, \mathbf{y}_{\rm lb}, \mathbf{y}_{\rm ub}, \mathbf{r_{prv}})$ returns the next integer ray after $\mathbf{r}_{\rm prv}$ in a sequence of new rays constructed from $\mathbf{r}$, $\mathbf{y}_{\rm lb}$ and $\mathbf{y}_{\rm ub}$ (see Section 2.3).

Algorithm 1 Integer Ray Method for optimizing over $\mathcal{P}$ in (1.1)

1: $A \leftarrow$ `initialConstraints()`
2: $\mathbf{r} \leftarrow \mathbf{b}$ ▷ scale $\mathbf{b}$ down if all b_i are too large
3: $(t^*, \mathbf{a}, c_a) \leftarrow$`IRM-subprob`$(\mathbf{r})$ ▷ in the code, it might be easier to manipulate $\frac{1}{t^*}$ than t^*
4: $A \leftarrow A \cup \{[c_a, \mathbf{a}]\}$
5: **repeat**
6: $\quad \mathbf{y}_{\text{lb}} \leftarrow t^* \mathbf{r}$
7: $\quad \mathbf{y}_{\text{ub}} \leftarrow$`optimize`$(\mathcal{P}_A, \mathbf{b})$ ▷ max. dual obj. func. $\mathbf{b}$ on polytope $\mathcal{P}_A$ described by constraints 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`$(\mathbf{r}, \mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}, \mathbf{r}_{\text{prv}})$ ▷ 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` $(\mathbf{r}, \mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}, \mathbf{0}_n)$ ▷ ii) re-try ray generation
14: $\quad\quad$ **end while**
15: $\quad\quad \mathbf{r}_{\text{prv}} \leftarrow \mathbf{r}_{\text{new}}$
16: $\quad\quad (t^*_{\text{new}}, \mathbf{a}, c_a) \leftarrow$`IRM-subprob`$(\mathbf{r}_{\texttt{new}})$
17: $\quad\quad newLb \leftarrow \left(\mathbf{b}^\top (t^*_{\text{new}} \mathbf{r}_{\text{new}}) > \mathbf{b}^\top \mathbf{y}_{\text{lb}}\right)$ ▷ better lower bound if the expression in parentheses is true
18: $\quad\quad newUb \leftarrow \left(\mathbf{a}^\top \mathbf{y}_{\text{ub}} > c_a\right)$ ▷ new dual constraint violated by current $\mathbf{y}_{\text{ub}}$ (from Line 7)
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: $\quad\quad A \leftarrow A \cup \{[c_a, \mathbf{a}]\}$ ▷ $\mathbf{y}_{\text{ub}}$ will be updated at the next loop at Line 7
25: $\quad$ **end if**
26: **until** $\left(\lceil \mathbf{b}^\top \mathbf{y}_{\text{ub}} \rceil = \lceil \mathbf{b}^\top \mathbf{y}_{\text{lb}} \rceil\right)$ ▷ current $\mathbf{y}_{\text{lb}}$ and $\mathbf{y}_{\text{ub}}$ are returned when the algorihm stops

2.2 The Intersection (IRM) Sub-problem

2.2.1 Formal Definition

Definition 1. *(Intersection (sub)-problem) Given ray $\mathbf{r} \in \mathbb{Z}^n$ and polytope $\mathcal{P}$ defined by constraints $\mathbf{a}^\top \mathbf{y} \leq c_a$ with $[c_a, \mathbf{a}] \in \overline{A}$ ($c_a \geq 0$), determine the maximum step length*

$$t^* = \min_{\substack{[c_a, \mathbf{a}] \in \overline{A} \\ \mathbf{a}^\top \mathbf{r} > 0}} \frac{c_a}{\mathbf{a}^\top \mathbf{r}} = \max\{t :\ t\mathbf{r} \in \mathcal{P}\}$$

The sub-problem also returns the first-hit constraint $[c_a, \mathbf{a}]$, i.e., a constraint $\mathbf{a}^\top \mathbf{y} \leq c_a$ that is tight for $t^*\mathbf{r}$. The min-max equivalence can simply be checked by observing that any $t > t^*$ would lead to an infeasible $t\mathbf{r}$, i.e., such $t\mathbf{r}$ would violate the constraint $a^\top \mathbf{y} \leq c_a$, because $\mathbf{a}^\top (t\mathbf{r}) = t(\mathbf{a}^\top \mathbf{r}) > t^*(\mathbf{a}^\top \mathbf{r}) = c_a$.

As described in Section 2.1, Algorithm 1 can be used to optimize over any polytope $\mathcal{P}$. When $\mathcal{P}$ is a dual polytope in the CG model (1.1) of a capacitated `Set-Covering` problem, $\overline{A}$ is a set of primal columns (not explicitly enumerated). The (dual) constraint $\mathbf{a}^\top \mathbf{y} \leq c_a$ is associated to a configuration (pattern, route, schedule, etc.) with *non-negative cost* c_a. Furthermore, we interpret $\mathbf{r}$ as a *vector of profits* and $\mathbf{a}^\top \mathbf{r}$ becomes the total profit of configuration $\mathbf{a}$. The ratio $\frac{c_a}{\mathbf{a}^\top \mathbf{r}}$ is interpreted as a *cost/profit ratio*; as such, the IRM sub-problem aims at *minimizing the cost/profit* ratio over all possible configurations in $\overline{A}$.

We propose in Section 2.2.2 a generic DP framework for solving the intersection sub-problem, but one should be aware that other alternatives do exist. For instance, a very general solution method consists of iteratively applying a *separation oracle* in a dichotomic manner. If a separation oracle certificates that

$t^- \mathbf{r} \in \mathcal{P}$ but $t^+ \mathbf{r} \notin \mathcal{P}$, then the sought maximum step length t^* belongs to $[t^-, t^+)$; a subsequent oracle call on $\frac{t^- + t^+}{2}\mathbf{r}$ can indicate the status of $\frac{t^- + t^+}{2}\mathbf{r}$, leading to a more precise interval for t^*, *i.e.*, either $t^* \in \left[t^-, \frac{t^- + t^+}{2}\right)$ or $t^* \in \left[\frac{t^- + t^+}{2}, t^+\right)$. By iterating this, one can solve the intersection sub-problem with a sufficiently good precision.

2.2.2 Dynamic Programming Framework

A general approach for intersection sub-problems consists of using a Dynamic Programming (DP) framework based on a set $\mathcal{S}$ of *states* described as follows. Given $s \in \mathcal{S}$, an s-configuration $\mathbf{a} \in s$ (*i.e.*, a configuration in state s) is characterized by a set of *state indices*:

- *the integer profit* p_s: all s-configurations $\mathbf{a} \in s$ need to 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. For instance, in classical `Cutting-Stock`, c_s is always 1, but it can have a more complex form in `Elastic Cutting-Stock` (see Section 3.1.1) and `Arc-Routing` (Section 3.2). In certain versions, this cost is only indirectly used for indexing (see Section 2.2.2.1 below);
- any other problem-specific state indicators—*e.g.*, route start and end points in routing problems. We will have more to say about this in Section 3.2, but for now it is enough to only focus on p_s and c_s.

We also use a *minimum weight function* $W : \mathcal{S} \to \mathbb{R}_+$ such that $W(s)$ is the minimum total weight required to reach state s, *i.e.*, the minimum weight of a valid configuration with cost c_s and profit p_s. For capacitated sub-problems, the most widely-used DP algorithms use the weights for state indexing and the recurrence relation to calculate maximum profit values. Since our profits $\mathbf{r}$ are integer, our DP framework reverses the role of weights and profits (as in the knapsack FPTAS [13]); our DP algorithm uses profits for state indexing and the recurrence relation to calculate *minimum weight values* $W(s)$ with $s \in \mathcal{S}$.

The main DP idea for calculating W can be sketched on the most general recursion formula example: $W(s) = \min_{i \in [1..n]} W\left(s_{\{-i\}}\right) + w_i$, 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 over all $s \in \mathcal{S}$, 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] In the end, one identifies a state $s^* = (p_{s^*}, c_{s^*})$ that minimizes $\frac{c_{s^*}}{p_{s^*}}$: the value $t^* = \frac{c_{s^*}}{p_{s^*}}$ is the sought maximum step length. The exact value of some $\mathbf{a} \in s^*$ can be recovered by following precedence relations between states. For this, it is enough to record for each $s \in \mathcal{S}$ the state $s_{\{-i\}}$ that triggered the last update on $W(s)$.

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

As for classical weight-indexed DP in CG sub-problems, the asymptotic running time of profit-indexed DP depends (linearly) 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 IRM, $|\mathcal{S}|$ depends closely on the number of realisable *profit* values. In CG, the number of weight-indexed 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, if $\mathbf{r} = \mathbf{b}$ contains very large values, Algorithm 1 can even scale $\mathbf{r}$ down (see Line 2). As such, the profits $\mathbf{r}$ can become the larger than weights $\mathbf{w}$ *only by* discretization refining, because the magnitude of $\mathbf{r}$ is doubled at each discretization refining operation. However, the numerical experiments confirm that the number of discretization refining operations can be quite easily kept below a reasonable value of 5—see Tables 1-4. This is enough to keep

[3] Care needs to be taken *not* to add element i more than b_i times by chaining and 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 A.

most of the time the profits at relatively low values (no larger than $2^5 = 32$, less than typical weight values, see Table 5, p. 32), and so, to keep $\mathcal{S}$ smaller than the state space in weight-indexed DP for CG sub-problems.

For both profit-indexed and weight-indexed DP, a combinatorial explosion risk is related to the number of feasible (column) cost values that can generate different states. By considering very high (or fractional) costs c_s, the number of states can become exceedingly large. In IRM, two techniques can (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 (3.1) in Section 3.1.1. In this case, the state cost c_s is actually determined from the *minimum* weight $W(s)$; there is no need to use a different state for each realisable weight (or cost) associated to the same profit. Secondly, one can prune states with exceedingly high costs, by detecting that they can *not* lead to a minimum cost/profit ratio. We only provide here a basic illustration (see also the `Arc-Routing` example in Section 3.2.3.3): 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.

2.3 Generating the Next Integer Ray

The ray generation resides in using a sequence (list) $rLst$ of new ray proposals that are provided *one by one* to Algorithm 1 by iteratively calling the routine (oracle) `nextRay(`$\mathbf{r}, \mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}, \mathbf{r}_{\text{prv}}$`)`. The input of `nextRay` is represented by the current ray $\mathbf{r}$, the current lower and upper bounds $\mathbf{y}_{\text{lb}}$ and $\mathbf{y}_{\text{ub}}$, and the last ray $\mathbf{r}_{\text{prv}}$ returned by the previous `nextRay` call ($\mathbf{r}_{\text{prv}} = \mathbf{0}_n$ if none). Each `nextRay` call returns the new ray situated right after $\mathbf{r}_{\text{prv}}$ in $rLst$, or the first new ray in $rLst$ if $\mathbf{r}_{\text{prv}} = \mathbf{0}_n$. The essential task is the construction of the sequence $rLst$; we describe it below by picking out the key points in boldface.

Rationale To locate promising new rays, let us focus on the segment $[\mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}] = \{\mathbf{y}^\alpha = \mathbf{y}_{\text{lb}} + \alpha\Delta : \ \alpha \in [0, 1]\}$ (with $\Delta = \mathbf{y}_{\text{ub}} - \mathbf{y}_{\text{lb}}$) that joins $\mathbf{y}_{\text{lb}}$ and $\mathbf{y}_{\text{ub}}$. Also, consider a projection of $[\mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}]$ onto the t^*-*scaled segment* $\{\mathbf{r}^\beta = \mathbf{r} + \beta\Delta : \ \beta \in [0, \beta_{\max}]\}$, where $\beta_{\max} = \frac{1}{t^*}$ (the case $t^* = 0$ is treated separately)—see Figure 2 (p. 9) for a graphical representation. If this later segment contains an *integer* solution $\mathbf{r}^\beta$, a new ray $\mathbf{r}^\beta$ can surely lead to an improvement of the optimality gap. Indeed, such integer new ray $\mathbf{r}^\beta = \mathbf{r} + \beta\Delta$ intersects the segment $[\mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}]$ at some point $\mathbf{y}^\alpha = t^*(\mathbf{r} + \beta\Delta) = \mathbf{y}_{\text{lb}} + t^*\beta\Delta$. By solving the intersection sub-problem on $\mathbf{r}^\beta$, one implicitly solves the intersection sub-problem on $\mathbf{y}^\alpha$. This also determines the feasibility status of $\mathbf{y}^\alpha$: (i) if $\mathbf{y}^\alpha \in \mathcal{P}$, the lower bound can be improved because the feasible solution returned by the IRM sub-problem on $\mathbf{r}$ has higher quality than $\mathbf{y}_{\text{lb}}$; (ii) if $\mathbf{y}^\alpha \notin \mathcal{P}$, then the upper bound can be improved, because both $\mathbf{y}^\alpha$ and $\mathbf{y}_{\text{ub}}$ can be separated from $\mathcal{P}$. We will formally prove this in Section 2.4 (Proposition 1), 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 $\lfloor \mathbf{r}^{\beta_\ell} + \frac{1}{2}\mathbf{1}_n \rfloor$. The lowest $\beta > \beta_\ell$ such that $\lfloor \mathbf{r}^\beta + \frac{1}{2}\mathbf{1}_n \rfloor \neq \lfloor \mathbf{r}^{\beta_\ell} + \frac{1}{2}\mathbf{1}_n \rfloor$ is achieved by the lowest $\beta > \beta_\ell$ with $r_i^\beta + \frac{1}{2} = \lfloor r_i^{\beta_\ell} + \frac{1}{2} \rfloor + 1$ on at least a coordinate $i \in [1..n]$. The latter relation can also be expressed as: $\left(r_i^{\beta_\ell} + \frac{1}{2}\right) + \Delta_i(\beta - \beta_\ell) = \lfloor r_i^{\beta_\ell} + \frac{1}{2} \rfloor + 1$, or $\Delta_i(\beta - \beta_\ell) = \lfloor r_i^{\beta_\ell} + \frac{1}{2} \rfloor + 1 - \left(r_i^{\beta_\ell} + \frac{1}{2}\right)$. As such, one could recursively generate a sequence $\beta_1, \beta_2, \beta_3, \ldots$ with the formula $\beta_{\ell+1} = \beta_\ell + \min_{i \in [1..n]} \frac{\lfloor r_i^{\beta_\ell} + \frac{1}{2} \rfloor + 1 - \left(r_i^{\beta_\ell} + \frac{1}{2}\right)}{\Delta_i}$. However, two consecutive integer rays $\mathbf{r}^{\beta_\ell}$ and $\mathbf{r}^{\beta_{\ell+1}}$ in this sequence can differ only on a single coordinate, *i.e.*, if only one $i \in [1..n]$ minimizes 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 $i = \max\{\Delta_j : \ j \in [1..n]\}$. This leads to a simpler update (recurrence) formula, based on a fixed i:

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

Algorithmic Construction of the New Ray Sequence Algorithm 2 completely describes the construction of the $rLst$ sequence. The above update formula 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). Algorithm 2 is actually only executed when the IRM updates $\mathbf{r}$, $\mathbf{y}_{\text{lb}}$ or $\mathbf{y}_{\text{ub}}$, *i.e.*, when IRM (Algorithm 1) calls $\texttt{nextRay}(\mathbf{r}, \mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}, \mathbf{0}_n)$. A last argument of $\mathbf{r}_{\text{prv}} = \mathbf{0}_n$ indicates that IRM requires the *very first* new ray for current $\mathbf{r}, \mathbf{y}_{\text{lb}}$ and $\mathbf{y}_{\text{ub}}$ and this triggers the construction of the $rLst$ sequence. 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, Algorithm 1). When all elements from $rLst$ are finished, `nextRay` returns $\mathbf{0}_n$.

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: $\mathbf{r},\ \mathbf{y}_{\text{lb}} = t^*\mathbf{r},\ \mathbf{y}_{\text{ub}} = \mathbf{y}_{\text{lb}} + \Delta$
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: $i \leftarrow \arg\max_{j \in [1..n]} \Delta_j$ ▷ break ties randomly if necessary
4: **repeat**
5: **if** $\lfloor \mathbf{r}^\beta + \frac{1}{2} \rfloor \neq \mathbf{r}^\beta + \frac{1}{2}$
6: $\beta \leftarrow \beta + \frac{\lfloor r_i^\beta + \frac{1}{2} \rfloor + 1 - \left(r_i^\beta + \frac{1}{2}\right)}{\Delta_i}$
7: **else**
8: $\beta \leftarrow \beta + \frac{1}{\Delta_i}$
9: **end if**
10: $\mathbf{r}^\beta \leftarrow \mathbf{r} + \beta\Delta$
11: $\texttt{append}\big(rLst, \lfloor \mathbf{r}^\beta + \frac{1}{2}\mathbf{1}_n \rfloor\big)$ ▷ add a new ray $\mathbf{r}_{\text{new}}$ at the end
12: **until** $\beta > \beta_{\max}$
13: $rLst \leftarrow \texttt{lastToFront}(rLst)$ ▷ better to try the last ray first

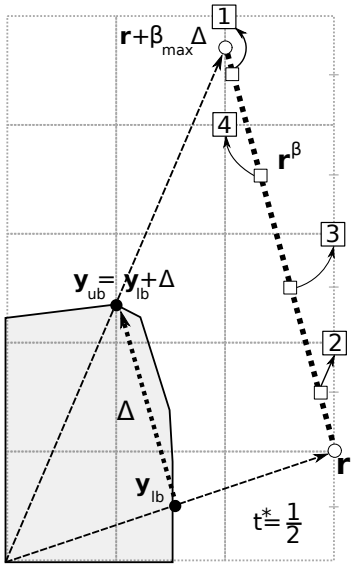

Figure 2: New rays for $n = 2$. The chosen i (Line 3) is the vertical coordinate; except at $\mathbf{r}^\beta = \mathbf{r}$, $\mathbf{r}_i^\beta + \frac{1}{2} \in \mathbb{Z}$, see the empty squares.

Sequence Reshuffling The order of rays in $rLst$ strongly influences the general IRM evolution. A very practical enhancement of the initial order simply consists of moving the last element of $rLst$ to the front (last line of Algorithm 2). The last discovered ray (close to the direction $\mathbf{0}_n \to \mathbf{y}_{\text{ub}}$) can be more useful for upper bound improvement (see Proposition 2 in Section 2.4 for more formal arguments), while the first rays are rather useful for lower bound improvement. The upper bounds are somehow more critical, because they can be *anywhere* in the search space. Reasonable lower bounds are more easily located by the IRM from the very first iteration, because the first ray $\mathbf{r} = \mathbf{b}$ makes IRM start out by moving on the fastest-improvement direction $\mathbf{r} = \mathbf{b}$, see also the lower bounding conclusions of the numerical experiments (§4.1.2 and §4.2.2).

The special case $t^* = 0$ In such exceptional situations, there is a $\mathcal{P}$ constraint $\mathbf{a}^\top\mathbf{y} \leq 0$ such that $\mathbf{a}^\top\mathbf{r} > 0$ that blocks any advance on direction $\mathbf{0}_n \to \mathbf{r}$, *i.e.*, this direction is not "open". In this case, the sequence $rLst$ only contains $\mathbf{r}_{\text{new}} = \lfloor \mathbf{y}_{\text{ub}} \rfloor$. However, after solving the IRM sub-problem on this $\mathbf{r}_{\text{new}}$, the constraint $\mathbf{a}^\top\mathbf{y} \leq 0$ is added to A, and so, IRM (continually) updates $\mathbf{y}_{\text{ub}}$ as in CG until an "open" direction is found.

2.4 Updating the Upper and Lower Bounds of (1.1)

We here analyze the way IRM decreases the gap between $\mathbf{b}^\top\mathbf{y}_{\text{lb}}$ and $\mathbf{b}^\top\mathbf{y}_{\text{ub}}$. We first present the best-case scenario and we finish with a worst-case investigation.

Proposition 1. *If* $\mathbf{r}^\beta = \mathbf{r} + \beta\Delta = \mathbf{r} + \beta(\mathbf{y}_{\text{ub}} - \mathbf{y}_{\text{lb}})$ *is integer for some* $\beta \in (0, \beta_{max}]$, *then Algorithm 1 can surely improve either the lower or the upper bound after calling* $\texttt{IRM-subprob}(\mathbf{r}_{new})$ *in Line 16 with*

$\mathbf{r}_{new} = \mathbf{r}^\beta$.

Proof. Recall (Section 2.3) that the segment $[\mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{lb}} + \Delta] = [t^*\mathbf{r}, t^*\mathbf{r} + \Delta]$ can be projected into the scaled segment $[\mathbf{r}, \mathbf{r} + \beta_{\max}\Delta] = [\mathbf{r}, \mathbf{r} + \frac{1}{t^*}\Delta]$, see also Figure 2. Using a simple multiplication by $\frac{1}{t^*}$, this scaling can project any $\mathbf{y}^\alpha = \mathbf{y}_{\text{lb}} + \alpha\Delta \in [\mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}]$ onto $\mathbf{r}^\beta = \mathbf{r} + \beta\Delta$, where $\alpha = t^*\beta$. We say that the ray (line) direction $\mathbf{0}_n \rightarrow \mathbf{r}^\beta$ intersects the segment $[\mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}]$ at $\mathbf{y}^\alpha$. The IRM sub-problem applied on $\mathbf{r}_{\text{new}} = \mathbf{r}^\beta$ returns the maximum step length t^*_β such that $t^*_\beta\mathbf{r}^\beta \in \mathcal{P}$.

case $t^*_\beta \geq t^*$ leads IRM directly to new lower bound solution $t^*_\beta\mathbf{r}^\beta$. We first observe that $\mathbf{b}^\top\mathbf{y}_{\text{ub}} > \mathbf{b}^\top\mathbf{y}_{\text{lb}} \geq 0 \implies \mathbf{b}^\top(t^*\mathbf{r} + \Delta) > \mathbf{b}^\top(t^*\mathbf{r}) \geq 0$, which leads to $\beta^\top\Delta > 0$ and further to $\mathbf{b}^\top\mathbf{r}^\beta = \mathbf{b}^\top\mathbf{r} + \mathbf{b}^\top(\beta\Delta) > \mathbf{b}^\top\mathbf{r} \geq 0$. Then, we show that $t^*_\beta\mathbf{r}^\beta$ strictly dominates $\mathbf{y}_{\text{lb}}$ in terms of objective value, *i.e.*, $\mathbf{b}^\top\left(t^*_\beta\mathbf{r}^\beta\right) > \mathbf{b}^\top\mathbf{y}_{\text{lb}}$. This comes from a simple transitivity with $\mathbf{b}^\top\mathbf{y}^\alpha = \mathbf{b}^\top\left(t^*\mathbf{r}^\beta\right)$ as middle term: $\mathbf{b}^\top\left(t^*_\beta\mathbf{r}^\beta\right) \geq \mathbf{b}^\top\left(t^*\mathbf{r}^\beta\right)$ (true based on $(t^*_\beta - t^*)\mathbf{b}^\top\mathbf{r}^\beta \geq 0$, owing to $\mathbf{b}^\top\mathbf{r}^\beta > 0$) and $\mathbf{b}^\top\left(t^*\mathbf{r}^\beta\right) > \mathbf{b}^\top(t^*\mathbf{r}) = \mathbf{b}^\top\mathbf{y}_{\text{lb}}$ (true owing to $\mathbf{b}^\top\mathbf{r}^\beta > \mathbf{b}^\top\mathbf{r}$).

case $t^* > t^*_\beta$ leads IRM to improve $\mathbf{y}_{\text{ub}}$ via an *upper bounding process* described by the following causal events in Algorithm 1:

1. the `IRM-subprob`($\mathbf{r}_{\text{new}}$) call (Line 16) on $\mathbf{r}_{\text{new}} = \mathbf{r}^\beta$ returns a constraint $\mathbf{a}^\top\mathbf{y} \leq c_a$ satisfied with equality by $t^*_\beta\mathbf{r}^\beta$;
2. $\mathbf{y}^\alpha = t^*\mathbf{r}^\beta$ violates above constraint ($t^* > t^*_\beta \implies \mathbf{a}^\top(t^*\mathbf{r}^\beta) > \mathbf{a}^\top(t^*_\beta\mathbf{r}^\beta) = c_a$), and so, $\mathbf{y}_{\text{ub}}$ also violates it, *i.e.*, $\mathbf{a}^\top\mathbf{y}_{\text{ub}} > c_a$ (otherwise, if $\mathbf{a}^\top\mathbf{y}_{\text{ub}} \leq c_a$, then $\mathbf{a}^\top\mathbf{y} \leq c_a,\ \forall\mathbf{y} \in [\mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}]$);
3. above property turns *newUb* `true` in Line 18, and so, $[c_a, \mathbf{a}]$ is added to A in Line 24, leading to a better upper bound at the next `optimize`($\mathcal{P}_A, \mathbf{b}$) call in Line 7.

□

The hypothesis of this property represents a very fortunate situation that does not necessarily hold very often. However, the main idea can be generalized to cases in which $\mathbf{r}^\beta$ is very close to an integer solution. If the generated ray $\mathbf{r}_{\text{new}} = \lfloor\mathbf{r}^\beta + \frac{\mathbf{1}_n}{2}\rfloor$ is close to $\mathbf{r}^\beta$, the ray line $\mathbf{0}_n \rightarrow \mathbf{r}_{\text{new}}$ is close to the segment $[\mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}]$: the closer this ray line is to $[\mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}]$, the more chances there are to improve any of the bounds as above.

2.4.1 Worst-Case Scenario of Upper Bound Improvement

We show that our ray generation procedure is able to eventually lead IRM to upper bound improvement. Sooner or later, the *upper bounding process* described in the second case ($t^* > t^*_\beta$) of the above proof (Proposition 1) can always be triggered. In the worst case, this can require very large ray coefficients, but they can be obtained by discretization refining. Section 2.5 will continue these ideas by fully describing the discretization refining mechanism, followed by proving the finite IRM convergence.

Proposition 2. *There exists a sufficiently small $t_\epsilon > 0$ such that if $\mathbf{y}_{\text{lb}} = t^*\mathbf{r}$ with $t^* < t_\epsilon$ (i.e., if $\mathbf{r}$ is large-enough), our ray generation procedure (Algorithm 2) produces at least a new ray that leads to upper bound improvement. Formally, for any input $\mathbf{r}$, $\mathbf{y}_{\text{lb}} = t^*\mathbf{r}$ and $\mathbf{y}_{\text{ub}} \notin \mathcal{P}$ 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 $\mathbf{a}^\top\mathbf{y} \leq c_a$ violated by infeasible $\mathbf{y}_{\text{ub}}$.*

Proof. The maximum feasible step length on direction $\mathbf{0}_n \rightarrow \mathbf{y}_{\text{ub}}$, denoted t_{ub}, corresponds to intersection point $\mathbf{r}_{\text{ub}} = t_{\text{ub}}\mathbf{y}_{\text{ub}}$. This $\mathbf{r}_{\text{ub}}$ can be seen as an ideal ray that we would like to be generated. Since $\mathbf{y}_{\text{ub}} \notin \mathcal{P}$ and $\mathbf{r}_{\text{ub}} \in \mathcal{P}$, it is clear that $t_{\text{ub}} < 1$. The set of $\mathbf{r}_{\text{ub}}$-tight constraints, denoted $A_{\text{ub}} \subset \overline{A}$, contains all constraints $[c_a, \mathbf{a}] \in \overline{A}$ such that $\mathbf{a}^\top\mathbf{r}_{\text{ub}} = c_a$(see Figure 3 below for an intuitive representation of all these notations).

All constraints from A_{ub} are violated by $\mathbf{y}_{\text{ub}}$, because $\mathbf{a}^\top\mathbf{r}_{\text{ub}} = c_a \implies \mathbf{a}^\top(t_{\text{ub}}\mathbf{y}_{\text{ub}}) = c_a \overset{t_{\text{ub}}<1}{\implies} \mathbf{a}^\top\mathbf{y}_{\text{ub}} > c_a$. To prove the required proposition, it is enough to show that one of the rays $\mathbf{r}_{\text{new}}$ constructed by Algorithm 2 leads the IRM sub-problem to a constraint from A_{ub}.

For this, we will first show there exists a small-enough box $B(\mathbf{y}_{\text{ub}}, \epsilon)$ with the following ϵ-*box property*: for any $\mathbf{y}_{\text{ub}}{}' \in B(\mathbf{y}_{\text{ub}}, \epsilon)$, `IRM-subprob`$(\mathbf{y}_{\text{ub}}{}')$ returns a constraint from A_{ub}. In other words, by advancing on direction $\mathbf{0}_n \rightarrow \mathbf{y}_{\text{ub}}{}'$, the first hit constraint belongs to A_{ub} (see Figure 3). We need a formal ϵ-box definition; given $\overline{\mathbf{y}} \in \mathbb{R}^n$ and $\epsilon > 0$, the ϵ-box centered at $\overline{\mathbf{y}}$ is:

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

Let us take any $\mathbf{y}_{\text{ub}}{}' = \mathbf{y}_{\text{ub}} + \mathbf{y}_\epsilon \in B(\mathbf{y}_{\text{ub}}, \epsilon)$. The maximum feasible step length on direction $\mathbf{0}_n \rightarrow \mathbf{y}_{\text{ub}}{}'$, denoted t'_{ub}, corresponds to intersection point $\mathbf{r}'_{\text{ub}} = t'_{\text{ub}}\mathbf{y}_{\text{ub}}{}'$ and to a (first hit) constraint $[c_a, \mathbf{a}]$ that is $\mathbf{r}'_{\text{ub}}$-tight, *i.e.*, $\frac{c_a}{\mathbf{a}^\top \mathbf{r}'_{\text{ub}}} = 1$, or $\frac{c_a}{\mathbf{a}^\top(\mathbf{y}_{\text{ub}}+\mathbf{y}_\epsilon)} = t'_{\text{ub}}$. This latter value $\frac{c_a}{\mathbf{a}^\top(\mathbf{y}_{\text{ub}}+\mathbf{y}_\epsilon)} = t'_{\text{ub}}$ can be arbitrarily close to $\frac{c_a}{a^\top \mathbf{y}_{\text{ub}}}$, because $\mathbf{a}^\top \mathbf{y}_\epsilon$ can be as close to 0 as one needs, *i.e.*, since $|\mathbf{y}_\epsilon| \leq \epsilon \mathbf{y}_{\text{ub}}$, by using a small-enough ϵ, the difference $\left| \frac{c_a}{\mathbf{a}^\top(\mathbf{y}_{\text{ub}}+\mathbf{y}_\epsilon)} - \frac{c_a}{a^\top \mathbf{y}_{\text{ub}}} \right|$ can be as small as one needs. We analyse two cases: (i) $[c_a, \mathbf{a}] \in A_{\text{ub}}$ and (ii) $[c_a, \mathbf{a}] \notin A_{\text{ub}}$. The first case is realisable, as it corresponds to a value of t'_{ub} that can be arbitrarily close to $\frac{c_a}{\mathbf{a}^\top \mathbf{y}_{\text{ub}}} = t_{\text{ub}}$. Case (ii) would lead to a value of t'_{ub} that could be arbitrarily close to $\frac{c_a}{\mathbf{a}^\top \mathbf{y}_{\text{ub}}} > t_{\text{ub}}$ (since $[c_a, \mathbf{a}] \notin A_{\text{ub}}$), and so, strictly greater than t_{ub}. A t'_{ub} value associated to case (i) is strictly inferior to one associated to case (ii). As such, only case (i) can lead to a correct step length t'_{ub}, showing that $[c_a, \mathbf{a}]$ needs to belong to A_{ub}.

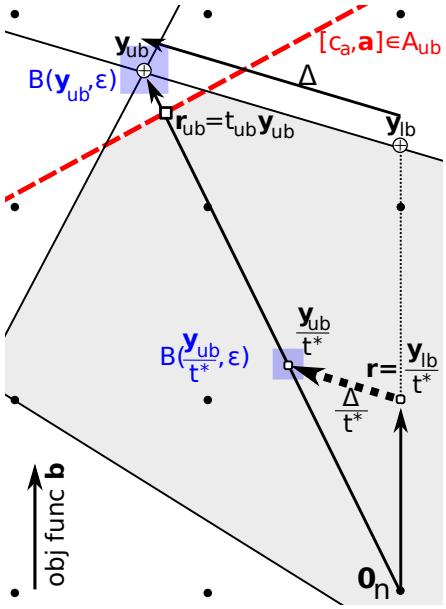

Figure 3: An A_{ub} constraint (dashed red line) that can be found by advancing on any direction inside any (blue) box. $B\left(\frac{\mathbf{y}_{\text{ub}}}{t^*}, \epsilon\right)$ is very small for this value of $t^* > 1$, but it can be arbitrarily large if t^* is small enough.

This only proves the existence of a box $B(\mathbf{y}_{\text{ub}}, \epsilon)$ such that all rays (not necessarily integer) in $B(\mathbf{y}_{\text{ub}}, \epsilon)$ lead the IRM sub-problem to a constraint from A_{ub}. We now show that our ray generation procedure does return *integer* rays $\mathbf{r}_{\text{new}}$ with the same property. Recall from Section 2.3 (Algorithm 2) that such rays $\mathbf{r}_{\text{new}}$ do not even belong to the above box; each $\mathbf{r}_{\text{new}}$ is obtained by rounding some solutions $\mathbf{r} + \beta\Delta$ generated by advancing on the line from $\mathbf{r}$ to $\mathbf{r} + \frac{1}{t^*}\Delta = \frac{\mathbf{y}_{\text{ub}}}{t^*}$, *i.e.*, by advancing along the thick dotted arrow in Figure 3. However, one can find a small enough t_ϵ such that, for any $t^* \leq t_\epsilon$, the box $B\left(\frac{\mathbf{y}_{\text{ub}}}{t^*}, \epsilon\right)$ contains as many integer solutions as needed, because each "edge" of this box has a length proportional to $\frac{1}{t^*}$. Recalling Algorithm 2, we observe that, for any coordinate $i \in [1..n]$ (as chosen in Line 3), $\mathbf{r} + \beta\Delta$ (as calculated in Line 10) can get deep inside $B\left(\frac{\mathbf{y}_{\text{ub}}}{t^*}, \epsilon\right) = B\left(\mathbf{r} + \frac{1}{t^*}\Delta, \epsilon\right)$ and lead, by rounding, to an integer $\mathbf{r}_{\text{new}} \in B\left(\frac{\mathbf{y}_{\text{ub}}}{t^*}, \epsilon\right)$. Since the intersection problem returns the same first-hit constraint from $\mathbf{r}_{\text{new}}$ or $t^*\mathbf{r}_{\text{new}} \in B(\mathbf{y}_{\text{ub}}, \epsilon)$, this first-hit constraint needs to belong to A_{ub}. By the A_{ub} definition, all constraints of A_{ub} are violated by $\mathbf{y}_{\text{ub}}$. □

2.5 Discretization Refining and Theoretical Convergence

The aim of discretization refining is to tackle the following (unfortunate) situation: all the rays *rLst* constructed by the ray generation procedure (Algorithm 2, p. 9) fail to improve any of the bounds. As mentioned in Section 2.4 (see Proposition 1 and the subsequent paragraph), this can only happen if Algorithm 2 finds no integer ray $\mathbf{r}_{\text{new}}$ such that the ray line $\mathbf{0}_n \rightarrow \mathbf{r}_{\text{new}}$ intersects the segment $[\mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}]$. Intuitively, the more distant the segment $[\mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}]$ is from $\mathbf{0}_n \rightarrow \mathbf{r}_{\text{new}}$, the less chances there are to improve any of the bounds. After finishing all rays in *rLst*, `nextRay` returns $\mathbf{0}_n$; this makes IRM activate the condition $\mathbf{r} = \mathbf{0}_n$ in Line 11 (see Algorithm 1) to trigger *discretization refining* in Line 12.

This refining operation is rather simple: multiply $\mathbf{r}$ by $\lambda \in \mathbb{Z}_+$ and adjust t^* so as to keep $\mathbf{y}_{\text{lb}} = t^*\mathbf{r}$ constant; λ is a discretization step parameter (we simply used $\lambda = 2$). This allows the next calls of the ray generation procedure to find more directions $\mathbf{0}_n \rightarrow \mathbf{r}_{\text{new}}$ that get closer to $[\mathbf{y}_{\text{lb}}, \mathbf{y}_{\text{ub}}]$, leading to more chances of optimality gap improvement (see Proposition 2, Section 2.4, p. 10). The clear disadvantage of discretization refining is that the IRM sub-problems become more difficult, as the ray coefficients become larger, *i.e.*, more states to be scanned by the DP intersection algorithms.

Theoretical Convergence Proposition 2 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 $\mathbf{y}_{\text{ub}}$-separating constraint (*i.e.*, that separates $\mathbf{y}_{\text{ub}}$ from $\mathcal{P}$). 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 $t^* < t_\epsilon$, enough to turn *newUb* `true` (*i.e.*, by finding a $\mathbf{y}_{\text{ub}}$-separating constraint as in Proposition 2). However, the stopping condition of this inner loop (*newLb* or *newUb*, see Line 19) can be triggered without upper bound improvement if *newLb* becomes `true` before *newUb*. However, *newLb* becomes `true` only when there is a strict improvement of $\mathbf{y}_{\text{lb}} = t^*\mathbf{r}$, leading to an update of $\mathbf{r}$ in Line 21. This strict improvement condition for updating $\mathbf{r}$ shows that $\mathbf{r}$ can never take the same value twice.

Since there is only a finite number of integer rays $\mathbf{r}$ such that $t^*\mathbf{r} \in \mathcal{P}$ *and* $t^* \geq t_\epsilon$ (for any fixed t_ϵ), 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. Sooner or later, either *newUb* = `true`, or $t^* < t_\epsilon$. As in the first case, the later case would also lead to an $\mathbf{y}_{\text{ub}}$-separating constraint (as above, via Proposition 2). The queue reshuffling in the last line of Algorithm 2 can accelerate the discovery of such $\mathbf{y}_{\text{ub}}$-separating constraints. Indeed, by first providing to Algorithm 1 ray directions close to $\mathbf{0}_n \rightarrow \mathbf{y}_{\text{ub}}$, the constraints cutting off infeasible $\mathbf{y}_{\text{ub}}$ can be more rapidly discovered.

This shows that, as long as $\mathbf{y}_{\text{ub}} \notin \mathcal{P}$, Algorithm 1 can surely find an $\mathbf{y}_{\text{ub}}$-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 $\mathbf{y}_{\text{ub}}$ equal to an optimum $\mathcal{P}$ vertex. Next, we prove that the lower bound $\mathbf{y}_{\text{lb}}$ also converges towards this $\mathcal{P}$ optimum, *i.e.*, after making $\mathbf{y}_{\text{ub}}$ reach the $\mathcal{P}$ optimum, $\mathbf{y}_{\text{lb}}$ can get arbitrarily close to it. 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 $\mathbf{y}_{\text{ub}}$, and so, lead the IRM sub-problem to a lower bound arbitrarily close to $\mathbf{y}_{\text{ub}}$. The IRM stopping condition $\lceil \mathbf{b}^\top \mathbf{y}_{\text{ub}} \rceil = \lceil \mathbf{b}^\top \mathbf{y}_{\text{lb}} \rceil$ can surely be triggered in finite time.

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

A detailed analysis of Algorithm 1 reveals that one only needs to provide the following problem-specific routines to apply IRM on a new problem.

- routine `initialConstraints` (Line 1);
- routine `IRM-subprob` (Lines 3 and 16).

The first routine only provides a list of variables and their domains, and so, we here focus on routine `IRM-subprob`. The cutting and routing problems from Section 3.1 and 3.2 show that that the (IRM) intersection sub-problem is not necessarily more difficult than the (CG) separation problem when solved by dynamic programming. 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.

3.1 Cutting-Stock and Elastic Cutting-Stock

3.1.1 Model Formulation

The well-known Gilmore-Gomory model for `Cutting-Stock` is a direct particularization of (1.1): the pattern cost is always $c_a = 1$ and the pattern feasibility condition is a pure knapsack constraint based on capacity C and weights $\mathbf{w}$: $\mathbf{a}^\top \mathbf{w} \leq C$. `Elastic Cutting-Stock` is a `Cutting-Stock` variation that allows the *base capacity* C to be (slightly) extended by paying a penalty. The larger the excess of the base capacity C, the higher the elastic pattern penalty. A maximum elastic extension of $C_{ext} > C$ is imposed. We 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) at the cost of some additional cost (*e.g.*, for unwanted damage risk).

A maximum elastic extension of $C_{\text{ext}} = 2C$ is reasonable, considering that elasticity can not feasibly more than double the base capacity. Formally, the elastic cost of pattern $\mathbf{a} \in Z_+^n$ is:

$$c_a = f\left(\frac{\mathbf{a}^\top \mathbf{w}}{C}\right), \tag{3.1}$$

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 well-known in the `Cutting-Stock` literature, this problem is not new. First, a stair-case function f can lead to `Variable-Size Bin Packing`, as in example f_{VSBP} below. More generally, the new problem can be interpreted as `Residual Cutting Stock` in the typology of [26] and reformulated as `Bin-Packing with Usable Leftovers`. In the residual or leftover 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, $C_{ext} - \mathbf{a}^\top \mathbf{w}$ can be seen as 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 [24], 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 (3.1); 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 $[0, 1]$) in (3.1). 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 (VSBP)` [26, § 7.13] with $m + 1$ bin sizes.

The main `Set-Covergin` CG model (1.1) becomes:

$$\left.\begin{array}{ll} \max b^\top y & \\ a^\top y \le c_a = f\left(\frac{\mathbf{w}^\top \mathbf{a}}{C}\right), & \forall [c_a, \mathbf{a}] \in \overline{A} \\ y_i \ge 0 & i \in [1..n] \end{array}\right\} \mathcal{P}\ , \tag{3.2}$$

where $\overline{A}$ is formally $\left\{[c_a, \mathbf{a}] \in [1, \infty) \times \mathbb{Z}_+^n \ : \ c_a = f\left(\frac{\mathbf{a}^\top \mathbf{w}}{C}\right), \mathbf{a}^\top \mathbf{w} \le C_{\text{ext}} = 2C\right\}$. In all our implementations, we always imposed $a_i \le b_i, \forall i \in [1..n]$, so as to obtain a tighter model (3.2).

To fully evaluate the interest in the IRM sub-problem with integer rays (Section 3.1.3), let us first present the CG sub-problem of (3.2) and discuss its difficulty for certain functions f (Section 3.1.2). If C and $\mathbf{w}$ consists of bounded integers, the CG sub-problem is tractable (at least by dynamic programming). 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 §3.1.2.1. By only using integer rays $\mathbf{r}$, the IRM can always avoid such very hard computational obstacles by $\mathbf{r}$-indexed DP in the IRM sub-problem.

3.1.2 The CG Sub-problem

The CG sub-problem can be written:

$$\text{SUB}_{CG}(\mathbf{y}) = \max_{\substack{\mathbf{a} \in \mathbb{Z}_+^n \\ \mathbf{w}^\top \mathbf{a} \le C_{\text{ext}}}} \mathbf{y}^\top \mathbf{a} - f\left(\frac{\mathbf{w}^\top \mathbf{a}}{C}\right) \tag{3.3}$$

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

A straightforward CG method for f-`Elastic Cutting Stock` applies an extension of standard knapsack Dynamical 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 $[1..C]$, this interval is simply extended to $[1..C_{\text{ext}}] = [1..2C]$. For each value $\mathbf{w}^\top\mathbf{a} \in [1..2C]$, one evaluates 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.

Compared to the IRM profit-indexed DP (later described in Section 3.1.3), the above weight-indexed DP can often require substantially more states. The number of states in profit-indexed DP is equal to the number of realizable profit values $\mathbf{r}^\top\mathbf{a}$. As long as the values of $\mathbf{r}$ are rather small (*e.g.*, in `Bin-Packing`, IRM starts out with $\mathbf{r} = \mathbf{b} = \mathbf{1}_n$), the number of profit-indexed states can often stay in the order of tens or hundreds. The number of states in weight-indexed DP above is equal to $2C$ (using states recorded in a table of $[1..2C]$ positions) or to the number of realizable total weight values $\mathbf{w}^\top\mathbf{a}$ (using states recorded with linked lists). Since most instances use weight values of hundreds or thousands, the weight-indexed DP can easily require more numerous states than profit-indexed DP (unless $\mathbf{r}$ becomes too large by discretization refining). However, as long as C is not prohibitively large, weight-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 any function f. The applicability of other such algorithms is discussed in the experimental section (§4.1.1).

3.1.2.1 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 input vectors $\mathbf{y}$. We provide below some examples that both consider $\mathbf{y} = \mathbf{w}$ (profits equal to the weight). However, this is only an illustration of more general phenomenons: if the vectors $\mathbf{y}$ are uncontrollable, worst-case scenarios of CG sub-problems can more often arise.

Consider elastic function f_{r} defined by: $f_{\text{r}}(x) = 1,\ \forall x \leq 1$ and $f_{\text{r}}(x) = 1 + (x-1)C - \epsilon \cdot [x = x_{\text{rnd}}]$ if $x > 1$, where ϵ is a sufficiently small value, x_{rnd} is a randomly chosen value and "$[x = x_{\text{rnd}}]$" is the Iverson bracket operator (*i.e.*, it takes value 1 if $x = x_{\text{rnd}}$ and 0 otherwise). We first evaluate $\text{SUB}_{CG}(\mathbf{y})$=$\text{SUB}_{CG}(\mathbf{w})$ on $\mathbf{a} \in \mathbb{Z}_+^n$ such that $\mathbf{w}^\top\mathbf{a} \leq C$; the objective value in such $\mathbf{a}$ is: $\mathbf{w}^\top\mathbf{a} - f_{\text{r}}\left(\frac{\mathbf{w}^\top\mathbf{a}}{C}\right) = \mathbf{w}^\top\mathbf{a} - 1 \leq C - 1$. We now evaluate $\text{SUB}_{CG}(\mathbf{y})$ on $\mathbf{a} \in \mathbb{Z}_+^n$ such that $\mathbf{w}^\top\mathbf{a} = xC > C$; the objective value is $xC - f_{\text{r}}(x) = xC - 1 - (x-1)C + \epsilon \cdot [x = x_{\text{rnd}}] = C - 1 + \epsilon \cdot [x = x_{\text{rnd}}]$. A maximum objective value $C - 1 + \epsilon$ can be reached at $\mathbf{w}^\top\mathbf{a} = x_{\text{rnd}}C$. Unless C and $\mathbf{w}$ consist of bounded integers, the number of realizable $\mathbf{w}^\top\mathbf{a}$ values is easily exponential. In this case, the resulting CG sub-problem requires finding some (random) element x_{rnd} in an exponential-size subset of $[1, 2]$.

More general definitions can be found if the above "$\epsilon \cdot [x = x_{\text{rnd}}]$" term is replaced by "$g(x)$", using any function $g : [1, 2] \to [0, \epsilon)$. This sufficiently small ϵ is needed to keep f_{r} non-decreasing, *i.e.*, ϵ needs to be lower than the difference between the closest two values $x_1, x_2 \in [1, 2]$ such that x_1C and x_2C do represent *realizable total weights* of feasible patterns. Using such definitions, solving the CG sub-problem is exactly as hard as optimizing any bounded function g over a prohibitively large discrete subset of $[1, 2]$.

The fractionality of C and $\mathbf{w}$ play an important role here. If C and $\mathbf{w}$ consist of fractional (or large-range) data, the number of realizable total weights can easily be exponential. In such cases, the CG process can really encounter prohibitively hard sub-problems, *i.e.*, 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 $[0, 1]$. IRM can always avoid such overly-hard sub-problems by only using integer profits. This reduces the set of realizable profits, and so, the number of states in the DP algorithm proposed below (Section 3.1.3).

3.1.3 The IRM Approach: the Intersection Sub-problem via Dynamical Programming

By particularizing Definition 1 from Section 2.2 (p. 6), 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 $\mathbf{a} \in \mathbb{Z}^n_+$ ($\mathbf{a}^\top \mathbf{r} > 0$) such that:

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

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_ub ← ∞                                          ▷ an upper bound that will converge to t*
2: states ← {0}                                      ▷ a linked list of realisable profits p
3: W(0) ← 0                                          ▷ a linked list records the minimum weight W(p), ∀p ∈ states
4: for i in [1..n]
5:     for j in [1..b_i]                             ▷ a ≤ b leads to better IRM bounds
6:         for p in states
7:             if p + r_i ∉ states
8:                 states ← states ∪ {p + r_i}       ▷ add p + r_i to the list of realisable profits
9:                 W(p + r_i) ← ∞
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) ← newW                 ▷ do not completely delete the lost state
14:                cost = f(newW/C)                  ▷ to keep track of precedence relations.
15:                t_ub ← min(t_ub, cost/(p+r_i))
16:            end if
17:        end for
18:    end for
19: end for
20: return t_ub        ▷ An associated pattern a can be built from precedence relations between states
```

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 (realisable) profit. Since all realisable profits are integer, one can use a conventional dynamic programming algorithm with states indexed by profits. The state associated to profit p only needs to record the minimum total weight $W(p) = \mathbf{w}^\top \mathbf{a}^*_p$ needed to realize profit $p = \mathbf{r}^\top \mathbf{a}^*_p$. Any 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 $\mathbf{w}^\top \mathbf{a}_p > W(p)$ has a cost $f(\frac{\mathbf{w}^\top \mathbf{a}_p}{C})$ of at least $f\left(\frac{W(p)}{C}\right)$—we used the fact that f is non-decreasing. Such patterns a_p are hereafter referred to as dominated patterns: their cost/profit ratio is no smaller than $\frac{f\left(\frac{W(p)}{C}\right)}{p}$. The DP pseudo-code is provided in Algorithm 3; we observe that dominated patterns can be safely ignored: any sequence of updates of W (Line 12) generated from $\mathbf{a}_p$ can also be generated from $\mathbf{a}^*_p$. As for the CG sub-problem DP, the practical implementation actually uses linked list to record states,

3.2 Capacitated Arc-Routing

We now present a more complex problem whose *intersection sub-problem* does not apparently seem "vulnerable" to profit-indexed DP. The configurations of the `Capacitated Arc Routing` Problem (CARP) represent paths in a graph; their costs do not depend on weights, but on distances. New problem-specific concepts (*e.g.*, deadheading edges) need to be taken into account. We start out with the classical CARP CG model in

Section 3.2.1, we reformulate it in Section 3.2.2 so as to make the intersection sub-problem more well-suited to profit-indexed DP, and we finish by providing the main DP scheme in Section 3.2.3 (the full pseudo-code is completely specified in Appendix A).

3.2.1 Problem Definition and Classical CG Model

Input Data and Notations 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, *i.e.*, all $e_i \in E_R$ requires service b_i times (set-covering demand), each service delivering a weight (supply demand) of $w_i = w(u, v) \in \mathbb{R}_+$. There is a limit (vehicle capacity) of C on the total route weight (supply delivered). The value of C belongs to $O(n)$ in existing instances, but it can also be unbounded (large-range) in our (scaled) instance versions from Section 4.2.1. 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 set of routes that service b_i times each $e_i \in E_R$ and that minimizes the total (traversal) cost. Please accept a slight notation abuse to lighten the text; $\mathbf{w}$, $\overline{\mathbf{c}}$ and $\mathbf{b}$ represent: (i) column vectors 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 for the route (traversal) cost, but not for the total route weight (load or 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) of $\mathbf{a}$ is $\mathbf{w}^\top \mathbf{a}$. While vector $\mathbf{a}$ does not indicate the non-serviced edges, the total traversal cost $\overline{c}_a$ is exactly 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 a different route that services a_i times each $e_i \in E_R$, but travels a longer distance than a *shortest* $\mathbf{a}$*-route*. The classical CARP dual CG model is:

$$\begin{array}{lll} & \max \mathbf{b}^\top \overline{\mathbf{y}} & \\ \mathbf{a}^\top \overline{\mathbf{y}} \leq \overline{c}_a, & \forall [c_a, \mathbf{a}] \in \overline{A} & \\ \overline{y}_i \geq 0, & i \in [1..n] & \end{array}, \tag{3.4}$$

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

$$\mathbf{a}^\top \overline{\mathbf{y}} \leq \mathbf{a}^\top \overline{\mathbf{c}} + \mathbf{a}_{\mathrm{D}}^\top \overline{\mathbf{c}} \tag{3.5}$$

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

3.2.2 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 (3.5), *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. This reduces the number of states for two reasons. First, the dead-heading cost of a route is naturally much smaller (even 0) than the total traversed distance. Secondly, states with smaller dead-heading cost will be discovered more rapidly and they will be later used to prune many high-deadheading states throughout the DP process. The new formulation only requires a simple variable substitution in (3.4):

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

The objective function $\mathbf{b}^\top \overline{\mathbf{y}}$ simply becomes $\mathbf{b}^\top (\overline{\mathbf{c}} + \mathbf{y})$; $\overline{y}_i \geq 0$ becomes $y_i \geq -\overline{c}_i, \ \forall i \in [1..n]$. The main constraint (3.5) reduces to:

$$\begin{array}{rl} \mathbf{a}^\top (\overline{\mathbf{c}} + \mathbf{y}) & \leq \mathbf{a}^\top \overline{\mathbf{c}} + \mathbf{a}_{\mathrm{D}}^\top \overline{\mathbf{c}}, \text{ or} \\ \mathbf{a}^\top \mathbf{y} & \leq \mathbf{a}_{\mathrm{D}}^\top \overline{\mathbf{c}} \end{array} \tag{3.6}$$

The right-hand dead-heading term is interpreted as follows. We consider a route is an alternating sequence of *service segments* and *segment transfers*. A *service segment* $v_{\text{st}} \rightarrowtail v_{\text{end}}$ consists of a continual sequence of serviced edges that start at $v_{\text{st}} \in V$ and finish at $v_{\text{end}} \in V$. Given two consecutive service segments $v^1_{\text{st}} \rightarrowtail v^1_{\text{end}}$ and $v^2_{\text{st}} \rightarrowtail v^2_{\text{end}}$ (with $v^1_{\text{end}} \neq v^2_{\text{st}}$) in a route, v^1_{end} and v^2_{st} 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 $\mathbf{a}_{\text{D}}^\top \overline{\mathbf{c}}$ becomes $\sum_{\{u,v\} \in T(a)} c(u, v)$, *i.e.*, (3.6) can be written as:

$$\mathbf{a}^\top \mathbf{y} \leq c_a = \sum_{\{u,v\} \in T(\mathbf{a})} c(u, v) \tag{3.7}$$

The IRM model can now be written below with the new $\mathbf{y}$ variables. It fits (1.1) completely; the only term not arising in (1.1) is $\mathbf{b}^\top \overline{\mathbf{c}}$, which is merely a constant added to the objective function.

$$\left.\begin{array}{ll} \multicolumn{2}{c}{\mathbf{b}^\top \overline{\mathbf{c}} + \max \mathbf{b}^\top \mathbf{y}} \\ \mathbf{a}^\top \mathbf{y} \leq c_a, & \forall [c_a, \mathbf{a}] \in \overline{A} \\ y_i \geq -\overline{c}_i, & i \in [1..n] \end{array}\right\} \mathcal{P}, \tag{3.8}$$

where $\overline{A} = \left\{[c_a, \mathbf{a}] \in \mathbb{R}_+ \times \mathbb{Z}^n : \ \mathbf{a}^\top w \leq C, c_a = \text{total } \textit{dead-heading} \text{ cost of a shortest } \mathbf{a}\text{-route}\right\}$. This set $\overline{A}$ is basically defined as in the original model (3.4); the only difference arises in the cost definition: the new $\overline{A}$ replaces classical costs $\overline{c}_a$ with dead-heading costs c_a—computed via (3.7) above. We observe that $\mathbf{y}$ can be negative, but $\mathbf{0}_n \in \mathcal{P}$.

3.2.3 IRM Sub-problem Algorithm

3.2.3.1 IRM sub-problem definition for model (3.8)**.** By particularizing Definition 1 from Section 2.2 (p. 6) to CARP, one obtains the following objective. Given integer $\mathbf{r} \in \mathbb{Z}^n$, determine

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

where the minimization is carried out over all $[c_a, \mathbf{a}] \in \overline{A}$ such that $\mathbf{a}^\top \mathbf{r} > 0$. This latter condition is slightly more important than in other cases, because $\mathbf{y}$ and $\mathbf{r}$ can often have negative components. If $\mathbf{a}^\top \mathbf{r} < 0$, the best feasible solution of the form $t\mathbf{r}$ with $t \geq 0$ is $\mathbf{0}_n$.

This defintion has a straightforward cost/profit ratio interpretation: c_a is the deadheading non-negative cost of any shortest $\mathbf{a}$-route and $\mathbf{r}^\top \mathbf{a}$ is the profit of such a route. Any shortest $\mathbf{a}$-route that provides all service without dead-heading has a cost of 0 and leads to a maximum step length $t^* = 0$. Such a case could arise, for instance, if all required edges could be serviced by a unique route with no deadheading. If all service can be provided without any deadheading, we obtain an optimal primal solution with objective value $\mathbf{b}^\top \overline{\mathbf{c}}$, *i.e.*, this is the minimum cost required to traverse all edges that need to be serviced. The new model (3.8) above reports this objective value at $\mathbf{y} = \mathbf{0}_n$; it can be proved optimal because the IRM sub-problem returns $t^* = 0$ for any input $\mathbf{r}$.

3.2.3.2 Data Structures and State Indexing A DP state s is characterized by the indices below, some of which have been already generally described in Section 2.2:

- 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 $\sum_{\{u,v\} \in T(a)} c(u, v)$, constant for 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 2.2, let $W(s) = W(p_s, c_s, v_s) \leq C$ represent the minimum weight (load) of a configuration in state s, *i.e.*, the minimum load (supply delivered) by an open route starting from v_0 and 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 lower cost/profit ratio t^* ever reached by a state (p^*, c^*, v_0).

3.2.3.3 The DP recursion and comparisons with CG pricing We provide below the main recursion formula for W. Please accept a slight abuse of notation c: in formula below, c refers to the cost of a state and $c(u, v)$ is the shortest path between $u, v \in V$; in general, we also use c_a to indicate the total cost of configuration $\mathbf{a}$.

$$W(p, c, v) = \min \begin{cases} W\left(p, c - c\left(u, v\right), u\right) & , \forall u, v \in V \\ w(u, v) + W\left(p - r\left(u, v\right), c, u\right) & , \forall \{u, v\} = e_i \in E_R, \exists \mathbf{a} \text{ in state } (p, c, v) \text{ } s.t. \text{ } a_i \geq 1 \end{cases} \tag{3.9}$$

This recurrence is exploited by Algorithm 4 (Appendix A, p. 30) that provide the full pseudocode of our DP scheme for the IRM sub-problem. 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 load (total weight) values in $[1..C]$, leading to a pricing complexity of at least $O(C(|E| + |V| \log |V|))$ [16]. 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 [16, § 3.1]—see also the more recent pricing algorithm discussed in [2, § 7.1]. However, the term C is always present; all traditional instances (see Section 4.2) have an average capacity of 119 and the maximum capacity is $C = 305$. On the other hand, state-of-the-art CG pricing algorithms can be very fast in terms of the $|V|$, $|V_R|$ or $|E_R|$ complexity factors.

The number of states in our DP scheme is at most $|V| p_{\max} c_{\max}$, where $p_{\max}$ and $c_{\max}$ are reasonably-large integers in many practical instances (either standard-size or large-range):

- $p_{\max}$ is the maximum number of realisable profits, *i.e.*, no larger (usually much smaller) than the number of realisable profits in a knapsack problem with profits $\mathbf{r}$, weights $\mathbf{w}$ and capacity C.
- $c_{\max}$ is the maximum number of states discovered by the DP process for any v and p. Such states need to correspond to costs $c_1 < c_2 < \ldots c_{\max}$ and weights $W(p, c_1, v) > W(p, c_2, v) > \ldots W(p, c_{\max}, v)$. Given an existing state (p, c_ℓ, v), a new state $(p, c_{\ell+1}, v)$ with $c_\ell < c_{\ell+1}$ can only be useful it uses a lower weight, *i.e.*, only if $W(p, c_\ell, v) > W(p, c_{\ell+1}, v)$. To reduce $c_{\max}$, 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 A).

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

This section performs an IRM evaluation[4] on several *capacitated* `Set-Covering` problems that fit the model (1.1), *i.e.*, on four `Elastic Cutting Stock` variants and on `Capacitated Arc Routing`. For all experiments, we will consider two types of instances: (i) an original standard-size instance and (ii) a *scaled* instance version. Given a standard instance with capacity C and weights w_i ($\forall i \in [1..n]$), the *scaled* version simply uses 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 solvers that divide the weights by their greatest common divisor. Both instance versions use integer data, but the scaled weights can be seen as fractional weights

[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).

recorded with a precision of 3 decimals. Preliminary experiments show that a higher precision (scale factors larger than 1000) would change certain 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 ($\sum_{i=1}^{n} a_i w_i \leq C \implies \sum_{i=1}^{n} a_i(w_i - \rho_i) \leq 1000C$), and (ii) any configuration exceeding C is turned into a configuration exceeding $1000C$ ($\sum_{i=1}^{n} a_i w_i \geq C+1 \implies \sum_{i=1}^{n} a_i(1000w_i - \rho_i) \geq 1000C + 1000 - \sum_{i=1}^{n} a_i\rho_i > 1000C$). 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 "`IT (DS)/TM`$^{\texttt{f}}$", where `IT` is the number of (CG or IRM) sub-problem calls (in parentheses, the number `DS` of discretization refining steps for IRM) and `TM` is the CPU time in seconds; `f` is optional and indicates the number of (if any) failed instances—*i.e.*, *not* solved within the allowed time. If `f` 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).

4.1 Cutting-Stock and Elastic Cutting-Stock

4.1.1 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 3.1). 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 algorithm was chosen because its evaluation on the (rich) knapsack benchmark sets from [21] indicates that `Minknap` "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 correlation between profits and weights can be natural in CG sub-problems. However, even for rather strongly correlated knapsack instances, `Minknap` is one of the best three algorithms from the literature on instances with large weight ranges [21, Table 6-8]. We adapted the code available at `www.diku.dk/~pisinger/minknap.c` to our setting.

`Cplex` 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 each pricing approach above on the four `Elastic` problem variants; we identify a `Elastic` variant by the elastic penalty function, using the notations from Section 3.1.1.

$f_{\mathbf{CS}}$ This `Elastic` variant is the pure `Cutting-Stock`; the pricing sub-problem is the knapsack problem that can be solved by all 3 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 $C_0 = C, C_1, C_2, \ldots C_m$. 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 (3.3). 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 that can no longer be used, *e.g.*, the simple LP bound becomes a quadratic programming bound;

$f_3(x) = x^3, \forall x > 1$ The CG sub-problem is now a Cubic IP that could only be solved by `Std. DP`. The `Cplex` solver does not address Cubic IPs; `Minknap` could not be adapted to it either. 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 perform a deep study 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 C (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.

4.1.2 Numerical Comparison IRM-CG: Standard and Scaled Instances

The results are provided in Table 1 for scaled instances (observe suffix "$_{\text{scl}}$" in the instance name) and Table 2 for standard unscaled instances. Both tables use the following format: the first two columns indicate the instance set (the `Elastic` variant in Column 1 and the benchmark set in Column 2), Column 3 reports the average IRM *computing effort* (in the format "`IT (DS)/TM`", see the third paragraph of Section 4) to reach a gap of 10% between the lower and the upper bound ($\mathbf{b}^\top \mathbf{y}_{\text{lb}} \geq \frac{9}{10} \mathbf{b}^\top \mathbf{y}_{\text{ub}}$), Column 4 provides the average IRM *computing effort* for complete convergence (we stop when $\lceil \mathbf{b}^\top \mathbf{y}_{\text{lb}} \rceil = \lceil \mathbf{b}^\top \mathbf{y}_{\text{ub}} \rceil$). The last three columns indicate the CG *computing effort*, *i.e.*, there is one column for each of the three CG pricing approaches.

The main conclusions of these tables are presented below; we pick out the key claims along the way in boldface.

For scaled (large-range) instances, the IRM is generally the fastest solution, globally over **all** `Elastic` variants. In fact, IRM requires a similar computing effort for scaled (large-range) or unscaled (standard) instances. Confirming theoretical arguments, the 1000-fold weight (scaling) increase leads to a limited IRM time running time increase (in $[-10\%, 30\%]$). The only CG approach that can be applied on all `Elastic` variants is CG[`Std. DP`], but it can not compete with IRM on scaled instances. The 1000-fold weight increase can make CG[`Std. DP`] 10 times slower. CG[`Cplex`] seems quite slow and it can never compete with IRM. Although CG[`Minknap`] reaches results on a par with IRM, it can only be applied to pure `Cutting-Stock` and `Variable-Sized Bin Packing`; more specific `Minknap` comparisons are provided below.

For standard-size instances, IRM represents a very reasonable solution, globally over **all** `Elastic` variants. IRM is generally faster than CG[`Std. DP`], although the number of sub-problem calls is similar. This comes from the fact that the $\mathbf{r}$-indexed DP in IRM is faster than the $\mathbf{w}$-indexed `Std. DP` in CG. Confirming theoretical arguments from Section 2.2.2.1 and 3.1.2, the state space is often smaller in $\mathbf{r}$-indexed DP than in $\mathbf{w}$-indexed DP. Indeed, `Std. DP` indexes states using weight values that are usually (substantially) larger than the values of the rays $\mathbf{r}$ used for indexing DP states in IRM. The very first value of $\mathbf{r}$ is $\mathbf{b}$ (*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 $\mathbf{r}$ can still be generally smaller than values of the (unscaled) weights $\mathbf{w}$. In the first iteration, IRM can also scale $\mathbf{r} = \mathbf{b}$ down if $\mathbf{b}$ contain some very large values (as stated in Line 2 in Algorithm 1). We also provide below a more specific comparison with CG[`Minknap`], limited to pure `Cutting-Stock` and `Variable-Sized Bin Packing`.

Prob. Ver.	Instance Name	IRM *Computing Effort*		CG *Computing Effort*		
		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".*

Prob. Ver.	Instance Name	IRM *Computing Effort*		CG *Computing Effort*		
		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".*

On all four `Elastic` variants, IRM provides reasonable lower bounds rather rapidly. Even for scaled instances, one second is often enough to reach bounds within 90% of the upper bound *for all* `Elastic` variants (see Columns "Gap 10%" in both Tables 1 and 2). One can (try to) compare such bounds with Lagrangean bounds in CG. However, determining high-quality Lagrangean bounds is not always straightforward (see Appendix B), and so, we only performed some comparisons on pure `Cutting-Stock`. A well-known Lagrangean bound in the `Cutting-Stock` literature is the Farley bound—see [3, § 2.2], [23, §. 3.2] or [18, §. 2.1] for interesting descriptions or Appendix B. By investigating the Farley bound values (publicly available at `www.lgi2a.univ-artois.fr/~porumbel/irm/farley`), the CG-IRM comparison can be briefly summarized as follows. On `Bin-Packing` instances `m01`, `m20` and `m35`, IRM reaches a state with $\mathbf{b}^\top \mathbf{y}_{\text{lb}} \geq \frac{9}{10}\mathbf{b}^\top \mathbf{y}_{\text{ub}}$ long before converging (see Column "Gap 10%" in Table 2 for $f_{\texttt{CS}}$), while the CG Farley bound reaches a similar state only towards the end of the search. The most spectacular IRM lower bound is found on the `Triplets`: it is clear that the optimal dual solution $\mathbf{y} = \frac{\mathbf{1}_n}{3}$ is located at the very first IRM iteration, when $\mathbf{r} = \mathbf{b} = \mathbf{1}_n$ and $t^* = \frac{1}{3}$. The Farley bound is far from reaching a comparable performance on these instances. On most `vb` instances, the IRM and the CG (Farley) bounds are however in the same order of magnitude.

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 $y_i + y_j \leq 1$ $(i, j \in [1..n])$; since the values of $\mathbf{y}$ 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 stability. 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 $\mathbf{r}$-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.

4.2 `Capacited Arc Routing`

4.2.1 Benchmark Sets

To our knowledge, there are four `Capacitated Arc Routing` (CARP) benchmark sets that are publicly available (see `www.uv.es/~belengue/carp.html`): `gdb` [11], `kshs` [15], `val` [4] (also called `bccm`) and `egl` [17]. In all cases, the data is integer; 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 4, we generate for each such instance a scaled (large-range) version with 1000 times larger 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 [16, Table 6] show, broadly speaking, that: (i) `gdb` and `kshs` instances both require a relatively-low running time (even less than one second), (ii) the `val` instances requires 10-20 times more running time and (iii) the `egl` instances requires 100-300 more time then `gdb` or `kshs`. Using this information, we impose the following time limits to IRM: 100 seconds to `gdb` and `kshs` (in fact, a rarely reached limit), 500 seconds to `val`, and 5000 seconds to `egl`.

The number of vehicles is not taken into account in our IRM model (3.8), p. 17. It would be possible to add a fleet size primal constraint and a corresponding dual variable in the sub-problem algorithm, but this is not an essential issue for evaluating the IRM. To reduce clutter, please accept the use of an unlimited fleet. This relaxes the original problem, and so, our IRM optimum can be smaller than the CG optimum reported using the original CARP formulation. 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); typically, the fleet size constraint is more important when there are fewer vehicles (*e.g.*, the original `val` instances of class A have 2 or 3 vehicles).

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

Table 3: IRM results for scaled (large-scale capacity) CARP instances. The *computing effort* in Columns 5-6 is reported under the format "`IT (DS)/TM`", see the third paragraph of Section 4. The final values lb and ub represent lower and upper bounds to the IRM optimum.

Instance			10% Gap	Final IRM Solution			OPT_{CG}[a]	OPT_{IP}
Name	n	$\|V\|$	$(lb \geq \frac{9}{10} ub)$	iters/time[s]	lb	ub	$[16]_{\text{bst}}$, $[22]_{\text{bst}}$	(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	313, 235	319
val2c	34	24	91(1)/30	140(7)/169	453	453	528, 457	528
val3c	35	24	104(1)/28	146(5)/97	131	131	155, 131	162
val4d	69	41		time out	418	1164	621, 500	652
val5d	65	34		time out	442	819	689, 547	720
val6c	50	31	158(0)/86	257(5)/459	296	296	405, 299	424
val7c	66	40		time out	265	394	407, —	437
val8c	63	30		time out	431	551	638, —	657
val9d	92	50		time out	299	612	484, —	518
val10d	97	50		time out	434	1340	695, —	735
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	10305*
egl-e4-C	98	77		time out	7621	13093	11416, 11411	11601*
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	16524*
egl-s3-C	159	140		time out	3380	78561	17014, 17014	17234*
egl-s4-C	190	140		time out	4187	92326	20197, 20144	20591*

Table 4: IRM results for standard (unscaled) CARP instances. The CG optima from [16] and [22] (next-to-last column) indicate the best value of all CG bounds reported in these papers. Each CG model defines its routes in slightly different manners.[a] The main particularity of our model is the use of an unlimited fleet size (relaxed primal constraint); our IRM sub-problem (Appendix A) eliminates certain route cycles.

[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 pricing (Appendix A) allows non-elementary routes, eliminates 2-cycles and forbids the repetitive use of the same service segment; the fleet size constraint is ignored. 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.

4.2.2 Numerical Results and Comparison with Existing CG Results

The CARP results are first provided 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 "`IT (DS)/TM`", see the third paragraph of Section 4) 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 "$[16]_{\text{bst}}$" (best CG optimum in columns E and NE in Tables 1-4) and "$[22]_{\text{bst}}$" (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 otherwise an upper bound marked with a star) in the last column.

We provide below some conclusions of Tables 3 and 4, by picking out the key claims along the way in boldface.

IRM can tackle *scaled* instances with 1000-fold larger capacities then in existing standard-size instances, *i.e.*, we are not aware of other CARP work dealing with fractional or large-range capacities. A classical CARP CG algorithm would naturally risk an important slowdown from using a 1000-fold increased scaled capacity of $1000C$. As for all CG pricing algorithms we are aware of, they could generate a state for each value in $[0..1000C]$ and for each $v \in V_R$ (see Section 3.2.3.3). To evaluate the gap between the IRM bound (Table 3) and the IP optimum of scaled instances, one can refer to the IP optimum of unscaled standard instances (last columns in Table 4). Under reasonable route length conditions (see the second paragraph of Section 4), the set of valid routes does not change by scaling. Generally speaking, the IP optimum is the same for both scaled and unscaled versions of the same instance. One can thus observe that the IRM optimality gap is generally comparable to the typical CG optimality gap reported in other papers—see last columns of Table 4, or directly [16, 22].

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 [16, 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 3.2.3.3, the CG pricing routines are usually faster in terms of complexity factors $|V|$ or $|E|$.

IRM provides reasonable CARP lower bounds before fully converging. IRM could be used to produce *some* lower bounds even if the full convergence (either for CG or IRM) takes prohibitively long time, *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 $C = 160000$!). As for the authors of this paper, they are skeptical that a CG algorithm could fully converge on such an instance within a comparable time, at least when a pricing DP with $160000 \cdot 77 \approx 10^7$ states is being used. Even *without scaling*, the standard `egl` instances can require from 10 and 1000 minutes for full CG convergence [16, Table 6]. While the Farley bound is a well-acknowledged intermediate lower bound for `Cutting-Stock`, we are not aware of analogous lower bounds commonly used in `Arc-Routing` CG work.

To solve scaled instances, IRM requires a similar number of iterations as for standard instances, but 2-10 more CPU time. Such CPU time increases come from a sub-problem algorithm slowdown; this is rather due to the noise 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 A) 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.

5 Conclusions

We presented the ray projection technique for optimizing (dual) LPs with prohibitively many (dual) constraints. The resulting Integer Ray Method (IRM) was tested on some `Set-Covering` dual LPs that are typically solved by Column Generation (CG). The IRM differs from CG in that it employs a new ray projection technique, *i.e.*, IRM resides on an *intersection sub-problem instead of a separation* (CG) sub-problem. We present below certain IRM advantages, by underlying the key claims along the way.

IRM can tackle certain large-range (scaled) instances that could require prohibitive time for classical CG. Recall that the IRM rays $\mathbf{r}$ represent the input of the intersection (IRM) sub-problem in the same way the dual vectors $\mathbf{y}$ represent the input of the separation (CG) sub-problem. The advantage of the IRM is that it can more easily control (choose) the input data $\mathbf{r}$ for its sub-problem. The IRM sub-problem is "vulnerable" to $\mathbf{r}$-indexed (profit-indexed) DP if it fits well the general DP framework from Section 2.2.2. As long as $\mathbf{r}$ contains reasonable-sized integers, the IRM sub-problem could thus be tractable, even when *no other integer input* data (weights) is available for state indexing; for such cases, the classical weight-indexed DP can require prohibitive time for the CG sub-problem (see theoretical examples in Section 3.1.2.1, or the large-range versions of `Capacitated Arc Routing`, f_2-`Elastic` and f_3-`Elastic Cutting-Stock`). We are not aware of any stabilized CG algorithm that performs a similar control to keep the dual vectors $\mathbf{y}$ integer.

IRM offers intermediate lower bounds $\mathbf{b}^\top\mathbf{y}_{\text{lb}}$ as a built-in component, while CG only offers an option (not systematically used) to determine (Lagrangean) intermediate lower bounds. The numerical tests show that, for many standard-size instances (of either problem, see columns "10% Gap" in Tables 2 and 4), the IRM can generally reach an optimality gap of 10% (*i.e.*, $\mathbf{b}^\top\mathbf{y}_{\text{lb}} \geq \frac{9}{10}\mathbf{b}^\top\mathbf{y}_{\text{ub}}$) long before fully converging, often in less than 1 second. In the best case, the IRM can even provide nearly-optimal lower bounds from the first iterations. For example, certain `Bin-Packing` instances (*e.g.*, the triplets) have an optimal dual solution of the form $\mathbf{y}^* = t^*\mathbf{b}$ (with $\mathbf{b} = \mathbf{1}_n$), which is always reached at the very first IRM iteration. Recall that IRM starts out by advancing on the fastest-improving direction, *i.e.*, the first ray is $\mathbf{r} = \mathbf{b}$. Such lower bounding features could be very useful when (very large) instances require very long time for full convergence (either for IRM or CG), *i.e.*, the IRM could provide reasonable lower bounds (long) before fully converging.

For standard-size (non scaled) instances of pure `Cutting Stock`*, the results of IRM are on a par with those of CG with* `Minknap` *pricing.* One should be aware that, by using `Minknap` pricing in CG, we actually chose one of the most competitive algorithms for the (famous) knapsack (sub-)problem, selected from a rich literature based on decades of research. This is not the case for our IRM implementation of the $\mathbf{r}$-indexed (profit-indexed) DP for the IRM sub-problem. Generally speaking, an ad-hoc implementation of $\mathbf{r}$-indexed DP allows IRM to solve *both* standard-size and scaled instances within similar (reasonable) running times, as a consequence of the IRM sub-problem tractability discussed above. We also proved the theoretical convergence of IRM (Section 2.5).

If CG can solve a `Set-Covering` *model* (1.1) *using DP for the sub-problem, so could the IRM.* There is no restriction on the column costs (they have large variations in `CARP`), on the covering demand $\mathbf{b}$ (they vary in `Cutting-Stock`), or on other specific configuration properties. We hope that this work can shed useful light for solving other `Set-Covering` problems with large-range weights or fractional input. Last but not least, we provide the IRM practical implementation (code source) and it only requires one to write two routines in order to solve a new problem: (i) a routine for the intersection sub-problem and (ii) a routine that simply lists the dual variables and their bounds. This implementation resides on a common core of IRM routines that implements the IRM pseudo-code from Algorithm 1. In general, one only needs a dual LP model (1.1) for which the intersection sub-problem is reasonably tractable, either by $\mathbf{r}$-indexed (profit-indexed) DP or by any other method. For $\mathbf{r}$-indexed DP, it is preferable to express the column costs c_a as a function of $\mathbf{a}$ only, or, at least, to limit the influence of external terms in the cost expression. For instance, in Section 3.2.2, we preferred (3.8) to (3.4) so as to have lower external (dead-heading) values in the cost expression and more easily exploit ray integrality properties in the $\mathbf{r}$-indexed DP.

Acknowledgements

.....

References

[1] 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.

[2] 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.

[3] 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.

[4] E. Benavent, V. Campos, A. Corberán, and E. Mota. The capacitated arc routing problem: lower bounds. Networks, 22(7):669–690, 1992.

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

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

[7] 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.

[8] 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.

[9] 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.

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

[11] B. Golden, J. Dearmon, and E. Baker. Computational experiments with algorithms for a class of routing problems. Computers & Operations Research, 10(1):47 – 59, 1983.

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

[13] 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.

[14] 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.

[15] M. Kiuchi, Y. Shinano, R. Hirabayashi, and Y. Saruwatari. An exact algorithm for the capacitated arc routing problem using parallel branch and bound method. In Spring National Conference of the Oper. Res. Soc. of Japan, pages 28–29, 1995.

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

[17] L. Y. Li and R. W. Eglese. An interactive algorithm for vehicle routeing for winter-gritting. Journal of the Operational Research Society, pages 217–228, 1996.

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

[19] 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.

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

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

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

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

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

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

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

A Algorithm for the CARP Intersection Sub-problem

The states from Section 3.2.3 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.

The (rather self-explanatory) pseudocode is provided in Algorithm 4 and commented upon below. 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.

[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 [16, §3.1] and VRP [14]. 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. These issues only concern the case $\mathbf{b} = \mathbf{1}_n$, but this arises in virtually all CARP work.

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

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: `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** $(u \in V,\ (p, c, u) \in$ `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 $(p + r(u, v), c, v) \notin$ `states`$[v]$
9: `states`$[v] \leftarrow$`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** $(p, c, u) \in$`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 $(p, c + c(u, v), v) \notin$ `states`$[v]$
19: `states`$[v] \leftarrow$ `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** $(p, c, v_0) \in$`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** $(v \in V,\ (p, c, v) \in$ `states`$[v])$ **do**
27: $wRes = C - W(p, c, v)$
28: $pMaxRes =$ `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: `states`$[v] \leftarrow$`states`$[v] - \{(p, c, v)\}$
31: **end if**
32: **end for**
33: **while** `states`$\neq \emptyset$
34: **return** t_{ub} ▷ An associated route is reconstructed by following precedence relations between states

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 realisable values $wRes \in [0..C]$ in a pre-processing stage.

B 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. We used them for pure `Cutting-Stock` in Section 4.1.2, *i.e.*, the Farley bound is a specialization of the Lagrangean CG bound. While such CG bounds are a very general CG tool, their calculation is not always straightforward, *e.g.*, in `Arc-Routing`, we are not aware of any lower bound similar to the Farley `Cutting Stock` bound. We argue below that these bounds are only well-adapted for problems with certain properties, *e.g.*, constant column costs, fixed upper bounds on primal variables.

We come back to the main `Set-Covering` CG model (1.1), p. 1. Based on work such as [3, § 2.2], [23, §. 3.2], [18, §. 2.1] or [5, § 1.2], CG Lagrangean bounds can be constructed as follows. Consider a given iteration of the CG process with dual values $\mathbf{y}$. We use these values $\mathbf{y}$ as multipliers in the formula of the Lagrangean relaxation, *i.e.*, they are the penalties for ignoring the set-covering constraints. The relation between the CG optimum OPT_{CG} and the Lagrangean bound $\mathcal{L}_{\mathbf{y}}$ can be written:

$$\text{OPT}_{CG} \geq \mathcal{L}_{\mathbf{y}} = \min_{\mathbf{x} \geq \mathbf{0}} \left(\sum_{[c_a, \mathbf{a}] \in \overline{A}} (c_a - \mathbf{a}^\top \mathbf{y}) x_a \right) + b^\top \mathbf{y} \geq \min_{\mathbf{x} \geq \mathbf{0}} \left(M_{\text{RC}} \sum_{[c_a, \mathbf{a}] \in \overline{A}} x_a \right) + b^\top \mathbf{y}, \tag{B.1}$$

where vector $\mathbf{x}$ is composed of values x_a of primal variables for columns $\left[\begin{smallmatrix} c_a \\ \mathbf{a} \end{smallmatrix}\right]$; $M_{\text{RC}} \leq 0$ is the minimum reduced cost at the current iteration, *i.e.*, $M_{\text{RC}} = \min_{[c_a, \mathbf{a}] \in \overline{A}} c_a - \mathbf{a}^\top \mathbf{y}$. For pure `Cutting-Stock`, the fact that c_a is always 1 leads to:

$$\sum x_a = \sum c_a x_a \leq \text{OPT}_{CG}, \tag{B.2}$$

where both sums are carried out over all primal columns $[c_a, \mathbf{a}] \in \overline{A}$. Using this, (B.1) reduces to:

$$\text{OPT}_{CG} \geq M_{\text{RC}} \cdot \text{OPT}_{CG} + b^\top \mathbf{y}$$

The Farley lower bound $\mathcal{L}^F_{\mathbf{y}}$ follows immediately: $\mathcal{L}^F_{\mathbf{y}} = \frac{b^\top \mathbf{y}}{1 - M_{\text{RC}}} \leq \text{OPT}_{CG}$.

This approach could not be (directly) applied to `Arc-Routing`. One can not determine an upper bound of $\sum x_a$ using a formula like (B.2) above. Recall that the considered `Arc-Routing` version does not impose an upper bound on the number of vehicles. A different Lagrangean bound could be determined by imposing some artificial limits on the primal variables $\mathbf{x}$. However, this can require more specific modelling work; if these artificial limits are too large, the resulting Lagrangean bounds are not necessarily very effective.

Regarding `Elastic Cutting-Stock`, one could have replaced (B.2) with $\sum x_a \leq \sum c_a x_a \leq \text{OPT}_{CG}$, but this would actually lead to the same Farley bound *i.e.*, we would use a pure `Cutting Stock` bound to a generalized problem in which the column costs are larger than in pure `Cutting-Stock`.

C `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 [23]: `CSTR10b50c[1-5]*` files[a]
`vb20`	20	10000	$[10, 100]$	$[1, \frac{1}{2}C]$	25 random instances [23]: `CSTR20b50c[1-5]*` files[a]
`vb50-c1`	50	10000	$[50, 100]$	$[1, \frac{3}{4}C]$	20 random instances [23]: `CSTR50b50c1*` files[a]
`vb50-c2`	50	10000	$[50, 100]$	$[1, \frac{1}{2}C]$	20 random instances [23]: `CSTR50b50c2*` files[a]
`vb50-c3`	50	10000	$[50, 100]$	$[1, \frac{1}{4}C]$	20 random instances [23]: `CSTR50b50c3*` files[a]
`vb50-c4`	50	10000	$[50, 100]$	$[\frac{1}{10}C, \frac{1}{2}C]$	20 random instances [23]: `CSTR50b50c4*` files[a]
`vb50-c5`	50	10000	$[50, 100]$	$[\frac{1}{10}C, \frac{1}{4}C]$	20 random instances [23]: `CSTR50b50c5*` files[a]
`vb50-b100`	50	10000	$[1, 210]$	$[\frac{1}{10}C, \frac{1}{2}C]$	20 random instances [23]: `CSTR50b100c4*` files[a]
`m01`	100	100	1	$[1, C]$	1000 random `bin-packing` instances [6];
`m20`	100	100	1	$[\frac{20}{100}C, C]$	1000 random `bin-packing` instances [6];
`m35`	100	100	1	$[\frac{35}{100}C, C]$	1000 random `bin-packing` instances [6];
`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 b and w indicate the (approximate) interval of the demand value, and, respectively, item weights.

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